Master in Control and Robotics

AMORO Lab Report

Simulation of a Biglide Mechanism

Authors:
Conti Fabio,
Pagano Francesco

Contents

1 Computing Trajectories

After ensuring the accuracy of our geometric, kinematic, and dynamic models with respect to the *GAZEBO* simulation, we exploited them through the computation of a *Computed Torque* control algorithm. Such control technique is capable of managing the trajectory of the robot's motion through a set of points in space considering not only desired position in space but also the velocity and acceleration to track during the movements towards the target position. Therefore, the focus of the implementation is to provide a torque command in the joint space capable of regulating the motion of the biglide robot to reach a certain target position by following a predefined trajectory of the end effector in the Cartesian space.
The trajectory we generated for the current application derives from a method of trajectory generation which is capable of computing the coefficients of a polynomial trajectory starting from the initial and final conditions. In our specific case the trajectory for the (x, y) Cartesian coordinates can be expressed as:

$$\begin{aligned} x(t) &= x_A + s(t)(x_B - x_A) \\ y(t) &= y_A + s(t)(y_B - y_A) \end{aligned} \tag{1}$$

with:

$$\begin{aligned} s(t) &= 10\left(\frac{t}{t_f}\right)^3 - 15\left(\frac{t}{t_f}\right)^4 + 6\left(\frac{t}{t_f}\right)^5 \\ \dot{s}(t) &= \frac{30}{t_f}\left(\frac{t}{t_f}\right)^2 - \frac{60}{t_f}\left(\frac{t}{t_f}\right)^3 + \frac{30}{t_f}\left(\frac{t}{t_f}\right)^4 \\ \ddot{s}(t) &= \frac{60}{t_f^2}\left(\frac{t}{t_f}\right) - \frac{180}{t_f^2}\left(\frac{t}{t_f}\right)^2 + \frac{120}{t_f^2}\left(\frac{t}{t_f}\right)^3 \end{aligned} \tag{2}$$

Where:

- (x_A, y_A): coordinates of the starting point of the trajectory.
- (x_B, y_B): coordinates of the ending point of the trajectory.
- t_f: time duration of the trajectory following.

2 Functions

For computing the above mentioned trajectories in the case of the biglide robot we wrote down two functions:

2.1 set_traj()

This function calculates the trajectory in Cartesian space as a *numpy* array given some initial position, final position and duration. These trajectories must be tracked by the robot end effector. The function will output position, velocity and acceleration for the (x, y) Cartesian coordinates with a $10ms$ sampling time. We chose a 2 seconds duration and therefore we obtained 200 samples trajectory.

2.2 compute_traj()

This second function computes the defined trajectory in the joint space. It requires as input the trajectory vectors to be translated from Cartesian to joint space. As a result, these computations depends on the accuracy of the implemented inverse geometric and kinematic models. This function will output the position, velocity and accelerations in the joint space as vectors. The output vectors will be exploited as the reference joint position, velocities and accelerations in the controller.

3 Computed Torque Control

The Computed Torque Control (CTC) is a feedback linearization-based control technique. It uses the robot's inverse dynamic model to find a decoupled solution for individual control of each robot joint. The *Computed Torque Control* law is:

$$\tau = M[\ddot{q_t} + K_d(\dot{q_t} - \dot{q_a}) + K_p(q_t - q_a)] + c \tag{3}$$

where:

- $q_t, \dot{q_t}, \ddot{q_t}$ vectors come from our self-developed function;
- $q_a, \dot{q_a}$ are the values representing the actual joint position, velocities and accelerations;
- K_p, K_d are the proportional and derivative gains. They are positive definite 2×2 matrices.
- M is the inertia matrix.
- c is the Coriolis and Centrifugal matrix.

Since we retrieved all the actual accelerations, positions, and velocities directly from the encoders (which publish data on the robot class), we could easily compute the input efforts in the main control loop by exploiting the control law [3].

