COLABS REPORT

SPACE ROBOTICS LAB.

Led by Prof. Yoshida Kazuya

GRIEEL STABILITY

legged-robot stability analysis, simulation set-up and software update for the usage of GRIEEL transformable module.

Alessandro Puglisi
[C4TL1107]

15/08/2024

INTRODUCTION

Have you ever wondered what hides inside of volcanic craters, the depth of the sea or remote celestial bodies? The exploration of areas not accessible by humans is crucial for scientific development in several fields: from mining to terraforming but also for climate change insight and to acquire new knowledge. To overcome dangerous and inaccessible environments, autonomous or teleoperated intelligent robotic systems, usually called “rovers” are usually deployed [1, 2].
Different environment calls for different form of locomotion (Fig.1).
Realizing a robot as adaptable as possible helps a lot in the task of exploration.

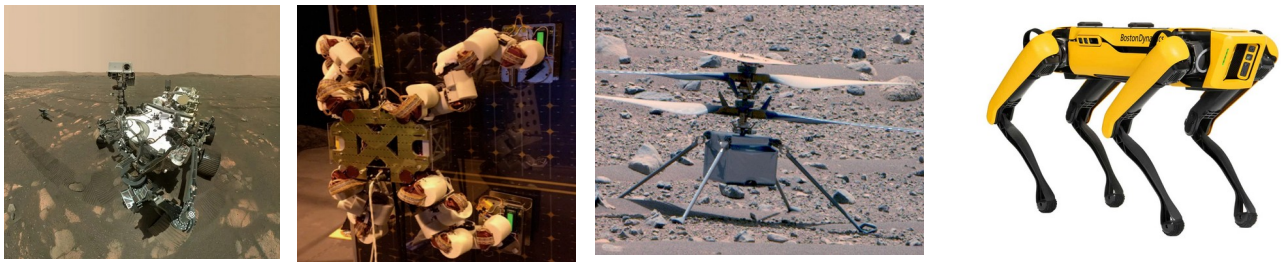


Fig.1: Mobile robot with different locomotion system.

Imagine that your robot is sent to a volcanic area or a celestial body, considering energy efficiency and speed, wheeled locomotion is the most common choice, as several past space exploration missions demonstrate. But wheels are not always the best solution, what if the terrain is extremely rough or with big slopes and walls? In those cases, bio inspired solutions help us to conceive that legged locomotion can achieve robust motion in presence of uneven terrains and even climbing motion to reach locations inaccessible by wheels [1, 2]. Unfortunately, legged robot requires more complex control system and autonomous navigation algorithm, coming also with worse energy efficiency and motion speed. That’s why this approach is not used a lot in space exploration missions especially because, due to the huge investment it is necessary to guarantee robustness and efficiency of the system. So, why not exploiting both locomotion systems? Here comes into play Grieel.
In the field of smart robot locomotion, an example of legged+wheeled locomotion system is the ETH research project [3] with the legged robot ANYmal (Fig.2), using wheels instead of foot as limb end. With this “unnatural” locomotion system (human invention and animal locomotion together), very flexible and agile movement become possible.

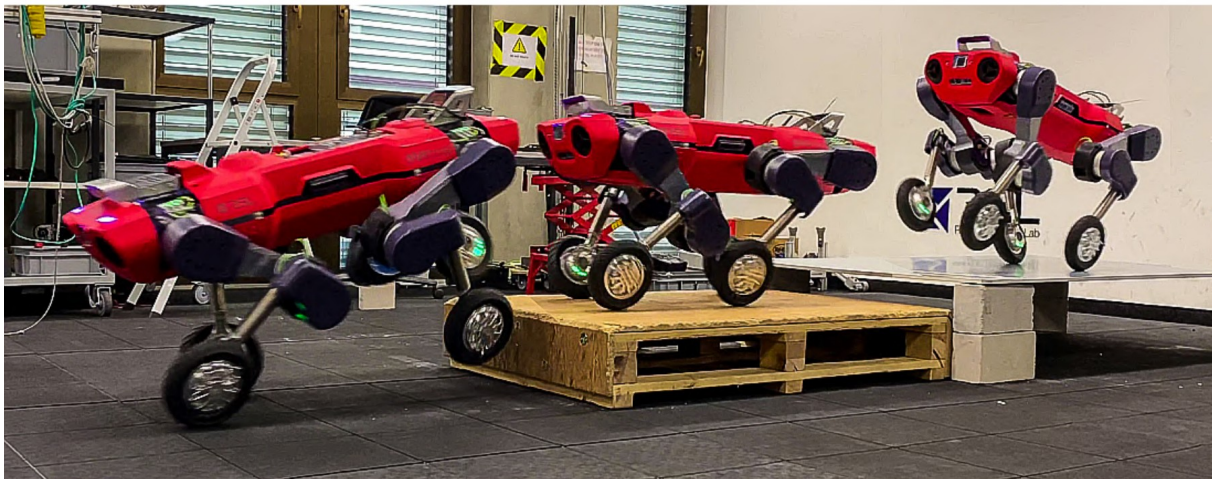


Fig.2: ETH legged robot ANYmal with wheel end-effector

The usage of wheeled legged locomotion is for sure interesting, but not the main focus of Grieel, which has the objective of enabling the usage of 2 different mode for the end-effectors itself, making possible a switch between a legged motion through rocky terrain or climbing with the usage of grippers, or moving quickly in a planar terrain driving on wheels.

Grieel (gripper + wheel) is a transformable module developed at SRL, with the ability of transitioning between two functional modes: wheeled locomotion system and versatile gripper for manipulation and legged motion (Fig.3). Mounting Grieel as end effector (final link of the robot arm) of a legged rover allows to realize both forms of locomotion explained above. In fact, it guarantees and avoids the previously mentioned advantages and drawbacks by transforming accordingly to the terrain characteristic through the usage of visual terrain estimation.

Fig.3: Grieel Module in its two locomotion modes (wheel on left, gripper on rigth).

Currently the transition of all Grieel modules is achieved by lowering and laying the robot body on the ground and rising all legs. Even though this method is simple and practical, it is not robust for two main reasons. In the first case, highly uneven terrain can damage the robot surface. In the latter, the overall procedure is not flexible especially when transforming only one module into gripper mode for manipulation while keeping the others in wheel mode for locomotion. That's why the focus of my research is Implementing a different transitioning algorithm to better exploit Grieel characteristics. This is done with a new algorithm that considers the possibility of transforming while performing motion (for instance, in case of wheeled-to-wall climbing locomotion, the rover can transform the front wheels into grippers and cling on the wall, without losing additional time).

For the new algorithm, legged robot stability is crucial. This is tested on the 4-legged robot LIMBERO (implemented by SRL) with Grieel mounted as end effector (Fig.4) . First, the procedure places the body's center of mass in a stable position by using three end-effectors as points of contact to keep the body above the ground. After that, it performs the transformation of the 4th module and positions it back on the ground. This is done iteratively for every single leg by either following a predefined or more flexible order.

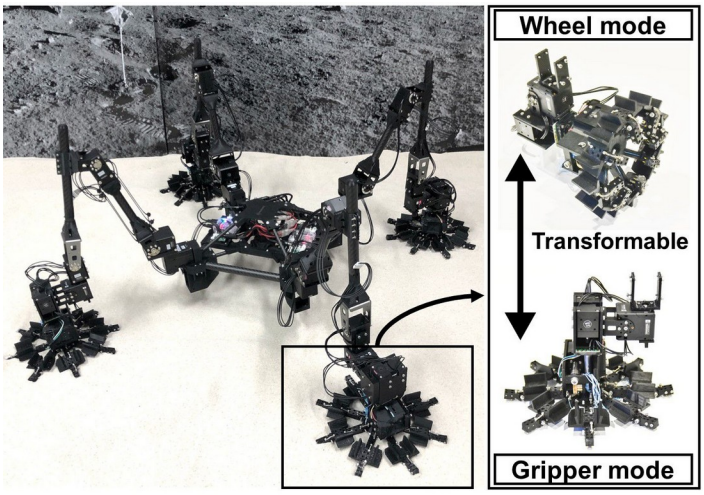
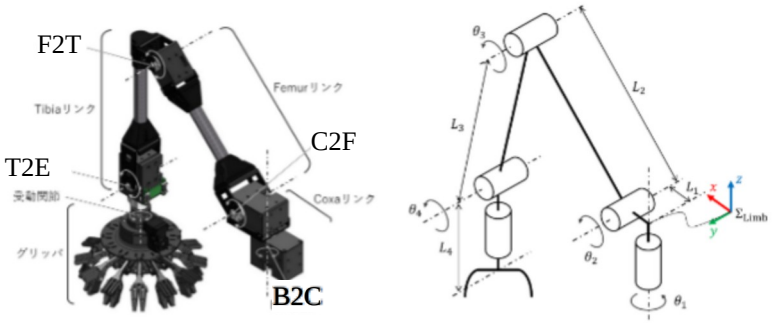


Fig.4: LIMBERO with GRIEEL end-effector

Overall, the goal of this project is to have a smoother transition of the modules to enable better flexibility and improved exploration possibility and capabilities.

LIMBERO (Briefly)

LIMBERO is a 4-legged insect configuration prototype robot of the limb team of SRL. Its structure follow the typical legged robot structure, characterized by a floating base moved by 4 legs. Each limb if seen as a single manipulator has 4 degrees of freedom (DOF) given by 4 joints (actuated by 4 motors) as Fig.4 above represent (not considering Grieel).
Those joints are referred as body-to-coxa (B2C), coxa-to-femur(C2F), femur-to-tibia (F2T) and tibia-to-end-effector (T2E), and are used to move the end-effector in the workspace of the robot. Fig.5 below give a simple representation of the arm configuration and its joints.



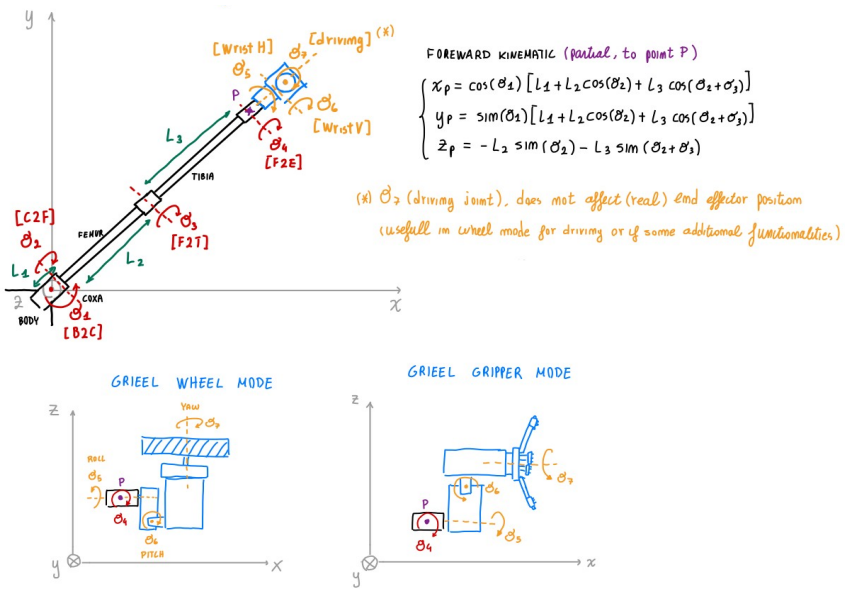
(a) 脚のリンクおよび関節の名称.　　(b) 脚のスケルトンダイヤグラム.

図3.3: 1脚の詳細図.

Fig.5: LIMBERO limb kinematic representation.

GRIEEL PROTOTYPE

At the time I work on this project, GRIEEL module is actuated by 2 joint motors, 1 driving motor and 1 locking motor (Fig.6) . the first three motors give 3 DOF to the end effector as yaw, pitch, roll angle, we call those joints wrist H, wrist V, driving. While locking motor is used to perform the transition between the two modes and to open/close the gripper for grabbing rocks, while walking or climbing.



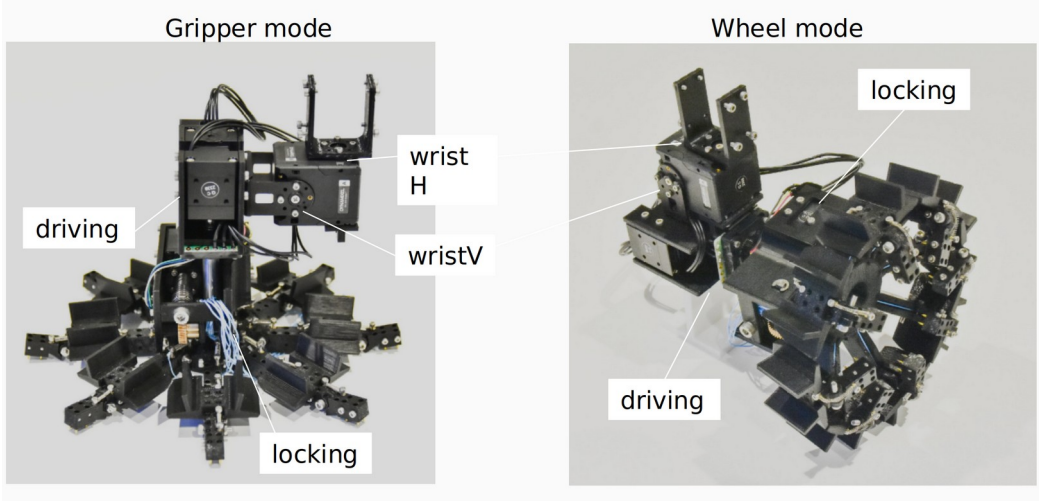


Fig.6: Grieel module, gripper mode on the left, wheel on the right

The Wheel-to-Gripper (W2G) and Gripper-to-Wheel (G2W) transition actuated by the locking motor is achieved using wires that keep the finger closed when in wheel mode and open in gripper mode. So the role of that motor is to keep finger wires in tension or release it by rolling it.

In between the two mode not only locking state change, but also other joint angle, defining two different configurations as seen in Fig.6.
If we take the gripper mode as standard one (and I will keep this choice also in the software development), Wheel mode is obtained by a rotation of WristH of 180° and wristV of 90° and also locking rotation until fully closed.

This model has some disadvantages:

- heavy weight, that require high torque by leg motors to be moved around.
- Unnecessary pitch angle. WristV joint is not needed, LIMBERO limb joint T2E, as seen before, already ensure end-effector pitch angle DOF. Also, wheen the base is up, high torque is required to that joint in order to hold the overall weight.
- Wheel depth is small, decreasing stability and driving capability.

Currently new model is in development by Kohta Naoki, removing the additional pitch (wristV) DOF (which is not needed) and moving the locking motor to the robot body (decreasing severally Grieel weight), with the usage of a BMX breaking system.
Also, wheel depth is increased thanks
to a different mechanical design (Fig.7).

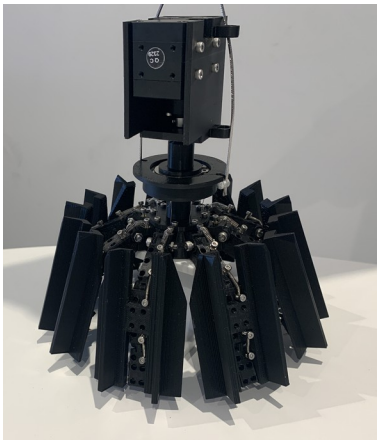


Fig.7: New Grieel module

RESEARCH STEPS

As said in the introduction, the final goal is achieving a smooth transition of limb end effector Grieel, maintaining stability.
Before entering into details of theory and implementation, I want to briefly summarize the main steps that I take to reach the final goal, to clarify the importance of each step discussed and what comes next.

1. Study the basics of legged robot.
2. Learn ROS2, ros2_control and Gazebo.
3. Understand how LIMBERO software works.
4. Update LIMBERO software to control GRIEEL.
5. Deep study of Gazebo parameters for reliable simulation.
6. Define and implement Grieel transition algorithm.
7. Test algorithm in simulation, and fix ros2_control gains.
8. Tune controllers of real motors (Dynamixel)
9. Single Limb experiments.
10. Whole Robot experiments.

I will skip the part regarding the self-learning of the necessary background knowledge to face the project, and just give some quick theoretical insight needed to approach each sub-problem.

SOFTWARE UPDATE

The software infrastructure is based on the Robot Operating System ROS2 [4], a very common middleware used for robotics application. ROS2 is the second version of it, as a completely new implementation (not just an update) that improve some aspect of the previous one, like real-time reliability and communication without the need of a “central entities” (the roscore).
In very simple words, ROS2 is used for message exchange between computation unit (nodes), through channels (topics). It is based on a publish and subscribe communication logic, a publisher “send” a message of any type (standard, custom or imported from libraries) on a topic (named channel), than the subscriber listen to that topic and when a message arrive, it calls a specific function (callback) and use that information to make something. This publisher and subscriber entities, and also callback function and internal variable are defined inside nodes, which are simple programs written in C++ or Python (currently LIMBERO SW uses mainly C++ thanks to the faster low level computations). Another advantage of ROS2 is that nodes in C++ and python can communicate between each other.
In detail, each node will be a C++ class, with its own attribute as useful variables and method as callback functions and for other useful computation.
In a complete ROS2 based project, several nodes are established and more connection between them is created, leading to a complex interconnection of nodes through topics, this form what is called the ros network.

As a common programming practice, programs are separated accordingly to the function they have in the overall project, nodes with similar purpose are collected inside packages, and packages (that can be grouped in meta-packages) are stored inside the ros2 workspace.

Briefly, regarding simulation aspect (we will see it in details later on), the simulator used is Gazebo, a reliable physical simulator that from robot model and physical parameter (like mass, friction, etc) simulate the evolution of the system by solving set of differential equations.

In between ROS2 architecture and simulation there is ros2_control, a framework that is used to simplify the coding of controllers (feedback laws like PID joint controller, differential drive controllers etc are already implemented, just tune the free parameters). In our case we use the joint_trajectory_controller of the framework, which realize the low level feedback PID control of the joints.

I will list the main packages that compose LIMBERO architecture [6] (ros network/ communication graph in Fig.8, easy to read high level graph in Fig.9) , and give an high level description of the necessary update to manage the additional DOF and hardware functionality inside square brackets: (notice that here each package is made up of just one node, with the same name as the package)

>**TOOLS**:

- **lbr_command_interface**: used to send high level commands to the robot, like moving the base or end-effector in a specific way. Those commands can be sent by joystick (buttons mapped properly) or by command line.

[additional commands like single Grieel module transition, all module transition algorithm, driving, etc needs to be mapped]

- **lbr_sim**: when in simulation map single limb state (contact and configuration) to a whole joint state for easier management of it. If real robot, collect into a single data structure the overall encoder state from the motors. Also, after collecting the joints, it is responsible of publishing it to the joint_trajectory_controller node established by ros2_control, in the proper topic.

[additional wristH, wristV, driving DOF pf each limb has to be included]

- Inside TOOLS meta-package, many useful launch file are collected, a **launch** file is a python (or xml) script that define a specific set of nodes to run and the configuration of each.
[several launcher modified to store the initial configuration of GRIEEL when program is launched, and stored into internal node parameters. This information is useful also in the simulation environment to load the correct model, as we will see later on].

