ROS Bootcamp - Day 4

Introduction To ROS1

JC Cruz and Alec Graves

UTSA RAS

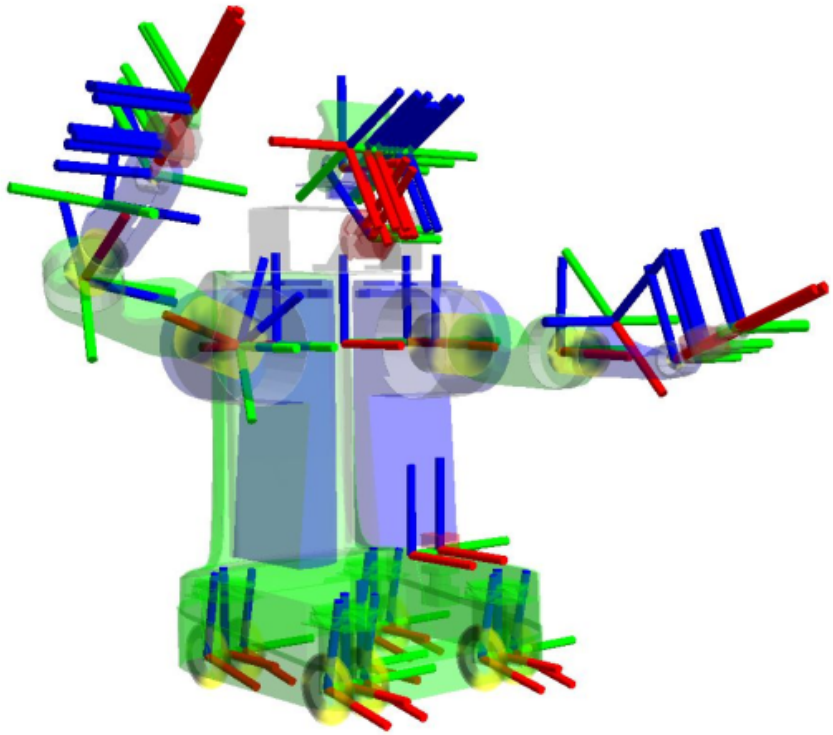
May 18, 2023

Overview

- TF Transformation System
- RQT User Interface
- Robot models (URDF)
- Simulation descriptions (SDF)
- ROS Control
- Intro to ROS2
- **Activity:** Services, Bag file, RQT Plot

TF Transformation System

- TF (transform) is a ROS package for managing coordinate transformations
- Tracks the relationships between coordinate frames over time
- Allows transforming data between different coordinate frames
- Useful for tasks like combining sensor data or navigating a robot



Broadcasting Transforms

- Broadcast transforms using a `tf2_ros::TransformBroadcaster`

```
#include <tf2_ros/transform_broadcaster.h>
//...
tf2_ros::TransformBroadcaster tf_broadcaster;
geometry_msgs::TransformStamped transform;
// Fill in the transform values...
tf_broadcaster.sendTransform(transform);
```

Listening to Transforms

- Listen to transforms using a `tf2_ros::Buffer` and `tf2_ros::TransformListener`

```
#include <tf2_ros/transform_listener.h>
#include <tf2_ros/buffer.h>
//...
tf2_ros::Buffer tf_buffer;
tf2_ros::TransformListener tf_listener(tf_buffer)
    ;
//...
geometry_msgs::TransformStamped transform;
transform = tf_buffer.lookupTransform(
    target_frame, source_frame, ros::Time(0));
```

