Industrial Mobile Cable-Driven Parallel Robots Simulation and Manipulation

Duan Xin
duanxin2023@shanghaitech.edu.cn

Xinzhe Liu
liuxzh12023@shanghaitech.edu.cn

Abstract

This project implemented the simulation task of an industrial 4-Cable-Driven Parallel Robots (CDPR) transporting solar rack with the Robot Operating System (ROS). Firstly, We will simplify the existing original Computer-Aided Design (CAD) robot model, extracting the compact structure as STL files with SolidWorks. Then we will export XML Macros(Xacro), which could be used by the simulation platform Gazebo to spawn robot. Then we could use MoveIt! to do kinematic and inverse kinematic computations, motion planning and manipulation for the robot.

1 Introduction

Our project is part of a big Industrial project – Bolight Industrial Robot, shown in Figure 1. The overall purpose of the industrial project is to transport large solar panels in the desert with a large wheeled mobile lifting robot of Ackerman structure. If collects sensor data such as LiDAR and dGPS to recognize the environment state and then plan to place the solar board to the corresponding position. Our project mainly concentrate on the upper structure of the robot, which a lifting device driven by "4-ropes", which could consider it as a 4-Cable-Driven Parallel Robots (CDPR).

Our project is a sub-module of the overall industrial project with the following targets:

- Export and improve URDF files from the original CAD file for simulation.
- Simplify the CDPR model into a compact structure with revolute joint and Prismatic joint and then compute the kinematic and inverse kinematic transformations from the original "rope" to the "virtual bar" model.
- Use ROS packages including MoveIt to control our own model.
- Implement the interface and simulation of the real robot.

2 System Description

At the beginning, we will extract the main structure as STL files from the CAD files provided by the industrial company and then finish the XACRO files describing the simplified compact structure. As it's hard to describe the CDPR mechanism structure in URDF format, We need to model it as a virtual revolute and prismatic joint. We also need forward and inverse kinemtatic for that, which means given cable lengths, calculating prismatic and revolute joint values and given prismatic and revolute joint values, calculating cable length. Since we don't know whether our simplification will lose some accuracy of the original CDPR model. We need to take this step carefully and refer to some literature related to CDPR. Then We will also use MoveIt! to manipulate this robot model, doing kinematic and inverse kinematic computations, motion planning and manipulation. Our project consists of 3 ros packages: megatron_description, megatron_publisher_simulator and megatron_arm_interface.

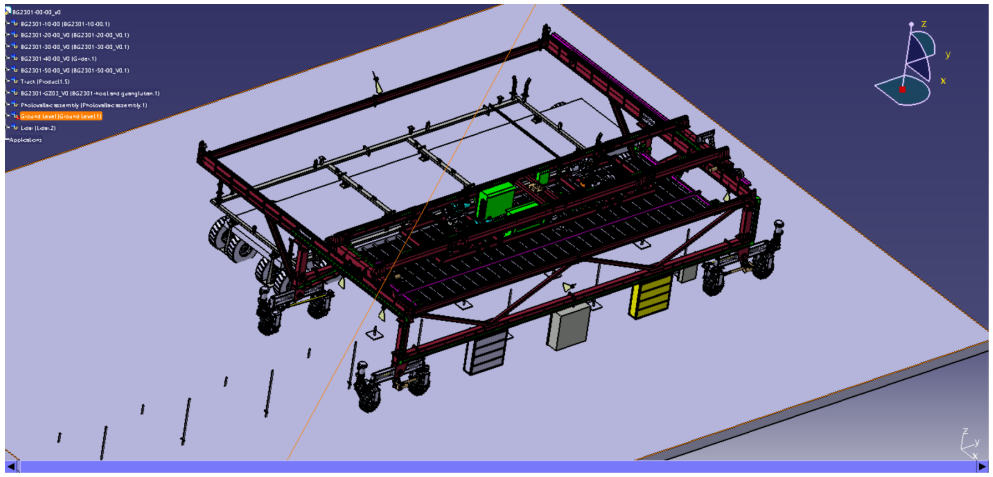

Figure 1: This is a CAD demo model of the industrial robot for transporting large solar panels in the desert.

2.1 Megatron_description

This package contains the urdf model of the robot. We extract the stl model of each important joint from CAD model, and measured the frame relationship between the joints. Based on the frame relationship, we write the xacro files. There are three xacro files: base, track and lidar. Since there are four tracks and and three lidars, one xacro file can be reused in base files. Then when we need to use this model, we use command $xacro[file\_name]$ in the launch file, then ros can generate urdf model by the xacro files. The urdf model is shown as Figure 2.