> **SLAM**:

- **lbr_state_estimator**: based on the overall joint state (collected by lbr_sim) , it is used for visualization purpose (publish each limb end-effector motion command as marker information for rviz2, a visualization tool used with ROS2). In order to send end effector state from joint one it rely on direct kinematic, which has the purpose of mapping joint state to end-effector (operation space) state, from the kinematic description of the manipulator (limb) [5].
Another crucial function of this node is using the end effector position to estimate the support polygon of the robot (a geometrical entity that I will describe later, essential for stability analysis).

[again, as done for lbr_sim, this node require an update on the overall list of the robot joints]

> **PARAM**:

- **lbr_description**: collection of the robot model description, in terms of mesh (3D representation of the links) and URDF (Unified Robot Descriptive Format), which is a standard way to define how links are interconnected by joints using xml format. Also URDF contains information useful for Gazebo simulation (like physical parameters) and other plug-in. This description is useful not only during simulation but also for rviz2 to have a real-time visualization of the robot in the screen,

looking at the evolution of each reference frame and with possibility of visualizing additional useful information. To keep URDF clean, this is written using the xacro formatting method, that allow to divide the xml overall description in multiple file (and also use some variables and macro-functions) that later on xacro program will merge in a single one. Also ros2_control set-up parameters are defined inside a xacro file and ros2 controller gains in a yaml configuration file.

[URDF of GRIEEL is needed to simulate LIMBERO+GRIEEL, my project mate Kohta Naoki make the mesh and URDF of wheel and gripper mode for Grieel. Also URDF has to manage which Grieel Mode we want to load, so using xacro syntax I set-up some if/else condition all over the different files. ros2_control related urdf is update with the new joints]

- **lbr_messages**: contains all useful custom messages like limb task and base task (definition of xyz, rpy movement of base or end-effector).

[I add some messages for management of Grieel state like a mode change message that store the limb id and the new mode]

> **CONTROL:**

- lbr_high_level_controller: map user command to low level messages in terms of base and limb task. Is also responsible for other high level functionalities like checking if base center is inside support polygon received from lbr_state_estimator (I'll explain later why it is useful).

[New Grieel commands are mapped with set of base/motion tasks, some Grieel joints actuation is also performed here accordingly to the user command (a better implementation should require to define some custom Grieel task and an additional package for managing Grieel joints, but my plan is to consider wristH,wristV as part of limb joints and not separately. Grieel transition algorithm high level commands and logic are written here separating it in 2 methods, a full algorithm function and a single limb transition. In this way the algorithm is easy to modify accordingly to necessity].

- **lbr_low_level_controller**: map low level command to limb command, as end effector position of each limb. Received a base task in terms of xyz, rpy this node map it to required end-effector position using geometrical computation, similarly from a single limb task it publish the end-effector sequence of positions. This end-effector trajectory is properly fitted using the initial pose, final and a middle pose if defined, using a 7 degree polynomius trajectory.

[I ass some callback function here to map high level Grieel command as single limb command accordingly to the specified limb_id]

- lbr_limb_controller: single limb controller, received the end-effector trajectory step by step rely on inverse kinematic (from end-effector desired position find joint state required angle) to map this into the required joints state, needed for actuating the motors. Also use direct (forward) kinematic to update at runtime the current joint state (when encoder information is not available, so in simulation). So it is the lower level part of the controller before the hardware actuation.

[Again as in other nodes, joints updated including Grieel ones, also in terms of upper/lower limits. Grieel trajectory received from low_level is here collected in the overall limb joint state than used by lbr_sim. At this point inverse kinematic is update considering a fixed Grieel joint angle accordingly to the state of an internal parameter that keep track of the state of the module. This is a simplification due to the complexity of solving inverse kinematic]

- **lbr_dynamixel_controller:** In principle, commands computed by ros2_control can be mapped directly to the motors, but this require the definition of an hardware interface, which is not straight forward. That's why at the current state of the Software architecture, motor commands are sent by this additional node, responsible of writing each lbr_limb_controller joint state to the correct port where the motor is attached, and reading the encoder state. To write this functionality the library dynamixel_sdk has been used, written directly by ROBOTIS [7] to control Dynamixel motors.

[This code again is updated with the new joints, plus other update due to the different control of driving and locking motors, which are controlled at velocity level (respect the others at position level). Additional methods to read and write velocity have been written, and also reading current become useful as we see later on. Also due to the fact that I proceed with a retuning of Dynamixel PID gains, a method to write in the correct address the new gains when dynamixel are setted-up is needed].

Other packages form the overall LIMBERO architecture, but are out of interest for my analysis, notice that whenever the overall robot is controlled, a launch file start one instance of the node explained above (from the package with that name) except for lbr_limb_controller which is launched in different namespaces as /LF, /LH, /RH, /RF and the same for lbr_dynamixel_controller.

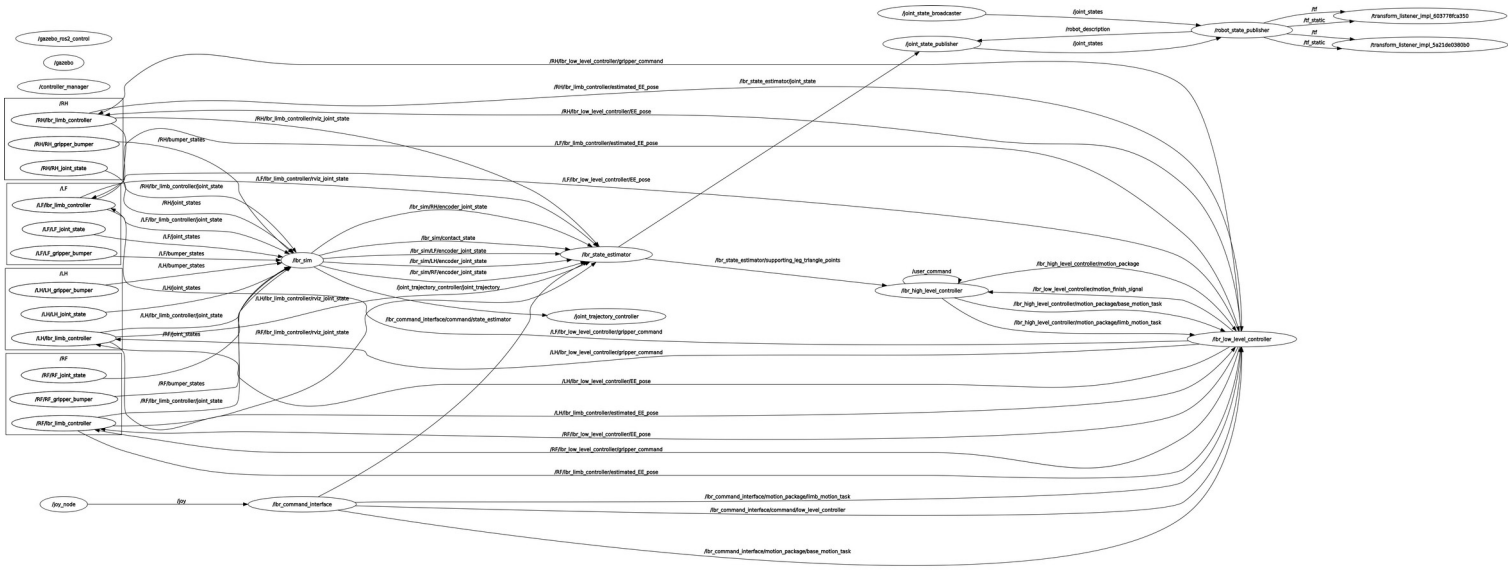


Fig.8: LIMBERO rqt_graph (ros network representation)

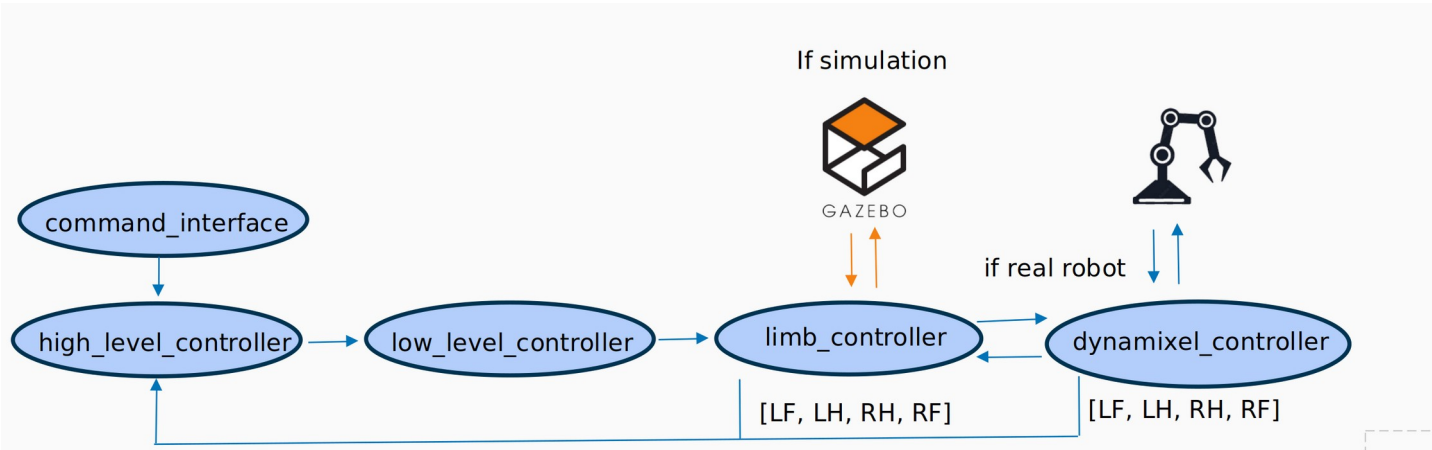


Fig.9: LIMBERO simplified communication graph.

SIMULATION SET-UP

Simulation is a crucial part when developing the robot prototype, each commands and algorithm should be tested in a reliable simulated environment before working with the hardware. Money and assembly effort comes with hardware, before using it we need to be sure that everything will be fine as much as possible (some inconsistency is inevitable).

As summarized in the previous section, the simulator we use with ROS2 architecture is Gazebo (which exists as a stand-alone outside of ROS).
Even if in terms of graphics this is not as astonishing as other simulator that rely on usage of video game engines (like isaacsim), in terms of reliability in many cases it is able to provide a realistic simulation by solving a set of high order differential equation obtained from the model description and the physical settings (Both for the word and the entities populating it).

After the update of the URDF attaching Grieel as end-effector and updating ros2_control joints, using the default entity spawner, the robot collapsed on itself and nothing worked.
Several debugging procedure show me that the problem was not in the control code updated but in the simulation and control gains parameter, that required fine tuning to make the model more realistic and the controllers stable for the overall new mechanical structure.
I perform a fine trial and error iterative tuning of PID gains of all limb joints, as I will explain later on, until the simulation was stable, and than retune accordingly to some desired tracking performances and using an intuitive tuning procedure defined by myself.

Meanwhile, I also fine tune physical parameters used by gazebo, defined in the URDF by some gazebo reference tag. In fact gazebo uses not only the inertial tag of mass and moment of inertia, but also joint friction and damping and ground-end-effector contact friction and damping are important for a realistic simulation.

After several retuning of this (briefly explained in the section related to the parameter tuning), I am able to spawn the robot both in gripper and wheel mode in the Gazebo environment.
And have a real-time visualization of it in rviz2.
(rviz2 is just a tool for visualization of link position and joint state, while Gazebo is supposed to be the real environment, a substitution of your real robot).

Finally, in order to make the simulation more “entertaining”, which seems useless but can be crucial when showing the project to costumers to capture their interest, I set up a mars-like word, with a mesh base as mars surface and some spread rocks to simulate a possible environment where Grieel module power can be exploited. Unfortunately due to time limitation I am able to spawn properly in this world just in gripper mode, a
fine tuning of contact state between wheel and ground is needed to spawn correctly also wheel mode.
Notice that for simulation and check of the working of the algorithm I faced the big limitation of gazebo of wire simulation. The real open/close motion of Grieel cannot be simulated because wire simulation is very hard and gazebo doesn’t allow it.
Also simulating each finger joint for each module is a crazy request (8 finger per module, with 3 joints per finger, 24 additional joints per module). Also, a runtime update of the URDF is somehow possible in rviz2, but almost impossible in Gazebo, after one month of research and tentative of changing the model at runtime after rising one limb (I don’t go into the details of the Gazebo software solution, because of the final fail of all the tries), I give up on this runtime model update and just test LIMBERO+GRIEEL as a sequence of motion starting from Wheel or Gripper mode.

Maybe a different simulation environment should be used if a 100% reliable stability check is required.

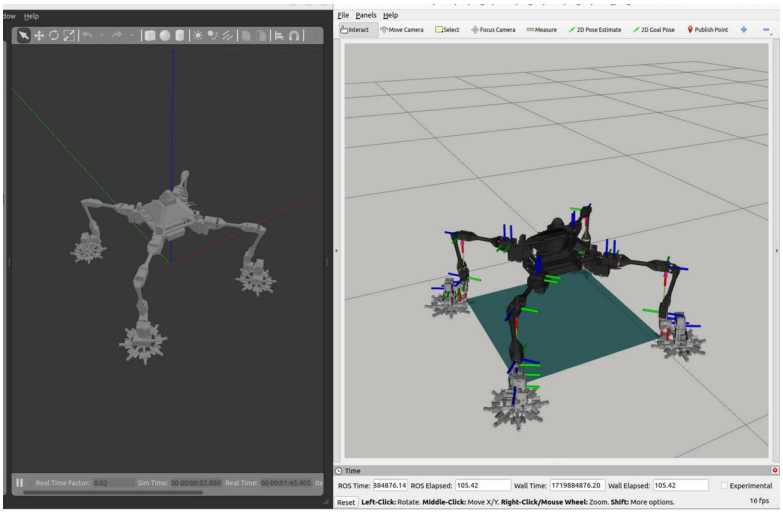


Fig.10.1: LIMBERO+GRIEEL in gripper mode

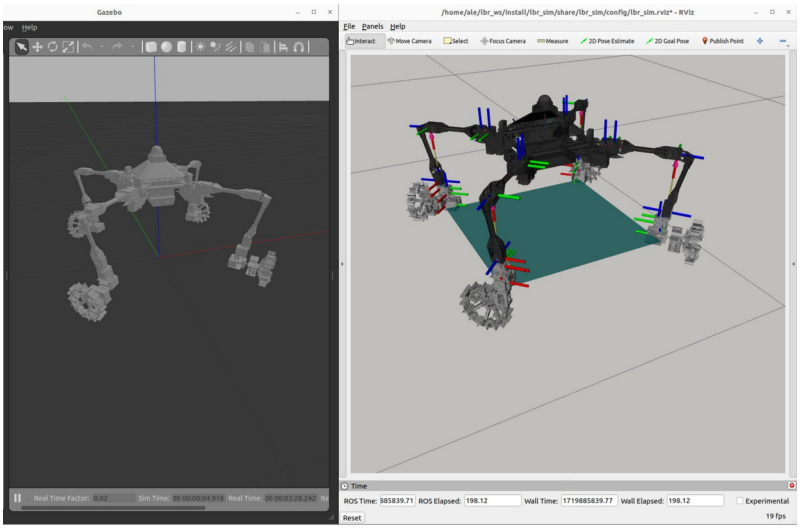


Fig.11: LIMBERO+GRIEEL in wheel mode

Fig.10.b: Mars Gazebo environment simulation

STABLE ALGORITHM

Once both ROS2 software architecture for the control of the robot and simulation are working, It is time to ideate and implement the algorithm that I briefly explain in the introduction.
In order to achieve my desired sequence of transformation keeping the body up, the stability of legged robot needs to be faced [1, 2]. Several stability theory and margin exist in order to evaluate the configuration in which the robot is in risk of falling or it is in a good configuration, but for my procedure I take a simple assumption of flat (maybe with some rocks) terrain. I can assume that when a transition W2G or G2W is occurring, in the first case the robot was driving in a flat terrain and now want to transform to gripper state all limbs to start climbing or moving in highly uneven terrain exploiting legged locomotion and grippers. In the former, after gripper mode, it reaches a planar surface where it want to drive faster using wheels. So in both case the transformation occurs when below the robot the terrain is flat (in general with negligible slope).

In those cases, stability of legged robot rely on the concept of support polygon and support polygon stability criterion.
The support polygon is a simple geometrical entity defined as the polygon formed by interconnecting the foot in contact with the ground (the one substaining the robot). For a 4-legged robot, when base is kept above the ground by all limbs this is a rectangle, when one limb is up it is a triangle.
The simple stability criterion ensure that whenever the base center of mass (which is an approximation of the overall body center of mass) projection in the ground fall inside of the support polygon, than the robot is stable, otherwise not (Fig.12).

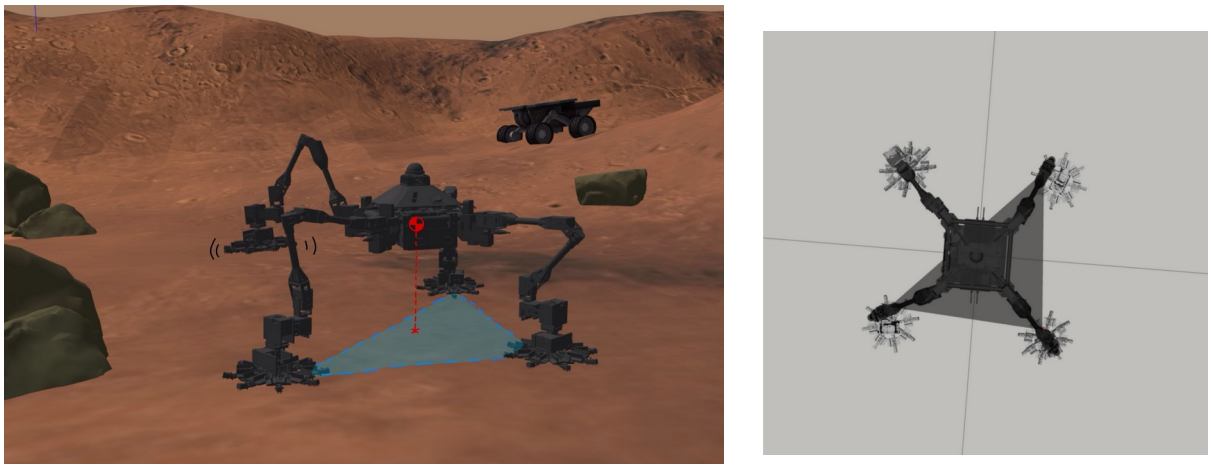


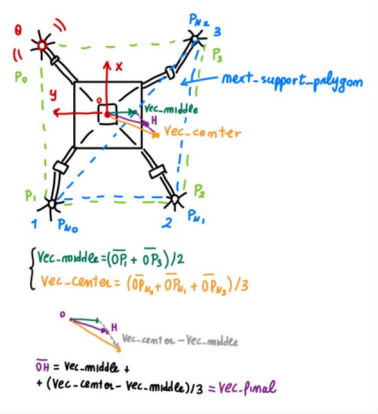
Fig.12: Stable configuration accordingly to support polygon stability theory.

With this simple stability idea, I define an algorithm that first define a limb transformation order (this can be generalized selecting the next limb at run-time accordingly to some external environment information or for other reasons). Than from the id selected, start a procedure that move the base to a stable position, accordingly to the future support polygon that will occur when the selected limb will be raised (it is a sort of simplified MPC (very simplified!) that predict what happens in the future and act to maintain good state of the system). Than after base task end, it can rise the limb selected, transform Grieel, put the foot back on the ground. Than select the next limb and so on… A graphical representation is for sure better in explaining this concept (see below)

Regarding the computation of the base motion, this is done using a rule of thumb that avoid to make extreme base motion (like moving it in the center of the future supporting triangle, that is too much in terms of stability and lead to unnatural or even dangerous robot configuration). The computation of this vector is also summarized below as first image of the graphical representation.

