Advanced optimization-based robot control

Homework 2: Optimal Control Programming

Students:
Claudia Gava [257872]
Matthias Cordioli [258797]

Note: This report explains the code modifications made to the A2_code_path_template.py template and presents a complete analysis of the results. To limit the report's length, the problem's mathematical formulation, already provided in the assignment PDF, is not repeated here. Where necessary, however, the plot code for displaying the results has been changed compared to the original file.

Academic Year 2025/2026

1 Problem Definition with Path Tracking

The first part of the project requires defining the optimization problem presented in the assignment by completing the missing parts of the *A2_template.py* code. In other words, the cost function to be minimized (*cost*) and the constraints it is subject to (*opti.subject_to(eq/ineq)*) must be defined.

The cost function is divided into two parts: the running cost, implemented by modifying the $define\_running\_cost\_and\_dynamics()$ function, represents the costs from step 0 to the penultimate step ($k = N-1$). The terminal cost is defined for the final step ($k = N$) by the $define\_terminal\_cost\_and\_constraints($ function.

1.1 Code completion

In the running cost, I impose the initial and final path constraints:

$$X[0] == x\_init \qquad S[0] == 0.0 \qquad S[-1] == 1.0 \tag{Eq. 1}$$

Inside the loop, I define the dynamic constraints:

$$opti.subject\_to(X[k+1] = X[k] + dt * f(X[k], U[k])) \qquad opti.subject\_to(S[k+1] = S[k] + dt * W[k]) \tag{Eq. 2}$$

Next, I define the actual task constraints, forcing the robot to stay on the desired path. First, I calculate the end-effector position using the kinematics already defined in the *fk()* function:

$$q\_k = X[k][:nq] \qquad ee\_pos = fk(q\_k)$$

Then, I define the x, y, and z positions to create the infinity-shaped path:

$$p\_x = c\_path[0] + r\_path * cs.cos(2 * np.pi * S[k])$$

$$p\_y = c\_path[1] + r\_path * 0.5 * cs.sin(4 * np.pi * S[k])$$

$$p\_z = c\_path[2]$$

and I constrain the end-effector position to be equal to the created path:

$$p\_k = cs.vertcat(p\_x, p\_y, p\_z) \qquad opti.subject\_to(ee\_pos == p\_k) \tag{Eq. 3}$$

The last constraint to define is on the torques, ensuring they remain within a certain range:

$$tau\_k = inv\_dyn(X[k], U[k]) \qquad opti.subject\_to(opti.bounded(tau\_min, tau\_k, tau\_max)) \tag{Eq. 4}$$

Finally, I define the cost function by summing the three terms to be minimized, each multiplied by its respective task weight: joint velocity, joint acceleration, and path velocity.

$$dq\_k = X[k][nq:] \qquad cost+ = w\_v * cs.sumsqr(dq\_k)$$

$$u\_k = U[k] \qquad cost+ = w\_a * cs.sumsqr(u\_k)$$

$$w\_k = W[k] \qquad cost+ = w\_w * cs.sumsqr(w\_k)$$

In the terminal cost, I impose that the final end-effector position coincides with the desired infinity-shaped path. The procedure is identical to the one used in the running cost but limited to the final state. Therefore, the full code is not shown, only the imposed constraint:

$$opti.subject\_to(ee\_pos\_N == p\_N) \tag{Eq. 5}$$

In this first part, specifically, only a terminal constraint is set, not a terminal cost.

Once the code is completed, the script is executed, and the plots obtained from the simulation are discussed.

1.2 Results Analysis

Upon executing the script, the desired infinity-shaped reference path is immediately displayed (Fig. 1).

