Internship Report

Internship Report : Implementation of a whole-body MPC on TIAGO

GEPETTO LAAS-CNRS

ENSEEIHT
EEEA

Project Report TIAGO

Loic Barthe

Table des matières

1 Introduction

The objective of this project is to implement a whole-body Model Predictive Control (MPC) system on the robot to control the position of TIAGO's arm. This report serves to document the progress and methods used in the project. Typically, TIAGO's arm can be controlled independently in terms of torque, position, or velocity. However, for controlling the torso, only position control is feasible. Consequently, the decision was made to control the entire robot using position control.

This project builds upon the foundational work of Julien Gleyze, who developed and tested the Model Predictive Control (MPC) system in a simulation environment. The current task involves modifying the existing MPC framework, rigorously testing these modifications in simulation, and subsequently implementing the refined MPC system on the TIAGO robot. This progression from simulation to real-world application is crucial to ensure the robustness and reliability of the control system in a practical setting.

This report provides a step-by-step guide for installing everything necessary to simulate the robot effectively. It includes installation files, commands, and advice aimed at helping users become familiar with the robot as seamlessly as possible.

1.1 Context of the study

TIAGo is a mobile humanoïd service robot specifically designed for indoor environments by the Spanish company called *PAL_ROBOTICS*. Equipped with an extendable torso and a manipulator arm, TIAGO is capable of grasping tools and objects. Its comprehensive sensor suite enables it to perform diverse tasks including perception, manipulation, and navigation with precision and efficiency.

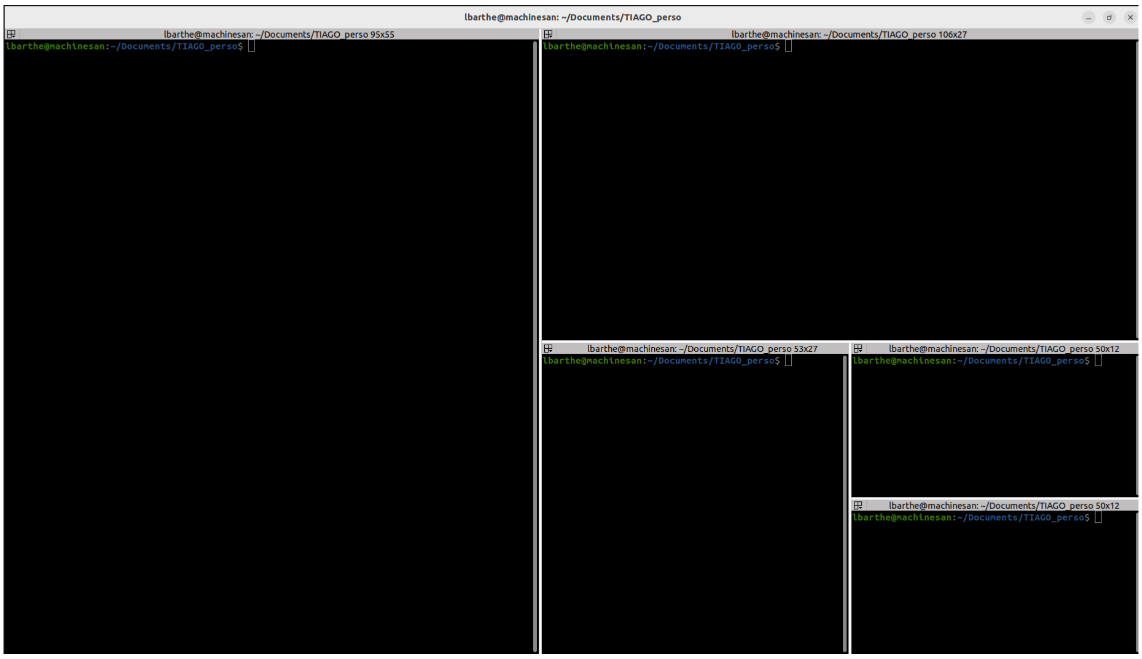
Initially developed on ROS1 and now updated to ROS2, the aim of this internship is to design a whole-body Model Predictive Controller (MPC) for controlling the arm position of the robot. Various control methods exist, including torque, position, and velocity control.

Given that the torso is limited to position control only, the arm will also utilize this method for consistency. Therefore, this report will exclusively focus on implementing position control for the arm.

2 Installation

2.1 Terminator

A highly effective tool for working on a Linux machine is **Terminator**. It allows you to manage multiple terminals within a single window, with the ability to split the interface both horizontally and vertically to create additional terminal instances. Let's explore this further below :


lbarthe@machinesan: ~/Documents/TIAGO_perso
lbarthe@machinesan: ~/Documents/TIAGO_perso 95x55
lbarthe@machinesan: ~/Documents/TIAGO_perso 106x27
lbarthe@machinesan: ~/Documents/TIAGO_perso 53x27
lbarthe@machinesan: ~/Documents/TIAGO_perso 50x12
lbarthe@machinesan: ~/Documents/TIAGO_perso 50x12

```
lbarthe@machinesan:~/Documents/TIAGO_perso$
```

```
lbarthe@machinesan:~/Documents/TIAGO_perso$
```

```
lbarthe@machinesan:~/Documents/TIAGO_perso$
```

```
lbarthe@machinesan:~/Documents/TIAGO_perso$
```

```
lbarthe@machinesan:~/Documents/TIAGO_perso$
```

Figure 1 – Screen using terminator

Some useful commands and shortcuts to effectively use Terminator :

- **Split Terminal Horizontally :**
  - Shortcut : `Ctrl + Shift + O`
- **Split Terminal Vertically :**
  - Shortcut : `Ctrl + Shift + E`
- **Open a New Tab :**
  - Shortcut : `Ctrl + Shift + T`
- **Close the Terminal :**
  - Shortcut : `Ctrl + Shift + W`
- **Navigate Between Terminals :**
  - Move to the Terminal Above : `Ctrl + Shift + Up`
  - Move to the Terminal Below : `Ctrl + Shift + Down`
  - Move to the Terminal Left : `Ctrl + Shift + Left`
  - Move to the Terminal Right : `Ctrl + Shift + Right`
- **Resize Terminals :**
  - Resize Vertically : `Ctrl + Shift + Up/Down`
  - Resize Horizontally : `Ctrl + Shift + Left/Right`
- **Broadcast Input to All Terminals :**
  - Shortcut : `Ctrl + Shift + A`
- **Toggle Full Screen :**
  - Shortcut : `F11`
- **Open Preferences :**
  - Shortcut : `Ctrl + Shift + P`
- **Search Terminal :**
  - Shortcut : `Ctrl + Shift + F`

WARNING : Each time a new terminal is open, do not forget to source your environement again !

2.2 ROBOTPKG

An easy way to use Pinocchio and Crocoddyl (two important libraries for the simulation), is to use directly **robotpkg**. robotpkg is a compilation framework and packaging system for installing robotics software developed by the robotic community. It also contains packages for some general, third-party open-source software that the robotics software depends on and that is not commonly packaged by major unix distributions. For more information about robotpkg, there is the website : http ://robotpkg.openrobots.org/

2.2.1 Installation

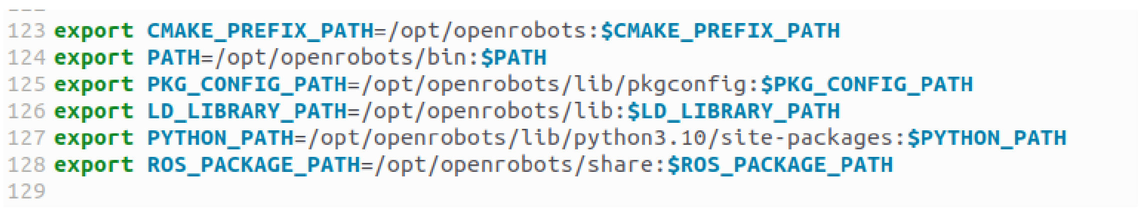
To install robotpkg, ...

The procedure to correctly install robotpkg can be found at the webpage :

`http://robotpkg.openrobots.org/debian.html`

2.2.2 Environment Variables

After being installed, it is necessary to setup the environement with good variables, to make possible for your computer to find the correct application. Below are mentionned, all the required environement variables :



```
export CMAKE_PREFIX_PATH=/opt/openrobots:$CMAKE_PREFIX_PATH
export PATH=/opt/openrobots/bin:$PATH
export PKG_CONFIG_PATH=/opt/openrobots/lib/pkgconfig:$PKG_CONFIG_PATH
export LD_LIBRARY_PATH=/opt/openrobots/lib:$LD_LIBRARY_PATH
export PYTHON_PATH=/opt/openrobots/lib/python3.10/site-packages:$PYTHON_PATH
export ROS_PACKAGE_PATH=/opt/openrobots/share:$ROS_PACKAGE_PATH
```

FIGURE 2 – Setup Environment

`https://stack-of-tasks.github.io/sot-doc/doxygen/HEAD/d_setting_up_environment.html`

Let's have a look about the PATH variable :

the PATH variable is a concatenation of all paths to executables. In other words, when you want to run a program, for example firefox, the computer will browse this list from the beginning until finding a path behavior the keyword "firefox" and that runs the web browser. This has as impact that the order in which we store our executables in this variable PATH has importance because the paths closer to the beginning of the list will have a higher priority than those farther in this list.