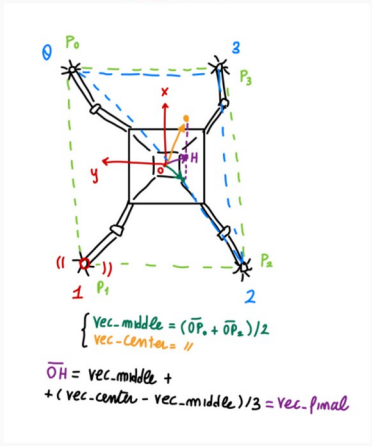
Notice that with a simple vector computation redefinition it is *possible* to adapt the code to the desired degree of stability, or even using a different stability criterion for a different setting we can compute the objective base position and define that vector accordingly, my algorithm is very flexible and so is the software implementation.

limb_id = 0



Symmetric for limb_id = 2

limb_id = 1

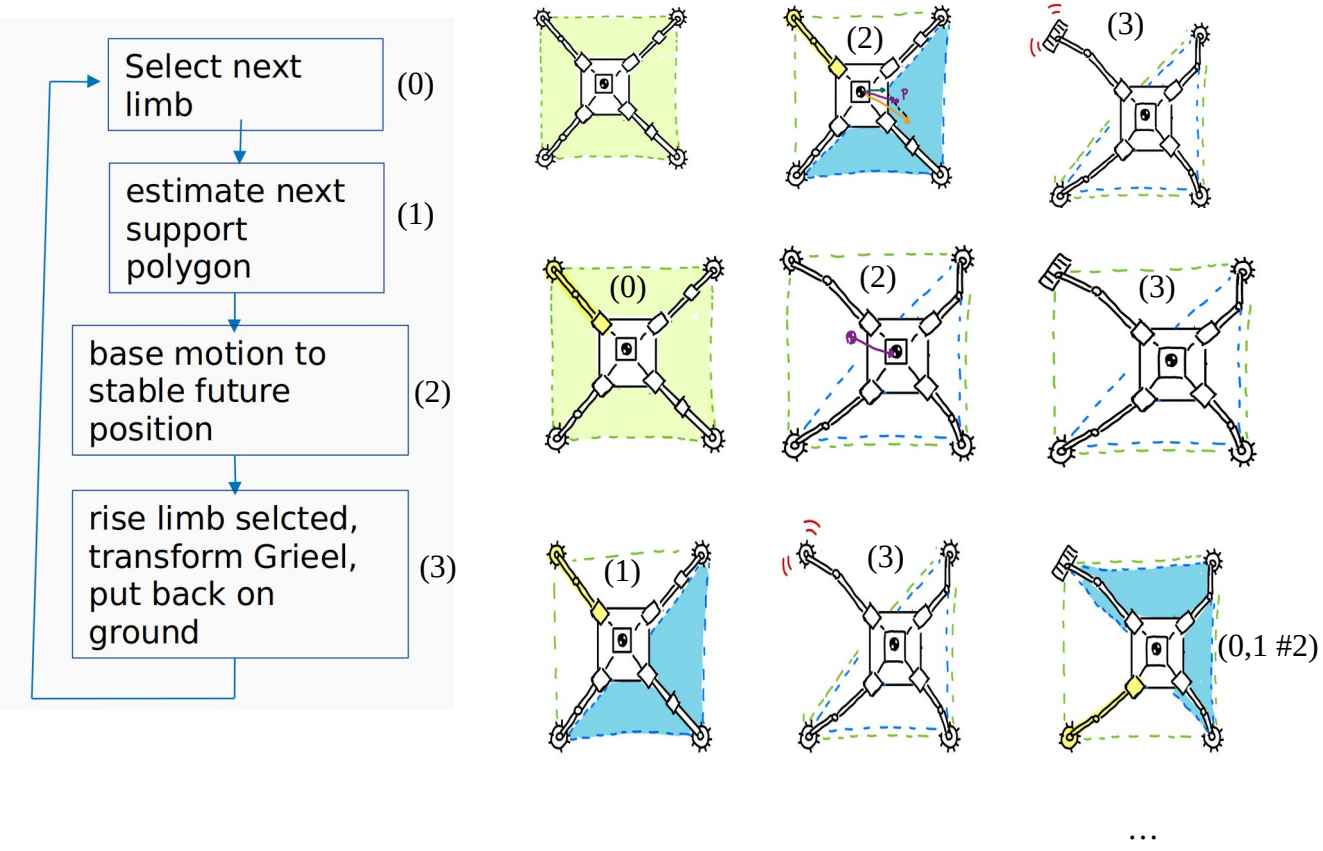


Symmetric for limb_id = 3

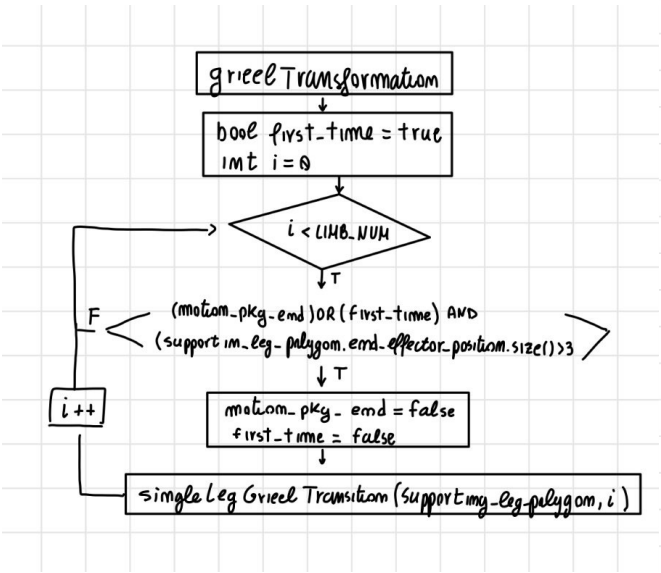
vec_middle := base to medium point of next_support_polygon's edge closer to the base

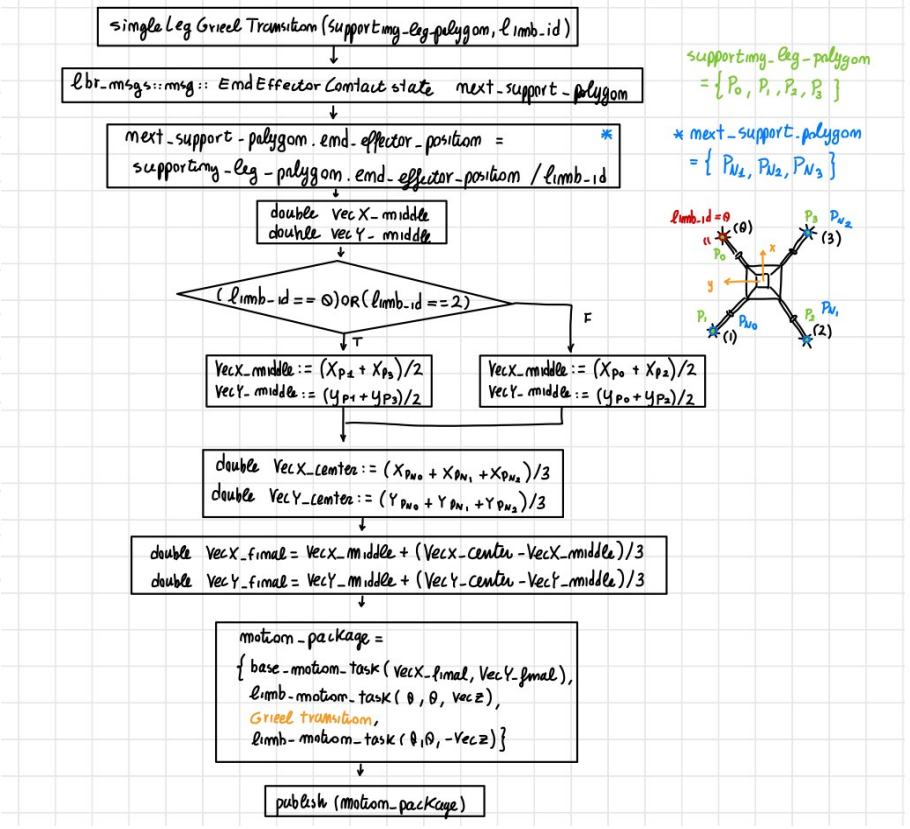
vec_center := base to center of next_support_polygon

vec_final := final base motion task to reach a stable position for the next limb raise



In terms of Software implementation, this algorithm procedure is written inside lbr_high_level_controller, fired by a user command (mapped from the “share” button of the joy-pad). Below I represent the Control Flow Graph of the software implementation used for simulation, but notice that when real robot is controlled, several additional feedback commands and checks are needed, Update that I made with great care during experiment due to the limitations of Gazebo simulation in performing the mode transition of Grieel module, as explained before.





CONTROL TUNING AND SIMULATION IMPROVEMENT

When the algorithm is ready to be implemented and the additional code for controlling Grieel Joints finally works, it is possible to test it in simulation and check if the controller gains defined in ros2_control are satisfactory (At this point of the retuning I start from a previous tuning done to eliminate some unrealistic oscillations and collapsing issues occurring on Gazebo with the default ros2_control gains).

Briefly, ros2_control here is used to implement feedback joint state controller for all limb joints, as independent controllers. This controllers are based on a widely used controller law, called Proportional-Integral-Derivative controller (PID), and on the configuration file you have the freedom of changing the parameters p,i,d. Related to each component of the feedback law.

$$u(t) = K_p e(t) + K_i \int_0^t e(z)dz + K_d \frac{de}{dt}$$

Feedback control is based on the computation of the tracking error, which is the difference between desired state and current state of the variable under control (in our case the joint angle), and than using this error e(t) to compute the best actuation command u(t) to try to reduce it. In PID control, gains have the following role:

p) proportional effect (kp): this define the reactiveness of the control action accordingly to the current error (present). An higher p gain make the controller faster but maybe too much aggressive, which can lead to overshoot/undershoot and possible instability.

I) integral effect (ki): define how much the error cumulation affects the control law (past). It is necessary when external disturbances act and just proportional law is not able to achieve a good tracking, maintaining some error also in steady state. This can cause oscillations when too high because of the reactiveness to small error cumulation.

d) derivative effect (kd) : define the importance of error “slope” in the control signal, if the error is increasing a lot, the action need to be stronger, vice versa if error is slowly decreasing. So it is related to the “future”. It is used to reduce overshoot and undershoot over the tracking performances, but can cause vibration in presence of unexpected noise that usually act at high frequency.

Based on the basic description of the role of PID gains, an iterative tuning is possible. With iterative tuning we refer to a tuning technique in which we try to change the gains by looking on the system behavior (in simulation), so modify those using the knowledge of the expected effect of each gain. But here we are not in the simple case of a single variable (what is called SISO system). Instead we are in a complex system with interconnected joint variables and a performance objective hard to define. The iterative tuning done for each joint without any criteria is not a good approach. So I think about an intuitive iterative tuning idea for a complex system like the one under analysis (notice that we tune the single joint for the limbs, but there is no reason to have different gains for same joints of different limbs).

PID tuning idea
iterative tuning by trial:

1) start tuning from “B2C” to final joint:

1.1) First set for all joints Ki=Kd= 0, and a Kp chosen by intuition according to joint role in the kinematic chain.

Indexing joints as B2C (i=0) to driving (i=7), start from i = 0

1.2) Consider joint i , if Kp cause a response too much aggressive, decrease it, otherwise increase it until the trajectory tracking performance ais fast enough with some small over(under)shoot or small oscillations acceptable.

1.3) Tune Kd of current joint to reduce oscillations and over(under)shoot as must as possible, carefully because an higher Kd can cause undesired noise due to reactiveness to joint speed.

1.4) When oscillations are reduced enough, if some steady-state error is present, tune Ki accordingly.

1.5) consider next joint i+1 and return on step 1.2

once reached the final joint, due to the complex dynamic maybe some performances are no more as expected, so proceed backward to trial by error performance improvement as possible

2) correct control gains from final joint to “B2C”:

2.1) proceed with the same step as 1.2 to 1.4, but now start from i=final_joint_num and iterative tune from last to first, so decreasing joint index considered at each step.

Usually in robotics arms, when iterative tuning is performed, the process goes from last joint to the first, but in this case the limb is used as a closed kinematic chain, bot actuating from the body or from the contact point according to the motion performed, that’s why I consider a two step tuning sequence up-bottom and bottom-up.

Before showing the results of the above tuning procedure, as quick note on simulation parameter set-up. Initially I have chosen friction and damping parameters for “realistic” simulation by “cheating a bit”, setting different ones for different joints accordingly to how I expect them to move.

Those parameters used by gazebo are defined in the <joint> tag of the URDF as
<dynamics damping="<joint_damping>" friction="<joint_friction>"/>

Dynamixel mechanical properties of friction and damping are not available on ROBOTIS e-manual, neither on other resources, I think that performing some mechanical identification experiments of this can be very helpful to improve simulation reliability and so SimtoReal testing becomes more meaningful.
During PID tuning, I make as hypothesis that all motors have the same characteristics, setting for all joints damping=0.1 and friction=0.4, this is reasonable in simulation, avoid unexplained driving oscillation issue on Grieel transition and is a better hypothesis than setting all joints parameters differently.

Now, I skip to report the previously used gains and start from the first step, suing just proportional control (with p gains as the one “optimal” in the previous full PID tuning without criterion).

(here are reported only LF limb joint state, all limb are tuned with same gains due to the symmetric characteristics of the system)

1) UP-BOTTOM (B2C to driving)

>“i = 1, B2C joint”

#1) B2C Tuning, just P control

decrease a bit B2C proportional gain to reduce aggressiveness

Tab.1: PID gains and performances step 1

JOINT	Kp	Ki	Kd	Comments on performances
B2C	50.0	0.0	0.0	Satisfactory but still some overshoot, damp required
C2F	110.0	0.0	0.0	A bit too much oscillating but good tracking
F2T	50.0	0.0	0.0	Non negligible delay, maybe caused by sustained mass during motion
T2E	125.0	0.0	0.0	Satisfactory
wristH	160.0	0.0	0.0	Small delay but fine
wrsitV	150.0	0.0	0.0	Satisfactory
driving	80.0	0.0	0.0	Satisfactory, WtoG transformation oscillations reduced a lot with new sim set-up

In order to analyze and plot the joint trajectory, I use an additional node written by me that subscribe to the correct topic, store it in a csv table and than a python script is responsable for plotting useful data.

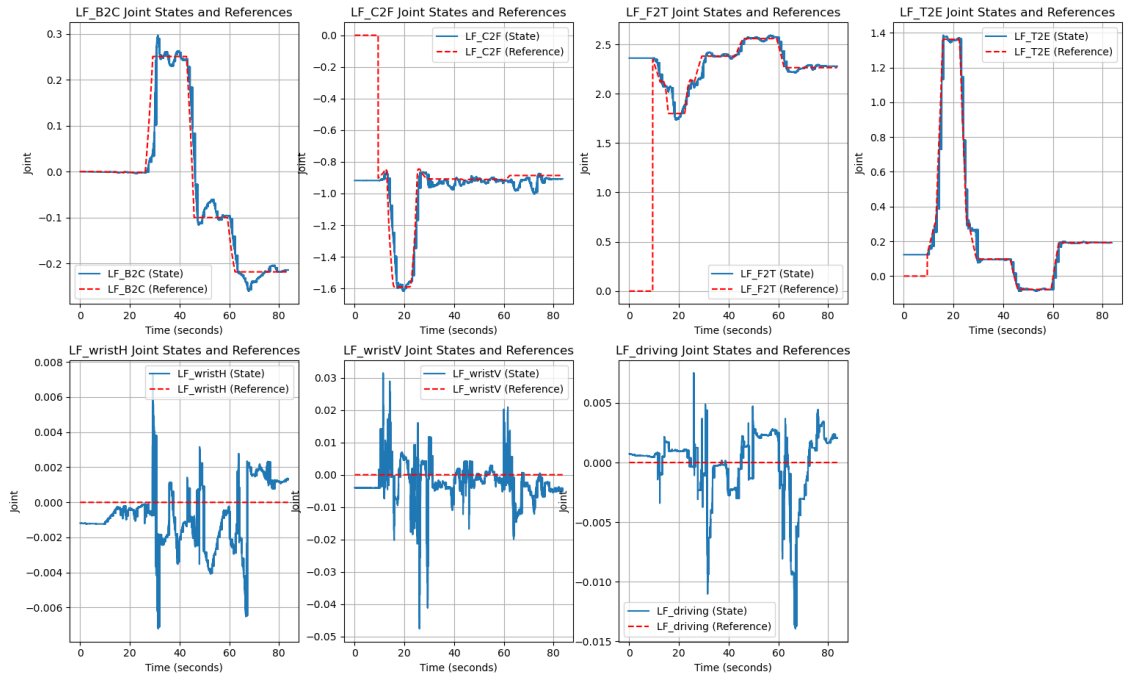
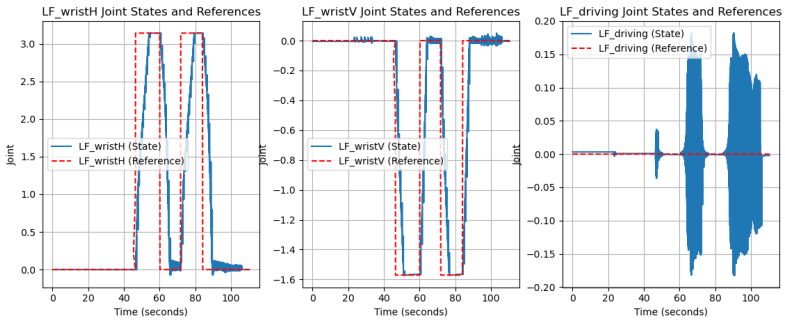


Fig.13: Initial PID tuning performances

Grieel Joint control G2W, W2G, G2W (unexplained driving oscillations)
(With past Grieel trajectory computation)



… here I skip all the intermediate tuning phase, that can be found in the Appendix, and go directly to the final optimal tuning, adding some final comments:

FINAL PID GAINS FOR EACH JOIN:

JOINT	Kp	Ki	Kd	Comments on performances
B2C	65.0	0.01	1.2	Satisfactory,Some additional over and undershoot introduced by wrist gains, but negligible
C2F	90.0	0.1	2.0	Satisfactory, small steady state error maybe caused by incoherent inverse kinematic solution
F2T	55.0	0.01	1.4	Satisfactory, small tracking error
T2E	100.0	0.0	0.0	Satisfactory
wristH	120.0	0.001	0.05	Satisfactory
wrsitV	120.0	0.0	0.01	Satisfactory
driving	80.0	0.0	0.0	Satisfactory, just small oscillations when W2G transition, maybe due to kinematic singularity, but negligible in simulation.