Here's the control scheme for the *Computed Torque Control*:

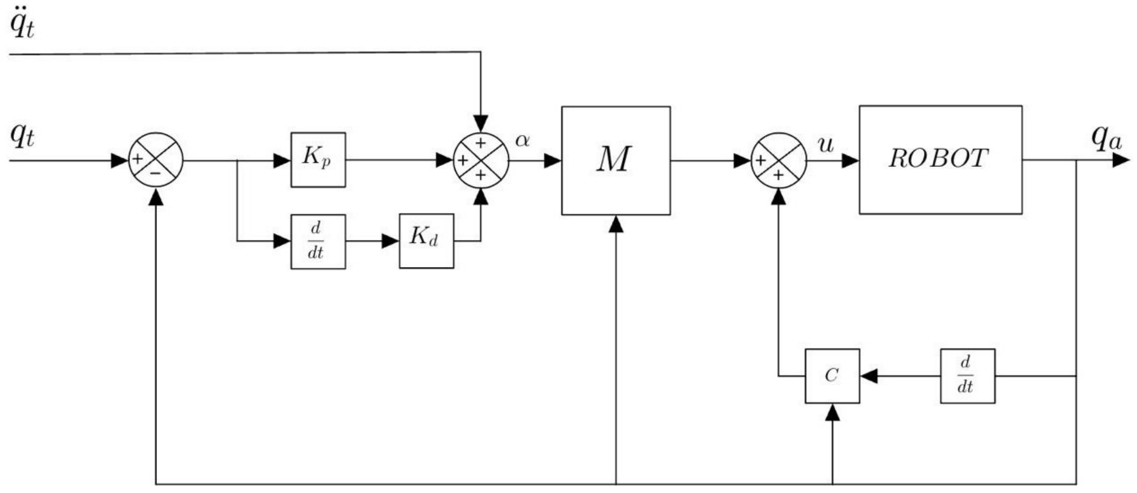


Figure 1: Computed Torque Control Scheme

4 Crossing Type 2 Singularities

The many singularity problems that parallel robots encounter during operations have an impact on how well they perform in terms of workspace, which suffers as a result of these issues. As a result, the workspace is smaller and parallel robot accessibility performance is compromised. Type 2 or parallel singularities are used to describe the most constraining singularities. One or more degrees of freedom become uncontrollable in such a unique combination. Therefore, encountering a singularity of type 2 means that the robot may lose its ability to move from one configuration to another. These singularities are undesirable because of the fact that the control of the robot is lost and the robot is not able to resist any effort or wrench applied to the platform. These singular configurations are located inside the workspace, which make the workspace be divided into different aspects.
In the case of the biglide parallel robot, the type 2 singularity is encountered whenever the mid-line parallel to the two linear actuators is crossed while the configuration $gamma_1 = -1 \quad and \quad gamma_2 = 1 \quad or \quad gamma_1 = 1 \quad and \quad gamma_2 = 1$ are in use. If the end effector would cross this configuration a type 2 singularity would be encountered since the vectors $\overrightarrow{A_1C}$ and $\overrightarrow{A_2C}$ would be parallel. In this case the determinant matrix A, $det(A)$ will be equal to 0 since the two vectors will be linearly dependent in the current configuration. In this case the matrix is not inevitable causing vector of efforts τ to tend to infinity and the robot's motors to brake. The picture 2 shows where the singularity line is:

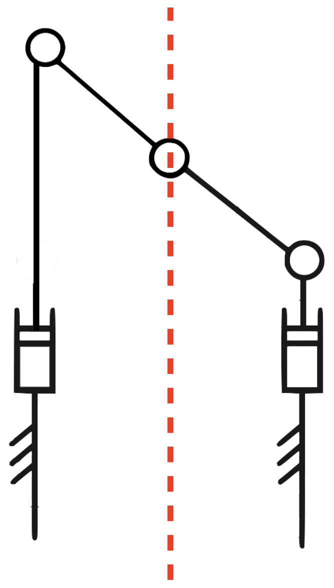
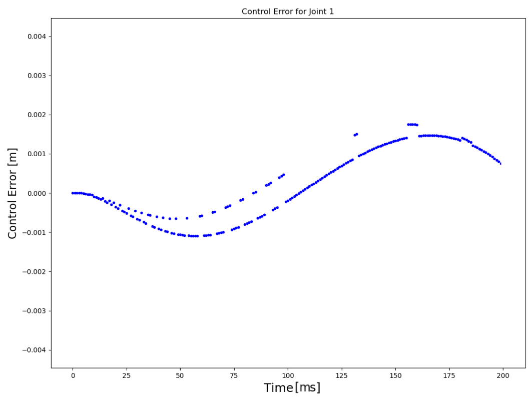