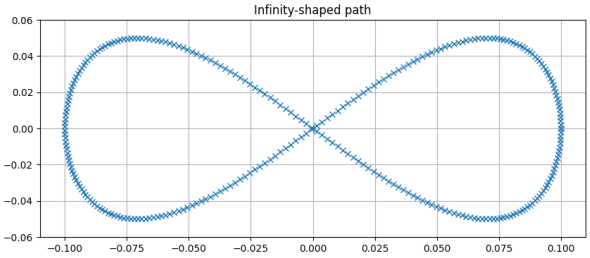
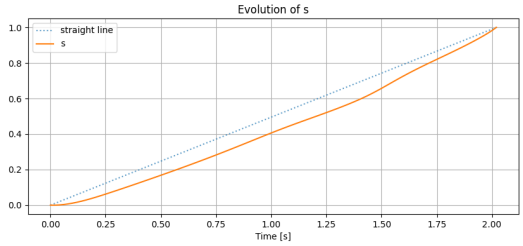


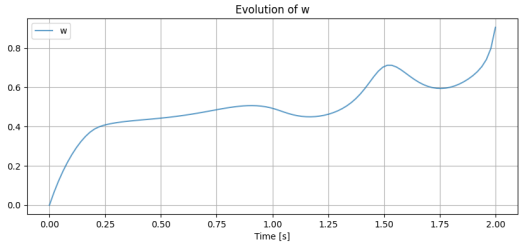
Figure Fig. 1: Reference Trajectory for end effector

Afterward, the result obtained from the optimization for the variable defining the "progression" along the path (s) is shown (Fig. 2). In the problem definition, the initial and final values of this variable were constrained to 0 and 1, respectively (Eq. 1), and it is clear that these two constraints are respected. The variable's evolution is monotonically increasing but not linear, as shown by the deviation from the dashed line in the plot. Since s represents the progress along the defined path, this means the robot continually moves forward without reversing. Because the slope is not constant, the robot advances progressively, starting very slowly and then speeding up.

In conjunction with the s variable, it is useful to immediately analyze the time evolution of the w variable (Fig. 2), which is strictly linked to s: if s represents the progress along the path, w is the velocity of advancement on that path. The plot of w shows the velocity strategy chosen by the optimizer. In the initial moments, w grows rapidly, then slows down, reaching a local maximum at around 1s. Afterward, two phases with similar behavior follow: w first decreases slightly and is then abruptly increased. Regarding the end-effector position versus the reference position, they are perfectly



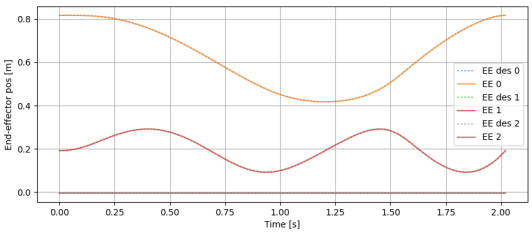
(a) Result of OCP for var S



(b) Result of OCP for var W

Figure Fig. 2

superimposed at every time step. This can be seen both in the plot showing the trends along the three axes and by looking at the overall trajectory (Fig. 3). This means the constraint imposing this was fully respected (Eq. 3) (Eq. 5). Regarding the behavior of the joints (Fig. 4), they all start with a zero



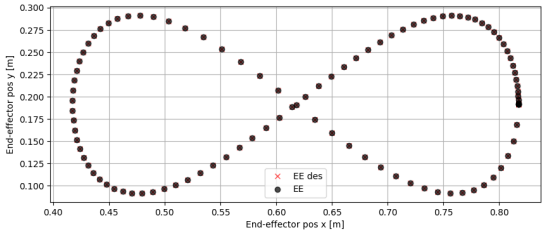
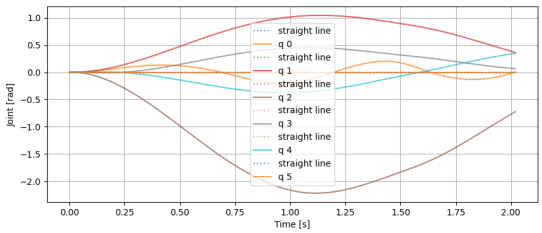


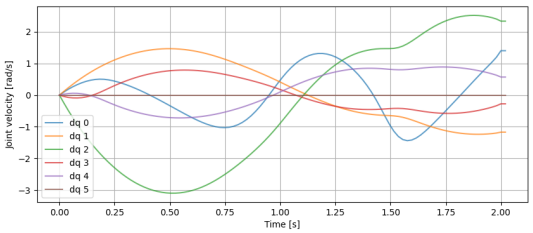
Figure Fig. 3: EE_pos vs Ref_pos along the three axis and along the all trajectory

configuration for the associated angle q. Their evolution over time is irregular, but in general, it can be