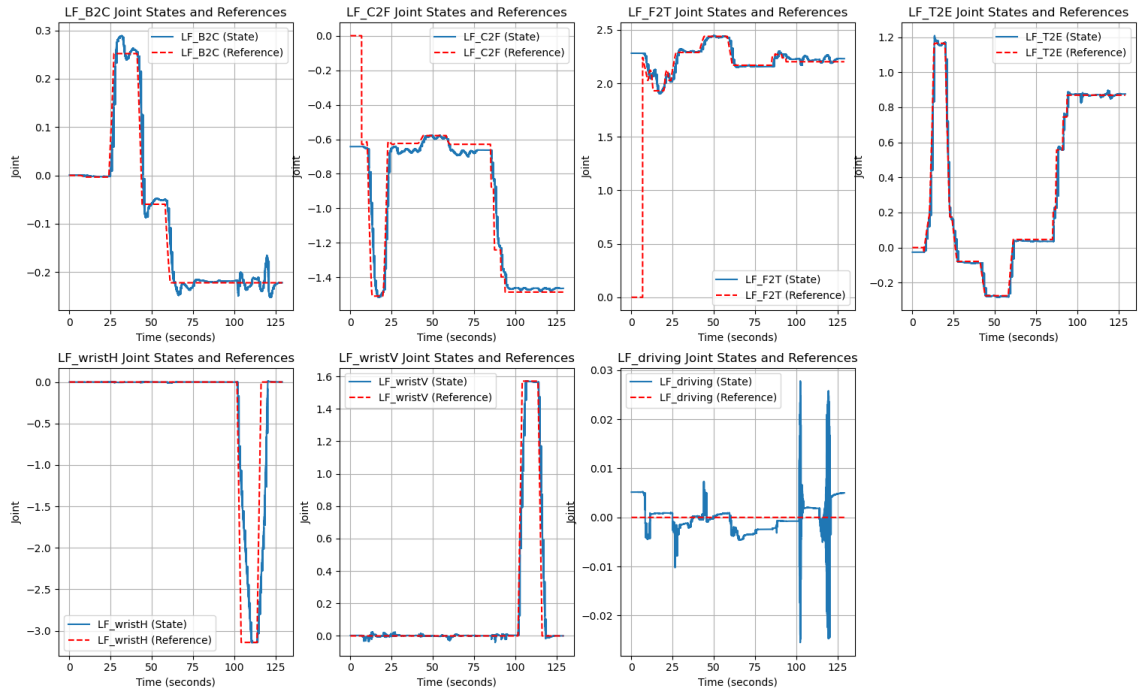


Fig.14: final Joint Tracking performances

Tuning Comments:

- The overall iterative tuning procedure has been done by testing the sequence of motion of my Grieel Transition automatic procedure, because of the wide range of task performed (base motion, limb motion, keep limb above the ground, Grieel joint command).

But to be sure about robustness of the tuned controller, even more tasks should be simulated and performances analyzed, for example base angle motions, limb x,y motions while it is above the ground, grasping, driving in wheel mode etc.

Unfortunately, when doing this tuning the software is not ready some of those advanced tasks.

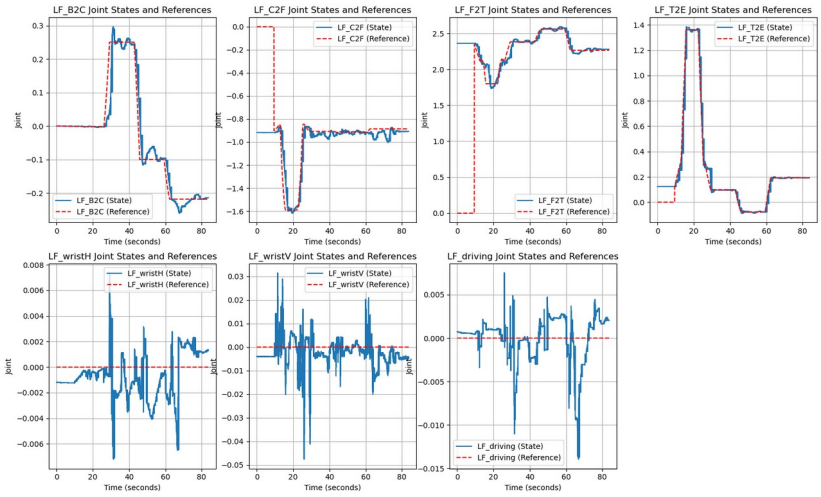
- When PID performance in simulation is satisfactory, next step is implementing those PID in the real joint motors, from simulation to real systems always differences show up, due to the complexity of estimating all physical parameters of the system.
  Simulation seems realistic but we are not sure of how much it is reliable. The implementation phase may require some adjustment on the gains, also accordingly to motors saturation and safety requirements.

- As we can see from simulation, G2W and W2G joint transition is a bit aggressive, and cause some limb oscillations, in simulation it doesn't seems to critical, but maybe wristH and wristV trajectory computation during this transition can be updated with a smoother motion, accordingly to real system behavior. Experiment will tell us how to proceed.

- In Gazebo simulation, from PID#0 to PID#16 performance improvement is clear. Body oscillations are reduced, limb motion is more stable, the movement is smoother and safer. (Video **Gazebo_GrieelTransition_PID#16** and **Gazebo_GrieelTransition_PID#0**)

- Finally we can compare the two joint trajectory performances from the joint graphs:

PID#0 (before re-tuning with Grieel)



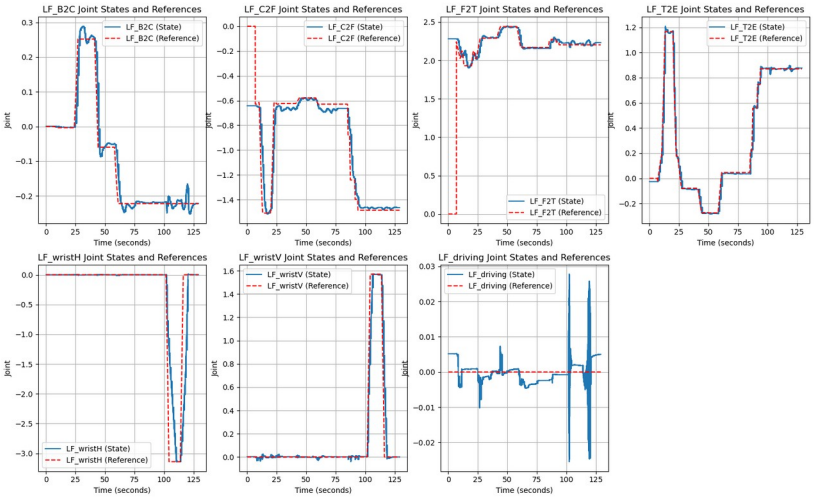
✗ significant overshoot and undershoot

✗ high frequency vibration in the joint angle (can be a problem for actuators)

✗ Non negligible oscillations in the tracking

✗ Unnatural behavior in simulation

PID (final one)



- ✔ negligible overshoot and undershoot just on B2C joint
- ✔ good tracking performances and quick convergence.
- ✔ Smooth and realistic simulation

Simulation Comments:

- Small physical parameters precautions can make a big difference in simulation reliability.
- A better tuning of contact friction of end effector allow to improve the reliability of the current simulation set-up. This is defined as a gazebo tag in the robot URDF:

```
<!-- Gazebo tag for support gripper/wheel phisical characteristics -->
<xacro:if value="$(arg wheel_mode)">
    <gazebo reference="${prefix}_wheel_Link">
        <mu1>1.5</mu1>
        <mu2>2.5</mu2>
        <kp>1000000</kp>
        <kd>1.0</kd>
    </gazebo>
</xacro:if>

<xacro:unless value="$(arg wheel_mode)">
    <gazebo reference="${prefix}_gripper_Link">
        <mu1>0.5</mu1>
        <mu2>0.5</mu2>
        <kp>1000000</kp>
        <kd>1.0</kd>
    </gazebo>
</xacro:unless>
```

Different ground contact conditions in wheel and gripper mode calls for different physical parameters.

<mu1> : “Friction coefficient 1”, define primary friction coefficient between contact surface, it affect frictional force along one direction of contact surface.

<mu2> : “Friction coefficient 2”, define secondary friction coefficient between contact surface, it affect frictional force orthogonal to the one defined by mu1.

(mu1, mu2: for good traction surfaces we use high values around 1.0, while low values close to 0 define slippery surfaces, a good tuning it can be done accordingly to experimental values. In my case I just tune it iteratively from meaningful values until simulation seems realistic.)

<kp>: “Stiffness coefficient”, represent proportional gain of spring-damper model used for contact dynamic simulation. (high kp means stiff spring, so less penetration with ground, that’s why here I use an high value, considering rocky terrain)
<kd>: “Damping coefficient”, represent damping gain, models energy loss in the contact (dissipation). (high kd means less contact oscillations)

Simulating and testing the controller for different realistic contact conditions can be useful to ensure robustness of the controller. It is good practice for SimToReal robust results.

- Finally, notice that until now robot performances has been tested and simulated only in Gripper mode of all Grieel modules.
  When switching between the two mode we can think about an adaptive control law that get loaded accordingly to the current mode, for example with a simple switching law control box based on a mode variable. Also to control in wheel mode in order to perform navigation, but also for climbing and legged locomotion, if we want to implement an autonomous navigation higher level controller that perform global and local path planning are needed. Also for complex manipulation and grasping task, or for complex maneuver optimal controllers like MPC and reinforcement learning control technique are needed to fully exploit the Flexibility of LOMBERO+GRIEEL.

EXPERIMENT and RESULTS

SINGLE LIMB TEST:
After setting properly the Dynamixel ID.
The first experiment is conducted on just one limb (single limb testbed Fig.15), using a proper launch file we start the dynamixel controller just on the limb tested.
From the experience of this experiment it is clear that the default Dynamixel PID gains are not satisfactory.

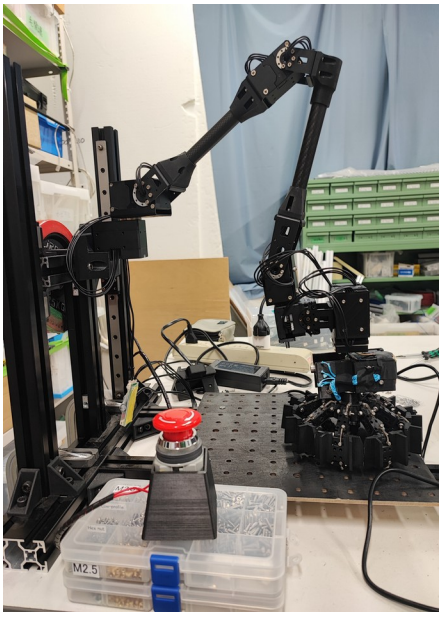


Fig.15: Single limb testbed

Dynamixel Re-tuning

Due to the fact that converting the gains from ros2_control to Dynamixel motor gains in Dynamixel wizard is not straight forward, I proceed with a quick retuning of the joint motors. This is done using the functionality of Dynamixel Wizard 2, plotting the encoder position (velocity) with respect to the goal position (velocity), and tuning the gains using the same reasoning as before.

Another approach, that would speed up this process and use the previously tuned controllers in ros2, is to implement the hardware interface between ros2 controllers and the Dynamixel, so that the input of the motors is directly applied as the one computed by the joint_trajectory_controller.

Notice that by tuning each joint one by one in this way, we are not controlling other motors joint angle, as done before in simulation. The previous consideration limit the robustness of the obtained tuning performance during this procedure, due to the avoided interference of complex motions.

So, after this single motor tuning (just considering the load of the other joints and possible external noise of other motor disturbance, but not complete motion of the limb), some adjustments may be needed when single limb test is done, and even when all robot together is mounted.

As final comment, this tuning is performed in a single limb testbed, so the additional degrees of freedom of limb-root moving with the base is lost here, and additional complex motion are not considered.

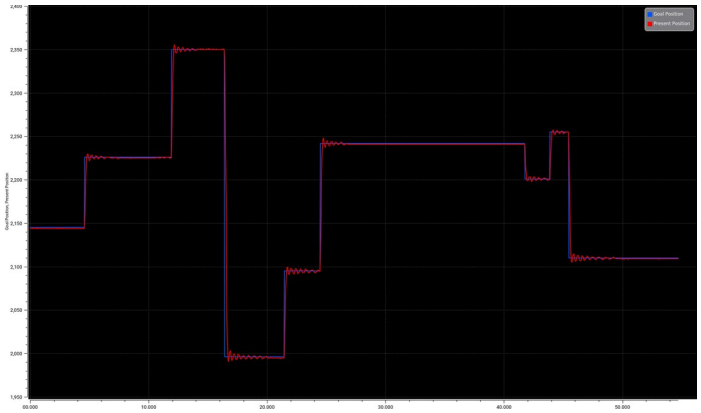
From the default PID gains of Dynamixel motors (not optimal):

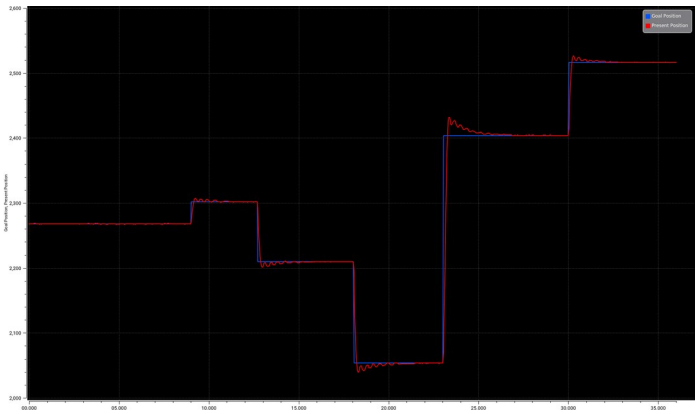
JOINT	Kp	Ki	Kd	Comments on performances
B2C	800.0	0.0	0.0	Long oscillations, causing long set-up time, and maybe not robust in case of unexpected body load
C2F	800.0	0.0	0.0	Non negligible steady state error, only proportional gain doesn’t react to unexpected gravitational force of next links
F2T	800.0	0.0	0.0	...
T2E	800.0	0.0	0.0	...
wristH	640.0	0.0	2000.0	...
wrsitV	640.0	0.0	2000.0	...
driving	100.0		1920.0	...

We tune by trial and error those gains, observing Dynamixel Wizard tracking plot and arm behavior, obtaining at the end reasonably good performance with the following motor gains:

JOINT	Kp	Ki	Kd	Comments on performances
B2C	750.0	200.0	300.0	Still oscillations but last less, sometimes overshoot but negligible. Robust to unexpected load by integral gain
C2F	1000.0	450.0	2200.0	Some overshoot but almost zero steady state error, feedback controller compensate even limb load.
F2T	650.0	300.0	1400.0	Again steady state error almost reduced up to zero!
T2E	900.0	300.0	350.0	...
wristH	800.0	400.0	1500.0	...
wrsitV	900.0	450.0	1300.0	...
driving	100.0		0.0	...

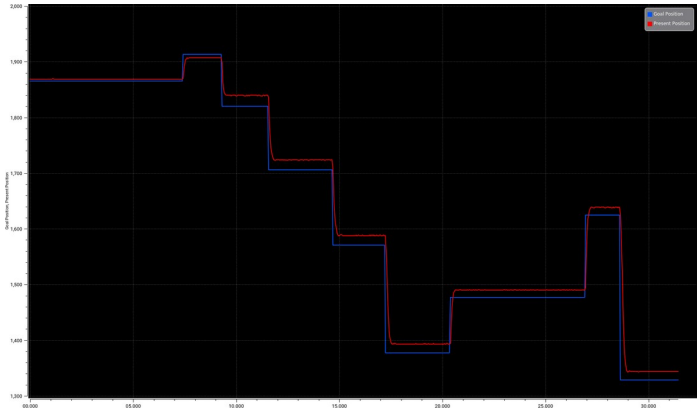
B2C:

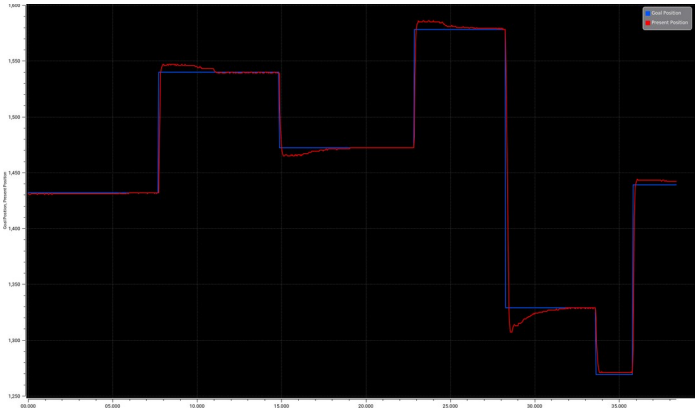




Notice that the main cause of the oscillation is Grieel gripper inertial effect, when a step position set point is sent, gripper finger are not so static and cause undesired oscillations.

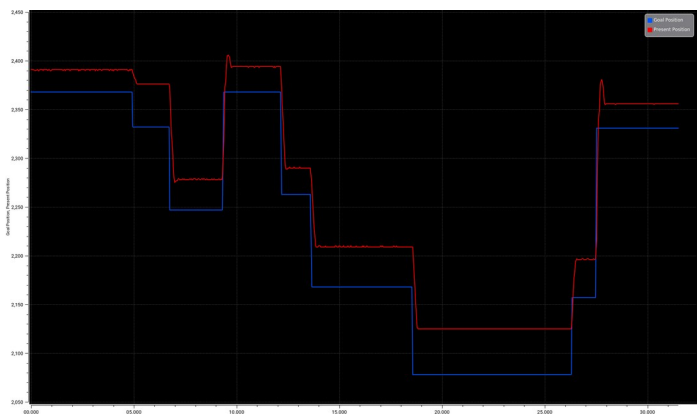
C2F:

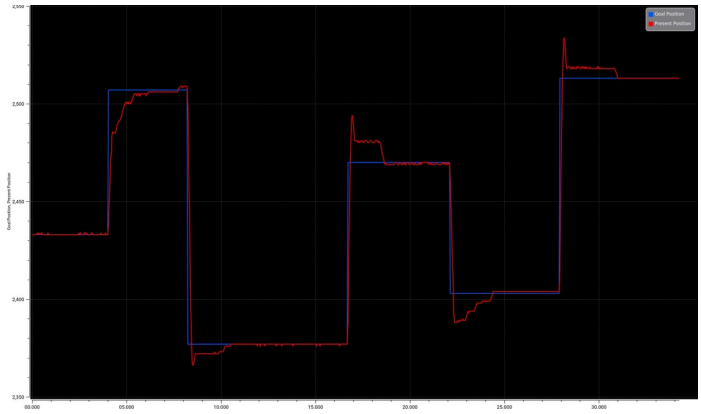




from a completely unacceptable tracking performance we obtain a good mass compensation thanks to the usage of PID controller power. Feedback and usage of error integration allow to obtain almost zero steady state error.

F2T:





and so on…

A second clear problem is related to the hardware limitation, Something that I don't think about until time of experiment, contact sensor are absent in Grieel.
This means that all support polygon estimation is not possible, than all the previously discussed algorithm cannot be tested. We try several idea to simulate a contact estimation, a good one by testing in the single limb test bed keeping limb root floating (removing the keeping column) was to use the current feedback [8]. As can be seen in Fig.16, reading the state of the C2F joint current (using readCurrent method in dynamixel controller) help us to estimate when the limb is applying torque to keep robot base up, and so if it is in contact.

