NXP PRESENTS AiM ARTIFICIAL INTELLIGENCE in MOBILITY

ARTIFICIAL INTELLIGENCE IN MOBILITY

ROS SOFTWARE STACK

IN COLLABORATION WITH
Time Of Sports®

Agenda

- About ROS
- Why ROS?
- Core Components
- Integration with other libraries

ABOUT ROS

The Robot Operating System (ROS) is an open-source, meta-operating system for your robot and provides a flexible framework for writing robot software.

- Provides a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.
- Provides OS-like services
- Provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

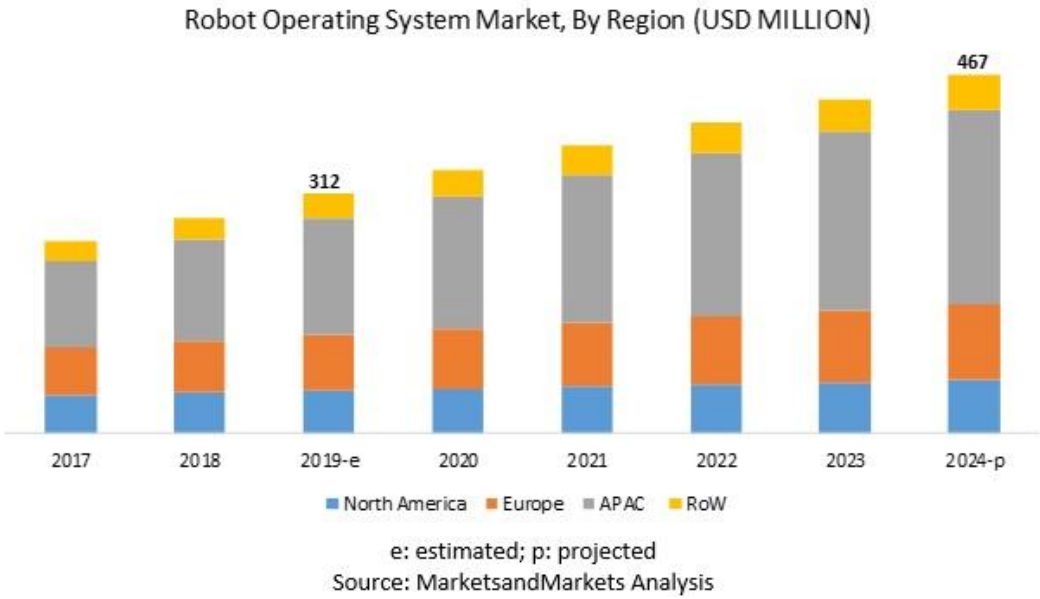
Currently supported on Unix-based platforms.

WHY ROS?

A Distributed, Modular Design

ROS was designed to be as distributed and modular as possible.

At last count there were over 3,000 packages in the ROS ecosystem, and that is only the ROS packages that people have taken the time to announce to the public.



Source: Robot Operating System Market by Types | ROS Market - 2024 | MarketsandMarkets

WHY ROS?

A Vibrant Community

ROS has grown to include a large community of active users worldwide ranging from research labs to adoption in the commercial sector, particularly in industrial and service robotics.

A Collaborative Environment

One of the core philosophies in ROS is shared development of common components.

WHY ROS?

Permissive Licensing

- The core of ROS is licensed under the standard three-clause BSD license. This is a very permissive open license that allows for reuse in commercial and closed source products.
- Commonly used licenses in community packages are the Apache 2.0 license, the GPL license, the MIT license, and even proprietary licenses
- Each package in ecosystem is required to specify a license

CORE COMPONENTS – COMMUNICATIONS INFRASTRUCTURE

Communications Infrastructure

At the lowest level, ROS offers a message passing interface that provides inter-process communication and is commonly referred to as a middleware.

The ROS middleware provides these facilities:

- publish/subscribe anonymous message passing
- recording and playback of messages
- request/response remote procedure calls
- distributed parameter system

CORE COMPONENTS – ROBOT SPECIFIC FEATURES

Robot-Specific Features

ROS provides common robot-specific libraries and tools :

- Standard Message Definitions for Robots
- Robot Geometry Library
- Robot Description Language
- Preemptable Remote Procedure Calls
- Diagnostics
- Pose Estimation
- Localization
- Mapping
- Navigation

CORE COMPONENTS

Standard Robot Messages

Years of community discussion and development have led to a set of standard message formats that cover most of the common use cases in robotics. There are message definitions for geometric concepts like poses, transforms, and vectors; for sensors like cameras, IMUs and lasers; and for navigation data like odometry, paths, and maps; among many others. By using these standard messages in your application, your code will interoperate seamlessly with the rest of the ROS ecosystem, from development tools to libraries of capabilities.