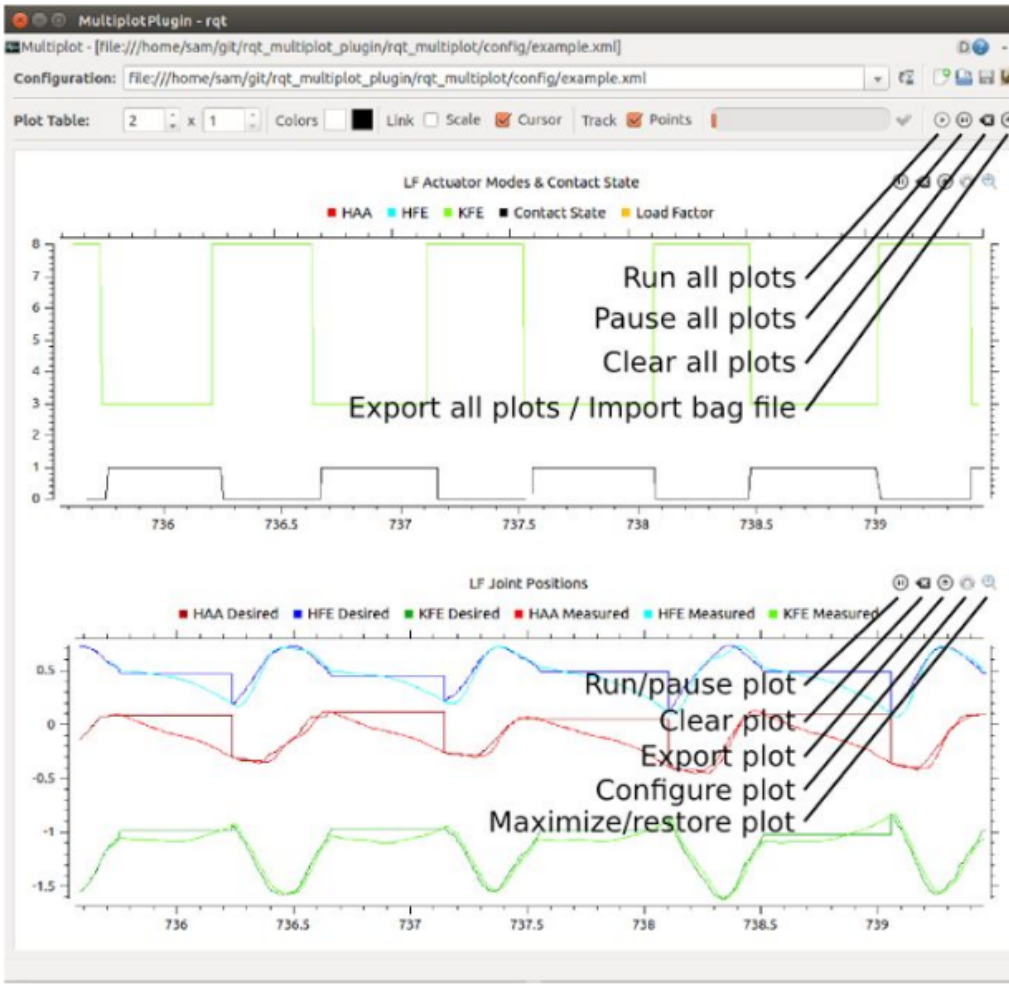
RQT User Interface

- RQT is a GUI framework for ROS, built on top of Qt
- Provides tools for visualizing and debugging a ROS system
- Offers a plugin system for extending its functionality
- Run RQT using: `rqt` or `rosrun rqt_gui rqt_gui`

More info: `http://wiki.ros.org/rqt`

RQT Plugins

- Some popular RQT plugins:
  - **rqt graph:** Visualize the computation graph of a running ROS system
  - **rqt plot:** Plot data from ROS topics
  - **rqt console:** Display log messages
  - **rqt tf tree:** Visualize TF frames and their relationships
- Run a specific plugin using: `rqt --standalone <plugin_name>`



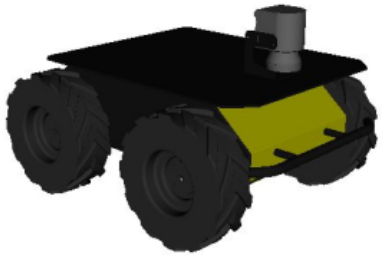
Creating Custom RQT Plugins

- To create your own RQT plugin:
  - Extend the `rqt_gui_cpp::Plugin` class (C++) or the `rqt_gui_py::Plugin` class (Python)
  - Implement the required methods
  - Add a plugin description XML file
  - Export the plugin using a ROS package

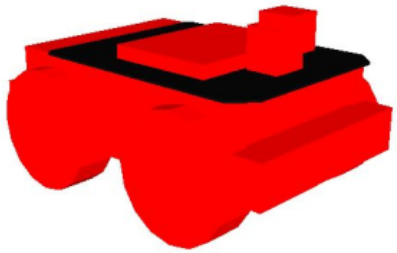
More info: `http://wiki.ros.org/rqt`

Robot Models (URDF)

- URDF (Unified Robot Description Format) is an XML format for representing a robot model
- Describes the robot's kinematic and dynamic properties, as well as visual and collision properties
- Allows for easy visualization and simulation of the robot