CMAKE_PREFIX_PATH

Description : CMAKE_PREFIX_PATH is used by CMake to search for configuration files. By adding /opt/openrobots to the beginning of the existing CMAKE

_PREFIX_PATH , it allows CMake to find the libraries and header files located in that directory. It is mainly used when compiling software with CMake.

PATH

Description : PATH is a system environment variable that specifies the directories in which the system should search for executables. By adding/opt/openrobots/bin to the beginning of the existing PATH, this allows the system to find and run programs in that directory. Used to run executables without specifying their full path.

PKG_CONFIG_PATH

Description : PKG_CONFIG_PATH is used by pkg-config to find .pc files (package configuration files). Adding /opt/openrobots/lib/pkgconfig to the beginning of the existing PKG_CONFIG_PATH allows pkg-config to locate the configuration information of the packages installed in that directory. Used when configuring and compiling software to find packages and their dependencies.

LD_LIBRARY_PATH

Description : LD_LIBRARY_PATH is an environment variable used by the dynamic library loader on Unix to search for shared libraries (.so). Adding /opt/openrobots/lib to the beginning of the existing LD_LIBRARY_PATH allows the system to find the dynamic libraries located in that directory.

PYTHON_PATH

Description : PYTHON_PATH is used by the Python interpreter to locate Python modules. By adding/opt/openrobots/lib/python3.10/site-packages to the beginning of the existing PYTHON_PATH, this allows Python to find the modules installed in this directory. Used to run Python scripts that depend on modules installed in specific directories.

ROS_PACKAGE_PATH

Description : ROS_PACKAGE_PATH is used by the Robot Operating System (ROS) package system to locate ROS packages. By adding /opt/openrobots/share at the beginning of the existing ROS_PACKAGE_PATH, this allows ROS to find the packages located in this directory. Used to run ROS programs and scripts that depend on packages installed in specific directories.

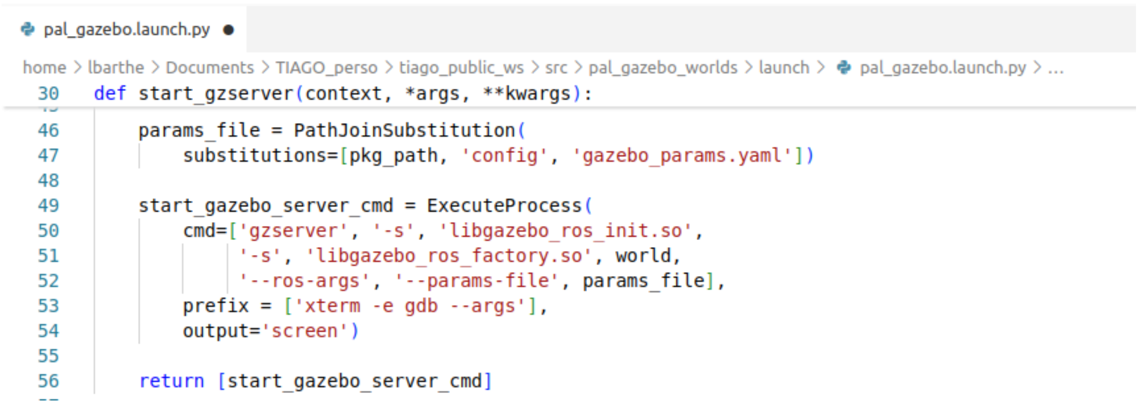
2.3 GDB Debugger

GDB (GNU Debugger) is a powerful tool for debugging programs written in languages like C and C++. It allows developers to inspect the state of a program while it runs or after it crashes, enabling them to set breakpoints, step through code, and examine variables. GDB helps identify and resolve bugs by providing detailed insights into program execution.

To install GDB, you need to use a prefix in the `executeProcess` of `pal_gazebo_worlds`. When you start the TIAGO simulation with the command :

```
ros2 launch tiago_gazebo tiago_gazebo.launch.py is_public_sim:=True
```

Various nodes will be launched concurrently. One of these nodes, `pal_gazebo_worlds`, is responsible for loading the robot and its environment. By adding the prefix `prefix=['xterm -e gdb -args']`, you can launch GDB, which will allow you to monitor and debug any issues.

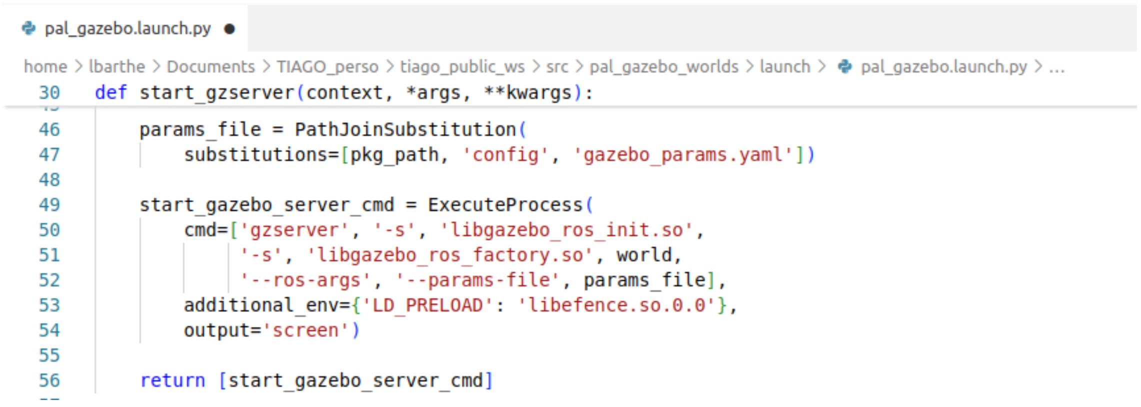

pal_gazebo.launch.py
home > lbarthe > Documents > TIAGO_perso > tiago_public_ws > src > pal_gazebo_worlds > launch > pal_gazebo.launch.py > ...

```python
def start_gzserver(context, *args, **kwargs):
    params_file = PathJoinSubstitution(
        substitutions=[pkg_path, 'config', 'gazebo_params.yaml'])

    start_gazebo_server_cmd = ExecuteProcess(
        cmd=['gzserver', '-s', 'libgazebo_ros_init.so',
             '-s', 'libgazebo_ros_factory.so', world,
             '--ros-args', '--params-file', params_file],
        prefix = ['xterm -e gdb --args'],
        output='screen')

    return [start_gazebo_server_cmd]
```

2.4 Electric Fense

Electric Fence (e.g., Electric Fence) is a debugging tool for detecting memory allocation errors in C and C++ programs. It helps identify issues like buffer overflows and memory leaks by allocating memory in a way that detects out-of-bounds accesses. The tool works by placing guard pages around allocated memory, so any invalid access triggers a segmentation fault. This allows developers to catch and fix memory-related bugs more effectively.

To install Electric Fence, the process is similar to that for GDB, but instead of using a prefix, you need to configure an additional environment variable. Specifically, set $additional\_env = 'LD\_PRELOAD' :' libefence.so.0.0'$ to preload the Electric Fence library. This setup allows Electric Fence to monitor memory allocation and detect errors during execution.


pal_gazebo.launch.py ●

home > lbarthe > Documents > TIAGO_perso > tiago_public_ws > src > pal_gazebo_worlds > launch > pal_gazebo.launch.py > ...

```
def start_gzserver(context, *args, **kwargs):
    params_file = PathJoinSubstitution(
        substitutions=[pkg_path, 'config', 'gazebo_params.yaml'])

    start_gazebo_server_cmd = ExecuteProcess(
        cmd=['gzserver', '-s', 'libgazebo_ros_init.so',
             '-s', 'libgazebo_ros_factory.so', world,
             '--ros-args', '--params-file', params_file],
        additional_env={'LD_PRELOAD': 'libefence.so.0.0'},
        output='screen')

    return [start_gazebo_server_cmd]
```