Robot Description Language

Another common robotics problem that ROS solves is how to describe your robot in a machine-readable way. ROS provides a set of tools for describing and modeling your robot so that it can be understood by the rest of your ROS system, including tf, robot_state_publisher, and rviz. The format for describing your robot in ROS is URDF (Unified Robot Description Format), which consists of an XML document in which you describe the physical properties of your robot, from the lengths of limbs and sizes of wheels to the locations of sensors and the visual appearance of each part of the robot.

Once defined in this way, your robot can be easily used with the tf library, rendered in three dimensions for nice visualizations, and used with simulators and motion planners.

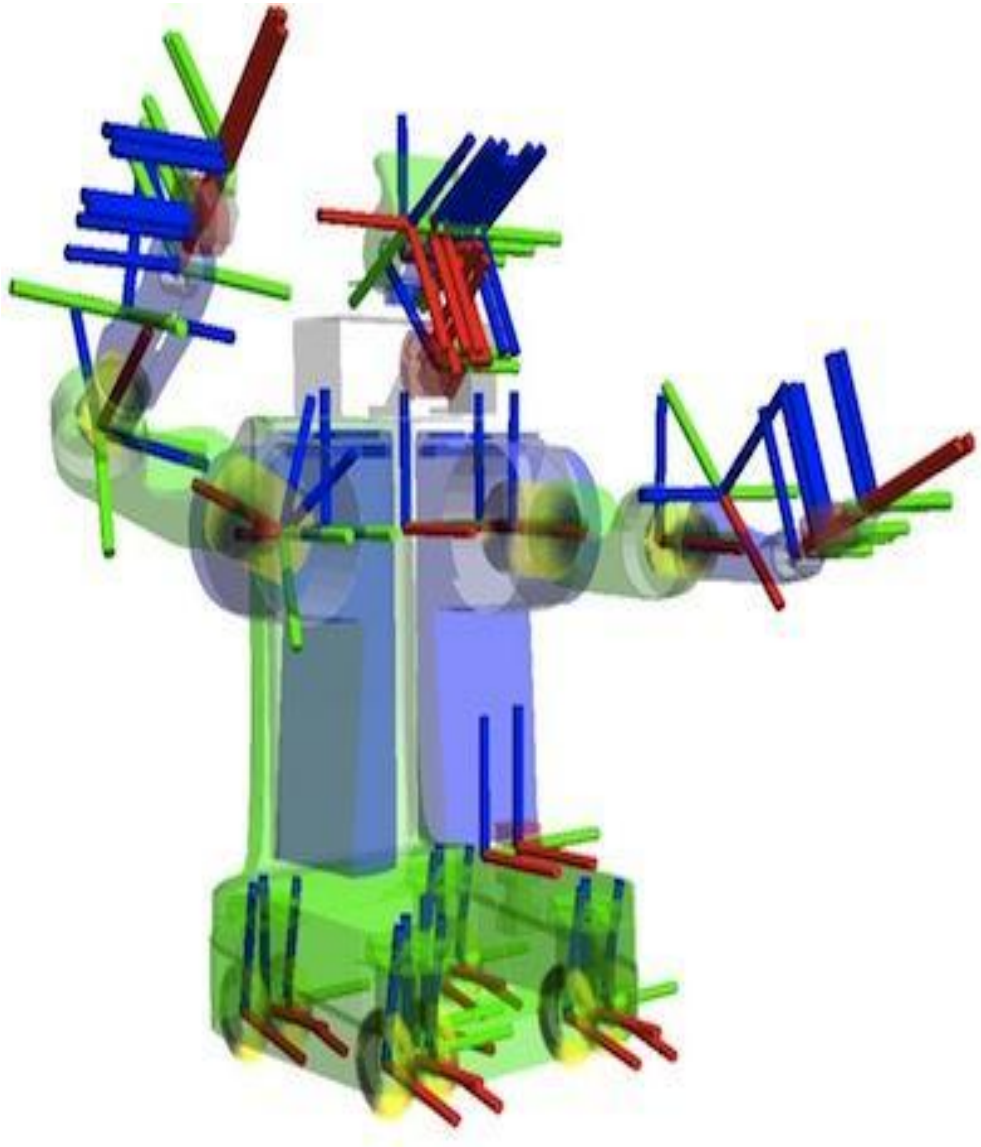
CORE COMPONENTS

Robot Geometry Library

A common challenge in many robotics projects is keeping track of where different parts of the robot are with respect to each other.

