Dbot_ROS_Humble

Release 0.1

Pejman Habibiroudkenar

Feb 19, 2024

CONTENTS

The Dbot, a differential-wheeled robot, is specially designed for autonomous indoor robotics research. It has undergone significant hardware updates. Presently, Dbot features a Velodyne VLP16 LiDAR, a Realsense camera, an Intel Core i7 CPU, and an NVIDIA RTX 2060 GPU. Originally configured for ROS Noetic, Dbot has been recently upgraded to ROS2 Humble. This guide aims to assist students in using and adapting Dbot for their projects. For further details about resources like Dbot, visit the Autonomus Mobility Lab website.

Note: This project is under active development.

CHAPTER

ONE

CONTENTS

1.1 ROS Humble

1.1.1 Why ROS Humble?

ROS is an open-source robotic platform designed for both academic and industrial applications. The latest iteration, ROS 2, builds upon the strengths of ROS 1 and introduces several significant improvements.

1.1.2 Improvements in ROS 2

1. **Improved Security Features** ROS 2, particularly in its Humble Hawksbill version, emphasizes security. It offers tools for secure node communication and access control, essential for sensitive or critical applications.
2. **Enhanced Real-Time Capabilities** Tailored for real-time computing, ROS 2 is crucial for applications requiring precise timing and high reliability. This is a notable enhancement over ROS Noetic's limited real-time support.
3. **Cross-Platform Compatibility** ROS 2 improves compatibility across Linux, macOS, and Windows, facilitating development and deployment in varied environments.
4. **Quality of Service (QoS) Settings** With customizable QoS settings, ROS 2 allows for refined control over node communication, crucial for system reliability and efficiency.
5. **Modernized Architecture and Communication Layer** Using DDS (Data Distribution Service) for its middleware, ROS 2 provides robust, scalable communication, a significant upgrade over ROS Noetic.

1.1.3 ROS 2 for Existing ROS Users

For those familiar with ROS, ROS 2 will feel intuitive yet distinct enough to operate without constant reference to new documentation. One key change in ROS 2 is the elimination of the *roscore* command; the system starts automatically with any ROS 2 command. Like ROS, ROS 2 uses a subscriber and publisher method.

1.1.4 Utilizing ROS Humble

This project uses ROS Humble, supported until 2027. For installation and usage instructions, refer to the ROS Humble documentation. This guide provides essential commands and cheat codes for operating robots with ROS 2. If you are new to ROS, reviewing this documentation is recommended. For an in-depth understanding, consider reading "A Concise Introduction to Robot Programming with ROS 2."

1.1.5 ROS Humble Cheat Sheet

This cheat sheet covers basic commands and operations in ROS Humble.

1.1.6 Basic Commands

```
Initialization:
ros2 run

Starting a node:
ros2 run [package_name] [node_name]

Listing nodes:
ros2 node list
```

1.1.7 Topic Management

```
Listing topics:
ros2 topic list

Publishing to a topic:
ros2 topic pub [topic_name] [msg_type] [args]

Subscribing to a topic:
ros2 topic echo [topic_name]
```

1.1.8 Service Management

```
Listing services:
ros2 service list

Calling a service:
ros2 service call [service_name] [srv_type] [args]
```

1.1.9 Parameter Management

```
Listing parameters:
ros2 param list

Setting a parameter:
ros2 param set [node_name] [param_name] [value]

Getting a parameter:
ros2 param get [node_name] [param_name]
```

1.1.10 Launch Files

```
Running a launch file:
ros2 launch [package_name] [launch_file_name]
```

1.1.11 Debugging and Logging

```
ROS2 logger levels: Debug, Info, Warn, Error, Fatal
Setting logger level:
ros2 logging set_logger_level [logger_name] [level]
```

1.1.12 Building and Compiling

```
Building a workspace:
colcon build

Sourcing the environment:
source install/setup.bash
```

1.1.13 Key ROS Tools

- Rviz for visualization
- Gazebo for simulation

1.1.14 Best Practices

- Regular backups of code
- Use of version control (e.g., Git)