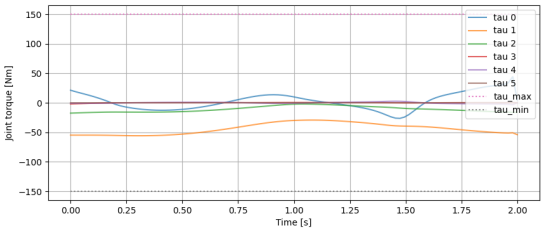
seen as an oscillation of varying amplitude around the zero value. This can be inferred from the behavior of joint 0, which, having a more limited amplitude, intersects the zero axis multiple times during the simulation. If the simulation time is increased (e.g., by increasing N), it becomes clear that other joints also exhibit similar oscillatory behavior, albeit with lower frequencies. The joint angles reach different maximum or minimum values, with the largest angles associated with joints 1 and 2. Joint 5 is the only one that remains constant at 0. In the final configuration, the joints remain at the position they are in at that last instant, as this was imposed as a terminal constraint. This overall behavior of the angles is also reflected in the velocity analysis (Fig. 4), where the initial value is zero for all joints, and oscillations are generally present. The final value is not zero, meaning some joints at most decelerate but do not come to a complete stop. For the velocities, the amplitude of oscillation varies both between different joints and for the same joint over time; in other words, each joint accelerates or decelerates non-uniformly throughout the simulation. In some places, temporary stalls in velocity can be observed, where they seem to settle on a value before starting to change again. As with the positions, joints 1 and 2 have the maximum excursion, and the velocity of the fifth joint is zero. Based on the model in the assignment, lower and upper limits should have been defined for position and velocity, but no saturation is observed for any joint. The torques associated with the motors (Fig. 4) are significantly more pronounced for joints 0, 1, and 2, while being much lower or null for the remaining three components. The torques also lack a regular pattern but nonetheless remain within the established range (Eq. 4).



(a) Joints Angles



(b) Joints Velocities



(c) Joints Torques

Figure Fig. 4

2 Cyclic Adaptation

The second part of the homework asks to adapt the previous work for a potential cyclic use of the robot, enabling it to perform the same action repeatedly while optimizing it only once. What is required is not to demonstrate this repetitive behavior, but rather to ensure it can be performed. To do this, it must be guaranteed that at the end of each cycle, the robot returns exactly to its initial configuration, from which it can restart the next cycle. As seen in the previous analysis, at the end of the simulation, the robot remains in the final configuration it reached, but this is not necessarily the same as the initial one. For this to occur, this constraint must be explicitly imposed, and the assignment requires inserting it as a terminal cost, thus minimizing the distance between the final state $X[N]$ and the initial configuration x_init.

2.1 Edit Code

In the code, the arguments in the $define\_terminal\_cost\_and\_constraints()$ function are modified, adding x_init as a variable. Finally, the *cost* variable, which was previously initialized to 0 and not modified, is redefined by adding the new term to be minimized:

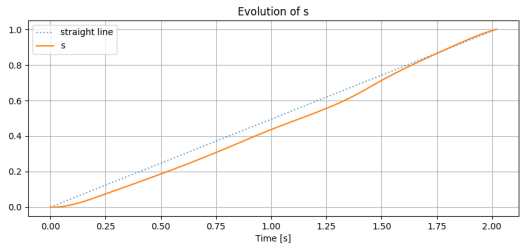
$$cost+ = w\_final * cs.sumsqr(X[-1] - x\_init) \quad \text{(Eq. 6)}$$

where w_final is set to 1 through the following relations:

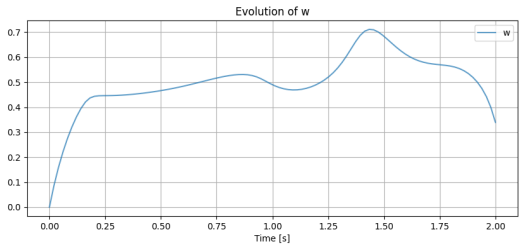
$$log\_w\_final = 0 \qquad w\_final = 10 * *log\_w\_final = 10^0 = 1$$