2.5 Valgrind

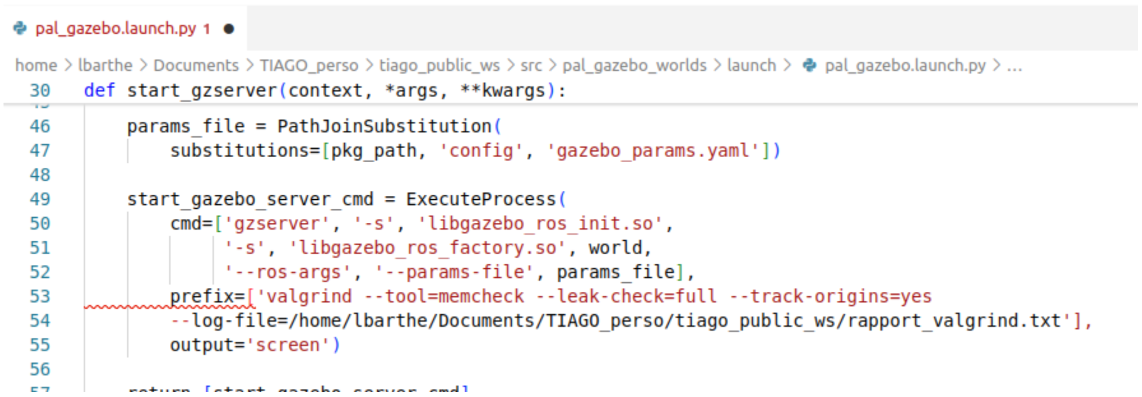
Valgrind is a powerful tool for memory debugging and profiling. It helps detect memory leaks, invalid memory accesses, and other memory-related errors in programs. By running a program under Valgrind, you can get detailed reports about memory usage and potential issues, which aids in improving code quality and reliability. It is commonly used in development to identify and fix memory management problems before deployment.

To install GDB, you need to use a prefix in the `executeProcess` of `pal_gazebo_worlds`. There are several options when using Valgrind, including :

— `-tool=memcheck` : Specifies that you want to use Valgrind's Memcheck tool, which is specialized in detecting memory management errors.

— `-leak-check=full` : Requests Valgrind to perform a comprehensive report on memory leaks, including those that are reachable, lost, indirectly lost, etc.

— `-track-origins=yes` : Instructs Valgrind to track the origins of uninitialized memory errors, which can be very helpful in pinpointing the exact source of the issue.



pal_gazebo.launch.py 1 ●

home > lbarthe > Documents > TIAGO_perso > tiago_public_ws > src > pal_gazebo_worlds > launch > pal_gazebo.launch.py > ...

```
def start_gzserver(context, *args, **kwargs):
    params_file = PathJoinSubstitution(
        substitutions=[pkg_path, 'config', 'gazebo_params.yaml'])

    start_gazebo_server_cmd = ExecuteProcess(
        cmd=['gzserver', '-s', 'libgazebo_ros_init.so',
             '-s', 'libgazebo_ros_factory.so', world,
             '--ros-args', '--params-file', params_file],
        prefix=['valgrind --tool=memcheck --leak-check=full --track-origins=yes
        --log-file=/home/lbarthe/Documents/TIAGO_perso/tiago_public_ws/rapport_valgrind.txt'],
        output='screen')

```

2.5.1 Difference between `prefix` and `additional_env`

Prefix

Usage : The `prefix` is typically used to prepend a command or set of commands that will run before the main command is executed. This is commonly used in contexts like launching or executing processes.
Purpose : It allows you to modify or wrap the main command with additional functionality. For example, you might use `prefix` to run a program inside a debugger or a profiler.
Example : If you want to run a program under GDB (GNU Debugger), you might set a prefix like `['xterm -e gdb -args']`. This ensures that the program is executed in an xterm terminal with GDB.

`Additional_env`

Usage : The `additional_env` is used to specify additional environment variables that should be set when executing a command. These variables can modify the behavior of the command or program being executed.
Purpose : It allows you to influence the runtime environment of the command by setting variables like `LD_PRELOAD`, `PATH`, or other environment-specific settings. This is useful for configuring how the program interacts with its environment.
Example : To use Electric Fence for memory debugging, you might set `additional_env={ĹD_PRELOAD:́ ĺibefence.so.0.0}́`. This instructs the system to preload the Electric Fence library, which will affect how memory allocation is handled by the program.

Summary

- **Prefix :** Used to modify the command itself or how it is executed (e.g., running under a debugger).
- **Additional Environment :** Used to set or modify environment variables for the command's execution (e.g., setting up a memory debugging library).

Both methods are used to customize and enhance the execution environment of programs, but they do so in different ways.

3 Ros2 Package Architecture

3.1 General Concept

Packages

- **Definition :** Packages are the fundamental units in ROS 2. They contain software components such as nodes, libraries, configurations, and dependencies.
- **Content :** A ROS 2 package can include source code, executables, configuration files, messages, services, and action definitions.

Nodes

- **Definition :** Nodes are individual executable processes within a package. Each node performs a specific task, such as handling sensors or controlling actuators.
- **Communication :** Nodes communicate with each other via topics, services, or actions.

Topics

- **Definition :** Topics are communication channels for messages. Nodes can publish messages to topics or subscribe to topics to receive messages.
- **Usage :** Ideal for continuous data streams, such as sensor readings.

Services

- **Definition :** Services provide a request-response communication mechanism between nodes. A node sends a request and waits for a response.
- **Usage :** Used for operations that require an immediate response, such as position requests.

Configuration Files

— **Definition :** Configuration files define parameters and settings for nodes.
— **Usage :** Allows customization of node behavior without modifying the source code.

Dependencies

— **Definition :** Packages can depend on other packages. Dependency management is crucial for ensuring that all necessary components are present.
— **Management :** Uses `package.xml` and `CMakeLists.txt` to declare and manage dependencies.

Build System

— **Definition :** The build system (such as `colcon`) is used to compile and assemble ROS 2 packages.
— **Usage :** Manages dependencies and compilation steps to create executables and libraries.

This modular and flexible architecture allows for the construction of complex robotic systems by combining various packages and facilitating code reuse.

3.2 Files for simulation

Firstly, it is important to list all the necessary files for modeling the robot in Gazebo. This report includes a specific section on how to correctly install each package. But first, let's take a look at the ROS 2 architecture.

To launch the simulation, four essential packages are required :

— **tiago_simulation** : Provided by PAL Robotics, this package defines both the robot and the entire environment.

— **cpcc2_tiago** : Developed by Julien Gleyze, responsible for controlling the robot's arm. This package will be located within the TIAGO_simulation workspace.

— **ros2_control** : This package facilitates hardware integration, essential for managing the robot's actuators and sensors.

— **crocoddyl** : A solver used to compute arm movement trajectories efficiently.

In ROS 2, package management adheres to a specific structure. First, create a directory to store all your workspaces. For instance, let's name it *TIAGO_perso*. Within this directory, it is important to organize all the packages in a particular way.

Let's take a look at the expected directory structure.