Designed with efficiency in mind, the tf library has been used to manage coordinate transform data for robots with more than one hundred degrees of freedom and update rates of hundreds of Hertz.
The tf library allows you to define both static transforms, such as a camera that is fixed to a mobile base, and dynamic transforms, such as a joint in a robot arm. You can transform sensor data between any pair of coordinate frames in the system.

The tf library handles the fact that the producers and consumers of this information may be distributed across the network, and the fact that the information is updated at varying rates.



CORE COMPONENTS

Preemptable Remote Procedure Calls

While topics (anonymous publish/subscribe) and services (remote procedure calls) cover most of the communication use cases in robotics, sometimes you need to initiate a goal-seeking behavior, monitor its progress, be able to preempt it along the way, and receive notification when it is complete. ROS provides actions for this purpose. Actions are like services except they can report progress before returning the final response, and they can be preempted by the caller..

Diagnostics

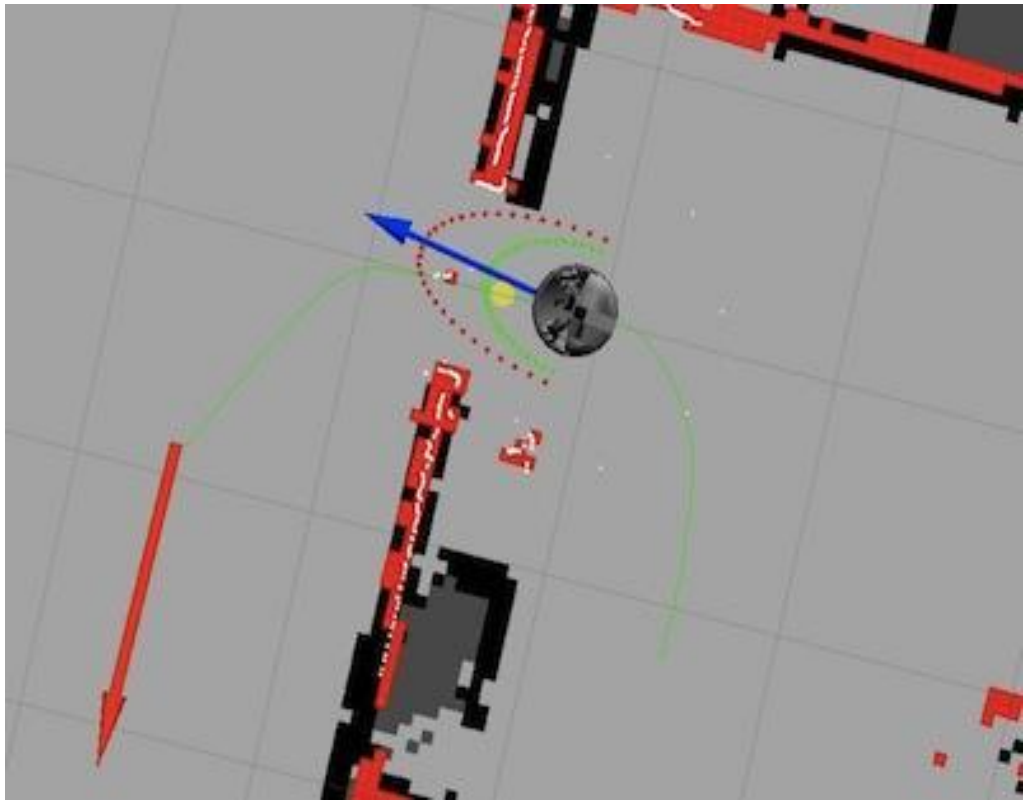
OS provides a standard way to produce, collect, and aggregate diagnostics about your robot so that, at a glance, you can quickly see the state of your robot and determine how to address issues as they arise.

CORE COMPONENTS

Pose Estimation, Localization, and Navigation

ROS also provides some "batteries included" capabilities that help you get started on your robotics project. There are ROS packages that solve basic robotics problems like pose estimation, localization in a map, building a map, and even mobile navigation.

Whether you are an engineer looking to do some rapid research and development, a robotics researcher wanting to get your research done in a timely fashion, or a hobbyist looking to learn more about robotics, these out-of-the-box capabilities will help you do more, with less effort.