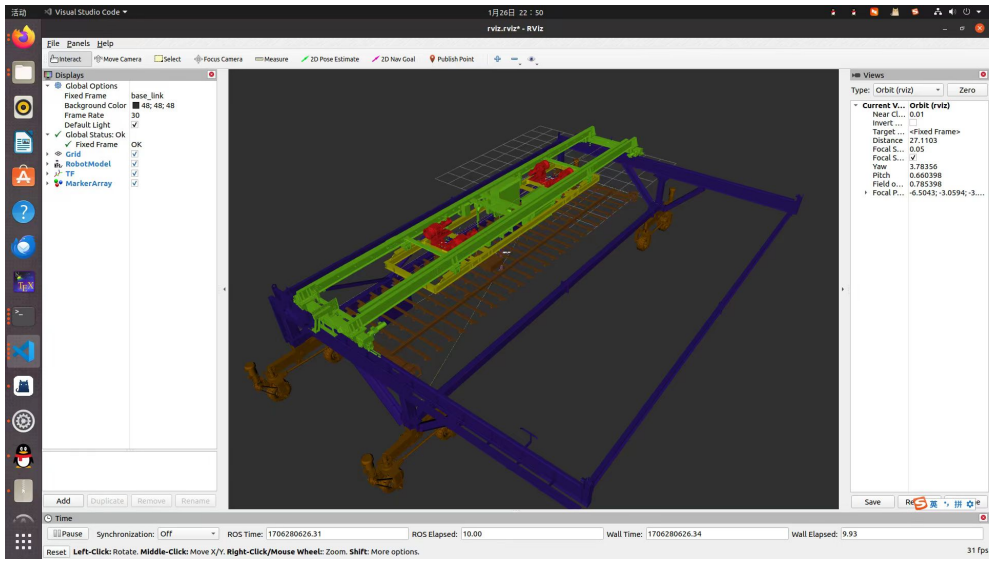

Figure 2: URDF model of the robot

2.2 Megatron_publisher_simulator

Because the real robot is very big and we can not use it frequently, we implement a robot simulator to simulate the real robot. The program defines a set of variables for each joint of the robot, which is used to maintain the state of the joint, the value, and so on. The gentry1 joint, for example, is a translational joint that moves along a fixed axis. We define the position and state of the variable,

which represent the current position of the joint and the motion state respectively. The program defines two sets of functions, which are the function that publishes its own state and the function that receives the control signal.

Self-state publishing function: Defines a set of ros topics that periodically publish information about all joints through a thread.

Control signal receiving function: When the outside world sends the target value of each joint to the simulator, the program will modify the value of the joint position at a certain speed through the callback function, so as to reach the target position.

The above two sets of functions simulate the data flow in two directions of the actual robot, the output of the robot's own state information, and the output of the target position to the robot by the upper computer.

2.3 Megatron_arm_interface

This is the main part of our projects. It contains two ros nodes: interface and controller.

2.3.1 Interface

This node is mainly used to receive information about its own state published by the robot. Since the actual robot and urdf model use different structures, when we receive the original joint data released by the robot, we need to solve the data of each joint in the urdf model through mathematical calculation. After solving the data of the joints in the urdf model, we can publish these data to rviz, and rviz will display the latest status of the urdf model in real time according to the latest data received, so as to display the status of the robot in the upper computer.

2.3.2 Controller

This node is mainly used to receive the target position published by other nodes, and send the target value of each joint to the robot. Workflow: Node subscribes to the topic of the target location. In the callback function, we first convert the quaternions to Euler angles. Then for each direction, we calculate the value of the robot joint corresponding to the target position. After the calculation is completed, the program sends the joint target position to the robot and monitors the joint movement status in real time. When all joints stop moving, we consider this control to be over and the callback function exits.

2.4 Integration

The overall workflow of the whole project is divided into two following parts.

- Upstream data flow: The simulator (which in real tests is replaced by a real robot) publishes the raw joint message, the interface subscribe to the message, calculates the joint value in the urdf, and publishes it to rviz for visualization.

- Downstream data flow: the external program publishes a target position message, and the controller subscribs to the message. According to the target position, the controller calculates the target value of the primal joint and publishes the value to the simulator, which moves each joint to the target position at a certain speed.The data flow in ros is shown in Figure 3.