2.2 Results Analysis

Rerunning the script with this modification reveals several differences, first and foremost a significant increase in the optimization problem's solving time: the solver time increases from about 6 seconds to 74 seconds, which means the change made the problem extremely more complex to solve. First, the behavior of the s variable in the final moments overlaps with the dashed line, meaning the progression speed along the path becomes constant (Fig. 5). As expected, the w variable also changes its behavior in the final part (Fig. 5). It does not show a positive acceleration in the last section, as in the previous case (Fig. 2), but rather a negative acceleration. This means the advancement along the path is slowing down to "buy time" and gradually return the robot to its initial conditions.



(a) Result of OCP for var S



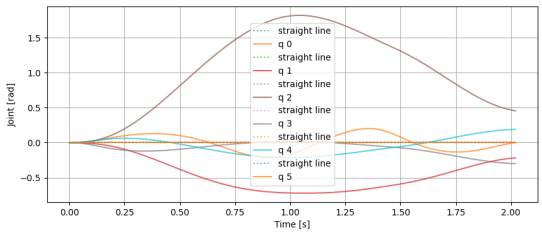
(b) Result of OCP for var W

Figure Fig. 5

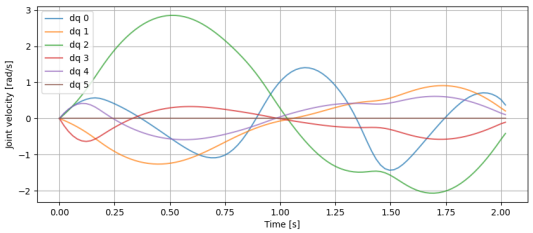
The end-effector position and the reference trajectory are once again perfectly superimposed, and the plots are identical to those in Fig. 3. The angle profiles for the various joints are not the same (Fig. 6), but they exhibit the same oscillatory behavior, albeit with slightly smaller amplitudes. The rotation directions for the joints with the most significant angles have been inverted, and *q5* remains the zero-angle joint. In this case, the dashed line in the plot serves as a reference because it constantly shows the initial position state: it can be seen, in fact, that at the end of the simulation, joint 5 and joint 0 have returned to the initial configuration, while joints 1, 2, 3, and 4 deviate from this value with errors of $0.219rad$, $0.457rad$, $0.302rad$, and $0.189rad$, respectively. By including the return to the initial state as a terminal cost, the optimizer solves the problem by finding the best compromise among all cost functions, and not all terms are necessarily zeroed. Terminal costs are treated as soft constraints and are optimized based on the weight assigned to them. Terminal constraints, on the other hand, are more binding and, as seen in previous analyses, represent a more restrictive limit for the solution. In other words, terminal costs are targets to be met as closely as possible, while terminal constraints are strict, non-negotiable conditions.

The velocity profiles are also similar to the previous ones (Fig. 6), showing an oscillatory behavior. However, in this case, the velocity curves show that all joints in the final moments are converging, with different accelerations, toward the initial zero velocity. At the final instant, this objective is not actually met, and all velocities show an error relative to the zero value, except for joint 5, which has no velocity. The errors in this case for joints 0 to 4 are $0.369rad/s$, $0.208rad/s$, $0.401rad/s$, $0.107rad/s$, and $0.089rad/s$, respectively. In this case, however, compared to the joint position analysis, it is clear

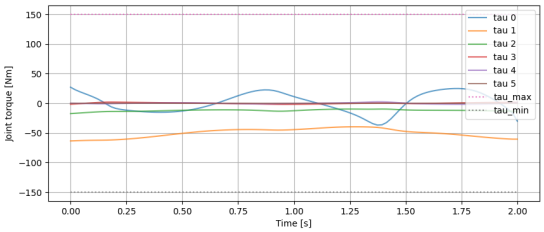
that by slightly increasing the time for the single cycle, the result required by the terminal cost (Eq. 6) would be achieved. The torques exerted by the motors (Fig. 6), on the other hand, are almost identical to those in the previous case (Fig. 4).