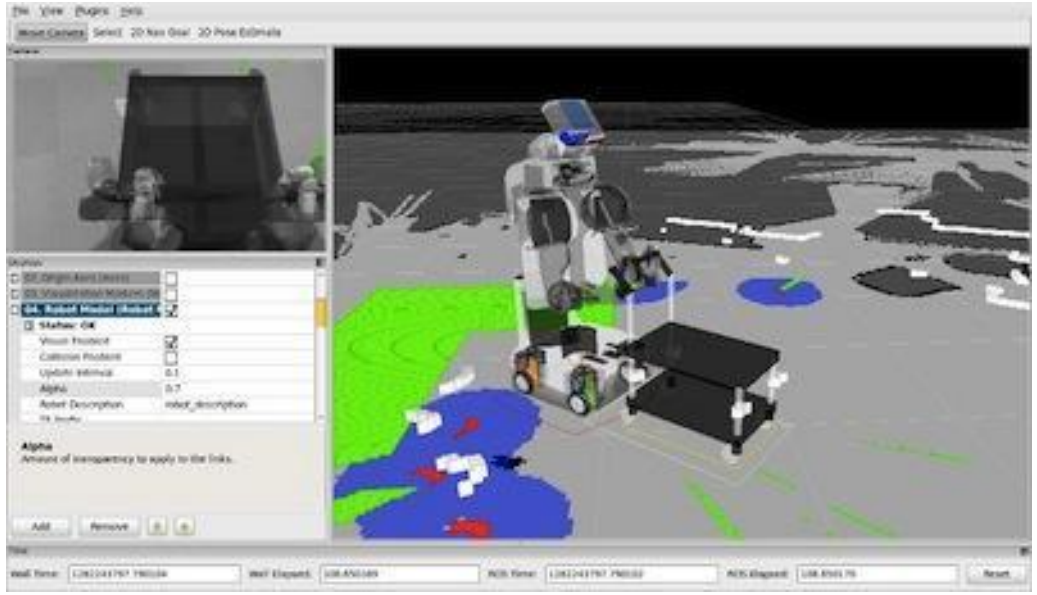
CORE COMPONENTS - TOOLS

Command-Line Tools

ROS can be used 100% without a GUI. All core functionality and introspection tools are accessible via one of more than 45 command line tools. There are commands for launching groups of nodes; introspecting topics, services, and actions; recording and playing back data; and a host of other situations.

Rviz – General purpose 3D visualization

Rviz provides general purpose, three-dimensional visualization of many sensor data types and any URDF-described robot.
rviz can visualize many of the common message types provided in ROS, such as laser scans, three-dimensional point clouds, and camera images.



CORE COMPONENTS - TOOLS

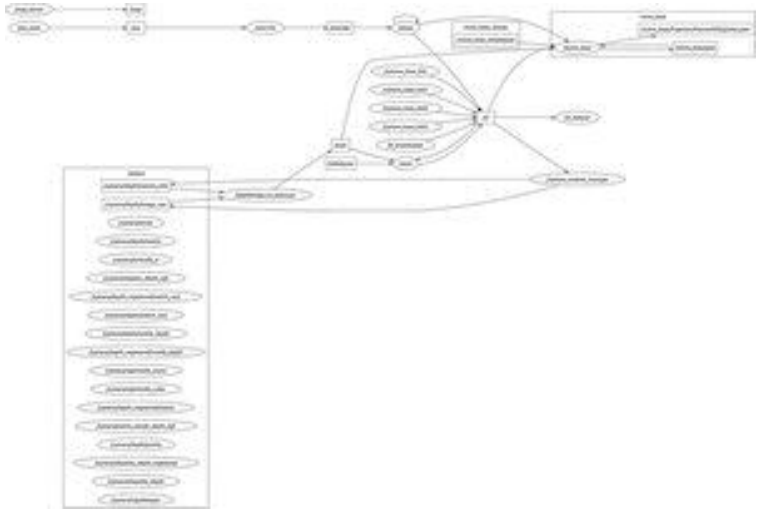
Rqt

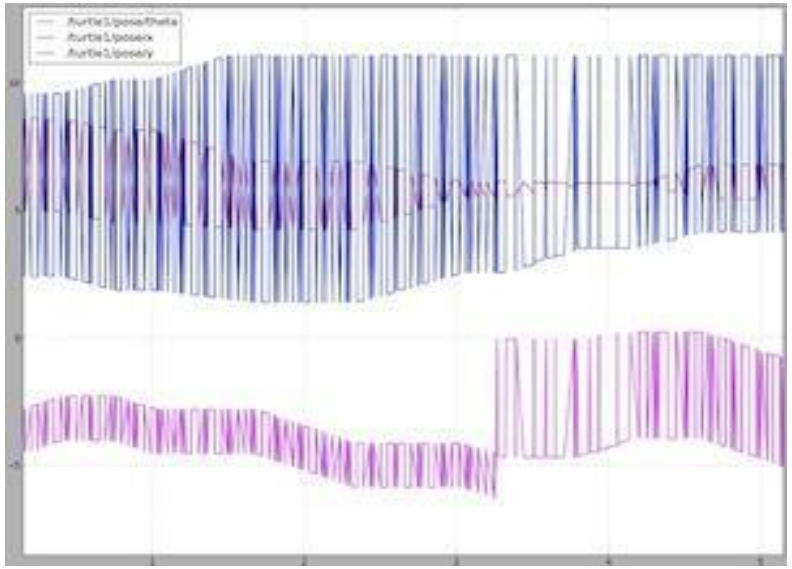
Qt-based framework for developing graphical interfaces for your robot.