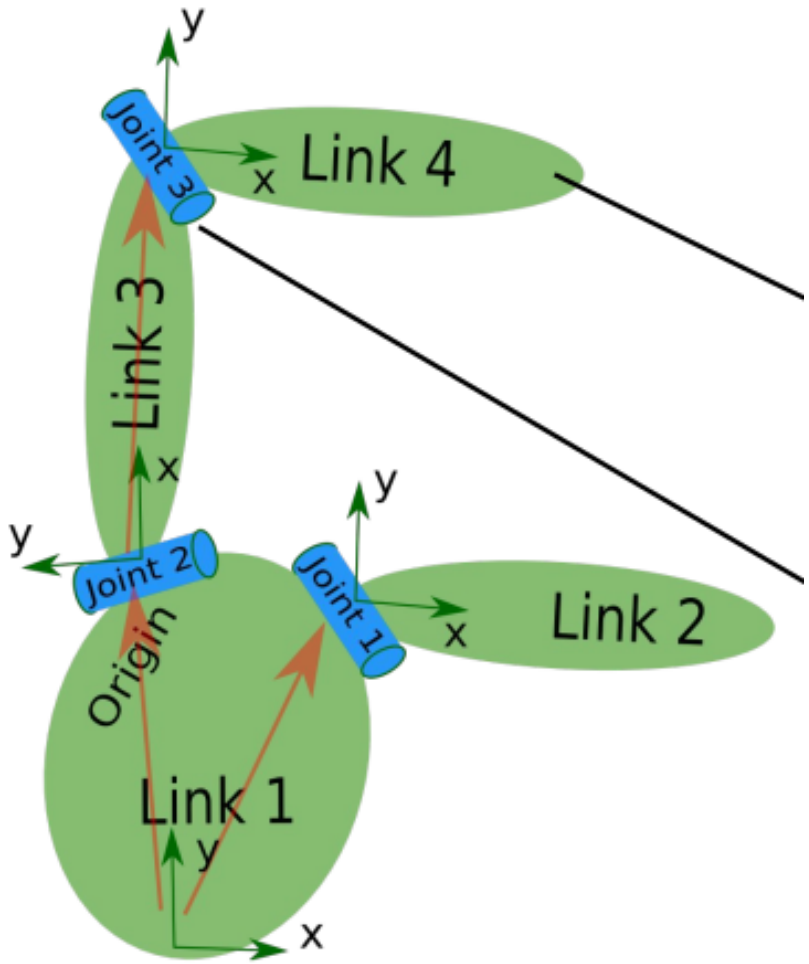
Mesh for visuals



Primitives for collision

URDF Components

- Key components of a URDF file:
  - **Links:** Rigid bodies with properties like mass, inertia, and visual/collision shapes
  - **Joints:** Define the connection and relative motion between two links
  - **Materials:** Define colors and textures for visual representation
  - **Transmission:** Describe the relationship between actuators and joints



Creating a URDF File

- To create a URDF file for your robot:
  - Define each link and its properties
  - Define each joint and its properties
  - Define materials (optional)
  - Define transmissions (optional)

```
<!-- Example URDF Code
    -->
<robot name="my_robot">
    <link name="base_link
    ">
        <!-- Link
    properties go here
    -->
    </link>
    <joint name="my_joint
    ">
        <!-- Joint
    properties go here
    -->
    </joint>
</robot>
```

Simulation Descriptions (SDF)

- SDF (Simulation Description Format) is an XML format for describing various objects within a simulation environment
- Used in Gazebo, a 3D robot simulator that integrates with ROS
- Can describe robots, sensors, terrains, buildings, and other objects in the simulation world

More info: `http://sdformat.org/`

SDF Components

- Key components of an SDF file:
  - **Model:** Represents a single object, such as a robot or a building
  - **Link:** Rigid body within a model with properties like mass, inertia, and visual/collision shapes
  - **Joint:** Define the connection and relative motion between two links
  - **Sensor:** Models various types of sensors, such as cameras, LIDAR, and GPS
  - **Light:** Represents light sources within the simulation environment

