ANNEX C. THE ROS SIMULATION ENVIRONMENT

In this annex a detailed description of the ROS package of the simulation environment employed in this Master's thesis will be offered. Consider this annex as a manual of instructions for an optimal usage.

C.1. Installation and Dependencies

Firstly, to operate this ROS environment, Ubuntu 20.04 is required. The ROS simulation environment is available at the following GitHub repository:

https://github.com/KP1598XD/AccelerationCompensation_ArmedMobileRobot

To obtain the ROS environment, proceed with cloning the repository using Git (ensure Git is installed beforehand):

git clone https://github.com/KP1598XD/AccelerationCompensation_ArmedMobileRobot.git

Once the repository has been cloned, several dependencies must be installed to ensure the ROS package functions properly:

Table 5: Libraries and Dependencies

Name	Source	Installation Command
ROS Noetic	ROS Wiki	`sudo apt-get install ros-noetic-desktop-full`
Python 3	Pyhton Official Site	`sudo apt-get install python3`
ROS Controllers	ROS Wiki	`sudo apt-get install ros-noetic-ros-control ros-noetic-ros-controllers`
Gazebo	Gazebo Official Site	`sudo apt-get install ros-noetic-gazebo-ros-pkgs ros-noetic-gazebo-ros-control`
PyQt5	PyQt5 Documentation	`pip3 install PyQt5`
Pandas	Pandas Official Site	`pip3 install pandas`
Matplotlib	Matplotlib Official Site	`pip3 install matplotlib`
Termcolor	Termcolor on PyPI	`pip3 install termcolor`
MoveIt	MoveIt Official Site	`sudo apt-get install ros-noetic-moveit`

C.2. File Structure and Organization

This section outlines the organization of the package, detailing the purpose and contents of each directory and key file. Understanding this structure will help users navigate the package, utilize its functionalities, and modify it according to their needs.