- Create custom interfaces by composing and configuring the extensive library of built-in rqt plugins into tabbed, split-screen, and other layouts.
- Introduce new interface components by writing your own rqt plugins .

The ***rqt_graph*** plugin provides introspection and visualization of a live ROS system, showing nodes and the connections between them

The ***rqt_plot*** plugin, allows to monitor encoders, voltages, etc. that varies over time.





CORE COMPONENTS - TOOLS

The ***rqt_topic*** is GUI plugin for displaying debug information about ROS topics including publishers, subscribers, publishing rate, and ROS Messages.

The ***rqt_publisher*** plugin provides a GUI plugin for publishing arbitrary messages with fixed or computed field values.

The ***rqt_bag_plugins*** contains visualization plugins for the rqt_bag ROSGUI plugin. It provide visualizers for messages of different types.

The ***rqt_bag*** is a graphical framework for analyzing bag (log) files.

INTEGRATION WITH OTHER LIBRARIES

Gazebo

Gazebo is a 3D indoor and outdoor multi-robot simulator, complete with dynamic and kinematic physics, and a pluggable physics engine.

- Integration between ROS and Gazebo is provided by a set of Gazebo plugins that support many existing robots and sensors.
- Develop ROS nodes that are compatible with simulation, logged data, and hardware.
- Develop application in simulation and then deploy to the physical robot with little or no changes in your code.

INTEGRATION WITH OTHER LIBRARIES

OpenCV

OpenCV is the premier computer vision library, used in academia and in products around the world. It provides many common computer vision algorithms and utilities.

- ROS provides tight integration with OpenCV, allowing users to easily feed data published by cameras of various types into OpenCV algorithms
- ROS builds on OpenCV to provide libraries such as image_pipeline, which can be used for camera calibration, monocular image processing, stereo image processing, and depth image processing.

INTEGRATION WITH OTHER LIBRARIES

Point Cloud Library (PCL)

The Point Cloud Library (PCL), is a perception library focused on the manipulation and processing of three-dimensional data and depth images.

- PCL provides many point cloud algorithms, including filtering, feature detection, registration, kd-trees, octrees, sample consensus, and more.

INTEGRATION WITH OTHER LIBRARIES

MoveIt

MoveIt is a motion planning library that offers efficient, well-tested implementations of state-of-the-art planning algorithms that have been used on a wide variety of robots, from simple wheeled platforms to walking humanoids.

- MoveIt can be used with any ROS-supported robot.
- Planning data can be visualized with rviz and rqt plugins, and plans can be executed via the ROS control system.

INTEGRATION WITH OTHER LIBRARIES

ROS-Industrial (ROS-I)

ROS-Industrial is an open-source project that extends the advanced capabilities of ROS to manufacturing automation and robotics.

The ROS-Industrial repository includes:

- Interfaces for common industrial manipulators, grippers, sensors, and device networks
- Provides software libraries for automatic 2D/3D sensor calibration, process path/motion planning, applications like Scan-N-Plan
- Developer tools like the Qt Creator ROS Plugin
- Training curriculum that is specific to the needs of manufacturers

ROS-I is supported by an international Consortium of industry and research members

REFERENCES

- Official ROS website (www.ros.org)
- http://wiki.ros.org/
- Robot Operating System Market by Types | ROS Market - 2024 | MarketsandMarkets