1.1.15 Useful Resources

- Official ROS documentation
- Community forums and Q&A sites

1.2 Motors

The wheel control is divided into three sections: Motor Controller (ZLTECH ZLA8015D), Hub Motors (ZLLG80ASM250 V3.0), and CAN Adapter (Peak-System PCAN-USB). OpenCAN is used to facilitate communication between the computer and the robot. "EXPLAIN CANOPEN, PDO AND SDO COMMUNICATION IN ONE PARAGRAPH". In the motor driver node, there are six files responsible for creating a connection between the computer and robot, each briefly explained as follows:

- ROS2wrapper.py: This file creates publishers (wheel velocity values) and subscribers (cmd_val) in ROS2.
- zlac80115d_canopen.py: A class that defines functions to control motors, such as setting target speed, halt, reading encoder speed, etc.
- od_definitions.py: An object dictionary for factors and units.
- Node_control.py: Conversion functions for a differential drive robot.
- log_connector.py: Connects Python logs to ROS2 logs.
- zlac8015d.eds: An electronic data sheet that zlac80115d_canopen.py reads for creating functions.

1.2.1 ROS2 Motor Topics

Note: The follwing command is implemented in system boot up so you do not need to add it:

```
$ sudo ip link set can0 up type can bitrate 500000
```

You can disable this by removing the "can0-setup.service" at /etc/systemd/system/can0-setup.service

You can now initiate the motor driver node by running:

```
$ ros2 run motor_driver motor_bringup
```

This node creates 1 subscriber, cmd_vel, receiving twist messages, and 2 publishers, left_velocity_rpm and right_velocity_rpm, as float64 messages in rpm.

You can now publish your target speed to cmd_val, and the motor will follow. In a new terminal, you can test it with the following command:

```
$ ros2 topic pub /cmd_vel geometry_msgs/msg/Twist "linear:
  x: 5.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0"
```

The dbot ROS humble package is also equipped with an odometry estimator based on wheel speed. You can run this node:

```
$ ros2 run odometry_estimator odometry_estimator
```

This node publishes the odom topic, where you can access x and y direction pose. A sample code to read positions is provided in the API section.

1.2.2 Joystick Control

The Dbot wheel can be controlled with a joystick. First, ensure the joystick is connected to the Dbot by testing the output of:

```
$ sudo jstest /dev/input/js0
```

If you have received any feedback, run the next command, which creates joy messages in ROS from Linux input:

```
$ ros2 run joy_linux joy_linux_node
```

In a new terminal, run the joycontrol node with:

```
$ ros2 run joycontrol joycontrol
```

This node reads the joy messages and publishes cmd_vel. You can now drive around with the Dbot!

1.3 Velodyne VLP16

The Velodyne VLP16 is connected to the computer via Ethernet, using a fixed IP address of 192.168.2.100 (factory setting value is 192.168.2.100). To access the LiDAR settings, enter 192.168.2.201 (factory setting value is 192.168.2.201) in your web browser. To initiate the Velodyne driver node, use the following command:

```
$ ros2 launch velodyne velodyne-all-nodes-VLP16-launch.py
```

This node publishes Velodyne packets and points. You can visualize these points using RVIZ in ROS2. To start RVIZ, execute:

```
$ rviz2 rviz2 -f velodyne
```

Once RVIZ is running, navigate to the lower left corner and click on the `"add"` option. In the 'Add' window, go to the "by topics" tab and select 'point_clouds'. You will now be able to see the Velodyne points in real-time.

1.3.1 Configuring Velodyne IP

Note: Do NOT change the IP address unless you are not receiving any points.

If the connection between the computer and the Velodyne is compromised, you can reset the IP address by trying the following commands:

```
$ sudo route add 192.168.2.100 enp89s0
$ sudo route add 192.168.2.201 enp89s0
$ sudo route add 192.168.2.255 enp89s0
```

1.4 Realsense Camera

Note: To be Written

1.5 Dbot Launch

You can launch all the nodes at the same time (motor_bring up, odometry_estimater, joycontrol, velodyne) by follwing line.

```
$ ros2 launch dbot dbot_bringup.py
```

1.6 ROS2 on Multiple Machines

Utilizing the DDS (Data Distribution Service) in ROS2, it's feasible to send and receive messages across multiple machines without explicit configuration. This functionality is employed to control a robot, referred to as DBot, and to collect data from it using an external computer.