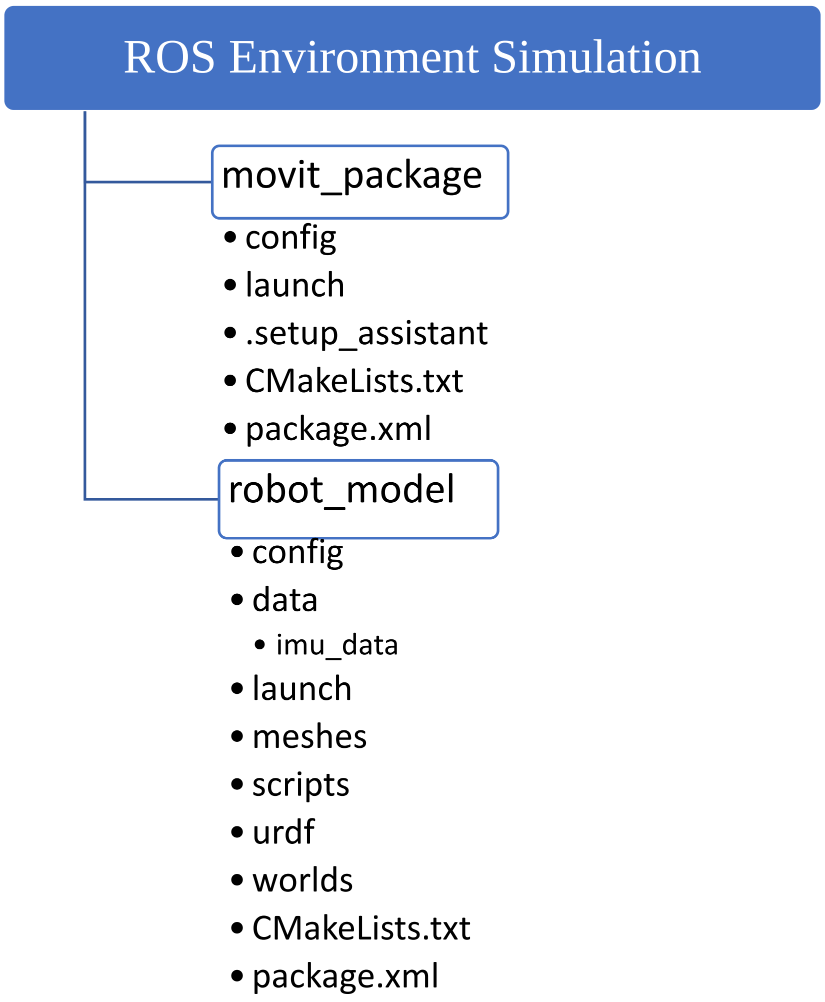


Figure 1: File organization scheme

As observed, the ROS environment is divided into two main packages: the *movit_package* and the *robot_model*. The first package includes configuration and launch files created using the MoveIt Setup Assistant. This powerful tool facilitates the generation and implementation of trajectory planning modules, encompassing capabilities such as manipulation, robotic arm control, mobile navigation, and perception. Within this package, the *KDL* inverse kinematics plugin is implemented. The second package, *robot_model*, contains various elements: the Xacro of the 3D robot model (located in the *urdf* folder), which configures links, joints, Gazebo parameters, virtual IMUs, and controllers; the meshes for the 3D visualization of the robot in STL files (located in the *meshes* folder); the launch file that executes the robot's visualization and various elements (found in the *launch* folder); scripts for the acceleration compensation algorithm and the visualization of information from IMUs and TFs (located in the *scripts* folder); the Gazebo world used as the simulation environment (located in the *worlds* folder); and finally, the results and gathered data regarding the calculated sloshing inclinations, acceleration, etc (located in the *data* folder).

C.3. ROS Nodes

The ROS environment simulation operates various nodes to perform the simulation. Some nodes are executed using libraries such as Gazebo, ROS controllers, or the MoveIt package, which provide foundational functionalities. Additionally, custom nodes have been developed to offer distinct capabilities. In this section, we will provide a brief overview of these custom nodes, all of which are contained within the */robot_model/scripts* folder.

Table 6: Create ROS list, topics and services used

Name	Functionality	Topics published	Topics subscribed	Services
acc_viewer.py	Base link IMU data visualizer	None	*/imu/data*	None
algorithm_ac.py	Joint positions calculation using *KDL*	None	*/pose_tcp*	*/compute_ik*
graphs_generator.py	Generation of graphs (accelerations, inclinations, TCP real orientations)	None	None	None
imu_conversion.py	TCP orientations viewer	*/orientatin_ee*	*/imu2/data*	None
initial_pose_1.py	Initialize position of the arm robot when starting the simulation environment	*/arm_robot_controller/command*	None	None
initial_pose_2.py	Initialize position of the arm robot when starting the tests	*/arm_robot_controller/command*	None	None
initialization.py	Initialize tests, creates test folder and selects acceleration profile data	None	None	None
mobile_robot_trajectory.py	Send velocities to the mobile robot controller to move it	*/cmd_vel*	*/imu_data*	None
move_joints.py	Send joint positions to the robot arm controller	*/arm_robot_controller/command*	*/joint_states* */orientation_ee*	None
orientation_planning.py	Calculation of inclination (normal, filtered and shifted)	None	None	None
pose_glass.py	Glass pose viewer using TF	*/pose_tcp*	None	None
starter.py	Starter for the tests of acceleration compensation algorithm	None	None	None

tf_broadcaster_wheels.py	Broadcast of TF for each mobile robot wheel	None	None	None

An overview of the relationships among the nodes can be observed in Figure 20. Please note that not all of the previously listed nodes are depicted in the schematic; only those that are permanently active during the execution of the launch file are shown. Other nodes are triggered independently and run only once during their own execution process.