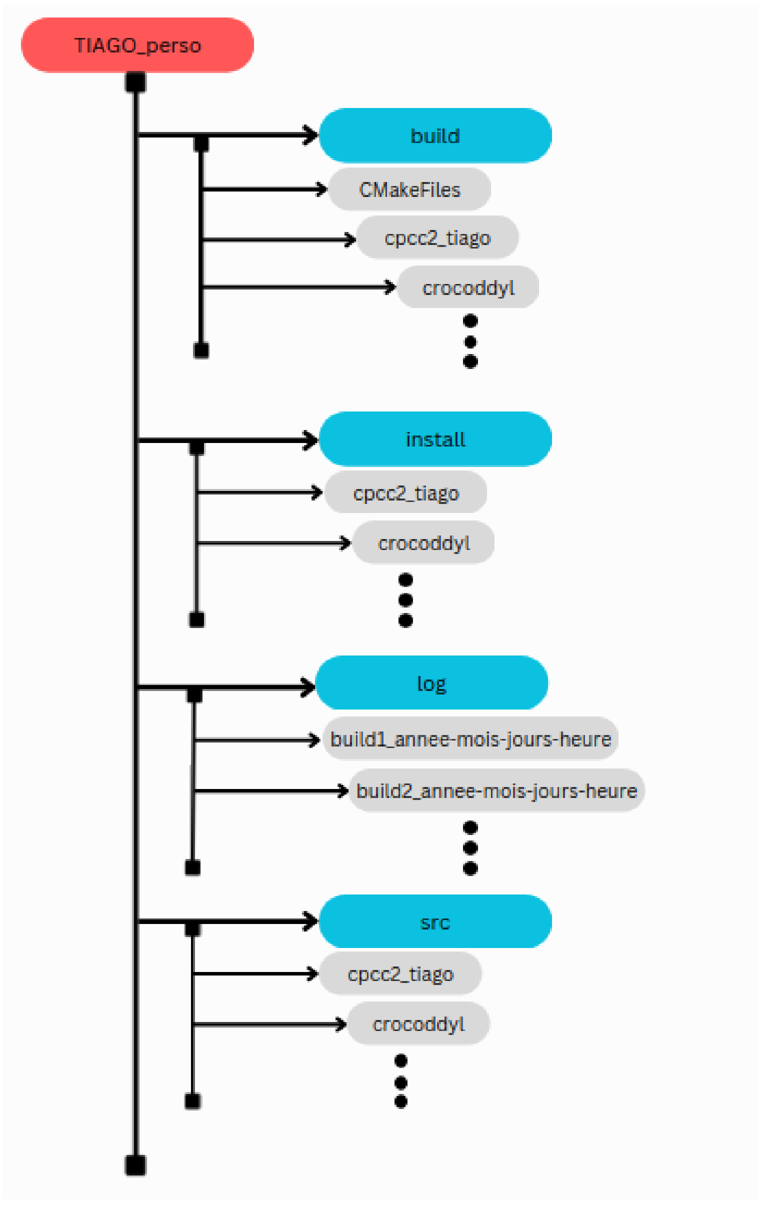


FIGURE 3 – Architecture of Ros2 workspace

ROS 2 encourages the use of colcon (CMake-based) as the default build tool for managing multiple workspaces. Each workspace serves to isolate and manage dependencies and packages independently. This modular approach helps in organizing and maintaining different components of a project, facilitating easier development, integration, and deployment processes.

By structuring your workspaces in this manner, you can effectively manage and develop different aspects of your robotics project within ROS 2. Each workspace can have its own set of dependencies, configurations, and build instructions, promoting modularity and maintainability.

3.3 Package compilation

This section is dedicated to the compilation process using **Colcon**.

Colcon is a command-line tool used for building, testing, and packaging multiple software packages. It is widely used in the ROS 2 (Robot Operating System) ecosystem to efficiently manage and compile large sets of interdependent packages.

— **Parallel Builds** : Colcon can build multiple packages in parallel, significantly speeding up the build process and reducing the overall compilation time.

— **Dependency Management** : It automatically handles dependencies between packages, ensuring they are built in the correct order. This prevents issues that can arise from missing or incorrectly ordered dependencies.

— **Workspaces** : Colcon supports the concept of workspaces, allowing developers to isolate different sets of packages and manage their environments separately. This modular approach helps maintain clean and organized development environments.

— **Extensibility** : Designed to be extensible, Colcon allows developers to add custom behaviors and extend its functionality through plugins. This flexibility makes it adaptable to a wide range of project requirements.

— **Unified Build System** : Colcon supports multiple build systems (e.g., CMake, Python setuptools), making it versatile for various types of projects. This unification simplifies the build process, even for projects with diverse requirements.

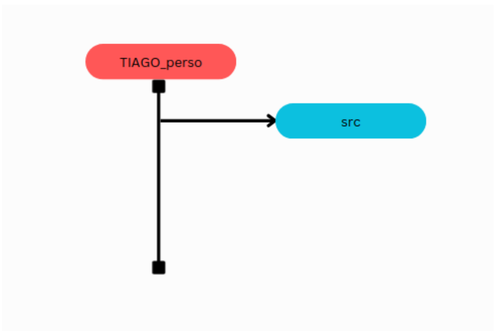
How to install Colcon

In order to install Colcon on a linux machine, enter this command in a terminal

```
sudo apt install python3-colcon-common-extensions
```

Understanding the compilation process is crucial as it translates the written code into a format that the computer can execute. However, for the code to be properly utilized, the environment must be sourced.

To compile your packages using Colcon, you need to be at the root of your workspace. This means you should be able to see the build, install, log, and src directories when you execute the ls command in the terminal. Here's an example of what the workspace structure looks like :



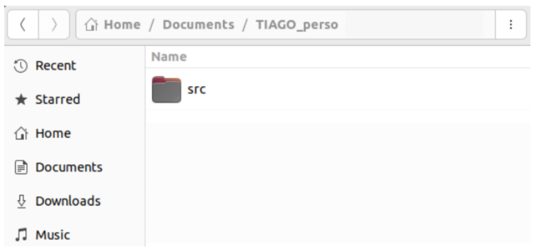


FIGURE 4 – Root of the workspace

Before the first compilation, it's normal if the build, install, and log files are not present. The compilation process will generate them. To compile, use the commande :

```
colcon build
```

To build only a specific package :

```
colcon build --packages-select <name_of_the_package>
```

Or to build in **Debug Mode** and only a specific package :

```
MAKEFLAGS="-j 1" colcon build --package-skip pinocchio
--cmake-args -DCMAKE_BUILD_TYPE=Debug
```

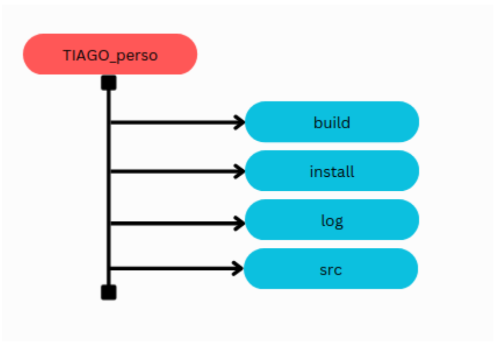
This type of compilation allows developers to use debugging tools such as GDB, ...

Once the code built (compiled with Colcon), it is necessary to source the environment.

```
source install/setup.bash
```

Sourcing the ROS 2 environment is crucial as it sets up essential environment variables, enabling access to ROS 2 packages and tools. It ensures proper dependency resolution, maintains compatibility with the ROS 2 distribution, simplifies launch script execution, and supports easy project or workspace switching. This process establishes a stable and consistent development environment, vital for reliable operation of robotic applications.

After compilation, in the root of the workspace, it is now possible to see 3 new files :



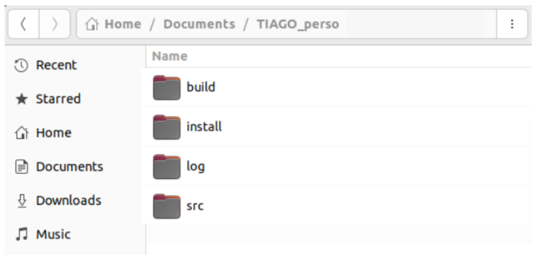


FIGURE 5 – Root of the workspace

- **The build directory** : This is where intermediate files are stored. For each package, a subfolder is created where tools like CMake are invoked.
- **The install directory** : Each package is installed here. By default, each package gets its own subdirectory.
- **The log directory** : Contains various logging information for each colcon invocation.

3.4 During the compilation

Depending on your RAM and CPU effiency, you may have trouble to compile your code. On linux, there is command or API to keep an eye on the RAM and CPU usage.

There are **htop** on the terminal but also "**System Monitor**" do that.
Keep an eye to this indicators in order to avoid issue like freeze of your computer.

It is possible also to use the following command to only allow the use of 1 thread during the compilation.

```
MAKEFLAGS="-j 1" colcon build --cmake-args -DCMAKE_BUILD_TYPE=Debug
```

4 GAZEBO

Gazebo is a widely used open-source robot simulation software that provides a realistic 3D environment for simulating robots and their interactions with the surrounding environment. It is a key tool in robotics research, development, and education

5 ROS2 CONTROL

Now let's have a look to Ros2 control.

ros2_control is a framework within the Robot Operating System 2 (ROS 2) ecosystem designed to simplify the development and integration of robot control systems. It provides a standardized and modular way to create controllers and hardware interfaces for various types of robots, enabling more efficient and flexible control architectures. Here are the key components and functionalities of *ros2_control* :

— Hardware Abstraction Layer (HAL) : *ros2_control* abstracts the hardware specifics, allowing developers to write controllers without needing to know the details of the underlying hardware. This layer includes hardware interfaces that interact with the robot's physical components such as sensors, actuators, and joints.

— Controller Manager : The controller manager handles the lifecycle of controllers, including loading, unloading, starting, and stopping them. It ensures that only one controller is active per resource at a time to avoid conflicts.

— Controllers : Controllers in *ros2_control* are reusable components that implement specific control strategies (e.g., position control, velocity control, effort control). They receive commands from higher-level nodes and send appropriate signals to the hardware interfaces.

— Real-Time Capabilities : *ros2_control* is designed with real-time performance in mind, which is crucial for many robotic applications that require precise timing and low-latency control loops.

— Modularity and Reusability : By separating hardware interfaces and controllers, *ros2_control* promotes modularity and reusability. This allows developers to mix and match different controllers and hardware interfaces as needed.

— Integration with ROS 2 : *ros2_control* integrates seamlessly with the ROS 2 ecosystem, utilizing ROS 2 topics, services, and parameters for communication and configuration. This allows for easy integration with other ROS 2 packages and tools.