NXP PRESENTS AiM ARTIFICIAL INTELLIGENCE in MOBILITY

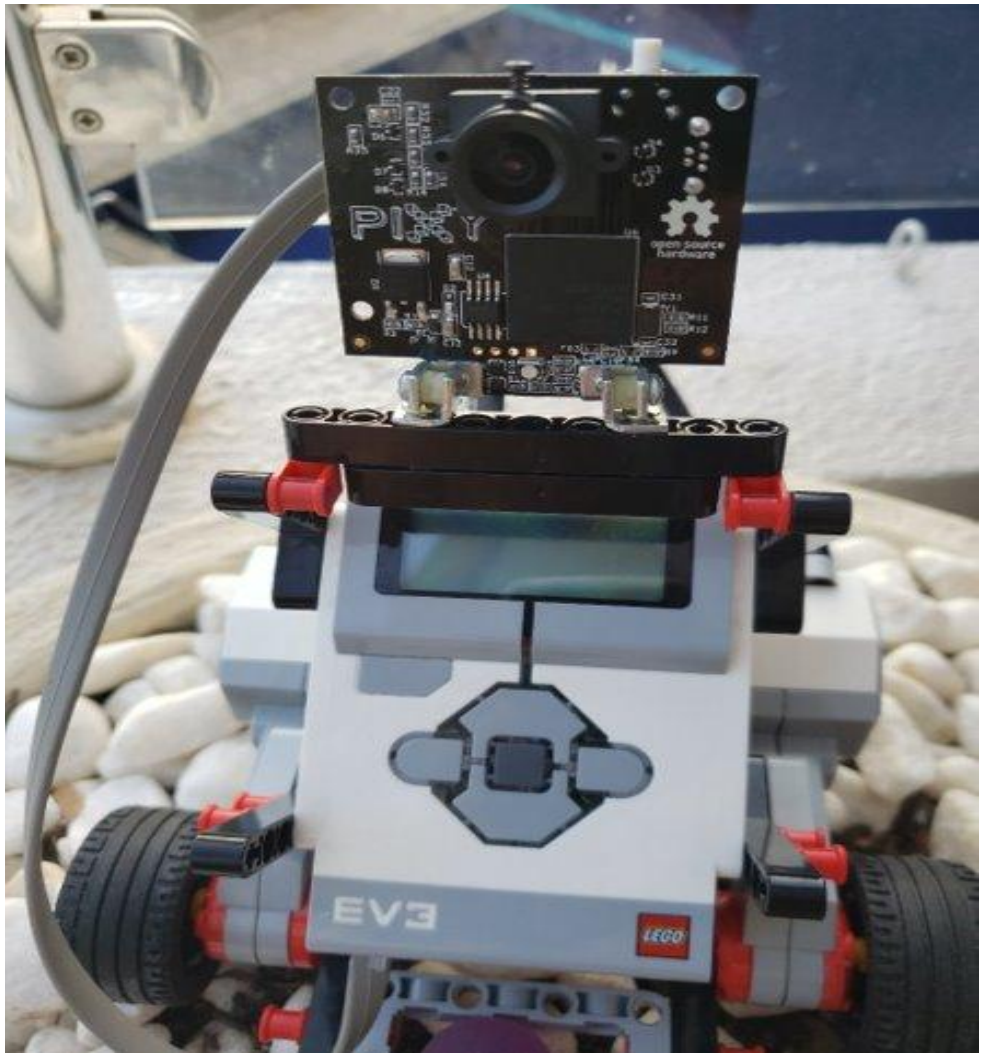
ARTIFICIAL INTELLIGENCE IN MOBILITY

OVERVIEW OF THE NXP GAZEBO STACK

IN COLLABORATION WITH
Time Of Sports®

SUMMARY

This document is designed to help contestants understand the inner-workings of the NXP Gazebo simulation stack. As a contestant, you will only need to focus on a small set of files within this massive workspace. In order to reduce confusion, we have created a detailed outline of all of the files and folders in the stack, and have documented the specific files and folders that contestants will be working with in order to write self-driving algorithms for their simulated NXP Cup car.



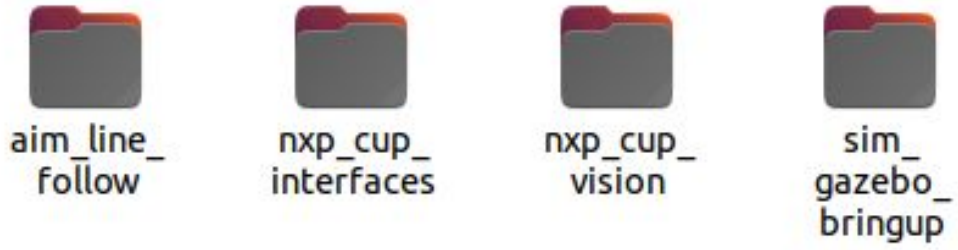
OUTLINE OF THE STACK

The NXP Gazebo simulation stack uses a diverse set of software to enable the simulation of the Cup car. Thankfully, as NXP AIM India contestants, you will only need to use specific portions of the stack to develop your self-driving software. The stack is located in the *~/ros2ws/* folder and is set up as follows:

Folder	Purpose
build	ROS2 specific build folder.
install	ROS2 specific install folder.
log	ROS2 specific log folder.
nxp_gazebo	Scripts and model files for simulation. Contains meshes and SDF files of car cup, camera and other necessary models to be used in simulation worlds
osrf	OSRF 3D model library for creating custom worlds
src	See the next section for an overview

SRC FOLDER CONTENTS

The src folder within the ROS2 workspace contains specific ROS, Camera vision packages as well as the sim_gazebo_bringup package.