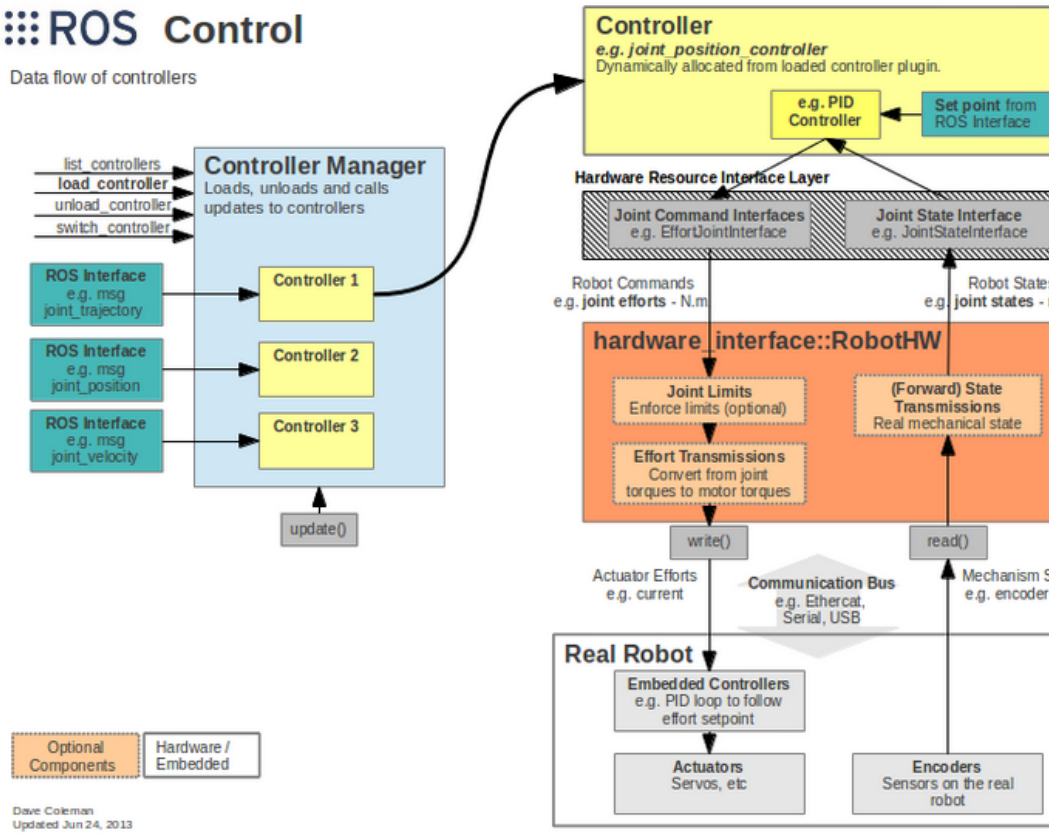
ROS Control Overview

- ROS Control is a framework for controlling robots using ROS
- Provides an abstraction layer for hardware interfaces, making it easier to switch between different robot hardware
- Supports various control schemes, such as position, velocity, and effort control

More info: `http://wiki.ros.org/ros_control`

ROS Control Components

- Key components of ROS Control:
  - **Hardware Interface:** Handles communication with the robot hardware
  - **Controller Manager:** Manages the loading, unloading, and switching of controllers
  - **Controller:** Implements a specific control algorithm, e.g., PID or trajectory control



Using ROS Control

- To use ROS Control with your robot:
  - Implement a hardware interface for your robot
  - Create a launch file to load the controller manager and controllers
  - Configure and tune the controllers for your specific robot

More info: `http://wiki.ros.org/ros_control/Tutorials`

ROS2 Overview

- Next-generation Robot Operating System designed for today's use cases
- Addresses limitations of ROS1 (not real-time, designed for research, mainly for PR2)
- Improved support for teams of robots, bare-metal microcontrollers, lossy networks
- API redesign, improved performance, security, and cross-platform support

More info: `https://design.ros2.org/articles/why_ros2.html`

Key Differences between ROS1 and ROS2

- **Middleware:** ROS2 uses the Data Distribution Service (DDS) instead of custom TCPROS
- **OS Support:** Native support for Ubuntu, Windows 10, macOS, and embedded systems
- **Languages:** C++11 and Python 3 support (C++03 and Python 2 in ROS1)
- **Build System:** Colcon build tool, isolated independent builds for packages
- **Launch Files:** Python-based launch files in ROS2, more configurable and conditioned execution
- **Real-time Support:** Deterministic real-time behavior with appropriate RTOS

More info: `https://design.ros2.org/articles/changes.html`

ROS2 Concepts

- Graph Concepts: nodes, messages, topics (similar to ROS1)
- ROS Client library (RCL): rclcpp for C++, rclpy for Python
- Discovery: nodes establish contact only if they have compatible Quality of Service settings

More info: `https://design.ros2.org/articles/changes.html`

ROS2 Workspace Environment and Building Packages

- Default workspace loaded with source /opt/ros/foxy/setup.bash
- Overlay workspaces and multiple ROS2 distributions
- Colcon build tool replaces catkin build in ROS1
- Clone, build, and source packages with colcon

More info:
`https://index.ros.org/doc/ros2/Tutorials/Colcon-Tutorial/`

ROS2 Launching and Creating Packages

- Launch files written in Python, no more XML launch files
- Create packages with ros2 pkg create command
- Create nodes and list dependencies during package creation

More info:
`https://index.ros.org/doc/ros2/Tutorials/Colcon-Tutorial/`

Using ROS1 and ROS2 Together

- ROS1 Bridge enables communication between ROS1 and ROS2 nodes
- Source ROS1 and ROS2, set `ROS_MASTER_URI`, and run the bridge

More info:

`https://index.ros.org/p/ros1_bridge/github-ros2-ros1_bridge/`