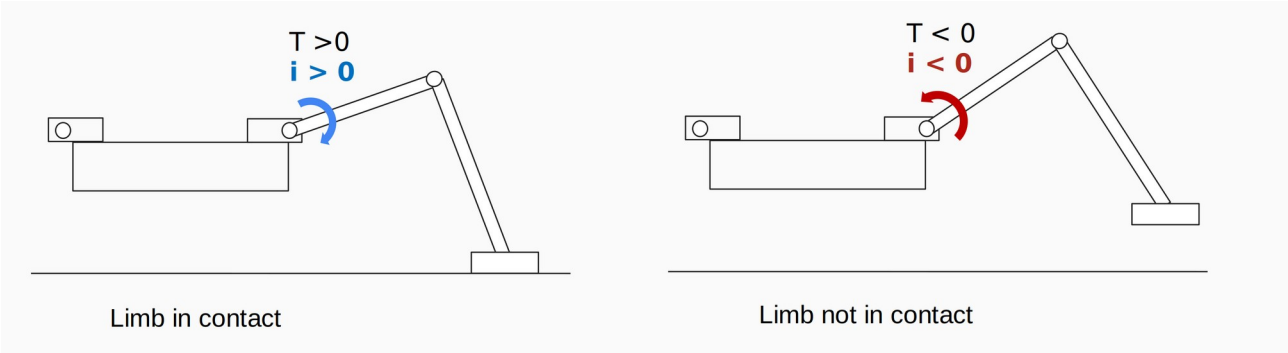


Fig.16: Possible contact state emulation using current feedback.

This solution works well for a single limb, after testing and fixing hardware configuration aspect and debugging the code for the all limbs, we can assemble LIMBERO+GRIEEL (Fig.17)

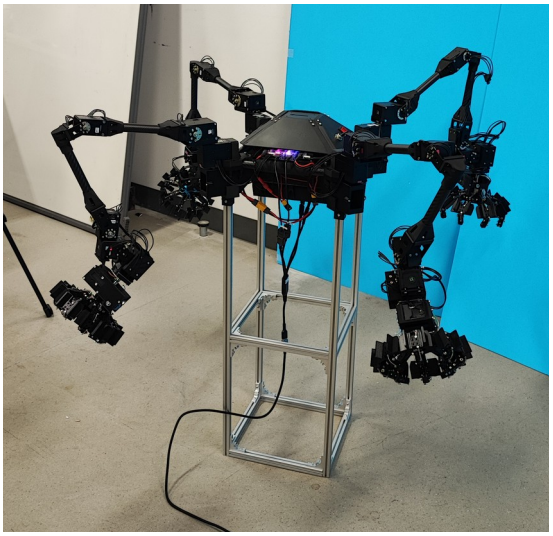


Fig.17: assembled robot in the holding column

LIMBERO+GRIEEL TEST:

We position the robot on the ground in a good configuration, connect all USB cables and launch the program, everything seems working, than a base motion task to rise it up is sent from joy-stick, still fine. Also contact state using current feedback seems to work (not 100% reliable but on average are fine).

A minute pass and first wristV joint dies, than a C2F joint of another limb die, and so on, the motors are not able to keep up the overall structure. The required torque is too much and dynamixel motors enter in protection mode even if not overheated. We try to start from different initial kinematic configuration, to reduce torque required but still some motors die.

The only possible way to keep robot up reducing the torque required to the limb joints (especially the one responsible for pitch angle, with axis of rotation parallel with the ground) is to proceed with a mass compensation. To do this we use some fishing wires attached to 4 edges of the robot base, and than connect those in a pulley on the roof which is itself connected to a weight, we try with several weight to obtain the best compensation that doesn't affect too much the experimental procedure. Unfortunately, using this not professional gravity compensation system, the contact estimation is not working by using current, somehow those external compensation forces reduces the torque on C2F joint and the previous assumption doesn't hold anymore.

For the few amount of times that I was able to visualize the overall support polygon, and start the algorithm, the base move to the desired position and Grieel is transformed properly rising the limb at the desired height (with some additional oscillations that may be caused by the wire that act as disturbances to the control system). Than In order to test the software overcoming the contact estimation limitation, I just send some fake command information accordingly to the expected current state of the robot. The algorithm implementation was able to transform 3 module, than unfortunately stability is lost due to motor fail.

I thought that a possible cause of the vibrations could the PID gains done by me for the dynamixel, but when we try to test the system with the default dynamixel gains, the absence of integral gain for gravity compensation is clear. When the rise up command is sent, the limb is raised but fail immediately to be kept in the desired position, C2F joint fall and transition start in a dangerous position close to the limb root. This experiment performed with default dynamixel setting show us the importance of PID tuning.

We test both in initial full wheel mode and full gripper mode, in both cases the stability of the robot is maintained bu moving the base to the position computed by my algorithm and than raising the limb. But unfortunately due to motors limitation the full transition was impossible to achieve, at least one motors of limb or Grieel dies before the end.

CONCLUSIONS

By merging the results of the simulation (which show up how during all transition, moving the base in the position computed by the algorithm ensure stability in 3 limb support) with the partial result of experiment (that give us some insight of stability in a mixed end-effector configuration, impossible to test in Gazebo), we can conclude that this stable transition algorithm works.
But some comments are needed for sure, both regarding the final result and on possible future improvement:

- During the experiment, the fact that forward and inverse kinematic is not properly updated for LIMBERO limb + Grieel seems to be not a big deal. Anyway as noticed before, in terms of performance and Real hardware exploit, this two aspect are crucial for the further update of the Software. At this point the best thing is to face this problem for the next prototype of Grieel, with a less DOF the solution may be “easier” but still infinite solution that may call for Newton-Rapson technique to solve inverse kinematic.

- After the experiments it is clear that and hardware update of the LIMBERO motors or joint structure is needed if it has to be used also for walking and not only for climbing (until now main experiment are conducted on climbing test bed with gravity compensation). More powerful motors or worm gear should be used to ensure higher torque or reduce the required torque. Different mechanical structure of the C2F or F2T joints should be considered if LIMBERO has to be used with Grieel.

- Maybe the next prototype of Grieel, without wristV motor limitation and continuous fail, with less weight and deeper wheel can ensure more stable movements and less torque request, but who will work on it has to take into account that some Software update are needed.

- The usage of sensor is crucial to perform autonomous tasks. Another must of the hardware upgrade consist in integrating contact sensors or using some force sensors to have knowledge of contact with the ground. Also, in order to have a feedback on Grieel locking state a tension sensor used in the wires that open and close the finger could help a lot to know the module state. During the experiment we rely on the number of locking motor rotations between the two mode, but it is not always reliable and depends on how much the module is wear.

- I write the software also for sending driving commands and to change Grieel Joint configuration in driving mode, but limitations of the experimental apparatus just to gravity compensated (fixed to the ground) limit the possible navigation workspace. Additional test should be done in the overall robot performing a smooth transition from wheeled locomotion to walking one, with my algorithm in the middle.

I’m sure that the potentiality of Grieel Module are above our expectation, the simple problem that I faced show up lot of improvement possibilities and also upgrade of the usage.
Using 3 wheels for driving and one gripper for manipulation is very important for specimen collection and exploration. Also exploiting the limb freedom while driving, as Anymall does (with the usage of high level reinforcement learning controllers) can enhance the flexibility, efficiency and application settings of Grieel Module. Finally, this is a very general end-effector, IN the future usage of heterogeneous and adaptable mobile robots, this can be used with lot of other tools and in legged robot with more limbs.

REFERENCES

[1] Warley Francisco Rocha Ribeiro, “Reaction-aware robotic locomotion in microgravity”, pp.34-37, 2023
[2] Kentaro Uno, “Autonomous Limbed Climbing Robots for challenging Terrain Exploration”
[3] Marko Bjelonic,, Planning and Control for Hybrid Locomotion of Wheeled-Legged Robots, 2021
[4] ros2 wiki, https://docs.ros.org/en/humble/index.html
[5] Bruno Siciliano, Robotics: Modelling, Planning and Control
[6] Limb-team LIMBERO readme in the devlopment branch, plus all the explanation by Kazuki Takada.
[7] dynamixel_sdk, https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/overview/
[8] Marek Wqsik, Leg’s tip-ground contact detection based on drive currents in a real walking robot, 2015, Journal of Automation, Mobile Robotics & Intelligent System.

ACKNOWLEDGMENT

This research could not be possible without the feedback of my supervisor Prof. Yoshida Kazuya, the continuous support and suggestions of Ass. Prof. Kentaro Uno and the resource and helps provided by Ass:Prof Shreya Santra. The help received by Kohta Naoki for the robot model, and especially during each step of the experiment has been crucial for my achievement. Him with my other mate of the Rover Team and Limb team Kota Ikeno, Riku Suzuki, yudai matsuura and Kazuki Takada literally keep LIMBERO up, remain with patients for any of my obsessive experiment until achieving the best result, with continuous feedback and suggestions. Takada “lessons” on limbero software help me from may to august, without its explanation I would have achieve nothing, his knowledge of legged robot and Limbero software had been like a lighthouse for me.
All of them have been research mate and assistant, but especially friends.
Thanks also to my friend Matteo for my everyday launch brake stress release after each fail, talking with him and listening to his advices help me in figuring out many solutions.
Last but not least, I’m grateful to all my exchange friends that transform bad days in good ones and have been with me in this adventure until the end.

APPENDIX, PID TUNING STEPS:

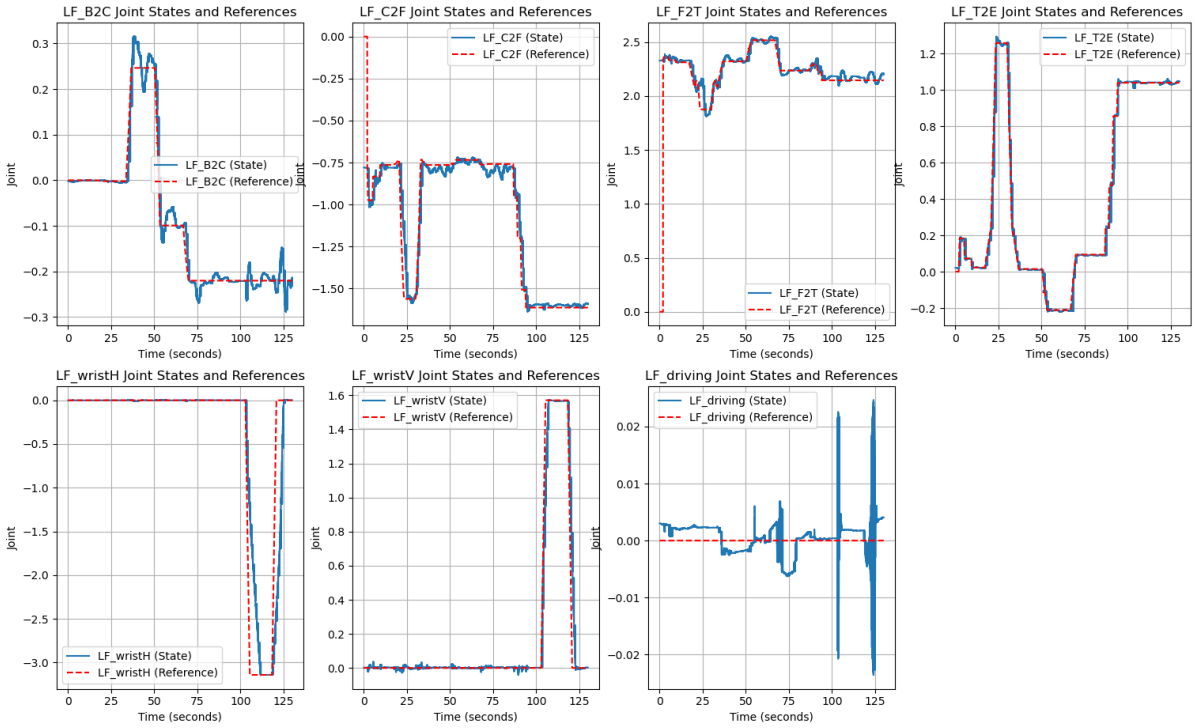
1) UP-BOTTOM (B2C to driving)

>“i = 1, B2C joint”

#1) B2C Tuning, just P control

decrease a bit B2C proportional gain to reduce aggressiveness

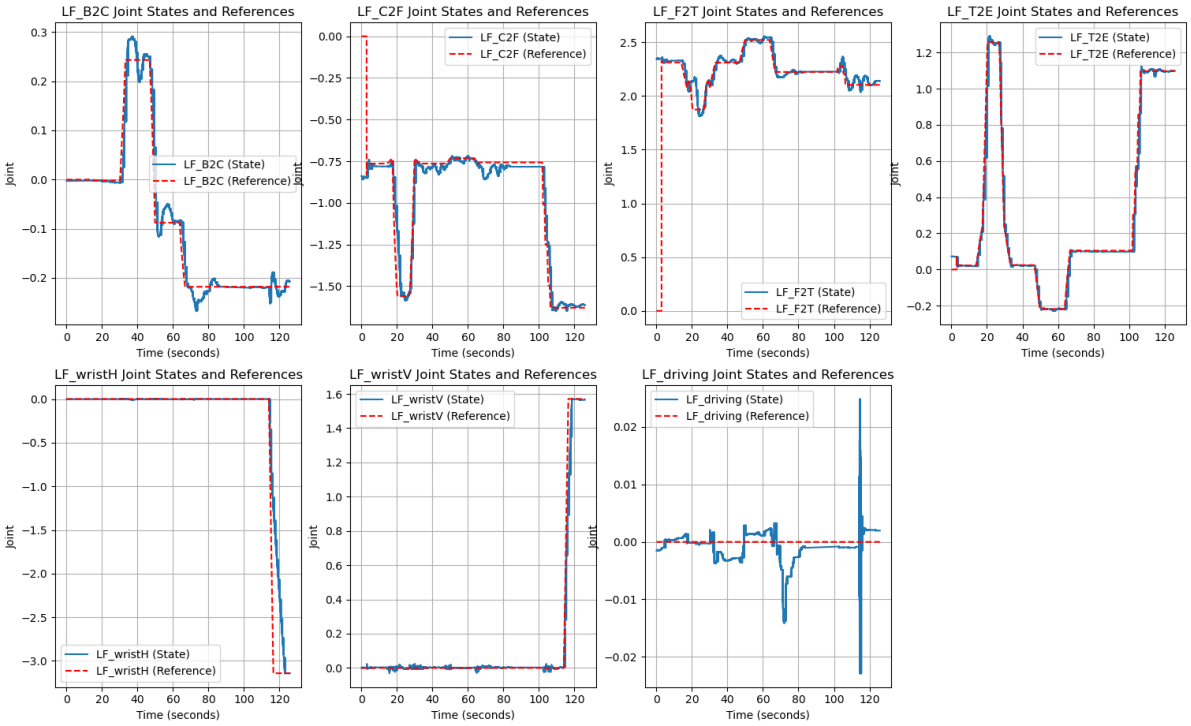
JOINT	Kp	Ki	Kd	Comments on performances
B2C	50.0	0.0	0.0	Satisfactory but still some overshoot, damp required
C2F	110.0	0.0	0.0	A bit too much oscillating but good tracking
F2T	50.0	0.0	0.0	Non negligible delay, maybe caused by sustained mass during motion
T2E	125.0	0.0	0.0	Satisfactory
wristH	160.0	0.0	0.0	Small delay but fine
wrsitV	150.0	0.0	0.0	Satisfactory
driving	80.0	0.0	0.0	Satisfactory, WtoG transformation oscillations reduced a lot with new sim set-up



#2) B2C Tuning, PD control

increase D gain to reduce overshoot and oscillations

JOINT	Kp	Ki	Kd	Comments on performances
B2C	50.0	0.0	0.5	Overshoot reduced a bit and less oscillations, still some more damp required
C2F	110.0	0.0	0.0	A bit too much oscillating but good tracking
F2T	50.0	0.0	0.0	Non negligible delay, maybe caused by sustained mass during motion
T2E	125.0	0.0	0.0	Satisfactory
wristH	160.0	0.0	0.0	Satisfactory
wrsitV	150.0	0.0	0.0	Satisfactory
driving	80.0	0.0	0.0	Satisfactory, WtoG transformation oscillations reduced a lot with new sim set-up

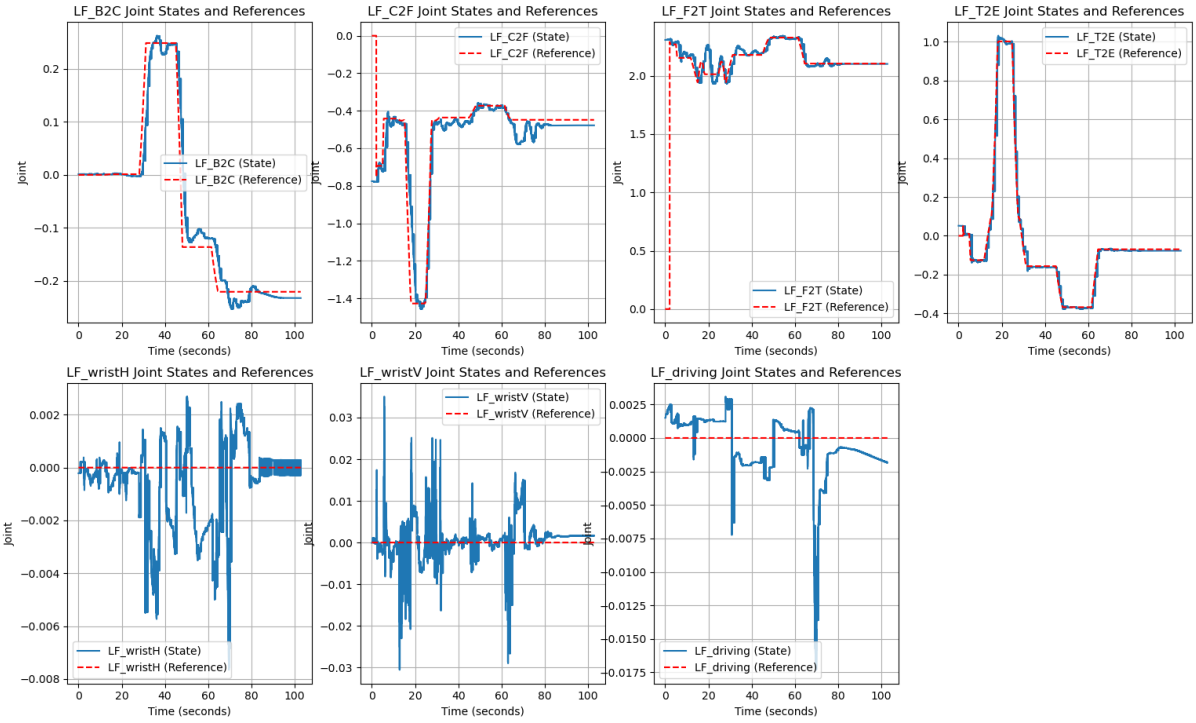


some reduction of the overshoot on LF_B2C joint is visible, this suggest that this is the correct tuning procedure to make B2C joint control better, let’s proceed in this way, from now on not all steps will be commented, but the interesting one.