— Simulation Support : *ros2_control* supports simulation environments, allowing developers to test and validate their control systems in a simulated setting before deploying them on physical robots.

To sum up, *ros2_control* provides a structured and efficient framework for developing and managing robot control systems, facilitating easier integration, testing, and deployment within the ROS 2 environment.

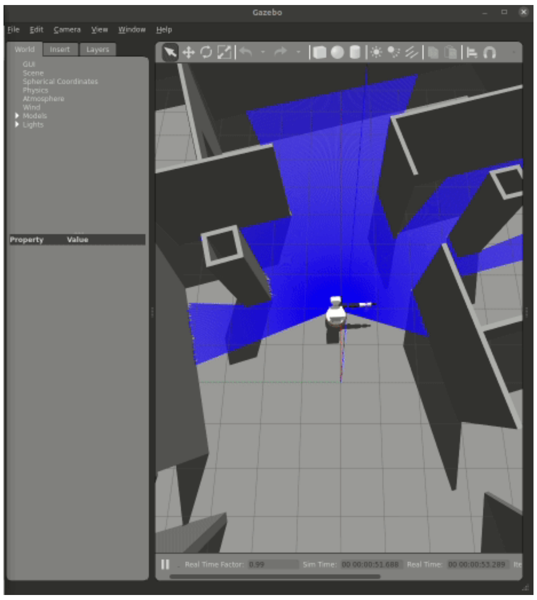
Thanks Ros2control, it is easy to manipulate controllers in a gazebo simulation

6 Launch

Tis section is dedicated to explain th more clearely as possible the right procedure to launch the simulation. After having created the workspace, compile and source, it is now possible to launch the spawning of TIAGO it the environnement. To do that, execute the following command :

```
ros2 launch tiago_gazebo tiago_gazebo.launch.py is_public_sim:=True
```

Now, on Gazebo, it is possible to see :



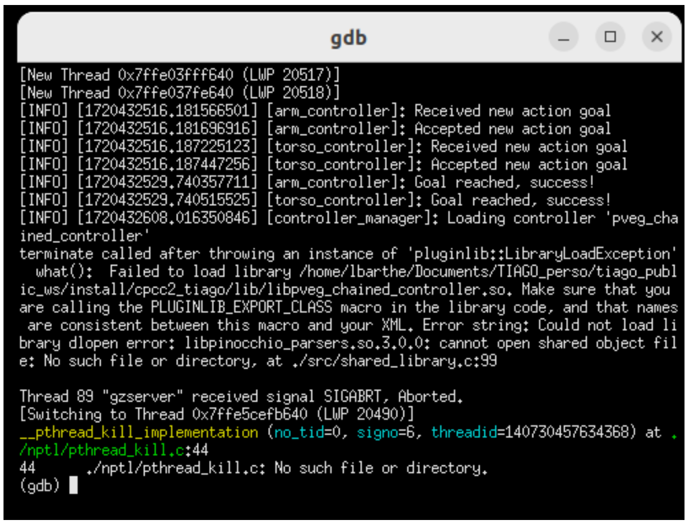
At this stage, TIAGO is controlled by controllers, that's why it is possible to observe its arm mouvement. It is possible to disable these cotrollers to upload anyone you want.

First, let's see the list of controllers like this :

This command shows which controller is used for each part of the robot. After that, it is possible to change of controllers by using ROS2 control.

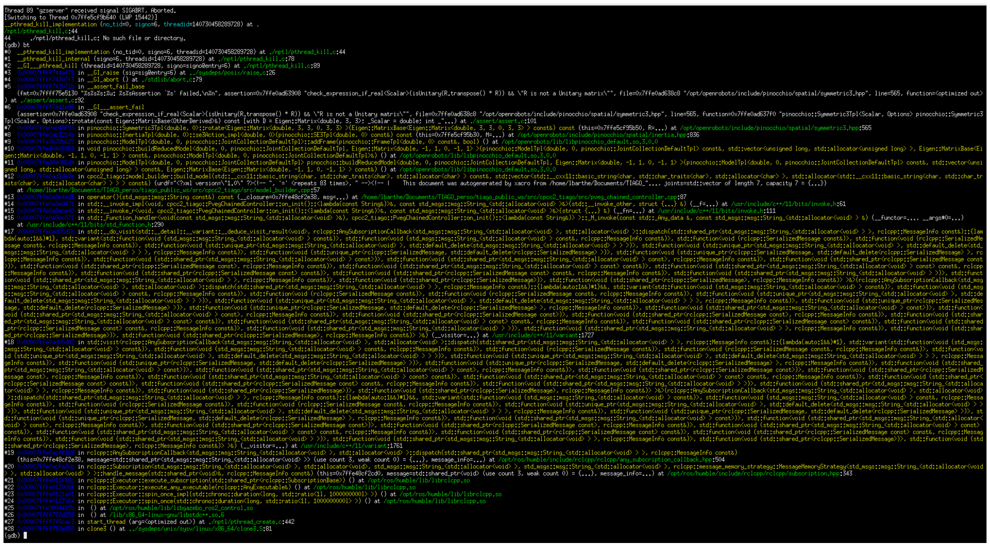
7 Issue during the simulation

This section is dedicated to the description of an issue occured during the execution of the simulation. First, the simulation has broken down following the error message :



```
[New Thread 0x7ffe03fff640 (LWP 20517)]
[New Thread 0x7ffe037fe640 (LWP 20518)]
[INFO] [1720432516.181566501] [arm_controller]: Received new action goal
[INFO] [1720432516.181696916] [arm_controller]: Accepted new action goal
[INFO] [1720432516.187225123] [torso_controller]: Received new action goal
[INFO] [1720432516.187447256] [torso_controller]: Accepted new action goal
[INFO] [1720432529.740357711] [arm_controller]: Goal reached, success!
[INFO] [1720432529.740515525] [torso_controller]: Goal reached, success!
[INFO] [1720432608.016350846] [controller_manager]: Loading controller 'pveg_cha
ined_controller'
terminate called after throwing an instance of 'pluginlib::LibraryLoadException'
  what():  Failed to load library /home/lbarthe/Documents/TIAGO_perso/tiago_publ
ic_ws/install/cpcc2_tiago/lib/libpveg_chained_controller.so. Make sure that you
are calling the PLUGINLIB_EXPORT_CLASS macro in the library code, and that names
 are consistent between this macro and your XML. Error string: Could not load li
brary dlopen error: libpinocchio_parsers.so.3.0.0: cannot open shared object fil
e: No such file or directory, at ./src/shared_library.c:99

Thread 89 "gzserver" received signal SIGABRT, Aborted.
[Switching to Thread 0x7ffe5cefb640 (LWP 20490)]
__pthread_kill_implementation (no_tid=0, signo=6, threadid=140730457634368) at .
/nptl/pthread_kill.c:44
44      ./nptl/pthread_kill.c: No such file or directory.
(gdb) █
```