(a) Joint Angles



(b) Joint Velocities



(c) Joint Torques

Figure Fig. 6

3 Comparison with Trajectory Tracking

In the third part, the method used to control the robot is modified. Until now, control was based on path tracking, where the path to follow and the total time to complete it were set. Now, we work with trajectory tracking, where a "timetable" is also defined for the robot, specifying that it must be at certain positions at specific fractions of the total time. To do this, the optimizer's freedom to choose how to execute the path (accelerating/decelerating) must be limited by removing the s and w variables from the *create_decision_variables*() function. Previously, the optimization result was a parabolic (non-linear) profile for s, and w was not constant. In this new approach, the reference trajectory must be calculated before the optimization, and the costs must be redefined.

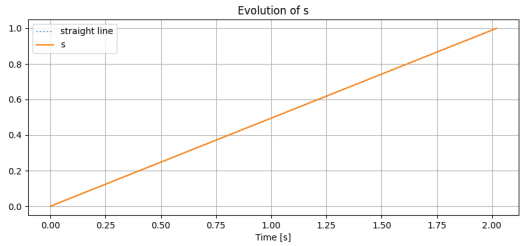
3.1 Edit Code

Summarizing all the modifications made to the code, it can be said that, first of all, s, previously used as a variable to be optimized, is now defined as the linear connection between 0 and 1 (initial and final state):

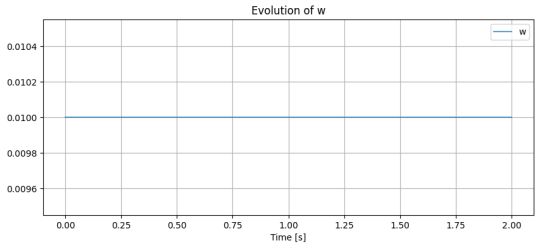
$$s\_ref = np.linspace(0, 1, N + 1) \qquad \text{(Eq. 7)}$$

that is, an array ranging from 0 to 1 in N perfectly linear steps, for which the progress in each single step $\Delta s\_k = s\_k + 1 - s\_k$ is constant and equal to $1/N$.
The reference position calculated as a function of s is defined, as before, by the equations for the infinity shape. Since s is no longer a variable, it is simpler to define s_ref and p_ref directly in the main function and then pass them to the various functions as pre-calculated variables. In the previous points, they were calculated directly within the cost function definitions, as S was available. The geometric meaning of this change remains the same.

As a consequence of this, the constraints on the initial and final states for s (Eq. 1) and the dynamic constraint defining its increment (Eq. 2) were removed from *define_running_cost_and_dynamics*(). The constraints for the initial state of x (Eq. 1), its dynamics (Eq. 2), and the torque limits (Eq. 4) were not



(a) Linear progression of S



(b) Constant progression velocity of W

Figure Fig. 7

modified. The running cost retains the tasks on joint velocity and acceleration, while the part concerning the path progression velocity is removed. This is replaced by a new task for trajectory tracking. Indeed, after extracting the reference position up to the penultimate state p_ref_k, the new cost term is defined by adding:

$$cost+ = w\_p * cs.sumsqr(ee\_pos - p\_ref\_k) \quad \text{(Eq. 8)}$$

dove $w\_p = 10 * *log\_w\_p$ e $log\_w\_p = 2$.

Consequently, the constraint on the end-effector position relative to the reference (Eq. 3) is also removed: the end-effector position is no longer strictly bound to overlap with the desired one. Instead, the optimization seeks to achieve the minimum possible error in the difference between the two positions, which means the correspondence will certainly not be perfect as it was before.