Folder	Purpose
aim_line_follow	Contains the sample code for self-driving car. This is the folder that participants are going to work on to design their algorithm
nxp_cup_vision	Contains the OpenCV vision code for simulating Pixy camera to detect lines
nxp_cup_interfaces	Contains the PixyVector message for simulated Pixy Camera line detection algorithm
sim_gazebo_bringup	Contains scripts for booting up the Gazebo simulation as well as setting up the simulation stack.

WRITING SELF-DRIVING CODE

The participants are required to use aim_line_follow folder to write their logic and algorithm of the self-driving car. The content of the folder is as shown:

Inside the aim_line_follow sub-folder, a sample code for the self-driving car has been provided inside ***aim_line_follow.py*** file. Participants are required to edit this file in order to complete the challenge. Participants are allowed to use any resource and practices to achieve their desired results

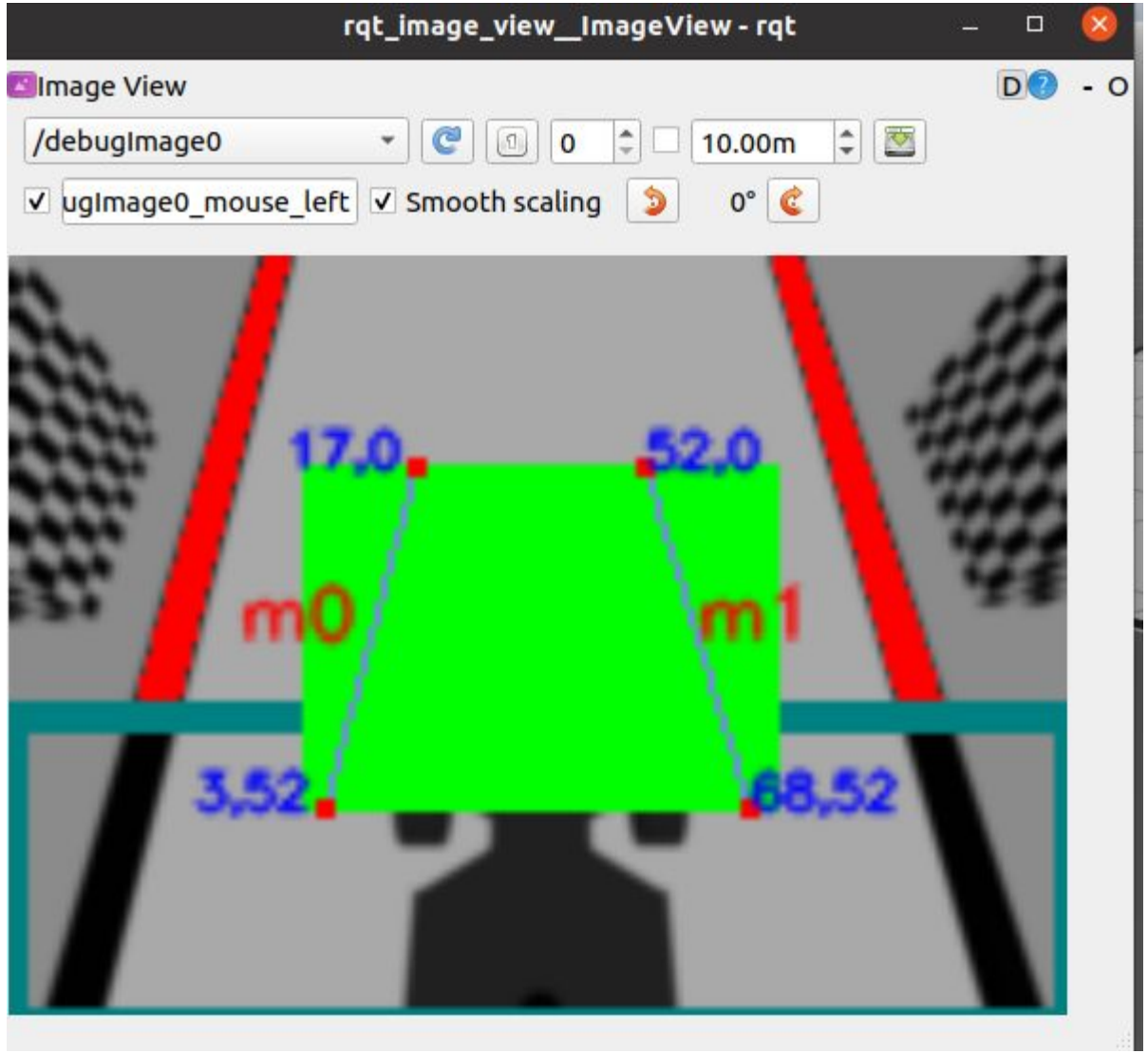
VIEWING THE SIMULATED PIXY CAMERA

Preface

In order to provide a true-to-life simulated environment for NXP AIM India contestants, we have written a simulated Pixy camera module that detects lines and outputs vector data just like the real Pixy camera. The source code for the simulated Pixy camera uses OpenCV to fit vectors to detected lines in simulation.