At first glance, something wrong with the model of the robot. It is possible to see the error backtrace from GDB each give a better understanding of files called during the simulation.



```
Thread 89 "gzserver" received signal SIGABRT, Aborted.
[Switching to Thread 0x7ffe5cf9b640 (LWP 15442)]
__pthread_kill_implementation (no_tid=0, signo=6, threadid=140730458289728) at .
/nptl/pthread_kill.c:44
44      ./nptl/pthread_kill.c: No such file or directory.
(gdb) bt
#0  __pthread_kill_implementation (no_tid=0, signo=6, threadid=140730458289728) at ./nptl/pthread_kill.c:44
#1  __pthread_kill_internal (signo=6, threadid=140730458289728) at ./nptl/pthread_kill.c:78
#2  __GI___pthread_kill (threadid=140730458289728, signo=signo@entry=6) at ./nptl/pthread_kill.c:89
#3  0x00007ffff744a476 in __GI_raise (sig=sig@entry=6) at ../sysdeps/posix/raise.c:26
#4  0x00007ffff74307f3 in __GI_abort () at ./stdlib/abort.c:79
#5  0x00007ffff743071b in __assert_fail_base
    (fmt=0x7ffff75e5130 "%s%s%s:%u: %s%sAssertion `%s' failed.\n%n", assertion=0x7ffe0ad63908 "check_expression_if_real<Scalar>(isUnitary(R.transpose() * R)) && \"R is not a Unitary matrix\"", file=0x7ffe0ad638c8 "/opt/openrobots/include/pinocchio/spatial/symmetric3.hpp", line=565, function=<optimized out>
) at ./assert/assert.c:92
#6  0x00007ffff7441e96 in __GI___assert_fail
    (assertion=0x7ffe0ad63908 "check_expression_if_real<Scalar>(isUnitary(R.transpose() * R)) && \"R is not a Unitary matrix\"", file=0x7ffe0ad638c8 "/opt/openrobots/include/pinocchio/spatial/symmetric3.hpp", line=565, function=0x7ffe0ad637f0 "pinocchio::Symmetric3Tpl<Scalar, Options> pinocchio::Symmetric3
Tpl<Scalar, Options>::rotate(const Eigen::MatrixBase<OtherDerived>&) const [with D = Eigen::Matrix<double, 3, 3>; _Scalar = double; int _"...) at ./assert/assert.c:101
#7  0x00007ffe0ab48f93 in pinocchio::Symmetric3Tpl<double, 0>::rotate<Eigen::Matrix<double, 3, 3, 0, 3, 3> >(Eigen::MatrixBase<Eigen::Matrix<double, 3, 3, 0, 3, 3> > const&) const (this=0x7ffe5cf95b50, R=...) at /opt/openrobots/include/pinocchio/spatial/symmetric3.hpp:565
#8  0x00007ffe0ab3204b in pinocchio::InertiaTpl<double, 0>::se3Action_impl<double, 0>(pinocchio::SE3Tpl<double, 0> const&) const (this=0x7ffe5cf95b30, M=...) at /opt/openrobots/include/pinocchio/spatial/inertia.hpp:836
#9  0x00007ffe3078b625 in pinocchio::ModelTpl<double, 0, pinocchio::JointCollectionDefaultTpl>::addFrame(pinocchio::FrameTpl<double, 0> const&, bool) () at /opt/openrobots/lib/libpinocchio_default.so.3.0.0
#10 0x00007ffe30406959 in void pinocchio::buildReducedModel<double, 0, pinocchio::JointCollectionDefaultTpl, Eigen::Matrix<double, -1, 1, 0, -1, 1> >(pinocchio::ModelTpl<double, 0, pinocchio::JointCollectionDefaultTpl> const&, std::vector<unsigned long, std::allocator<unsigned long> >, Eigen::MatrixBase<Ei
gen::Matrix<double, -1, 1, 0, -1, 1> > const&, pinocchio::ModelTpl<double, 0, pinocchio::JointCollectionDefaultTpl>&) () at /opt/openrobots/lib/libpinocchio_default.so.3.0.0
#11 0x00007ffe304082d4 in pinocchio::ModelTpl<double, 0, pinocchio::JointCollectionDefaultTpl> pinocchio::buildReducedModel<double, 0, pinocchio::JointCollectionDefaultTpl, Eigen::Matrix<double, -1, 1, 0, -1, 1> >(pinocchio::ModelTpl<double, 0, pinocchio::JointCollectionDefaultTpl> const&, std::vector<unsi
gned long, std::allocator<unsigned long> > const&, Eigen::MatrixBase<Eigen::Matrix<double, -1, 1, 0, -1, 1> > const&) () at /opt/openrobots/lib/libpinocchio_default.so.3.0.0
#12 0x00007ffe0ad1de3e in cpcc2_tiago::model_builder::build_model(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, std::vector<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >, std::allocator<std::__cxx11::basic_string<char, std::char_tr
aits<char>, std::allocator<char> > > > const&) (urdf="<?xml version=\"1.0\" ?><!-- ", '=' <repeats 83 times>, " --><!-- |    This document was autogenerated by xacro from /home/lbarthe/Documents/TIAGO_"..., joints=std::vector of length 7, capacity 7 = {...})
    at /home/lbarthe/Documents/TIAGO_perso/tiago_public_ws/src/cpcc2_tiago/src/model_builder.cpp:57
#13 0x00007ffe0a9e4ed5 in operator()(std_msgs::msg::String const&) const (__closure=0x7ffe48cf2e38, msg=...) at /home/lbarthe/Documents/TIAGO_perso/tiago_public_ws/src/cpcc2_tiago/src/pveg_chained_controller.cpp:87
#14 0x00007ffe0a9ed471 in std::__invoke_impl<void, cpcc2_tiago::PvegChainedController::on_init()::<lambda(const String&)>&, const std_msgs::msg::String_<std::allocator<void> >&>(std::__invoke_other, struct {...} &) (__f=...) at /usr/include/c++/11/bits/invoke.h:61
#15 0x00007ffe0a9ece40 in std::__invoke_r<void, cpcc2_tiago::PvegChainedController::on_init()::<lambda(const String&)>&, const std_msgs::msg::String_<std::allocator<void> >&>(struct {...} &) (__fn=...) at /usr/include/c++/11/bits/invoke.h:111
#16 0x00007ffe0a9ec571 in std::_Function_handler<void(const std_msgs::msg::String_<std::allocator<void> >&), cpcc2_tiago::PvegChainedController::on_init()::<lambda(const String&)> >::_M_invoke(const std::_Any_data &, const std_msgs::msg::String_<std::allocator<void> > &) (__functor=..., __args#0=...)
    at /usr/include/c++/11/bits/std_function.h:290
#17 0x00007ffe0ad05d1b in std::__do_visit<std::__detail::__variant::__deduce_visit_result<void>, rclcpp::AnySubscriptionCallback<std_msgs::msg::String_<std::allocator<void> >, std::allocator<void> >::dispatch(std::shared_ptr<std_msgs::msg::String_<std::allocator<void> > >, rclcpp::MessageInfo const&)::{lam
bda(auto:1&&)#1}, std::variant<std::function<void (std_msgs::msg::String_<std::allocator<void> > const&)>, std::function<void (std_msgs::msg::String_<std::allocator<void> > const&, rclcpp::MessageInfo const&)>, std::function<void (rclcpp::SerializedMessage const&)>, std::function<void (rclcpp::SerializedMe
ssage const&, rclcpp::MessageInfo const&)>, std::function<void (std::unique_ptr<std_msgs::msg::String_<std::allocator<void> >, std::default_delete<std_msgs::msg::String_<std::allocator<void> > > >)>, std::function<void (std::unique_ptr<std_msgs::msg::String_<std::allocator<void> >, std::default_delete<std_
msgs::msg::String_<std::allocator<void> > > >, rclcpp::MessageInfo const&)>, std::function<void (std::unique_ptr<rclcpp::SerializedMessage, std::default_delete<rclcpp::SerializedMessage> >)>, std::function<void (std::unique_ptr<rclcpp::SerializedMessage, std::default_delete<rclcpp::SerializedMessage> >, rc
lcpp::MessageInfo const&)>, std::function<void (std::shared_ptr<std_msgs::msg::String_<std::allocator<void> > const>)>, std::function<void (std::shared_ptr<std_msgs::msg::String_<std::allocator<void> > const>, rclcpp::MessageInfo const&)>, std::function<void (std::shared_ptr<rclcpp::SerializedMessage const
>)>, std::function<void (std::shared_ptr<rclcpp::SerializedMessage const>, rclcpp::MessageInfo const&)>, std::function<void (std::shared_ptr<std_msgs::msg::String_<std::allocator<void> > const> const&)>, std::function<void (std::shared_ptr<std_msgs::msg::String_<std::allocator<void> > const> const&, rclcpp
::MessageInfo const&)>, std::function<void (std::shared_ptr<rclcpp::SerializedMessage const> const&)>, std::function<void (std::shared_ptr<rclcpp::SerializedMessage const> const&, rclcpp::MessageInfo const&)>, std::function<void (std::shared_ptr<std_msgs::msg::String_<std::allocator<void> > >)>, std::funct
ion<void (std::shared_ptr<std_msgs::msg::String_<std::allocator<void> > >, rclcpp::MessageInfo const&)>, std::function<void (std::shared_ptr<rclcpp::SerializedMessage>)>, std::function<void (std::shared_ptr<rclcpp::SerializedMessage>, rclcpp::MessageInfo const&)> >&>(rclcpp::AnySubscriptionCallback<std_msg
s::msg::String_<std::allocator<void> >, std::allocator<void> >::dispatch(std::shared_ptr<std_msgs::msg::String_<std::allocator<void> > >, rclcpp::MessageInfo const&)::{lambda(auto:1&&)#1}&&, std::variant<std::function<void (std_msgs::msg::String_<std::allocator<void> > const&)>, std::function<void (std_msg
s::msg::String_<std::allocator<void> > const&, rclcpp::MessageInfo const&)>, std::function<void (rclcpp::SerializedMessage const&)>, std::function<void (rclcpp::SerializedMessage const&, rclcpp::MessageInfo const&)>, std::function<void (std::unique_ptr<std_msgs::msg::String_<std::allocator<void> >, std::de
