Télémaque BUZZI

Homework 1 :
Bring up your robot

Robotics Lab 2025 - Prof Mario Selvaggio

---

1) Definition and preparation of the model :

a) Launching the model

We first start with adding some dependencies that were missing for the homework :

```
sudo apt install ros-humble-joint-state-publisher
sudo apt install ros-humble-joint-state-publisher-gui
```

Then we create our own launcher armando_display.launch in the launch file and we modify the CMakeLists.txt to implement the file in the compilation of the environment/

To launch the file use the following commands :

```
cd ~/ros2_ws
colcon build --packages-select armando_description
source install/setup.bash
ros2 launch armando_description armando_display.launch.py
```

We stumbled on a visual error because the urdf was not found by the program so it couldn't materialize, we just corrected it and the model appears.

b) Saving a config and loading it

After modifying the parameters of the robot like the ground plane and adding the model itself we then save the file in config/armando.rviz and modify the node to use this saved file :

```
rviz_config_file = os.path.join(pkg_path, 'config', 'armando.rviz')
arguments=['-d', rviz_config_file]
```

To launch the model with ready for use :

```
cd ~/ros2_ws
colcon build --packages-select armando_description
source install/setup.bash
```

ros2 launch armando_description armando_display.launch.py

c) Creation of the physical body

To simulate the physical body of the robot we modify it piece by piece, the cylinders that represent the links are not visible but are defined in the urdf file.

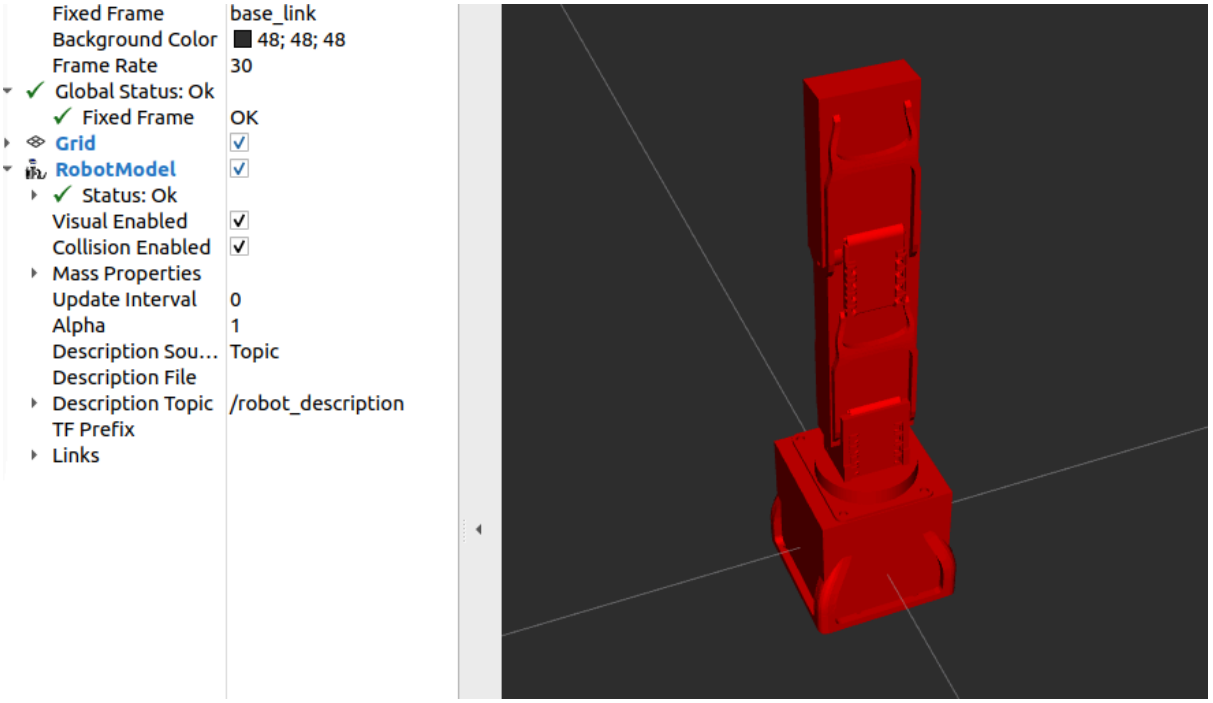

Figure 1 : The collision model of the robot

2) Adding sensors and controllers

a) Launching the garmando_gazebo

We created the new file armando_world.launch.py, now we just have to launch it using :

colcon build --packages-select armando_gazebo
source install/setup.bash
ros2 launch armando_gazebo armando_world.launch.py

We also had to install gz-sim :

sudo apt update
sudo apt install ros-humble-ros-gz ros-humble-ros-gz-sim

b) Creating the hardware

We create the urdf to add sensors called armando_hardware_interface to the robot and then we refer to it in the main file with xmlns:xacro="http://www.ros.org/wiki/xacro" in order to let arm.urdf.xacro use it

c) Launching and test of the controllers

We create a new folder config in armando_gazebo for the controllers and add the file armando_controllers.yaml. And then we add the inclusion of the .yaml file in the launch file.