Figure 2: Kinematic model of the Biglide in a singular configuration

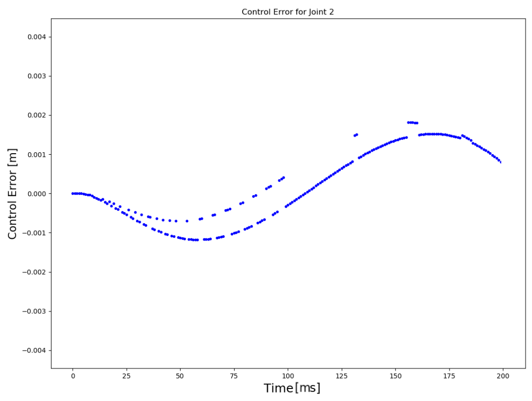
For testing this configuration, we decided to build a trajectory which is supposed to cross the red line passing from a positive x coordinate to a negative one. Since no control is implemented to avoid the singularity, the robot brakes inevitably also in the simulation.
The simplest method we came up with to prevent this behavior is to either apply reciprocal forces to the end effector's direction of motion or to stop the input torques when they cross the singularity line. These torque commands might prevent the robot from braking but they do not impose the crossing of the singularity. The goal would be to implement an advanced control algorithm in such a way that the problem of crossing type 2 singularities is solved and the parallel robot presents an optimal and stable performance around the parallel singularities. One possible approach coming from the literature consists in developing the dynamic condition for passing through the singular configurations. This dynamic condition allows motion generation in the presence of singularity by using an optimal force generation.

5 Results & Conclusions

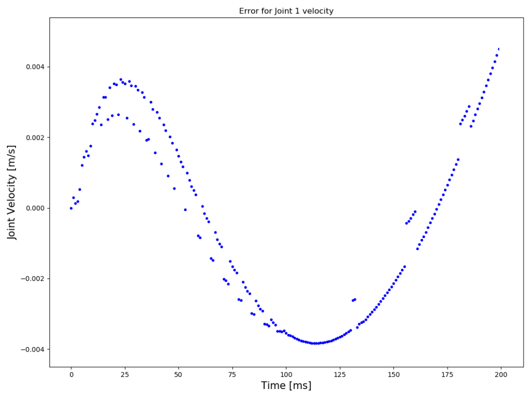
To have a clearer idea about the controller accuracy we decided to plot the position and velocity control errors which the control law tries to minimize. More precisely we obtained Fig 3a and Fig 3b by computing the difference between the reference joint space trajectory and the actual joint position directly obtained from the already implemented robot class. Likewise, we obtained Fig 4a and Fig 4b by computing the difference between reference joint space velocities and the actual joint velocities obtained by the robot class. We were able to create these plots with the help of the *matplotlib* Python library, and it is easy to see from the resulting graphs how small each control error envelop gain is. Therefore we can deduce from the plots that the robot's end effector correctly tracks the precomputed trajectories in terms of position and velocity.



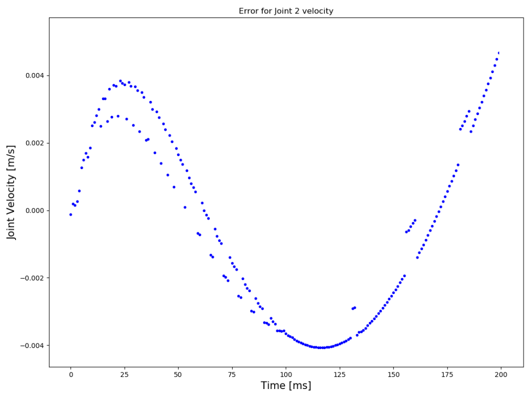
(a) Joint 1 position error over time



(b) Joint 2 position error over time



(a) Joint 1 velocity error over time



(b) Joint 2 velocity error over time

To further confirm the accuracy of the results we ran the *model_eval.py* python script and we noticed that the mean error of the two joints position and velocity were equal until the fourth decimal digit, and this is the reason why we obtained very similar plots for the two joints.

By using a Computed Torque control algorithm, the accuracy of the Geometric, Kinematic, and Dynamic models has been verified. The reference trajectories have been computed thanks to the aforementioned models. Instead, the M and c matrices come from the robot's inverse dynamic model. The control law depends on these matrices. Therefore, the accuracy of the robot models has a significant impact on controller performance. From the position control error plots we can see that the trajectories are tracked with an acceptable error (max error is about $0.002m$ for both joints) along all their duration. The same thing holds for the joint velocities. Thus we draw the conclusion that the robot's correct tracking behaviour of the pre-calculated trajectories demonstrates the accuracy of the presented models in describing the Biglide Kinematics and Dynamics. However, as we can see from the position error plots, the end effector convergence to the final position is not perfect. More precisely, we can state that the robot does not reach the exact desired position at the end of the trajectory.