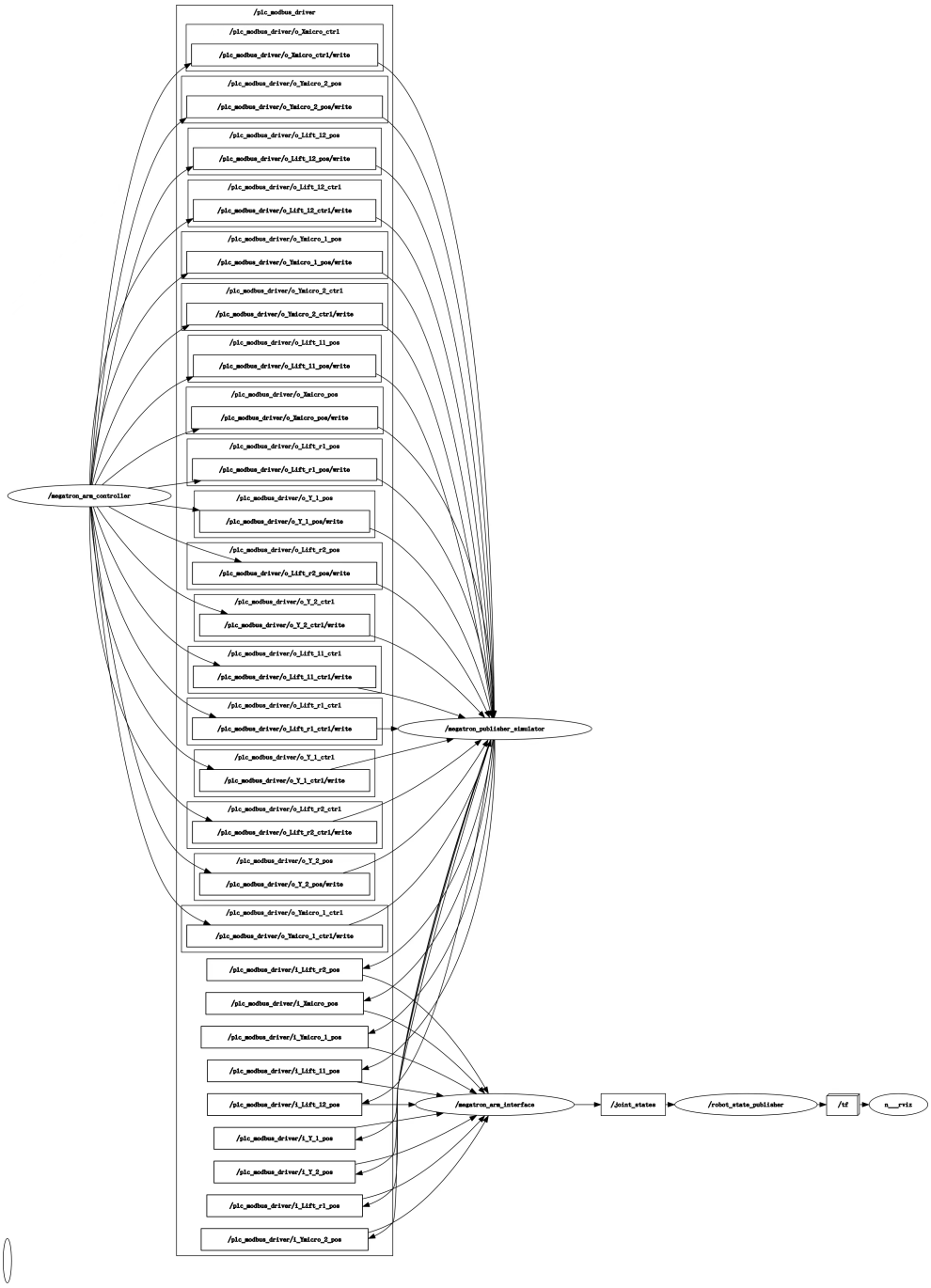

Figure 3: Data flow in ros topic

3 Experiment

In this section, we describe the experimental implementation of our system, which involved the integration of a visual demo in Rviz and testing on a real industrial robot. Our goal was to establish bidirectional communication between Rviz and the robot, enabling control over the robot's joints.

Implementation of Visual Demo in Rviz We successfully developed a visual demo in Rviz, leveraging its 3D visualization capabilities. The demo served as an interface for controlling the

robot's movements. We utilized Rviz's interactive markers and GUI features to ensure user-friendly interaction.

Testing on a Real Industrial Robot We conducted tests on a large industrial robot to evaluate the performance of our system in a real-world scenario. This allowed us to validate the bidirectional communication between Rviz and the physical robot.

Receiving Messages from the Real Robot to Control Rviz Demo A significant achievement was establishing a communication channel that enabled the real robot to send messages to control the Rviz demo. This allowed us to synchronize the robot's movements with the visual representation in Rviz.

Sending Messages to Control Robot Joints Furthermore, our system demonstrated the capability to send messages from Rviz to control the individual joints of the real industrial robot. This bidirectional control facilitated seamless coordination between the Rviz demo and the physical robot's movements.

4 Conclusions

Our project will finish XACRO files refer to given CAD files and then control and simulate it within ROS using packages including MoveIt. This would work as a sub-part of a real working industrial robot within ROS, transporting the solar racks to correct position and pose in the desert. Our work could work as the basic components for further meaningful tasks. Additionally, since the pose determination of the CDPR model is a difficult task due to the elasticity and changing shape of the ropes and it's also hard to simulate in most of the tools, we hope that the technique we used to simplify the model and computing the kinematic and inverse kinematic model could give an easy to approximately simulate CDPR models with acceptable accuracy.
Our experiments confirmed the successful implementation of the visual demo in Rviz and its integration with a real industrial robot. The bidirectional communication between Rviz and the robot allowed for effective control over the robot's joints, showing the practical applicability of our system.