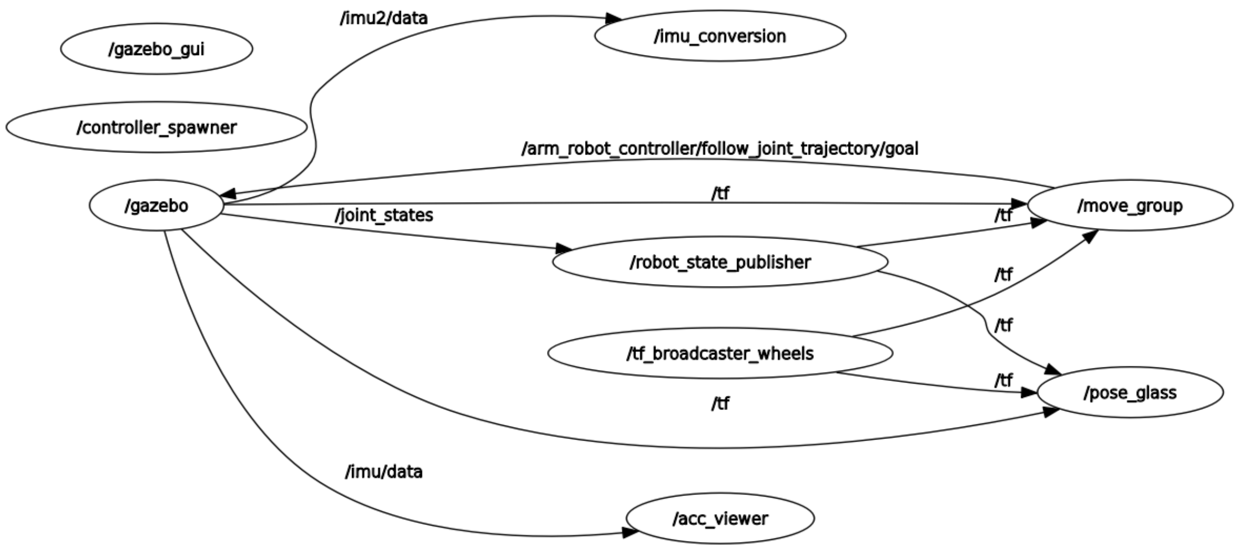


Figure 2: ROS Node relations scheme

C.4. Transformation Framework (tf) Tree Structure

In the context of this project, a comprehensive tf tree has been established to define the spatial relationships between various components of the robotic system. The tf tree is a hierarchical structure that encapsulates the parent-child relationships between frames, reflecting how each part of the robot is positioned and oriented relative to others over time. track multiple coordinate frames over time.

Presented below is the tf tree diagram for our robot system:

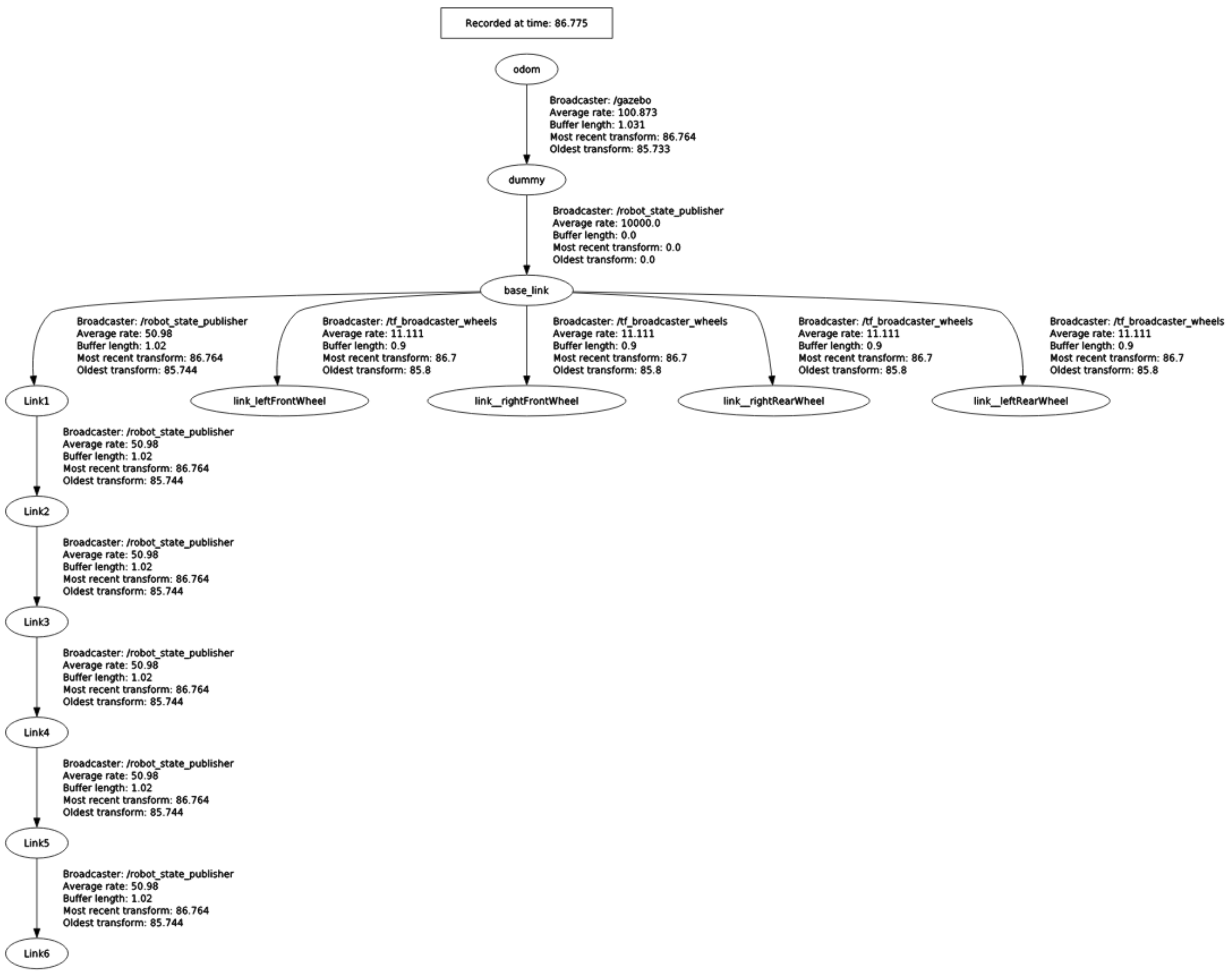


Figure 3: tf transformations tree

The root of this hierarchy is the *odom* frame, which symbolizes the robot's odometry frame, offering a frame of reference within the environment. A *dummy* frame is directly connected to *odom*, potentially fulfilling specific roles in the system such as testing or alignment procedures. The *base_link* frame sits at the core of the robot's frame structure, serving as the pivotal reference from which all other frames are derived:

- Frames *Link1* to *Link6* represent each link of the robotic arm.
- Wheel-related frames, namely *link_frontLeftWheel*, *link_frontRightWheel*, *link_rearLeftWheel*, and *link_rearRightWheel*, articulate the positions of the respective wheels relative to the base_link.

There are three distinct broadcasters:

- */gazebo,* which broadcast the *odom* frame.
- */robot_state_publisher,* which broadcast the frames related to the robotic arm: *Link1-6* and *base_link*.
- */tf_broadcaster_wheels*, which is one of the created nodes from the Table 6 and it broadcast the frames related to the mobile robot: *link_frontLeftWheel, link_frontRightWheel, link_rearLeftWheel, and link_rearRightWheel*.

C.5. Instructions for Use

This section provides detailed steps on how to utilize the functionalities of the ROS package developed for this project. These instructions are designed to help users quickly get started with the package and to familiarize themselves with its basic operations.

1. **Initial Setup**

   After cloning the repository and installed all the libraries and dependencies form section C.1, the next step would be the building the ROS workspace using *catkin_make* and the posterior *source devel/setup.bash*.

2. **Running the simulation environment**

   Once the workspace is configured, the ROS environment can be initiated, specifically the MoveIt and Gazebo setups. The MoveIt setup initializes and configures the controllers and kinematics plugins for the robotic arm. Concurrently, the Gazebo setup spawns the 3D robot model and activates all necessary plugins, including controllers and the IMU, for the simulation. This project utilizes two distinct launch files to operate the simulation environment:

   a. Launching with the Rviz from the MoveIt setup:

      roslaunch movit_package full_robot_arm_sim.launch

      In addition to running all the previously mentioned elements, this launch will also initiate Rviz with the MoveIt setup. For those interested, Rviz provides access to various tools and functions from trajectory planning plugins.

b. Launching without Rviz:

roslaunch movit_package robot_model_simulation.launch

This launch configuration will not initiate the Rviz visualizer; it is designed to activate only the essential elements required for conducting the tests.

3. Conducting the tests and results

To execute the tests, two distinct procedures are available:

- Capture of the acceleration profiles:

  The first procedure involves capturing the acceleration using the virtual IMU located at the base link of the robot. This step can be executed using the following node and command:

  rosrun robot_model mobile_robot_trajectory.py

  Upon execution, the node will prompt the user to input a series of values for distinct parameters, including acceleration/deceleration rate, duration of the movement, maximum speed, and the name of the file to be saved. Subsequently, the node will send the necessary velocity commands to the mobile robot controller to execute the desired acceleration profile. Finally, it will save the IMU data in a CSV file located in */robot_model/data/imu_data*.

- Acceleration compensation algorithm:

  Once the acceleration data from the previous step is obtained, we can proceed with the application of the algorithm. To run the algorithm, the following command and node are used:

  rosrun robot_model starter.py

  This node orchestrates the sequential execution of several sub-nodes:

  1- *initialization.py*: This node creates a folder named *test_i*, where *i* represents the lowest available number within the */robot_model/data* folder.