#3) B2C Tuning, PID control

increase more D gain and add some integral gain

JOINT	Kp	Ki	Kd	Comments on performances
B2C	50.0	0.4	2.0	Overshoot reduced but maybe too much dumping and unexplained delay respect before
C2F	110.0	0.0	0.0	A bit too much oscillating but good tracking
F2T	50.0	0.0	0.0	Tracking a bit worse than before
T2E	125.0	0.0	0.0	Satisfactory
wristH	160.0	0.0	0.0	Real tracking not checked for now (same Grieel gains)
wrsitV	150.0	0.0	0.0	Real tracking not checked for now (same Grieel gains)
driving	80.0	0.0	0.0	Satisfactory, WtoG transformation oscillations reduced a lot with new sim set-up

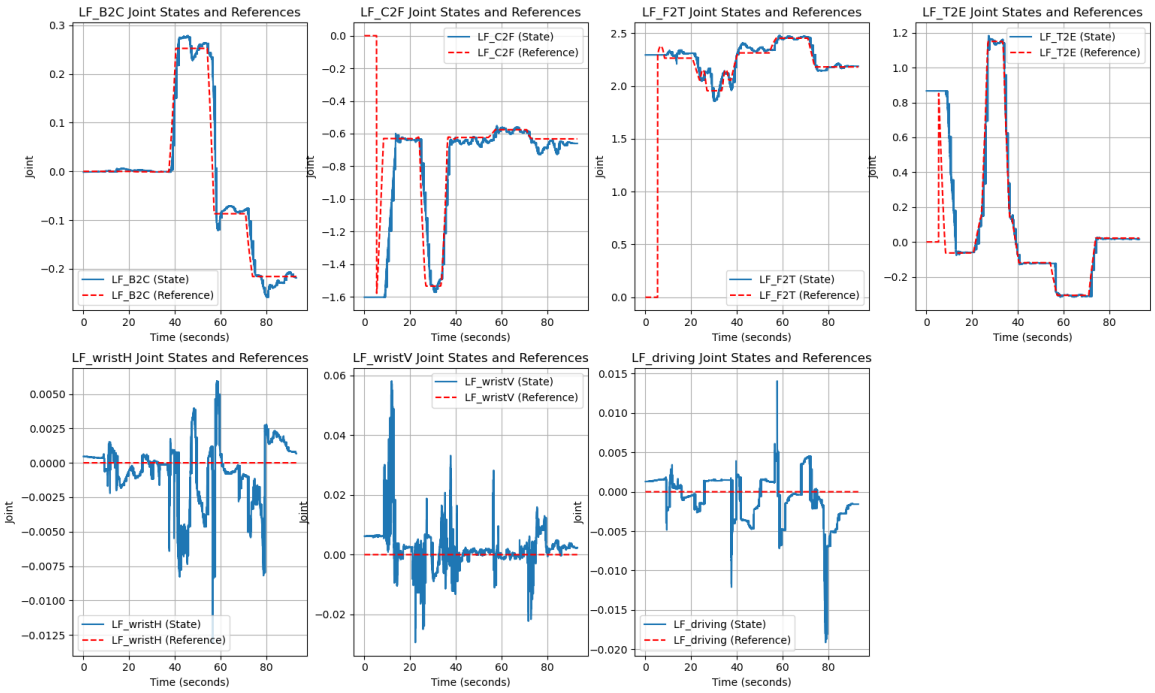


iteratively adjusting Ki and Kd we can try to achieve a good performance, case #2 was good in the sense of steady state error, but with too much overshot/undershot. Tune Kp and Ki until satisfied, and if needed maybe a change of Kp can help in achieving performances.

#4) B2C Tuning, PID control

thanks to D gain much reactiveness with an higher P gain can help in tracking trajectory, and I gain can be reduced

JOINT	Kp	Ki	Kd	Comments on performances
B2C	65.0	0.01	1.2	Overshoot reduced and also less delay respect case#3
C2F	110.0	0.0	0.0	A bit too much oscillating but good tracking, also too aggressive as seen from video when raise up
F2T	50.0	0.0	0.0	Tracking a bit worse than before
T2E	125.0	0.0	0.0	Satisfactory
wristH	160.0	0.0	0.0	Real tracking not checked for now (same Grieel gains)
wrsitV	150.0	0.0	0.0	Real tracking not checked for now (same Grieel gains)
driving	80.0	0.0	0.0	Satisfactory, WtoG transformation oscillations reduced a lot with new sim set-up



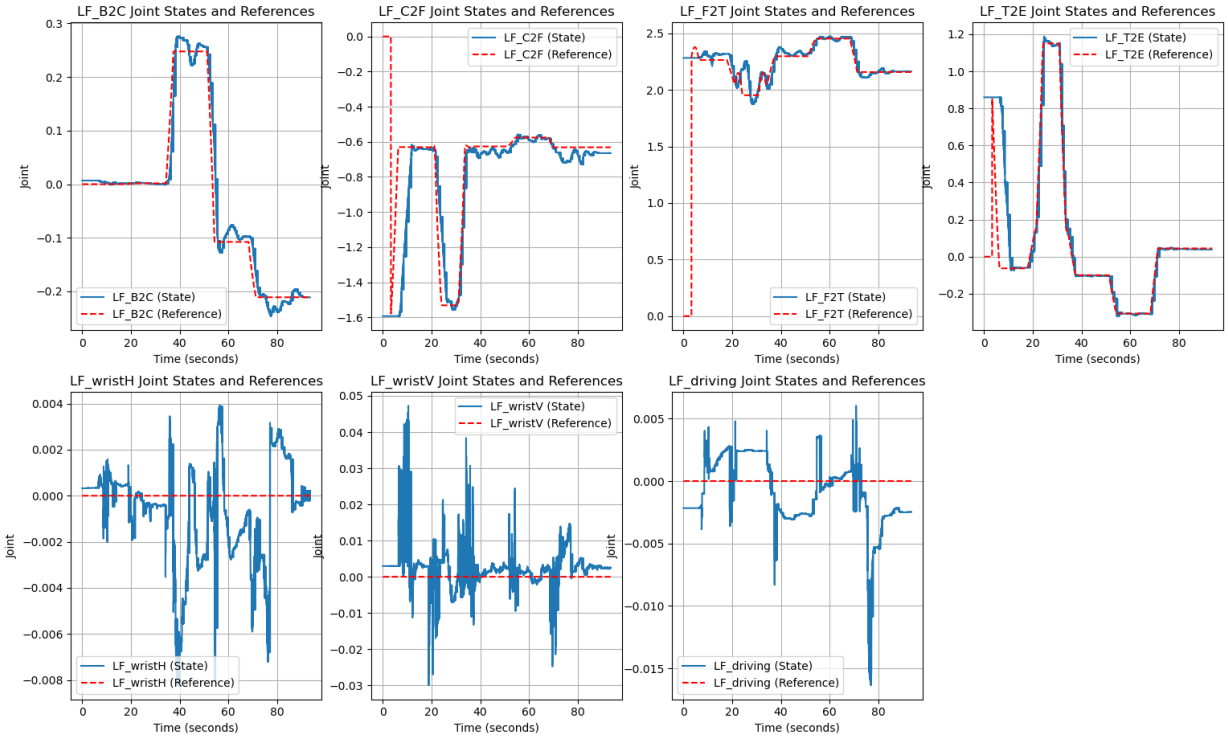
For now we are satisfied by this B2C tuning, next joint to tune is C2F. As we can see both from plot and when robot moves in simulation, it seems to oscillate too much and also when a body task of rising up is sent, the response is too aggressive.. This suggest a reduction of P gain and increase of D gain, let’s reduce a bit that P gain and then check if D gain helps against oscillations.

>“ i=2, C2F Joint”

#5) C2F Tuning, PD control

decrease a bit C2F P gain due to aggressive raise up/down and increase a bit D gain

JOINT	Kp	Ki	Kd	Comments on performances
B2C	65.0	0.01	1.2	satisfactory
C2F	90.0	0.0	0.5	Negligible changes
F2T	50.0	0.0	0.0	No changes
T2E	125.0	0.0	0.0	Satisfactory
wristH	160.0	0.0	0.0	Real tracking not checked for now (same Grieel gains)
wrsitV	150.0	0.0	0.0	Real tracking not checked for now (same Grieel gains)
driving	80.0	0.0	0.0	Satisfactory, WtoG transformation oscillations reduced a lot with new sim set-up

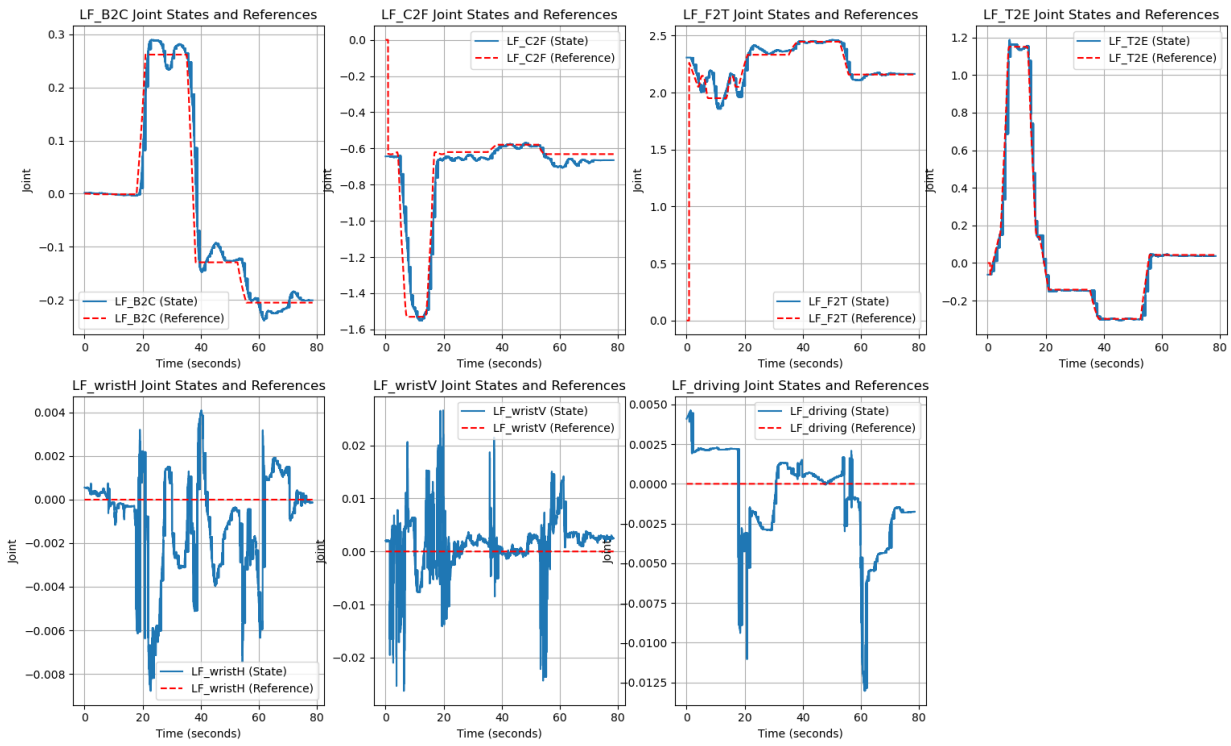


Try to reduce the oscillations and the tracking error if possible, a gravity compensation feed forward controller may help in achieving better performances, maybe just a feedback PID controller is not enough to achieve perfect tracking, also by iterative tuning on such a complex system we cannot easily obtain 100% desired performances.

#6) C2F Tuning, PID control

Try combinations of I and D gain able to decrease oscillations and reach reference trajectory

JOINT	Kp	Ki	Kd	Comments on performances
B2C	65.0	0.01	1.2	Satisfactory, negligible overhoot
C2F	90.0	0.1	2.0	Less oscillations and a bit better tracking (0 error cannot be achieved also because inv kin for now is solved for Limbero end effector, not with Grieel)
F2T	50.0	0.0	0.0	Some aggressiveness and overshoot reduction
T2E	125.0	0.0	0.0	Satisfactory
wristH	160.0	0.0	0.0	Real tracking not checked for now (same Grieel gains)
wrsitV	150.0	0.0	0.0	Real tracking not checked for now (same Grieel gains)
driving	80.0	0.0	0.0	Satisfactory, WtoG transformation oscillations reduced a lot with new sim set-up



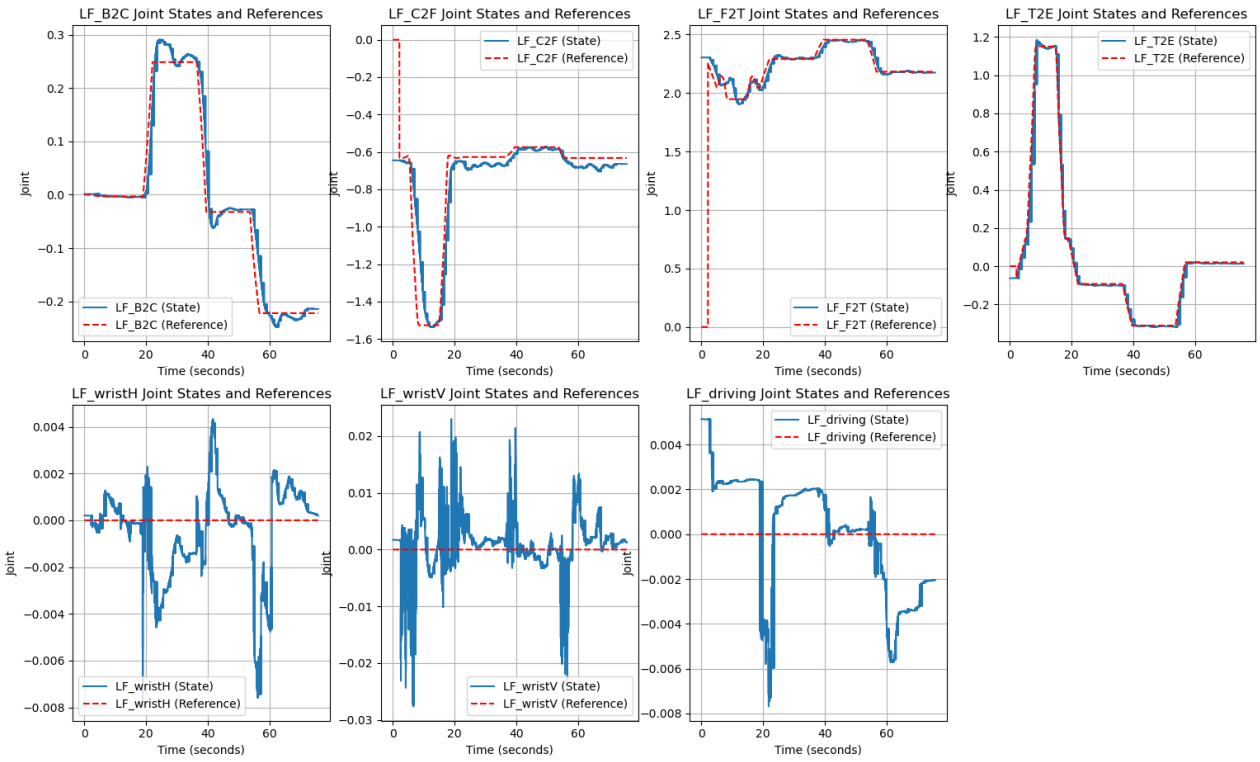
Also C2F PID is tuned, the final steady state error can be caused by the C2F offset when Grieel is in contact with the ground, due to the inverse kinematic solution not coherent.

>**“i=3, F2T”**, Here the tracking is very bad and important oscillations shows up, this joint is very hard to tune because it has to counteract the effect of Grieel mass, being also moved by the previous join, with respect to B2C that is fixed at the limb root, so we expect that the achievable performance without a model based tuning and advanced control techniques is limited.

#7) F2T Tuning, PD control

Try to increase D to reduce oscillations, a decrease of P may cause a reduction of the torque applied which we need to avoid to keep end effector raised during tasks

JOINT	Kp	Ki	Kd	Comments on performances
B2C	65.0	0.01	1.2	Satisfactory, negligible overshoot
C2F	90.0	0.1	2.0	Satisfactory, small steady state error
F2T	50.0	0.0	1.0	Significant oscillation and overshoot reduction
T2E	125.0	0.0	0.0	Satisfactory
wristH	160.0	0.0	0.0	Real tracking not checked for now (same Grieel gains)
wrsitV	150.0	0.0	0.0	Real tracking not checked for now (same Grieel gains)
driving	80.0	0.0	0.0	Satisfactory, WtoG transformation oscillations reduced a lot with new sim set-up

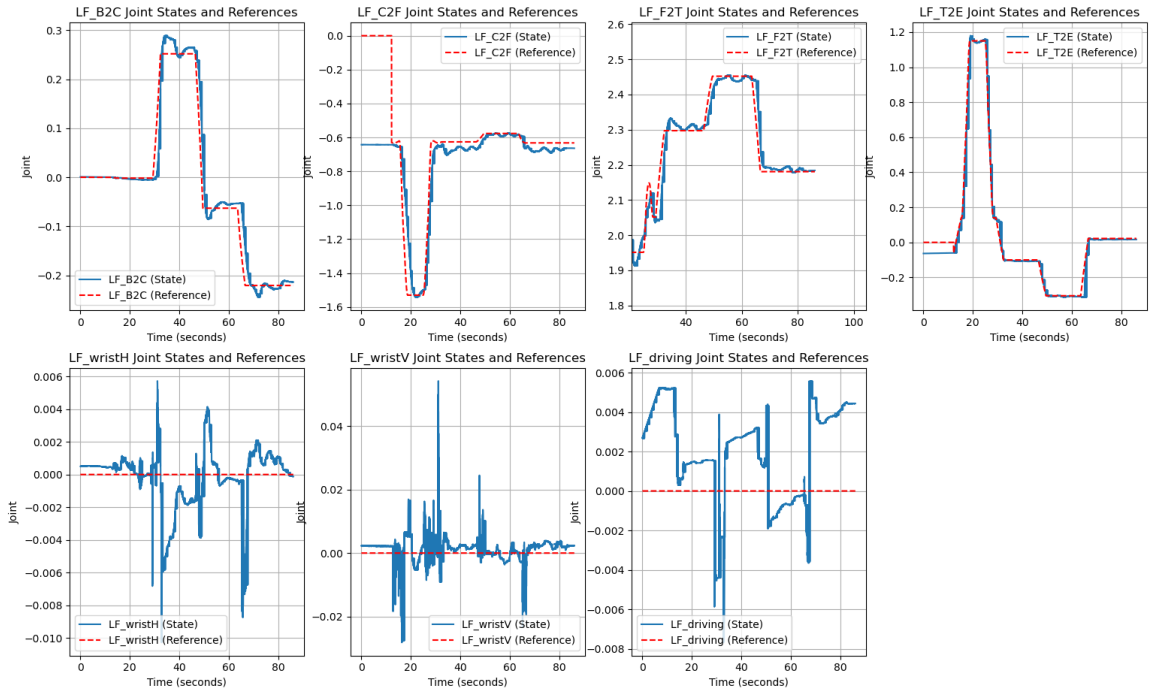


This F2T tuning is already satisfying, I gain may be not needed, but as good practice, verify if a different combination of I and D is able to do better. And maybe a small increase over P gain can be helpful.

#8) F2T Tuning, PID control

Try to use I action and different combination of P,D to reduce a bit tracking error, the trajectory is already very good, except for the initial part due to the noise caused by subsequent joints motion.