For detailed information, refer to this guide

Note: DBot operates on Ubuntu 22.04 with ROS2 Humble installed. The external computer should also have ROS2 installed, preferably the same distribution (ROS Humble).

1.6.1 Prerequisites

1. Ensure both machines are connected to the same network.
2. Verify that multicasting is enabled on the network. To check, run the following command on DBot to obtain its IP address:

   ```
   $ hostname -I
   ```

   This command will return an IP address, such as 192.168.1.102(). Then, from the external computer, run:

   ```
   $ ping 192.168.1.102
   ```

   If the ping is successful, communication between the external computer and DBot is established.
3. Confirm that all machines share the same domain ID.
4. To enable joystick control from the external PC, install the 'joy_linux' ROS2 package:

```
$ sudo apt-get install ros-humble-joy-linux
```

1.6.2 SSH Connection

To establish SSH connection make sure that dbot and the laptop are both connected to the same wifi. It is recommended to use wifi router mounted on top of dbot (Tp link M200). The password for the router is 52399565 and Dbot ip is set to 192.168.1.102, Therefore you can make ssh connection by:

```
$ ssh dbot2@192.168.1.102
```

1.6.3 Configuring Joystick Control

If your joystick ID is not 'js0', replace XXX with your joystick ID and run:

```
$ ros2 run joy_linux joy_linux_node --ros-args -p dev:=/dev/input/XXX
```

For instance, for ID 'js1', execute:

```
$ ros2 run joy_linux joy_linux_node --ros-args -p dev:=/dev/input/js1
```

This action sets the parameter and initiates the 'joy_linux' node, allowing DBot to be controlled from the external PC.

Note: With ROS functionalities shared between the two machines, access to all topics, nodes, etc., is available. However, be mindful of potential delays in data sharing, especially when recording messages (e.g., from a Velodyne sensor) on the external PC.

1.6.4 Setting the ROS_DOMAIN_ID

For ROS communication across multiple machines, a common 'ROS_DOMAIN_ID' parameter is essential. By default, this value is 0 and doesn't require explicit setting. To add more machines to the same network or to restrict communication, setting a domain ID is recommended:

```
$ export ROS_DOMAIN_ID=XX
```

Replace XX with any number between 0 and 101. Ensure the same ID is used for both DBot and the external PC.

1.7 Usage

Please use Dbot for your study, and email Pejman at pejman.habibiroudkenar@aalto.fi if you have a finished project with Dbot, so I can add your project to this page.

1.8 API

This section will provide 3 sample codes for your projects namely, simple publsiher, simpler listener, odometry position print.

1.8.1 Hello Dbot Publisher

The Hello Dbot publisher is a simple ROS2 node that publishes "Hello, Dbot" messages. This document provides an overview of its functionality and instructions for use.

Ensure ROS2 is installed on your system. If not, follow the installation instructions on the ROS2 website.

1.8.2 Setup Workspace

Create a new ROS2 workspace and a package for the publisher:

```
mkdir -p ~/ros2_ws/src
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_cmake --node-name hello_dbot_publisher hello_dbot
```

1.8.3 Publisher Node

The publisher node is written in Python and is responsible for publishing "Hello, Dbot" messages.

```
# hello_dbot_publisher.py
import rclpy
from rclpy.node import Node
from std_msgs.msg import String

class HelloDbotPublisher(Node):
    def __init__(self):
        super().__init__('hello_dbot_publisher')
        self.publisher_ = self.create_publisher(String, 'hello_dbot', 10)
        timer_period = 1  # seconds
        self.timer = self.create_timer(timer_period, self.timer_callback)

    def timer_callback(self):
        msg = String()
        msg.data = 'Hello, Dbot'
        self.publisher_.publish(msg)
        self.get_logger().info('Publishing: "%s"' % msg.data)

def main(args=None):
    rclpy.init(args=args)
    hello_dbot_publisher = HelloDbotPublisher()
    rclpy.spin(hello_dbot_publisher)
    hello_dbot_publisher.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

1.8.4 Build and Run

To build and run the publisher, use the following commands:

```
# Build the package
cd ~/ros2_ws
colcon build --packages-select hello_dbot

# Source the setup script
source ~/ros2_ws/install/setup.bash

# Run the Publisher
ros2 run hello_dbot hello_dbot_publisher
```

The publisher will start and publish "Hello, Dbot" messages at a 1-second interval.

1.8.5 Verify

To verify the publisher is working, listen to the topic in another terminal:

```
ros2 topic echo /hello_dbot
```

You should see "Hello, Dbot" messages being printed at regular intervals.

1.8.6 Hello Dbot Subscriber

The Hello Dbot subscriber is a ROS2 node that subscribes to messages on the "hello_dbot" topic. It prints out each "Hello, Dbot" message it receives.

1.8.7 Subscriber Node

The subscriber node is written in Python. It listens to the *hello_dbot* topic and logs each message received.

```
# hello_dbot_subscriber.py
import rclpy
from rclpy.node import Node
from std_msgs.msg import String

class HelloDbotSubscriber(Node):

    def __init__(self):
        super().__init__('hello_dbot_subscriber')
        self.subscription = self.create_subscription(
            String,
            'hello_dbot',
            self.listener_callback,
            10)
        self.subscription  # prevent unused variable warning

    def listener_callback(self, msg):
        self.get_logger().info('Received: "%s"' % msg.data)
```

(continues on next page)

(continued from previous page)

```
def main(args=None):
    rclpy.init(args=args)
    hello_dbot_subscriber = HelloDbotSubscriber()
    rclpy.spin(hello_dbot_subscriber)
    hello_dbot_subscriber.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

1.8.8 Usage

To use this subscriber node:

1. Ensure the ROS2 environment is sourced.
2. Run the subscriber node with:

   ```
   ros2 run [package_name] hello_dbot_subscriber
   ```

   Replace *[package_name]* with the name of your ROS2 package.
3. The subscriber will start and print out "Hello, Dbot" messages as they are received from the publisher.

This node can be used in conjunction with the Hello Dbot publisher to demonstrate basic ROS2 pub/sub functionality.

1.8.9 Odom Position Subscriber

The Odom Position Subscriber is a ROS2 node that subscribes to the *odom* (odometry) topic and prints the x and y positions. This is typically used in robotics to track the position of a robot.

1.8.10 Subscriber Node

The subscriber node is written in Python. It listens to the *odom* topic, which is of the type *nav_msgs/msg/Odometry*, and logs the x and y position coordinates.

```
# odom_position_subscriber.py
import rclpy
from rclpy.node import Node
from nav_msgs.msg import Odometry

class OdomPositionSubscriber(Node):

    def __init__(self):
        super().__init__('odom_position_subscriber')
        self.subscription = self.create_subscription(
            Odometry,
            'odom',
            self.odom_callback,
            10)
```

(continues on next page)

(continued from previous page)

```
        self.subscription  # prevent unused variable warning

    def odom_callback(self, msg):
        position = msg.pose.pose.position
        self.get_logger().info(f'Position: x={position.x}, y={position.y}')

def main(args=None):
    rclpy.init(args=args)
    odom_position_subscriber = OdomPositionSubscriber()
    rclpy.spin(odom_position_subscriber)
    odom_position_subscriber.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

1.8.11 Usage

To use this subscriber node:

1. Ensure the ROS2 environment is sourced.
2. Place the script in the *src* directory of your ROS2 package.
3. Build the package using *colcon build*.
4. Run the subscriber node with:

   ```
   ros2 run [package_name] odom_position_subscriber
   ```

   Replace *[package_name]* with the name of your ROS2 package.
5. The subscriber will start and print out the x and y positions as they are received from the *odom* topic.

This node is useful for tracking the real-time position of a robot in a 2D space, especially in a simulation or testing environment.