In $define\_terminal\_cost\_and\_constraints()$, the cost term related to the final state (Eq. 6) (the modification made in Part 2 for cyclic behavior) was kept, and a second term was added—exactly as in the running cost—to also optimize the final end-effector position against the final state reference p_ref_N:

$$cost+ = w\_p * cs.sumsqr(ee\_pos\_N - p\_ref\_N) \quad \text{(Eq. 9)}$$

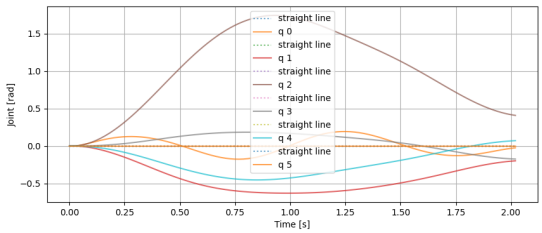
All affected functions were modified by removing s and w from the variables, also eliminating the w_w weight as it is no longer used, and no longer extracting s_sol and w_sol from the solution. This is why s_sol and w_sol are no longer plotted among the results.

3.2 Results Analysis

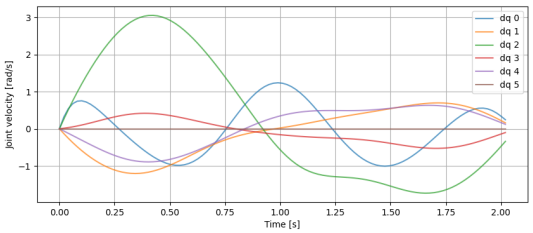
The time required by the solver, in this case, is shorter: about $2.15s$. This drastic decrease compared to the previous case is certainly due to the fact that the number of variables in the problem was halved, and therefore the optimizer is working in a smaller search space.

The behavior of the joints is not so different from the previous case: angles and velocities have an oscillatory behavior that converges toward the end to what should be the desired state (the initial state), but they do not quite reach it, similar to what happened in Part 2.

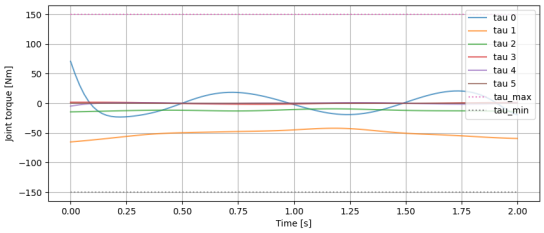
The substantial difference from the previous case is seen in the plot showing the relative positions of the end-effector and the reference (Fig. 9). In this case, as anticipated, there is no perfect overlap between the two values. This is because the perfect correspondence ($ee\_pos == p\_ref$) is no longer explicitly defined as a constraint; rather, the difference between the two positions is treated as a new task for the problem, and the error must be minimized (Eq. 9). This also means that the solver is not forced to have a perfect match between the end-effector and the reference, but can adapt this correspondence to best solve the entire problem and all the tasks that define it, according to their respective weights. To give this task more importance relative to the others, one could increase its specific weight, or, to have a tighter constraint, redefine it as a constraint instead of a cost. However, this would probably be too restrictive for the system. Indeed, a linear increment for s has already been imposed, which, unlike in previous cases, makes the progression velocity constant and non-adaptive. This assumption places a burden on the robot's physical limits (e.g., torques, velocities, max/min joint angles) and, above all, it



(a) Joint Angles



(b) Joint Velocities



(c) Joint Torques

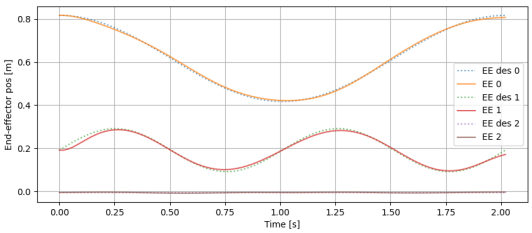
Figure Fig. 8

is not guaranteed that the problem would be solvable with this approach. To verify this hypothesis, an attempt was made to solve the problem by removing the new tasks (Eq. 8) (Eq. 9) and redefining them as constraints:

$$opti.subject\_to(ee\_pos == p\_ref\_k) \qquad opti.subject\_to(ee\_pos\_N == p\_ref\_N) \qquad \text{(Eq. 10)}$$

When running the optimization, the solver immediately appears to require much more time and ultimately fails to converge to a solution (exit message: $return\_status\ is\ 'Infeasible\_Problem\_Detected'$)

Optimizing a cost function, in contrast, seeks a compromise to ensure the solution is also physically feasible. This causes the real end-effector trajectory to not coincide precisely with the reference, especially in curved sections and during the initial acceleration phase, as seen in the two plots (Fig. 9). Obviously, the result obtained is a product of the "timetable" initially imposed with fixed integration steps. This means that by improving the mapping, for example, better results could be obtained.