JOINT	Kp	Ki	Kd	Comments on performances
B2C	65.0	0.01	1.2	Satisfactory, negligible overshoot
C2F	90.0	0.1	2.0	Satisfactory, small steady state error
F2T	60.0	0.02	1.2	Not big improvements..
T2E	125.0	0.0	0.0	Satisfactory
wristH	160.0	0.0	0.0	Real tracking not checked for now (same Grieel gains)
wrsitV	150.0	0.0	0.0	Real tracking not checked for now (same Grieel gains)
driving	80.0	0.0	0.0	Satisfactory, WtoG transformation oscillations reduced a lot with new sim set-up



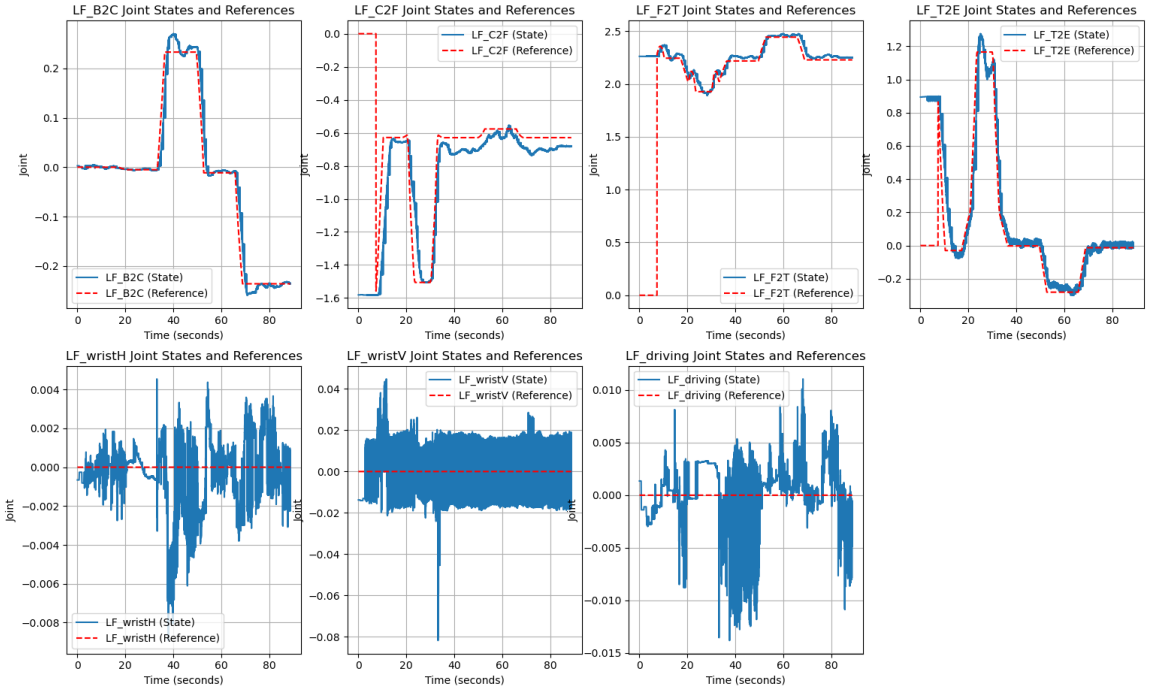
Trying several combinations of D and I for F2T, performance obtained on case#7 seems better, this case#8 is reported to show an additional tuning case for this joint. Also be carefully that the time scale of the two plots si different, and also vertical axis has different granularity, the two trajectories of F2T in case #7 and #8 are very similar. Just increase a bit Kp to 55.0 remain safe and ensure that if higher torque is required the controller is able to guarantee the request.

Then, keep PD#7 tuning and proceed with next joint
>“i=4, T2E”

#9) T2E Tuning, PD control

Performances of T2E trajectory are already very good, it is just used to change end-effector pitch angle and keep the desired angle with respect to Grieel. In principle we can leave it as it is, just for safety of the motors, to avoid torque saturation, we can decrease a bit proportional gain and insert some damp effect.

JOINT	Kp	Ki	Kd	Comments on performances
B2C	65.0	0.01	1.2	Satisfactory, negligible overshoot
C2F	90.0	0.1	2.0	Tracking error increased
F2T	55.0	0.0	1.0	Satisfactory, small tracking error
T2E	100.0	0.0	0.8	Vibrate more and track with higher error
wristH	160.0	0.0	0.0	High frequency vibration
wrsitV	150.0	0.0	0.0	High frequency vibration
driving	80.0	0.0	0.0	vibration



Tuning example shown to demonstrate how with small tuning differences in the controller gains, simulation becomes unstable and performance get worse. In Gazebo simulation we can see how this

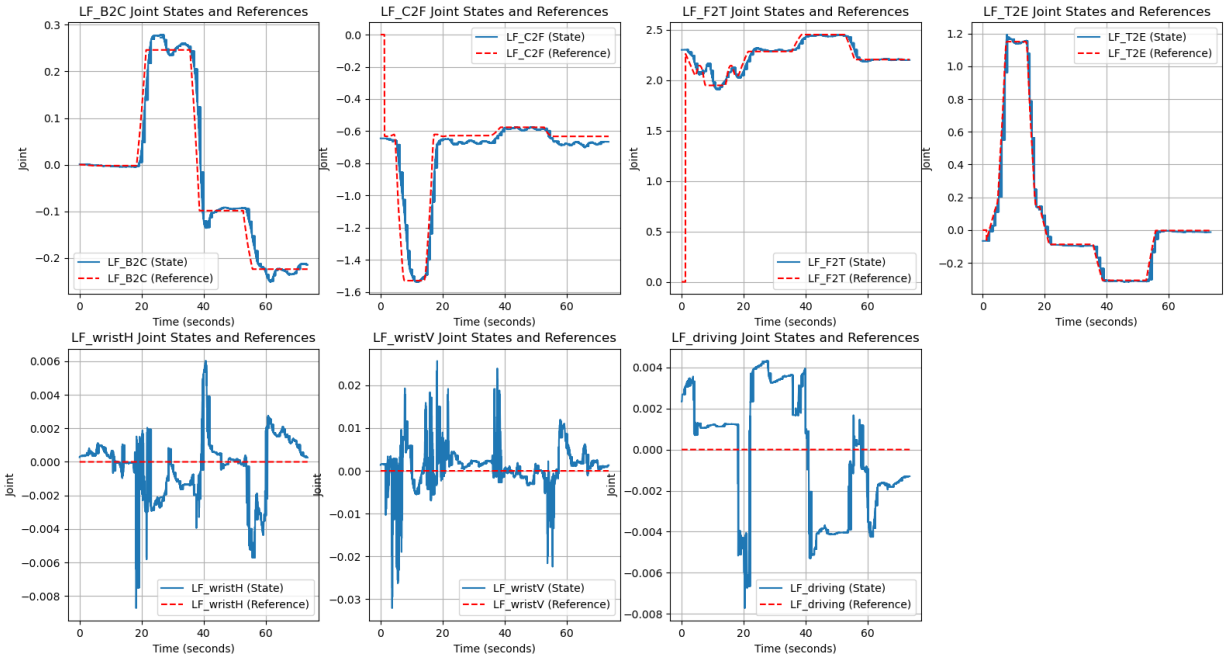
tuning is very bad and end effector movement becomes less natural, causing also skating motion with the ground.
(This is why, with the default ros2_control trajectory control PID tuning the simulation was completely unreliable)

Maybe some dumping can be useful in the controller, but we need to decrease it significantly to avoid this issue, because damping gain can cause noise amplification. And due to the natural oscillatory behavior of the wrist joints, to keep angle to 0, a derivative gain on T2E responds in this nervous way to the high frequency change (so big joint angle error rate) in Grieel.
A Proportional controller with maybe a bit less proportional gain is the best choice, than if in the real system we notice some low frequency oscillations or undesired oscillations we can think about small derivative gain increase on T2E dynamixel motor.

#10) T2E Tuning, P control

Performances of T2E trajectory are already very good, it is just used to change end-effector pitch angle and keep the desired angle with respect to Grieel. In principle we can leave it as it is, just for safety of the motors, to avoid torque saturation, we can decrease a bit proportional gain and insert some damp effect.

JOINT	Kp	Ki	Kd	Comments on performances
B2C	65.0	0.01	1.2	Satisfactory, negligible overshoot
C2F	90.0	0.1	2.0	Satisfactory, small steady state error
F2T	55.0	0.0	1.0	Satisfactory, small tracking error
T2E	100.0	0.0	0.0	Not big difference, satisfactory
wristH	160.0	0.0	0.0	Real tracking not checked for now (same Grieel gains)
wrsitV	150.0	0.0	0.0	Real tracking not checked for now (same Grieel gains)
driving	80.0	0.0	0.0	Satisfactory, WtoG transformation oscillations reduced a lot with new sim set-up



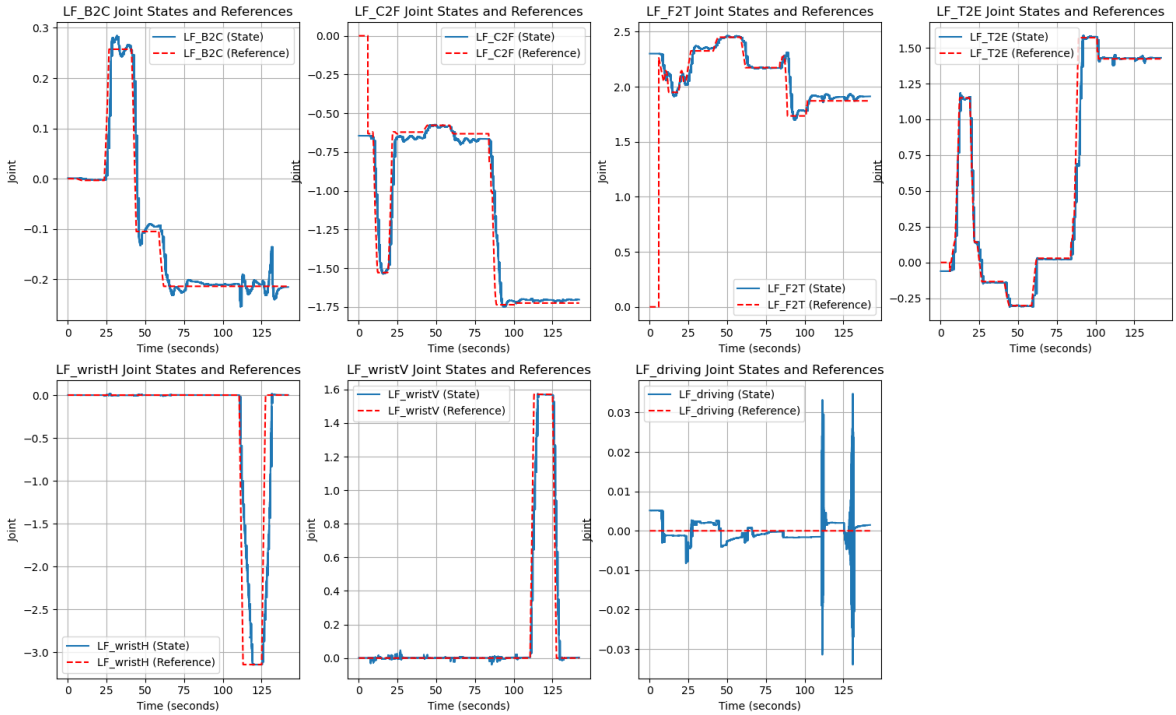
>"i=5, wristH"

All LIMBERO limb joints have been tuned, now its the turn of Grieel Joints, for this we will also simulate a change of mode configuration to track control performances after Grieel receive joint trajectory command from limb_controller

#11) wristH Tuning, PD control

wristH until now has been just actuated sending command 0, and the performances were very good considering that it has to account for friction with the ground and kept gripper configuration with almost 0 error. But to guarantee good tracking to Grieel commands lets proceed as for the other joints

JOINT	Kp	Ki	Kd	Comments on performances
B2C	65.0	0.01	1.2	Some additional over and undershoot introduced by new wristH gains
C2F	90.0	0.1	2.0	Satisfactory, small steady state error
F2T	55.0	0.0	1.0	Satisfactory, small tracking error
T2E	100.0	0.0	0.0	Not big difference, satisfactory
wristH	120.0	0.0	0.01	Good tracking
wrsitV	150.0	0.0	0.0	Good tracking
driving	80.0	0.0	0.0	Satisfactory, but oscillations increased by wrist control

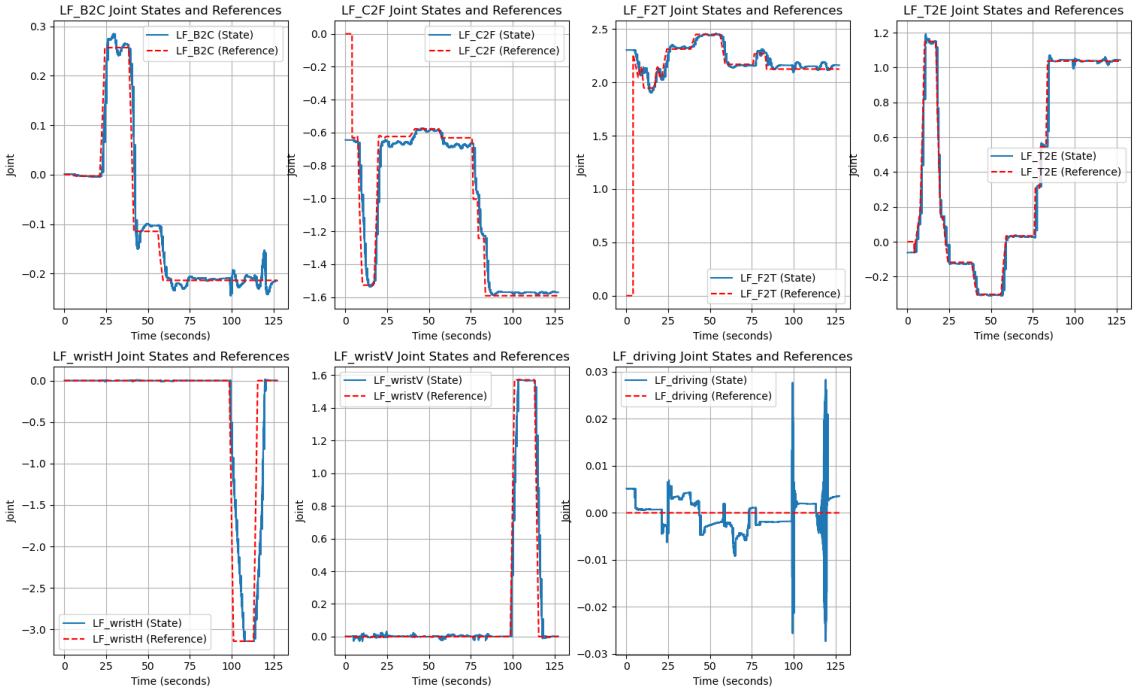


In terms of wristH trajectory tracking performances it is good, but notice that the derivative gain make it more reactive to high frequency error change rate, propagating to the driving joint that has a bit higher error than before, the 0.03 magnitude oscillations on driving joints appear when Grieel change configuration, they are not so crucial and also thanks to the new simulation coefficient they are almost 10 times smaller than the previous unreliable simulation. The cascade effect to B2C is caused by less proportional gain and so maybe less “violent” reaction to a motion of the B2C joint to keep wristH to 0, try to solve it with some different gains combination of WristH or later in the bottom-up retuning.
The necessity of decreasing wristH gain is to ensure realistic torque requirements to actuator when controller will be implemented on real motors.

#12) wristH Tuning, PID control

wristH until now has been just actuated sending command 0, and the performances were very good considering that it has to account for friction with the ground and kept gripper configuration with almost 0 error. But to guarantee good tracking to Grieel commands lets proceed as for the other joints

JOINT	Kp	Ki	Kd	Comments on performances
B2C	65.0	0.01	1.2	Satisfactory,Some additional over and undershoot introduced by new wristH gains, but less than case #11
C2F	90.0	0.1	2.0	Satisfactory, small steady state error
F2T	55.0	0.0	1.0	Satisfactory, small tracking error
T2E	100.0	0.0	0.0	Not big difference, satisfactory
wristH	120.0	0.001	0.05	Good tracking
wrsitV	150.0	0.0	0.0	Good tracking
driving	80.0	0.0	0.0	Satisfactory, but oscillations increased by wrist control



Not big improvement with respect to previous behavior. But for robustness it is good to keep this I and D gain, that doesn't affect simulation quality and can help in real robot control.

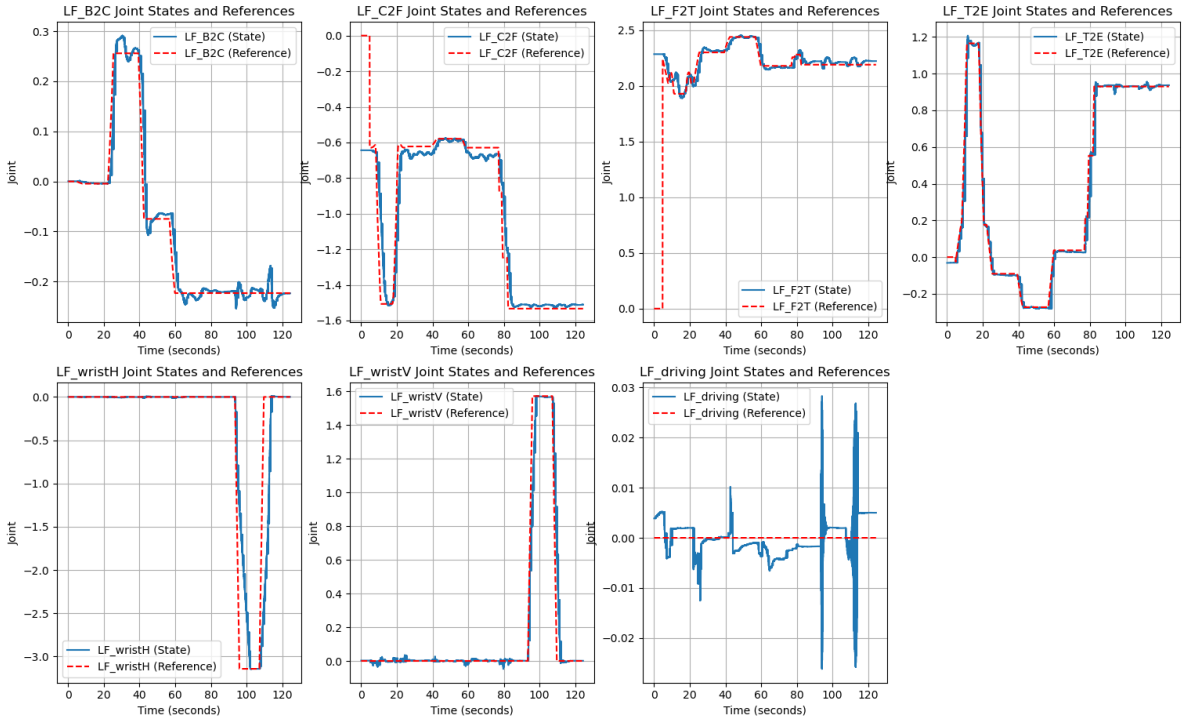
>"i=6, wristV"

Proceed in the tuning with similar idea to the driving joint tuning, this is also a pitch DOF joint, so it is reactive to driving mechanism oscillations, big attention has to be taken during tuning.