Additionally, it prompts the user to input the name of the file containing the acceleration data.

2- *initial_pose_2*.py: This node prompts the user to specify an orientation for Joint1 and transmits the selected configuration to the robotic arm controller. Its purpose is to facilitate testing of the algorithm's response to sloshing orientations affecting the XY plane.

3- *orientation_planning*.py: Using the acceleration data file, this node calculates the sloshing orientations (normal, filtered, and shifted) and saves all calculations in a CSV file within the test folder.

4- *algorithm_ac.py:* Leveraging the information from the previous node, this node utilizes the inverse kinematics plugin to compute distinct joint configurations necessary to compensate for the sloshing inclinations of the liquid. The resulting joint positions are saved in a CSV file

5- *move_joints*.py: This node transmits the previously calculated positions to the robotic arm controller, captures information from the second IMU located at the TCP, and saves this data in another CSV file.

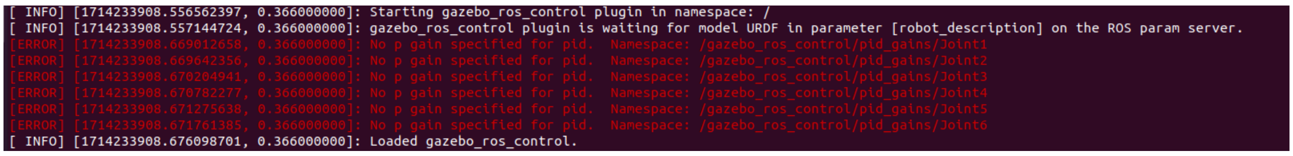
6- *graphs_generator.py*: Finally, using the CSV files generated by the preceding nodes, this node produces two different graphs. One graph contains information about the three different calculated inclinations with respect to acceleration, while the second graph displays the real orientations of the TCP, along with the shifted and filtered sloshing inclinations and the acceleration.

For proper utilization of these nodes, please adhere to all instructions and recommendations provided during their execution.

C.6. Considerations and Precautions

In this section, we outline important considerations and precautions to be aware of when working with the ROS simulation environment.

During the launch of the simulation environment, a series of error messages will be displayed in the terminal, similar to those shown in Figure 22.

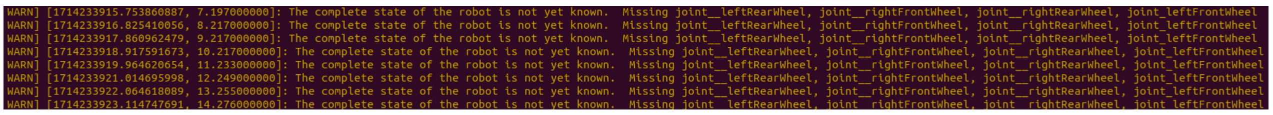


```
[ INFO] [1714233908.556562397, 0.366000000]: Starting gazebo_ros_control plugin in namespace: /
[ INFO] [1714233908.557144724, 0.366000000]: gazebo_ros_control plugin is waiting for model URDF in parameter [robot_description] on the ROS param server.
[ERROR] [1714233908.669012658, 0.366000000]: No p gain specified for pid.  Namespace: /gazebo_ros_control/pid_gains/Joint1
[ERROR] [1714233908.669642356, 0.366000000]: No p gain specified for pid.  Namespace: /gazebo_ros_control/pid_gains/Joint2
[ERROR] [1714233908.670204941, 0.366000000]: No p gain specified for pid.  Namespace: /gazebo_ros_control/pid_gains/Joint3
[ERROR] [1714233908.670782277, 0.366000000]: No p gain specified for pid.  Namespace: /gazebo_ros_control/pid_gains/Joint4
[ERROR] [1714233908.671275638, 0.366000000]: No p gain specified for pid.  Namespace: /gazebo_ros_control/pid_gains/Joint5
[ERROR] [1714233908.671761385, 0.366000000]: No p gain specified for pid.  Namespace: /gazebo_ros_control/pid_gains/Joint6
[ INFO] [1714233908.676098701, 0.366000000]: Loaded gazebo_ros_control.
```

Figure 4: Simulation error message

These error messages occur because the robotic arm is utilizing a *position_controllers/JointTrajectoryController,* which does not require specification of parameters and gains for the PID control of each joint. In this project, the dynamic behavior of the robot arm was not considered; therefore, the absence of PID implementation results in an unrealistic behavior of the robot. However, the primary focus of this project is on the kinematic behavior of the arm.