The controllers were missing in the installation packages so we add them :

sudo apt install ros-humble-gazebo-ros2-control ros-humble-ros2-controllers

To test the controllers, we launch the gazebo in one terminal and we launch in another one :

ros2 control list_controllers

We were blocked for some time on the verification of the controllers because, when launching the gazebo, nothing appeared. We couldn't launch the new model because we had a zombie program from a past test that kept the server busy. We found it using gazebo –verbose to check the availability.
To ensure a safe launch, we do :

killall -9 gzserver gzclient gazebo

3) Adding a camera to the robot

a) Modifying the urdf file

We modify the arm.urdf.xacro file to create a camera at the base of the arm.

```
  <link name="camera_link">
    <visual>
      <geometry>
        <box size="0.05 0.05 0.05"/>
      </geometry>
      <material name="white"/>
    </visual>
  </link>

  <joint name="camera_joint" type="fixed">
    <parent link="base_link"/>
    <child link="camera_link"/>
    <origin xyz="0.0 0 0.0" rpy="0 0 -1,57"/>
  </joint>
```

b) Creation of the xacro file

We then create a new folder in the armando_gazebo folder for the armando_camera.xacro file.

arm.urdf.xacro armando_camera.xacro

```
<?xml version="1.0"?>
<xacro:macro name="camera_sensor" prefix="">
  <sensor name="${prefix}camera" type="camera">
    <camera>
      <horizontal_fov>1.047</horizontal_fov>
      <image>
        <width>640</width>
        <height>480</height>
      </image>
      <clip>
        <near>0.1</near>
        <far>100</far>
      </clip>
    </camera>
    <plugin filename="libgz-sim-sensors-system.so" name="gz::sim::systems::Camera">
      <sensor_name>${prefix}camera</sensor_name>
      <topic>/${prefix}camera/image_raw</topic>
    </plugin>
  </sensor>
</xacro:macro>
```

And then we include it to the original urdf file with :

```
<xacro:include filename="$(find armando_gazebo)/armando_camera.xacro"/>
<xacro:armando_camera link="camera_link"/>
```

c) Test and modification of the launch file

We first modify the launch file to add the bridge to detect the new added camera :

```
bridge = Node(
    package='ros_ign_bridge',
    executable='parameter_bridge',
    name='bridge',
    output='screen',
    arguments=[
        '/camera/image_raw@sensor_msgs/msg/Image@gz.sim.sensors.Camera'
    ]
)
```

We then modify package.xml and the CMake to include the sharing of the new folder and add the bridge :

```
install(
  DIRECTORY urdf/
  DESTINATION share/${PROJECT_NAME}
)
```

```
  <depend>gazebo_ros</depend>
  <depend>ros_ign_bridge</depend>
  <depend>xacro</depend>

  <export>
    <build_type>ament_cmake</build_type>
    <armando_gazebo>
      <urdf>urdf</urdf>
    </armando_gazebo>
```

To test the modifications :

colcon build --packages-select armando_gazebo
source install/setup.bash
ros2 launch armando_gazebo armando_world.launch.py

Issue : (I've tried to change xml and cmake, rewrite cmake to verify the dependencies, tried to use ros_ign_bridge and ros_gz_bridge)

```
[INFO] [launch]: All log files can be found below /home/telemaque/.ros/log/2026-
01-09-11-16-50-213110-telemaque-Aspire-AV14-51-17796
[INFO] [launch]: Default logging verbosity is set to INFO
[ERROR] [launch]: Caught exception in launch (see debug for traceback): Caught m
ultiple exceptions when trying to load file of format [py]:
 - ExpatError: unbound prefix: line 2, column 0
 - InvalidFrontendLaunchFileError: The launch file may have a syntax error, or i
ts format is unknown
```

After verification it came from an error in the armando_camera.xacro because I forgot the <robot>.

Now we can verify the camera output with a second tab with :

ros2 run rqt_image_view rqt_image_view

4) Sending joint positions to the robot

a) Creation of the armando_controller package

We go into the /ros2_ws/src

ros2 pkg create --build-type ament_cmake armando_controller

The dependencies are rclcpp, sensor_msgs, and std_msgs. Consequently, the CMakeLists.txt and package.xml files have been appropriately modified to compile the new node according to the specified dependencies.