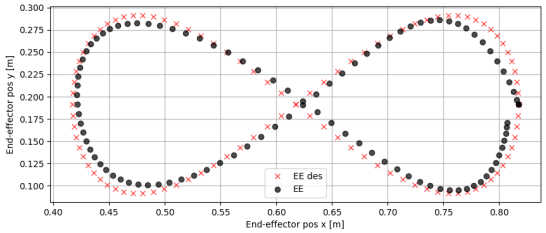


Figure Fig. 9: EE_pos vs Ref_pos along the three axis and along the all trajectory

4 Bonus Question: Minimum-Time Formulation

The fourth point involves revising the previous scripts by adding a cost function for optimizing the time taken to traverse the path:

$$cost+ = (10 * * - 1) * (dt * *2) \qquad \text{(Eq. 11)}$$

where the time step dt is a new decision variable for the problem, bounded between 0.001 and 0.1. For this reason, a constraint is added to bound it between these two extremes:

$$opti.subject\_to(opti.bounded(0.001, dt, 0.1)) \qquad \text{(Eq. 12)}$$

The new variable is then used in the running and terminal cost functions no longer as a constant, but as a variable. These functions are not modified, except for the addition of the time cost (Eq. 11). The *opts* variable is modified in the path tracking case as suggested in the assignment, and in addition to the solver time, the optimal value for dt obtained from the optimization, dt_sol, and the total path traversal time, $N * dt\_sol$, are also displayed in the output. Finally, the weights assigned to the individual tasks are modified as follows:

$$log\_w\_v = log\_w\_a = log\_w\_w = log\_w\_final = -6 \quad \text{(Eq. 13)}$$

Both the script for path tracking (cyclic case) and for trajectory tracking are therefore modified as explained.

4.1 Results Analysis - Comparison

The first analysis that makes sense to perform concerns the time values obtained with the two different methods:

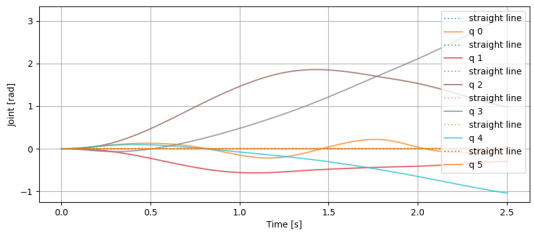
	Path tracking	Trajectory tracking
Optimal dt	0.0248s	0.0316s
Total time	2.48s	3.16s
Solver time	9.28s	6.30s

The solver time is not directly managed in this homework and depends on the solver used rather than the number of variables, which is lower in trajectory tracking, and for this reason, the solver time is shorter. More important is the optimal dt value reached by the optimization, as the total time is derived from it. The path tracking case achieved a lower dt, and therefore this method allows for completing the reference path in less time. Path tracking is faster because it has the freedom to optimize both the shape of the movement (s, w) and the time (dt), whereas trajectory tracking can only optimize the time (dt) for an imposed, sub-optimal trajectory.

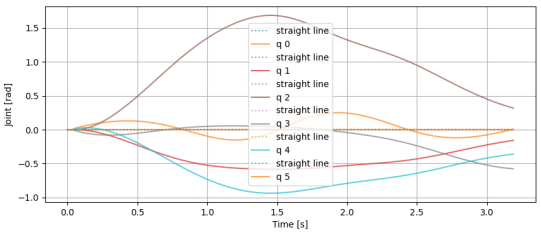
The results seen in the comparison between the actual end-effector position and the desired position are practically identical in both tracking methods, and the result is excellent in both cases. For the path tracking case, this is not new, as it is a result that was achieved from the very beginning (Fig. 3). For trajectory tracking, however, this aspect was a significant limitation on the result that could be obtained (Fig. 9), so in this case, making time a variable led to an improvement. Regarding the evolution of the s variable, on the contrary, in the trajectory tracking case, no differences are seen compared to the fixed-time case (Fig. 7) because s is not a variable, but rather a constant. In the path tracking case, however, the previously limited time likely forced the system to abruptly change its behavior, decreasing its velocity (Fig. 5). In this case, having adaptive time, the system can accelerate less sharply and, above all, does not have to slow down towards the end to return to the initial configuration, showing a behavior for the s and w variables similar to what was obtained when imposing a terminal constraint (Fig. 2).