Furthermore, an additional warning message will be displayed (see Figure 23). This warning is related to the absence of information regarding the state of the wheel joints. The message appears because the MoveIt setup considers all joints of the robot, even though only the robotic arm joints are being controlled in this setup.

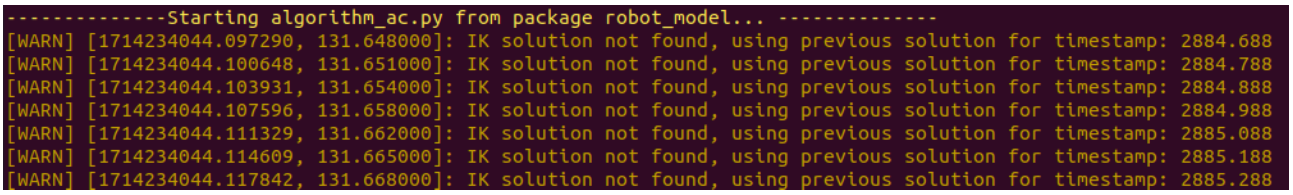


```
WARN] [1714233915.753860887, 7.197000000]: The complete state of the robot is not yet known.  Missing joint__leftRearWheel, joint__rightFrontWheel, joint__rightRearWheel, joint_leftFrontWheel
WARN] [1714233916.825410056, 8.217000000]: The complete state of the robot is not yet known.  Missing joint__leftRearWheel, joint__rightFrontWheel, joint__rightRearWheel, joint_leftFrontWheel
WARN] [1714233917.860962479, 9.217000000]: The complete state of the robot is not yet known.  Missing joint__leftRearWheel, joint__rightFrontWheel, joint__rightRearWheel, joint_leftFrontWheel
WARN] [1714233918.917591673, 10.217000000]: The complete state of the robot is not yet known.  Missing joint__leftRearWheel, joint__rightFrontWheel, joint__rightRearWheel, joint_leftFrontWheel
WARN] [1714233919.964620654, 11.233000000]: The complete state of the robot is not yet known.  Missing joint__leftRearWheel, joint__rightFrontWheel, joint__rightRearWheel, joint_leftFrontWheel
WARN] [1714233921.014695998, 12.249000000]: The complete state of the robot is not yet known.  Missing joint__leftRearWheel, joint__rightFrontWheel, joint__rightRearWheel, joint_leftFrontWheel
WARN] [1714233922.064618089, 13.255000000]: The complete state of the robot is not yet known.  Missing joint__leftRearWheel, joint__rightFrontWheel, joint__rightRearWheel, joint_leftFrontWheel
WARN] [1714233923.114747691, 14.276000000]: The complete state of the robot is not yet known.  Missing joint__leftRearWheel, joint__rightFrontWheel, joint__rightRearWheel, joint_leftFrontWheel
```

Figure 5: Warning simulation message

During the execution of the node for capturing acceleration data, it is strongly recommended to pay close attention to the parameters inputted. Abnormal values of these parameters can result in issues within the Gazebo environment, potentially causing the robot to stumble or encounter other problems.

Finally, another warning message will appear when executing the node for applying the acceleration compensation algorithm. This warning message was previously discussed in Chapter 4.



```
--------------Starting algorithm_ac.py from package robot_model... --------------
[WARN] [1714234044.097290, 131.648000]: IK solution not found, using previous solution for timestamp: 2884.688
[WARN] [1714234044.100648, 131.651000]: IK solution not found, using previous solution for timestamp: 2884.788
[WARN] [1714234044.103931, 131.654000]: IK solution not found, using previous solution for timestamp: 2884.888
[WARN] [1714234044.107596, 131.658000]: IK solution not found, using previous solution for timestamp: 2884.988
[WARN] [1714234044.111329, 131.662000]: IK solution not found, using previous solution for timestamp: 2885.088
[WARN] [1714234044.114609, 131.665000]: IK solution not found, using previous solution for timestamp: 2885.188
[WARN] [1714234044.117842, 131.668000]: IK solution not found, using previous solution for timestamp: 2885.288
```

Figure 6: Acceleration compensation algorithm warning message