Guide

The simulation stack will open a new window that shows the debug output of the simulated Pixy camera. Here's what it looks like:



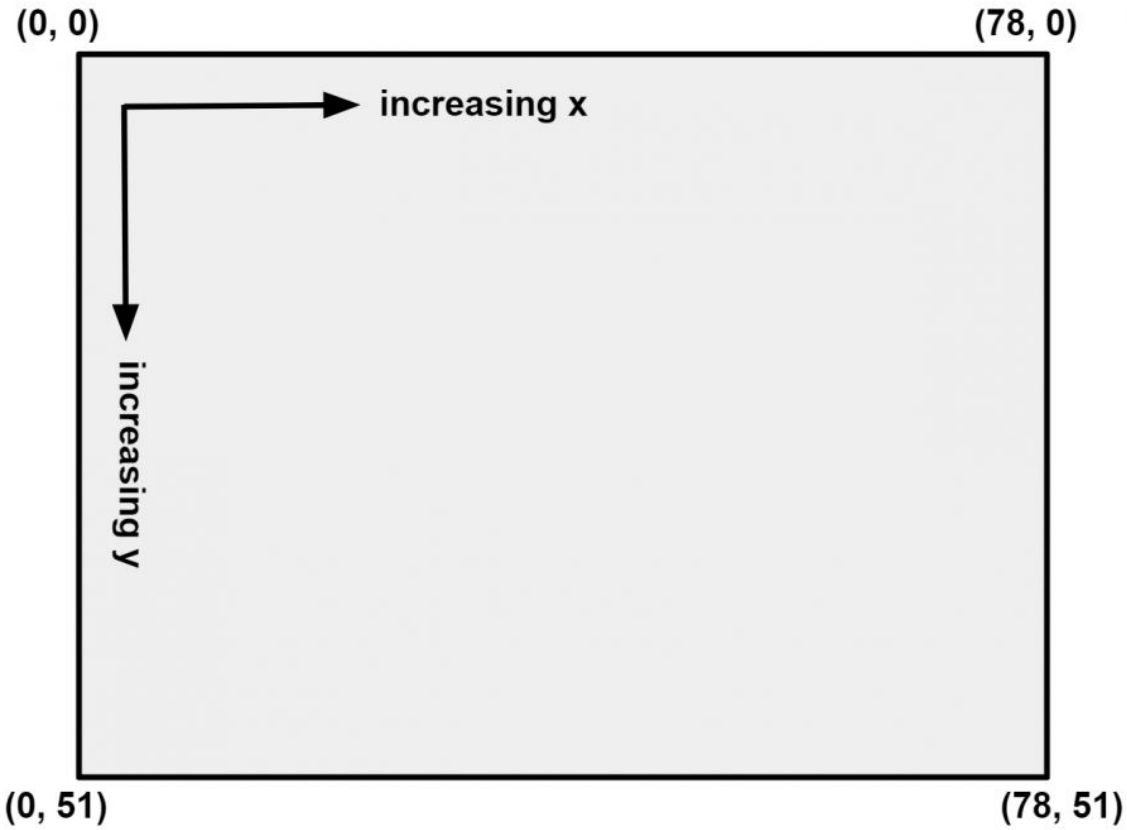
If you do not see the simulated Pixy camera output, use the drop-down at the top left of the window and select /debugImage0.

VIEWING THE SIMULATED PIXY CAMERA

The simulated Pixy camera detects the black lines in the environment and fits lines to them. Then, it will use those lines to create a simulated Pixy camera vector output in the Pixy camera frame space as seen:

As you can see in the simulated pixy camera output, the vector data returned is identical to the vector data that the real Pixy camera sends over I2C (vector head and tail coordinates for each vector). This allows contestants to use the same algorithms that are on their real NXP Cup cars.

The source code for this simulated Pixy camera is located at ***~/ros2ws/src/nxp_cup_vision/nxp_cup_vision/nxp_track_vision.py*** . You are free to edit this code if you see any potential areas of improvement!



SECURE CONNECTIONS
FOR A SMARTER WORLD

Thank you

SECURE CONNECTIONS
FOR A SMARTER WORLD

Thank you