The joint behaviors are quite different in the two tracking methods (Fig. 10). In path tracking, the angles tend to grow more slowly and then, at the end of the optimization, move far away from the required initial value; the velocities show analogous behavior. In trajectory tracking, the angles grow right from the start, but towards the end, they seem to approach the value requested in the terminal cost. The velocities follow this behavior less closely, growing rapidly at first and then continuing irregularly with various oscillations, without cyclically returning to the initial value. The motor torques, however, are similar in both cases.

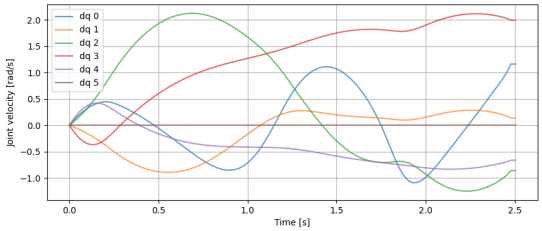
This behavior, at first glance, might not align with what was expected from Part 2, where, for example, the joint angles seemed to only need more time to return to the initial configuration (Fig. 6). However, the only reason this does not happen, despite the extra time, is because the weights of the various tasks were modified (Eq. 13). With a weight for w_final, one can see, instead, how the velocities do return to zero (Fig. 11).



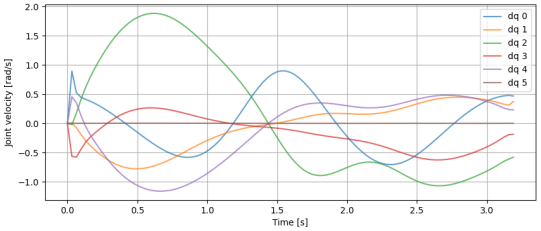
(a) Joint Angles path track.



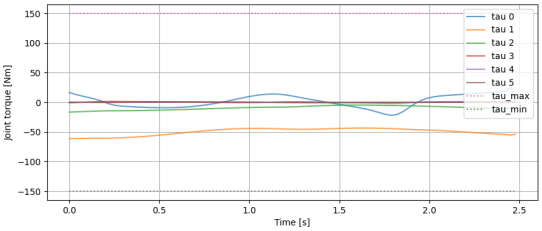
(b) Joint Angles traj. track.



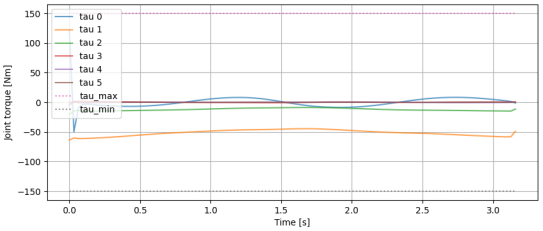
(c) Joint Velocities path track.



(d) Joint Velocities traj. track.



(e) Joint Torques path track.



(f) Joint Torques traj. track.

Figure Fig. 10

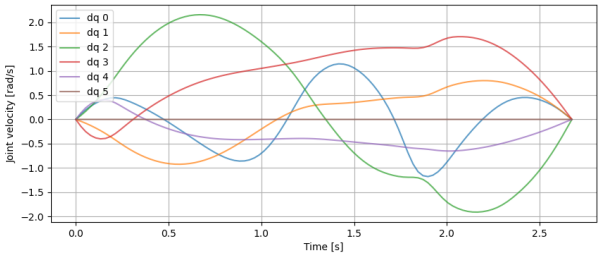


Figure Fig. 11: Joint Velocities path track. with different weight