fault_delete<std_msgs::msg::String_<std::allocator<void> > > >)>, std::function<void (std::unique_ptr<std_msgs::msg::String_<std::allocator<void> >, std::default_delete<std_msgs::msg::String_<std::allocator<void> > > >, rclcpp::MessageInfo const&)>, std::function<void (std::unique_ptr<rclcpp::SerializedMes
sage, std::default_delete<rclcpp::SerializedMessage> >)>, std::function<void (std::unique_ptr<rclcpp::SerializedMessage, std::default_delete<rclcpp::SerializedMessage> >, rclcpp::MessageInfo const&)>, std::function<void (std::shared_ptr<std_msgs::msg::String_<std::allocator<void> > const>)>, std::function<
void (std::shared_ptr<std_msgs::msg::String_<std::allocator<void> > const>, rclcpp::MessageInfo const&)>, std::function<void (std::shared_ptr<rclcpp::SerializedMessage const>)>, std::function<void (std::shared_ptr<rclcpp::SerializedMessage const>, rclcpp::MessageInfo const&)>, std::function<void (std::shar
ed_ptr<std_msgs::msg::String_<std::allocator<void> > const> const&)>, std::function<void (std::shared_ptr<std_msgs::msg::String_<std::allocator<void> > const> const&, rclcpp::MessageInfo const&)>, std::function<void (std::shared_ptr<rclcpp::SerializedMessage const> const&)>, std::function<void (std::shared
_ptr<rclcpp::SerializedMessage const> const&, rclcpp::MessageInfo const&)>, std::function<void (std::shared_ptr<std_msgs::msg::String_<std::allocator<void> > >)>, std::function<void (std::shared_ptr<std_msgs::msg::String_<std::allocator<void> > >, rclcpp::MessageInfo const&)>, std::function<void (std::shar
ed_ptr<rclcpp::SerializedMessage>)>, std::function<void (std::shared_ptr<rclcpp::SerializedMessage>, rclcpp::MessageInfo const&)> >&) (__visitor=...) at /usr/include/c++/11/variant:1727
#18 0x00007ffe0ad05d82 in std::visit<rclcpp::AnySubscriptionCallback<std_msgs::msg::String_<std::allocator<void> >, std::allocator<void> >::dispatch(std::shared_ptr<std_msgs::msg::String_<std::allocator<void> > >, rclcpp::MessageInfo const&)::{lambda(auto:1&&)#1}, std::variant<std::function<void (std_msgs:
:msg::String_<std::allocator<void> > const&)>, std::function<void (std_msgs::msg::String_<std::allocator<void> > const&, rclcpp::MessageInfo const&)>, std::function<void (rclcpp::SerializedMessage const&)>, std::function<void (rclcpp::SerializedMessage const&, rclcpp::MessageInfo const&)>, std::function<vo
id (std::unique_ptr<std_msgs::msg::String_<std::allocator<void> >, std::default_delete<std_msgs::msg::String_<std::allocator<void> > > >)>, std::function<void (std::unique_ptr<std_msgs::msg::String_<std::allocator<void> >, std::default_delete<std_msgs::msg::String_<std::allocator<void> > > >, rclcpp::Messa
geInfo const&)>, std::function<void (std::unique_ptr<rclcpp::SerializedMessage, std::default_delete<rclcpp::SerializedMessage> >)>, std::function<void (std::unique_ptr<rclcpp::SerializedMessage, std::default_delete<rclcpp::SerializedMessage> >, rclcpp::MessageInfo const&)>, std::function<void (std::shared_
ptr<std_msgs::msg::String_<std::allocator<void> > const>)>, std::function<void (std::shared_ptr<std_msgs::msg::String_<std::allocator<void> > const>, rclcpp::MessageInfo const&)>, std::function<void (std::shared_ptr<rclcpp::SerializedMessage const>)>, std::function<void (std::shared_ptr<rclcpp::SerializedM
essage const>, rclcpp::MessageInfo const&)>, std::function<void (std::shared_ptr<std_msgs::msg::String_<std::allocator<void> > const> const&)>, std::function<void (std::shared_ptr<std_msgs::msg::String_<std::allocator<void> > const> const&, rclcpp::MessageInfo const&)>, std::function<void (std::shared_ptr<
rclcpp::SerializedMessage const> const&)>, std::function<void (std::shared_ptr<rclcpp::SerializedMessage const> const&, rclcpp::MessageInfo const&)>, std::function<void (std::shared_ptr<std_msgs::msg::String_<std::allocator<void> > >)>, std::function<void (std::shared_ptr<std_msgs::msg::String_<std::alloca
tor<void> > >, rclcpp::MessageInfo const&)>, std::function<void (std::shared_ptr<rclcpp::SerializedMessage>)>, std::function<void (std::shared_ptr<rclcpp::SerializedMessage>, rclcpp::MessageInfo const&)> >&>(rclcpp::AnySubscriptionCallback<std_msgs::msg::String_<std::allocator<void> >, std::allocator<void>
 >::dispatch(std::shared_ptr<std_msgs::msg::String_<std::allocator<void> > >, rclcpp::MessageInfo const&)::{lambda(auto:1&&)#1}&&, std::variant<std::function<void (std_msgs::msg::String_<std::allocator<void> > const&)>, std::function<void (std_msgs::msg::String_<std::allocator<void> > const&, rclcpp::Messa
geInfo const&)>, std::function<void (rclcpp::SerializedMessage const&)>, std::function<void (rclcpp::SerializedMessage const&, rclcpp::MessageInfo const&)>, std::function<void (std::unique_ptr<std_msgs::msg::String_<std::allocator<void> >, std::default_delete<std_msgs::msg::String_<std::allocator<void> > >
 >)>, std::function<void (std::unique_ptr<std_msgs::msg::String_<std::allocator<void> >, std::default_delete<std_msgs::msg::String_<std::allocator<void> > > >, rclcpp::MessageInfo const&)>, std::function<void (std::unique_ptr<rclcpp::SerializedMessage, std::default_delete<rclcpp::SerializedMessage> >)>, st
d::function<void (std::unique_ptr<rclcpp::SerializedMessage, std::default_delete<rclcpp::SerializedMessage> >, rclcpp::MessageInfo const&)>, std::function<void (std::shared_ptr<std_msgs::msg::String_<std::allocator<void> > const>)>, std::function<void (std::shared_ptr<std_msgs::msg::String_<std::allocator<
void> > const>, rclcpp::MessageInfo const&)>, std::function<void (std::shared_ptr<rclcpp::SerializedMessage const>)>, std::function<void (std::shared_ptr<rclcpp::SerializedMessage const>, rclcpp::MessageInfo const&)>, std::function<void (std::shared_ptr<std_msgs::msg::String_<std::allocator<void> > const>
const&)>, std::function<void (std::shared_ptr<std_msgs::msg::String_<std::allocator<void> > const> const&, rclcpp::MessageInfo const&)>, std::function<void (std::shared_ptr<rclcpp::SerializedMessage const> const&)>, std::function<void (std::shared_ptr<rclcpp::SerializedMessage const> const&, rclcpp::Messag
eInfo const&)>, std::function<void (std::shared_ptr<std_msgs::msg::String_<std::allocator<void> > >)>, std::function<void (std::shared_ptr<std_msgs::msg::String_<std::allocator<void> > >, rclcpp::MessageInfo const&)>, std::function<void (std::shared_ptr<rclcpp::SerializedMessage>)>, std::function<void (std
::shared_ptr<rclcpp::SerializedMessage>, rclcpp::MessageInfo const&)> >&) (__visitor=...) at /usr/include/c++/11/variant:1761
#19 0x00007ffe0acff1b8 in rclcpp::AnySubscriptionCallback<std_msgs::msg::String_<std::allocator<void> >, std::allocator<void> >::dispatch(std::shared_ptr<std_msgs::msg::String_<std::allocator<void> > >, rclcpp::MessageInfo const&)
    (this=0x7ffe48cf2e38, message=std::shared_ptr<std_msgs::msg::String_<std::allocator<void> >> (use count 3, weak count 0) = {...}, message_info=...) at /opt/ros/humble/include/rclcpp/rclcpp/any_subscription_callback.hpp:504
#20 0x00007ffe0acfa7cb in rclcpp::Subscription<std_msgs::msg::String_<std::allocator<void> >, std::allocator<void>, std_msgs::msg::String_<std::allocator<void> >, std_msgs::msg::String_<std::allocator<void> >, rclcpp::message_memory_strategy::MessageMemoryStrategy<std_msgs::msg::String_<std::allocator<void
> >, std::allocator<void> > >::handle_message(std::shared_ptr<void>&, rclcpp::MessageInfo const&) (this=0x7ffe48cf2cd0, message=std::shared_ptr<void> (use count 3, weak count 0) = {...}, message_info=...) at /opt/ros/humble/include/rclcpp/rclcpp/subscription.hpp:343
#21 0x00007fffe011f88c in rclcpp::Executor::execute_subscription(std::shared_ptr<rclcpp::SubscriptionBase>) () at /opt/ros/humble/lib/librclcpp.so
#22 0x00007fffe012008f in rclcpp::Executor::execute_any_executable(rclcpp::AnyExecutable&) () at /opt/ros/humble/lib/librclcpp.so
#23 0x00007fffe0121a01 in rclcpp::Executor::spin_once_impl(std::chrono::duration<long, std::ratio<1l, 1000000000l> >) () at /opt/ros/humble/lib/librclcpp.so
#24 0x00007fffe0117c64 in rclcpp::Executor::spin_once(std::chrono::duration<long, std::ratio<1l, 1000000000l> >) () at /opt/ros/humble/lib/librclcpp.so
#25 0x00007ffe9804412b in  () at /opt/ros/humble/lib/libgazebo_ros2_control.so
#26 0x00007ffff772d253 in  () at /lib/x86_64-linux-gnu/libstdc++.so.6
#27 0x00007ffff749cac3 in start_thread (arg=<optimized out>) at ./nptl/pthread_create.c:442
#28 0x00007ffff752e850 in clone3 () at ../sysdeps/unix/sysv/linux/x86_64/clone3.S:81
(gdb) █
```