CMakeLists.txt :

```
find_package(rclcpp REQUIRED)
find_package(sensor_msgs REQUIRED)
find_package(std_msgs REQUIRED)
add_executable(${PROJECT_NAME}_node src/arm_controller_node.cpp)
ament_target_dependencies(${PROJECT_NAME}_node rclcpp std_msgs sensor_msgs)

install(TARGETS
  ${PROJECT_NAME}_node
  DESTINATION lib/${PROJECT_NAME})
```

And package.xml

```
<build_depend>std_msgs</build_depend>
<build_depend>sensor_msgs</build_depend>
<build_depend>rlcpp</build_depend>

<exec_depend>sensor_msgs</exec_depend>
<exec_depend>rlcpp</exec_depend>
<exec_depend>std_msgs</exec_depend>
```

b) Subscriber feed

In the arm_controller_node, we created a subscriber to the joint_states topic and a callback function that prints the current joint positions of the robot. We created the node class JointStateSubscriber by inheriting from rclcpp::Node. The constructor uses the node's create_subscription class to execute the callback.

```
class JointStateSubscriber : public rclcpp::Node
{
public:
  JointStateSubscriber()
  : Node("joint_state_subscriber")
  {
    subscription_ = this->create_subscription<sensor_msgs::msg::JointState>(
      "/joint_states", 10,
      std::bind(&JointStateSubscriber::topic_callback, this, _1)
    );
  }

private:
  void topic_callback(const sensor_msgs::msg::JointState & msg) const
  {
    for (size_t i = 0; i < std::min((size_t)4, msg.position.size()); i++)
    {
      RCLCPP_INFO(this->get_logger(), "The joint %zu: %.3f", i+1, msg.position[i]);
    }
  }

  rclcpp::Subscription<sensor_msgs::msg::JointState>::SharedPtr subscription_;
};
```

We then can receive the information with the text : “The joint 2 : “

c) Creation of the publisher node

We use differents target positions to test our system and a mode trajectory control and a position one :

```
class RobotControllerNode : public rclcpp::Node
{
public:
  RobotControllerNode() : Node("robot_controller_node"), step_index_(0)
  {
    // Initialize controller mode parameter
    this->declare_parameter("use_trajectory_mode", false);
    trajectory_mode_ = this->get_parameter("use_trajectory_mode").as_bool();

    // Define target positions for the robot joints
    joint_target_positions_ = {
      {0.0, 0.0, 0.0, 1.0},
      {0.7, 0.0, 0.7, 1.0},
      {0.7, -1.0, 0.7, 0.0},
      {0.0, 0.0, 0.0, 0.0}
    };

    // Set up appropriate publisher based on control mode
    if (trajectory_mode_) {
      trajectory_publisher_ = this->create_publisher<trajectory_msgs::msg::JointTrajectory>(
        "/trajectory_controller/commands", 10);
      RCLCPP_INFO(this->get_logger(), "Trajectory control mode activated");
    } else {
      position_publisher_ = this->create_publisher<std_msgs::msg::Float64MultiArray>(
        "/position_controller/commands", 10);
      RCLCPP_INFO(this->get_logger(), "Position control mode activated");
    }

    // Create timer for periodic position updates
    update_timer_ = this->create_wall_timer(
      std::chrono::milliseconds(5000),
      std::bind(&RobotControllerNode::update_joints, this)
    );
  }
```

And then we add to the function a system to update each joint one by one to reach the desired final position :

```
private:
  void update_joints()
  {
    const auto& target = joint_target_positions_[step_index_];

    if (trajectory_mode_) {
      // Publish trajectory message
      auto trajectory = trajectory_msgs::msg::JointTrajectory();
      trajectory.joint_names = {"joint_0", "joint_1", "joint_2", "joint_3"};
      trajectory.points.push_back({});
      trajectory.points.back().positions = target;
      trajectory.points.back().time_from_start = rclcpp::Duration(5s);
      trajectory_publisher_->publish(trajectory);
      RCLCPP_INFO(this->get_logger(), "Published trajectory step %d", step_index_);
    } else {
      // Publish position command
      auto command = std_msgs::msg::Float64MultiArray();
      command.data = target;
      position_publisher_->publish(command);
      RCLCPP_INFO(this->get_logger(), "Published position step %d", step_index_);
    }

    // Increment step counter (wraps around automatically)
    step_index_ = (step_index_ + 1) % joint_target_positions_.size();
  }

  size_t step_index_;
  bool trajectory_mode_;
  rclcpp::Publisher<trajectory_msgs::msg::JointTrajectory>::SharedPtr trajectory_publisher_;
  rclcpp::Publisher<std_msgs::msg::Float64MultiArray>::SharedPtr position_publisher_;
  std::vector<std::vector<double>> joint_target_positions_;
  rclcpp::TimerBase::SharedPtr update_timer_;
};
```

```
  bool use_trajectory_;
  size_t current_step_;
  std::vector<std::vector<double>> target_pos_;

  rclcpp::TimerBase::SharedPtr timer_;
  rclcpp::Publisher<std_msgs::msg::Float64MultiArray>::SharedPtr publisher_pos_;
  rclcpp::Publisher<trajectory_msgs::msg::JointTrajectory>::SharedPtr publisher_traj_;
};
```

d) Creation of a trajectory publisher

We add a new type of control in the armando_controller.yaml file to insert trajectories.

```
state_publish_rate: 200.0
action_monitor_rate: 20.0

open_loop_control: true
allow_integration_in_goal_trajectories: true

allow_partial_joints_goal: true

constraints:
  stopped_velocity_tolerance: 0.01
  goal_time: 0.0
```

We stumble upon an error in the modification of the launch file, the system used to change in between position and trajectory doesn't work properly.