#13) wristV Tuning, PD control

wristH until now has been just actuated sending command 0, and the performances were very good considering that it has to account for friction with the ground and kept gripper configuration with almost 0 error. But to guarantee good tracking to Grieel commands lets proceed as for the other joints

JOINT	Kp	Ki	Kd	Comments on performances
B2C	65.0	0.01	1.2	Satisfactory,Some additional over and undershoot introduced by wrist gains, but negligible
C2F	90.0	0.1	2.0	Satisfactory, small steady state error
F2T	55.0	0.0	1.0	Satisfactory, small tracking error
T2E	100.0	0.0	0.0	Not big difference, satisfactory
wristH	120.0	0.001	0.05	Good tracking
wrsitV	120.0	0.0	0.01	Good tracking , not big differences
driving	80.0	0.0	0.0	Satisfactory, but oscillations increased by wrist control



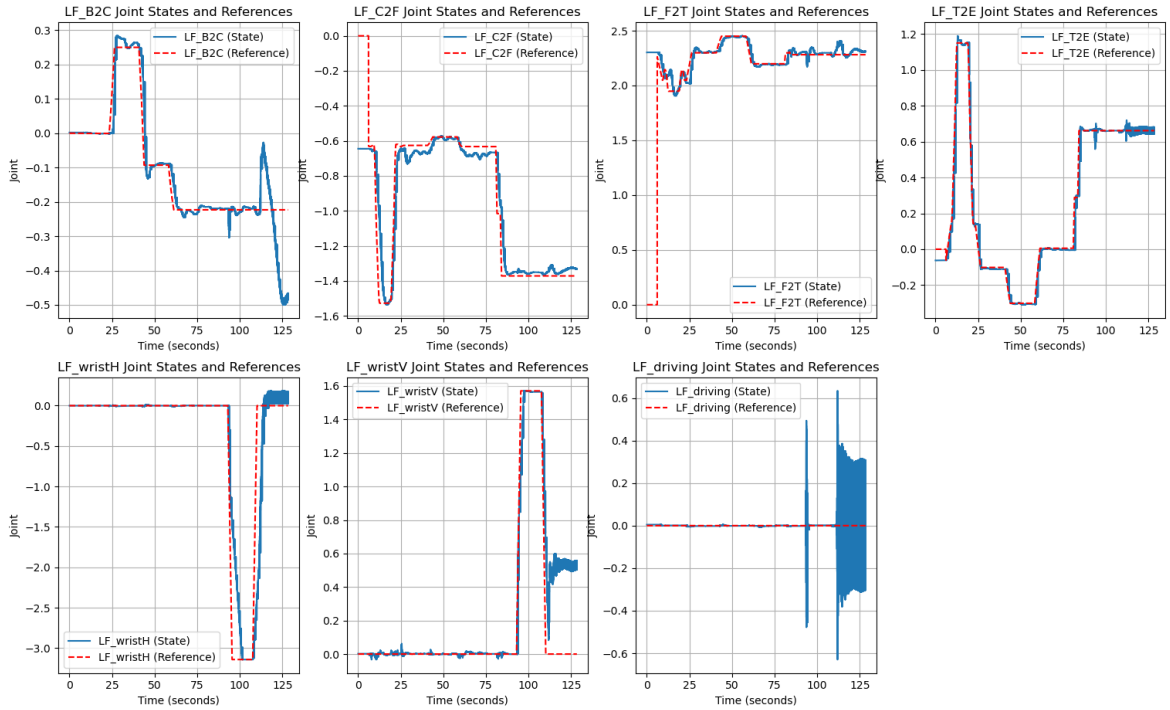
Not big improvement with respect to previous behavior.
For now let's see if we can decrease driving vibration with different driving joint tuning, otherwise when we do the bottom-up quick returning we will try to refix Wrist controller and check if simple P controllers for wristH and wristV are better.

> **"i=7, driving"**

#14) driving Tuning, PD control

We want to reduce driving joint vibration, but this is very reactive to controller gains changes due to the contact with ground contact in gazebo simulation, small changes can degenerate simulation easily.

JOINT	Kp	Ki	Kd	Comments on performances
B2C	65.0	0.01	1.2	Satisfactory,Some additional over and undershoot introduced by wrist gains, and instability due to driving
C2F	90.0	0.1	2.0	Satisfactory, small steady state error
F2T	55.0	0.0	1.0	Satisfactory, small tracking error
T2E	100.0	0.0	0.0	Not big difference, satisfactory but vibration due to driving
wristH	120.0	0.001	0.05	Good tracking , domino effect of driving when unstable
wrsitV	120.0	0.0	0.01	Good tracking , domino effect of driving when unstable
driving	90.0	0.0	0.02	High frequency and High magnitude oscillations, UNSTABLE



Unfortunately, driving mechanism D gain in simulation has to be avoided, due to the unrealistic vibration occurring in the mode transition, that noise cause an high frequency error rate, so a vibration that destabilize the system, and as can be seen in the simulation the robot behaves in a complete dangerous and undesired way.
We can keep a P controller with Kp=80 as used until now.
The retuning of driving joint should be done again when all kinematic is updated (both forward and inverse) and the simulation is capable of testing also wheel mode locomotion, in that case in fact the wheel driving controller is fundamental to achieve a good trajectory tracking in the sense of autonomous navigation, and no more as joint angle tracking (two parallel control problems will need to be faced)

2) BOTTOM-UP (driving to B2C)

tuned driving joint with simulation limits, we can proceed up and think about a better retuning of the joints accordingly to the overall new joints performance with respect to tuning#1.

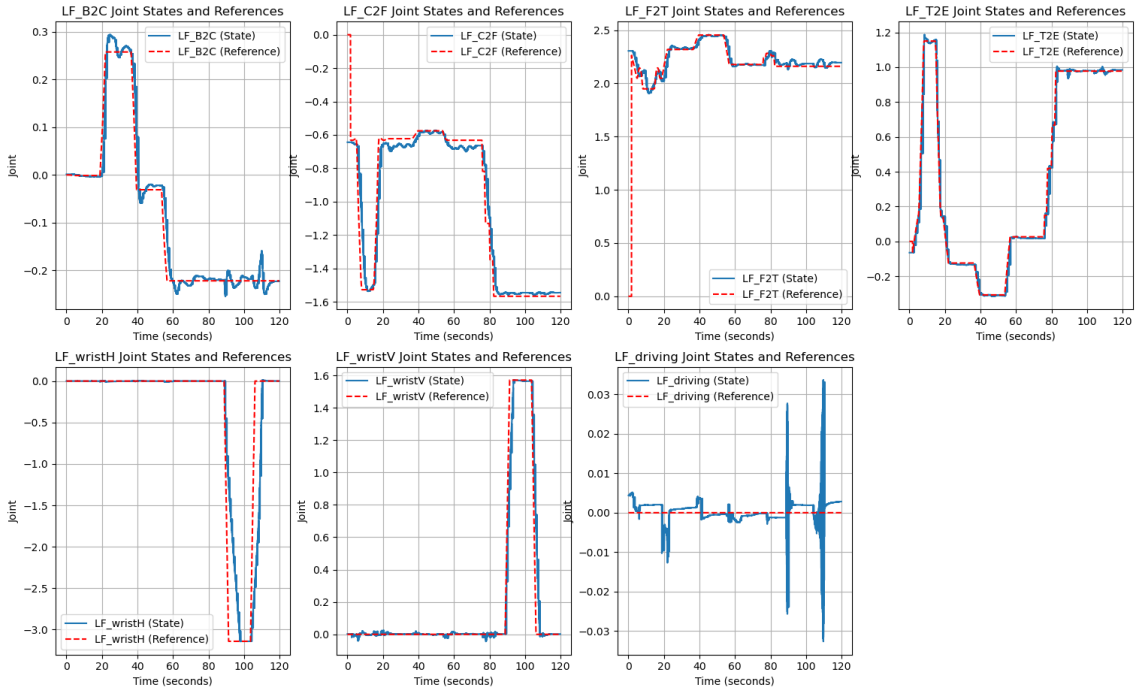
Bottom-up should be faster, we just want to correct the gains and not retune completely.

> “i=6, wristV”

#15) wristV Re-Tuning, PID control

Accordingly to PID#13 performances (the one before this possible re-tuning), performances are already satisfactory. There is just some small vibration that we may want to reduce, but as seen wrist joints are lot reactive to derivative gain changes, due to noise amplification even after small changes.

JOINT	Kp	Ki	Kd	Comments on performances
B2C	65.0	0.01	1.2	Satisfactory,Some additional over and undershoot introduced by wrist gains, but negligible
C2F	90.0	0.1	2.0	Satisfactory, small steady state error
F2T	55.0	0.0	1.0	Satisfactory, small tracking error
T2E	100.0	0.0	0.0	Satisfactory
wristH	120.0	0.001	0.05	Good tracking
wrsitV	110.0	0.001	0.04	Satisfactory, the same as PID#13
driving	80.0	0.0	0.0	Satisfactory, but oscillations increased more by wrist control



Magnitude of driving oscillation is increased and no relevant improvement on performances or in motion smoothness as can be seen from simulation in Gazebo, so we can keep PID#13 tuning.

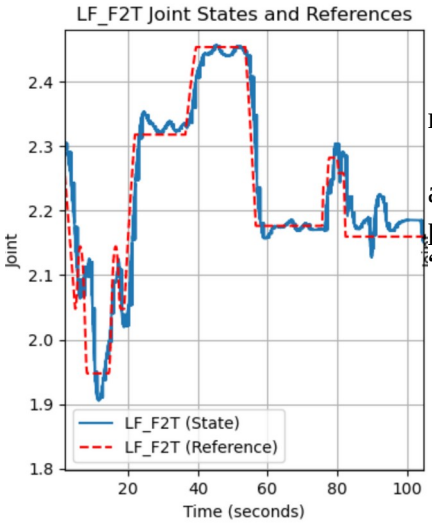
> “i=5, wristH”

There is no need to change this controller gains for now, simulation performances are satisfactory and also trajectory tracking, so for now let’s just keep PID#13 tuning and when this gains will be adapted for real motors we will see if adjustments are necessary.

>”i=4, T2E”

The only negligible oscillations on this joint occurs due to the unrealistic driving oscillations when Grieel W2G transition occurs, and the tracking is almost perfect, no need to re-tune.

> “i=3, F2T”

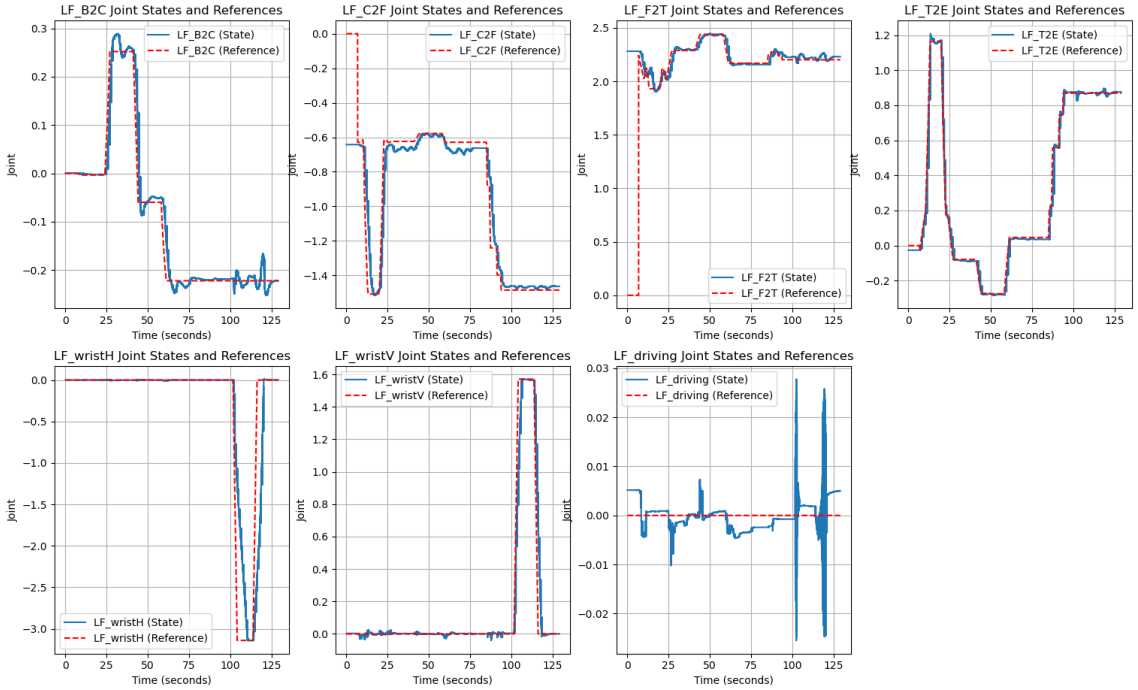


If we zoom on the tracking trajectory when stable transition movement is made, we notice that still some undesired oscillations occur, as said before this is a critical joint, we can try to retune properly now that cascade joints have been adjusted, but achieving “perfect” tracking is very hard.

#16) F2T Re-Tuning, PID control

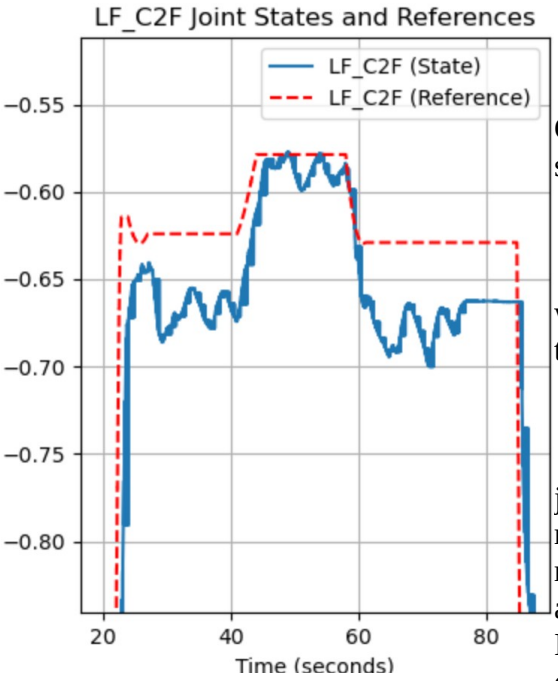
Accordingly to PID#13 performances (the one before this possible re-tuning), performances are already satisfactory. There is just some small vibration that we may want to reduce, but as seen wrist joints are lot reactive to derivative gain changes, due to noise amplification even after small changes.

JOINT	Kp	Ki	Kd	Comments on performances
B2C	65.0	0.01	1.2	Satisfactory,Some additional over and undershoot introduced by wrist gains, but negligible
C2F	90.0	0.1	2.0	Satisfactory, small steady state error
F2T	55.0	0.01	1.4	Satisfactory, negligible changes
T2E	100.0	0.0	0.0	Satisfactory
wristH	120.0	0.001	0.05	Satisfactory
wrsitV	120.0	0.0	0.01	Satisfactory
driving	80.0	0.0	0.0	Satisfactory, but oscillations increased more by wrist control



Changes with respect to PID#13 are almost negligible, but an Integral gain can help us to keep control action when for example additional unexpected load is carried by the interested limb. SO we keep this tuning.

>"i=2, C2F",
as seen before, we are not able to remove that tracking error easily by iteratively retuning C2F gains. Anyway let's try now that cascade joint has ben tuned if some gains combination improve performances of this joint.



Again looking into details of the achieved tracking by C2F joint gains, in some points reference trajectory seems not to be reached. And the error is not negligible.

Investigating on this critical part of the trajectory, this correspond to the moment during transition procedure when limb is put back on the ground after raising (and transitioning).
Than another joint trajectory is sent to move the body.

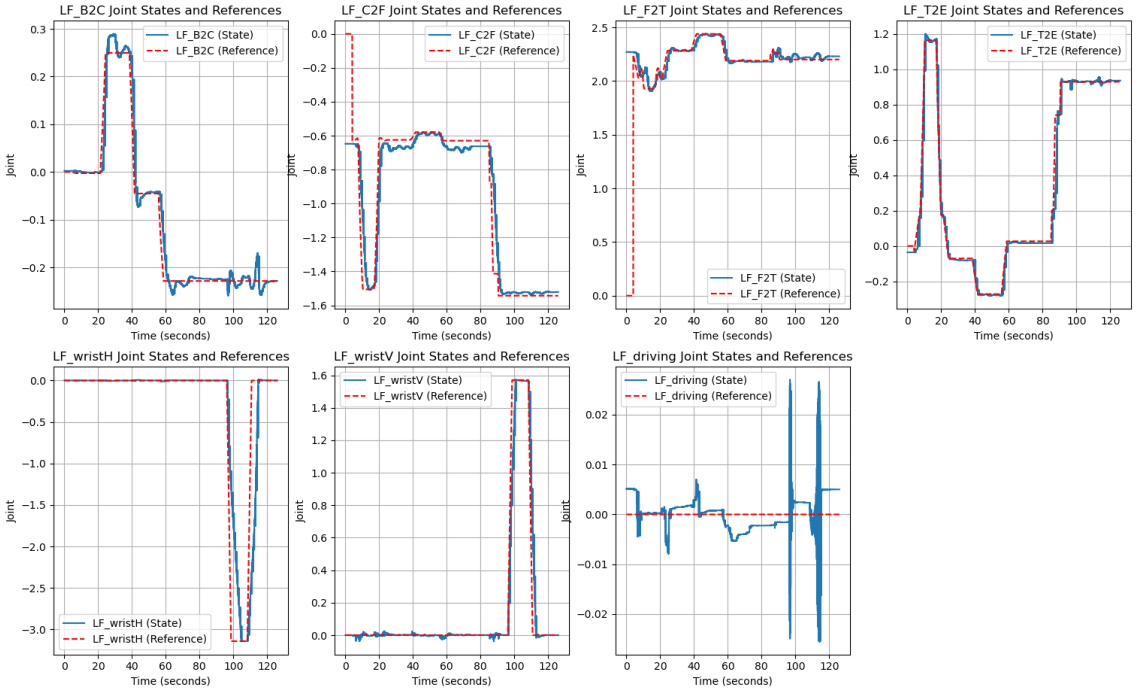
As noticed before, we need to consider the problem of joint trajectory generated for LIMBERO at the moment I'm tuning those PID gains, so limb task of moving the limb on z axis is computed with respect to a foot which is not the real one.
Instead, when we send a raise up limb command, the desired joint angle is reached with negligible error, meaning that maybe the controller is able to control properly the joints.

Anyway, let’s check if oscillations can be reduced, considering that this constant error is not a big problem.
(It is highly probable that we keep previous tuning case PID#16)

#17) C2F Re-Tuning, PID control

For oscillations reduction we can increase derivative gain, and maybe decrease a bit integral one. Notice that it is reasonable to keep proportional one with high value due to the torque requirements at this joint.

JOINT	Kp	Ki	Kd	Comments on performances
B2C	65.0	0.01	1.2	Satisfactory,Some additional over and undershoot introduced by wrist gains, but negligible
C2F	90.0	0.08	2.4	Satisfactory, small steady state error same as before
F2T	55.0	0.01	1.4	Satisfactory, negligible changes
T2E	100.0	0.0	0.0	Satisfactory
wristH	120.0	0.001	0.05	Satisfactory
wrsitV	120.0	0.0	0.01	Satisfactory
driving	80.0	0.0	0.0	Satisfactory, but oscillations increased more by wrist control



No need to retune this joint, we can keep PID#16 tuning for now.

> “i=1, B2C”

From the next joints retuning we introduce some additional overshoot, which anyway is of negligible magnitude. So we can conclude that PID#16 is the optimal gain combination.