it is possible to notice the succesive file call from the backtrace. The issue seems coming from model.hxx script, located in src/pinocchio/include/pinocchio/algorithm/model.hxx

In this file, the issue seems to stem from the buildReducedModel. The robot has a URDF, which is a file that describes the robot's geometry and the connections between all its components. However, since we are only focusing on the TIAGO arm, we want to lock all joints that are not being used. This requires modifying the original URDF to determine the equivalent block for all the locked parts (i.e., the equivalent inertia matrix). And this is where the problem arises.

7.1 Unit Test

It was possible to perform unit test in order to sure about the good behaviour of pinocchio during the simulation. To perform unit test, go at the path : cd build/-pinocchio/unittest and then put in the terminal : gdb ./$<name\_of\_the\_test>$, for instance, to launch the test about $joint\_revolute$, it is possible to launch gdb ./test-cpp-joint-revolute.

But running the test didn't enhance any problem linked to buildReducedModel. $/Documents/TIAGO\_perso/build/pinocchio/unittest$

7.2 ROS 2 Control Commands

–**List Available Controllers :**
Command : ros2 control list_controllers
Description : Lists all available controllers loaded in the system.

–**Load a Controller :**
Command : ros2 control load_controller <controller_name>
Description : Loads a specific controller onto the robot hardware.

–**Unload a Controller :**
Command : ros2 control unload_controller <controller_name>
Description : Unloads a currently running controller from the robot hardware.

–**Start a Controller :**
Command : ros2 control start_controller <controller_name>
Description : Starts a previously loaded controller.

–**Stop a Controller :**
Command : ros2 control stop_controller <controller_name>
Description : Stops a running controller, but keeps it loaded.

–**Configure a Controller :**
Command : ros2 control param set <controller_name> <parameter> <value>
Description : Sets parameters for configuring a controller.

–**Get Controller Status :**
Command : ros2 control list_controllers
Description : Displays status and state of all controllers.

–**Interface with Hardware :**
Command : ros2 control list_hardware
Description : Lists available hardware interfaces for communication with robot actuators and sensors.

–**Configure Hardware :**
Command : ros2 control param set <hardware_name> <parameter> <value>
Description : Sets parameters to configure a specific hardware interface.

Commandes Unix Utiles

12 septembre 2024

Gestion des fichiers et des répertoires

— **ls** : Liste les fichiers et répertoires du répertoire courant.
— **cd [répertoire]** : Change le répertoire courant.
— **pwd** : Affiche le chemin du répertoire courant.
— **cp [source] [destination]** : Copie des fichiers ou des répertoires.
— **mv [source] [destination]** : Déplace ou renomme des fichiers ou des répertoires.
— **rm [fichier]** : Supprime des fichiers.
— **rmdir [répertoire]** : Supprime des répertoires vides.
— **mkdir [répertoire]** : Crée un nouveau répertoire.
— **touch [fichier]** : Crée un fichier vide ou met à jour la date d'accès d'un fichier.

Affichage et manipulation de fichiers

— **cat [fichier]** : Affiche le contenu d'un fichier.
— **more [fichier]** : Affiche le contenu d'un fichier page par page.
— **less [fichier]** : Affiche le contenu d'un fichier page par page avec des fonctionnalités de navigation supplémentaires.
— **head [fichier]** : Affiche les premières lignes d'un fichier.
— **tail [fichier]** : Affiche les dernières lignes d'un fichier.

Gestion des processus

- **ps** : Affiche les processus en cours d'exécution.
- **top** : Affiche en temps réel les processus en cours d'exécution.
- **htop** : Affiche en temps réel les processus en cours d'exécution avec une interface plus conviviale que 'top'.
- **kill [PID]** : Termine un processus en utilisant son PID (Process ID).
- **killall [nom du processus]** : Termine tous les processus ayant le nom spécifié.
- **bg** : Reprend l'exécution d'un processus suspendu en arrière-plan.
- **fg** : Reprend l'exécution d'un processus suspendu en avant-plan.
- **jobs** : Liste les tâches en arrière-plan dans la session actuelle.

Historique et surveillance du système

- **history** : Affiche l'historique des commandes.
- **uptime** : Affiche depuis combien de temps le système fonctionne.
- **who** : Affiche les utilisateurs actuellement connectés.
- **w** : Affiche qui est connecté et ce qu'ils font.
- **uname -a** : Affiche des informations sur le système.
- **dmesg** : Affiche les messages du noyau.

Réseau

- **ping [adresse]** : Vérifie la connectivité réseau avec une adresse IP ou un nom de domaine.
- **wget [URL]** : Télécharge des fichiers depuis le web.
- **curl [URL]** : Transfère des données depuis ou vers un serveur.
- **traceroute [adresse]** : Affiche la route prise par les paquets pour atteindre un hôte.
- **ifconfig** : Configure les interfaces réseau.
- **ip** : Affiche et manipule les informations d'interface réseau.
- **netstat** : Affiche des statistiques réseau.
- **ss** : Affiche des informations sur les sockets.
- **tcpdump** : Affiche les paquets réseau capturés.
- **iptables** : Configure le pare-feu du noyau Linux.
- **firewalld** : Gère les règles du pare-feu dynamiquement.

Système de fichiers

- **df** : Affiche l'espace disque disponible sur les systèmes de fichiers.
- **du** : Affiche l'utilisation de l'espace disque.
- **mount** : Monte un système de fichiers.
- **umount** : Démontre un système de fichiers.

Administration système et gestion de paquets

- **apt-get [commande]** : Gère les paquets sur les systèmes basés sur Debian.
- **yum [commande]** : Gère les paquets sur les systèmes basés sur Red Hat.
- **dnf [commande]** : Gère les paquets sur les systèmes basés sur Fedora.
- **service [service] [commande]** : Gère les services système.
- **systemctl [commande]** : Contrôle le système et les services.
- **journalctl** : Affiche les journaux du système.
- **crontab -e** : Édite les tâches planifiées de l'utilisateur.

Utilitaires divers

- **echo [texte]** : Affiche un texte à l'écran.
- **date** : Affiche ou définit la date et l'heure du système.
- **chmod [options] [fichier]** : Change les permissions d'un fichier ou d'un répertoire.
- **chown [propriétaire] [fichier]** : Change le propriétaire d'un fichier ou d'un répertoire.
- **find [chemin] [options]** : Recherche des fichiers et des répertoires.
- **grep [motif] [fichier]** : Recherche des motifs dans des fichiers.
- **tar [options] [archive]** : Archive ou extrait des fichiers avec tar.
- **zip [archive] [fichiers]** : Compresse des fichiers en archive zip.
- **unzip [archive]** : Extrait des fichiers d'une archive zip.
- **ssh [utilisateur]@[hôte]** : Se connecte à un hôte distant via SSH.
- **scp [source] [destination]** : Copie des fichiers entre hôtes via SSH.
- **sftp [utilisateur]@[hôte]** : Transfère des fichiers via SFTP.
- **alias [nom]='[commande]'** : Crée un alias pour une commande.
- **unalias [nom]** : Supprime un alias.
- **sudo [commande]** : Exécute une commande avec les privilèges superutilisateur.
- **logout** : Déconnecte l'utilisateur de la session actuelle.
- **free** : Affiche la quantité de mémoire libre et utilisée dans le système.
- **vmstat** : Affiche des statistiques sur la mémoire, les processus, l'ES et le CPU.
- **iostat** : Affiche des statistiques sur l'ES et le CPU.
- **watch [commande]** : Exécute une commande périodiquement et affiche le résultat.
- **lsof** : Affiche les fichiers ouverts par les processus.
- **strace** : Trace les appels système