Robotics project Part 1

Luca Massini [844049]
Roland Reylander [868917]

May 29th 2019

1 Files inside the archive

The project folder contains the following files:

- `configs.cfg` used for dynamically set:
  - Differential Drive Kinematics or Ackerman model
  - initial x and y coordinates
- `odometry.launch` to launch the main node
- `custom_msg.msg` to publish the odometry with a custom message and `floatStamped.msg` used for retrieve data from the bag
- `odometry.cpp` where the code of the main node is written
- `CMakeLists.txt` and `package.xml` to specify what will be used.

```
project
├── cfg
│   └── configs.cfg
├── launch
│   └── odometry.launch
├── msg
│   ├── custom_msg.msg
│   └── floatStamped.msg
├── src
│   └── odometry.cpp
├── CMakeLists.txt
└── package.xml
```

2 Description of how to start/use the nodes

First place the folder inside your catkin workspace and run in the terminal:

```
$ catkin_make
```

There are two ways to start the node.
Using rosrun:

```
$ roscore
$ rosrun project odometry
```

Using roslaunch:

```
$ roslaunch project odometry.launch
```

After this, it is ready to read the bag file running:

```
$ rosbag play bag_name.bag
```

2.1 Read the published odometry

The Node works as subscriber of the bag as well as publisher of the odometry. To view the odometry published with the custom message run:

```
$ rostopic echo /odom_custom
```

To view the odometry published with nav_msgs/Odometry message run:

```
$ rostopic echo /odom_std
```

To see the published tf run:

```
$ rostopic echo tf
```

The custom message publishes `x`, `y` and the angle `theta`. In addition there is a parameter that tells if the model used is differential drive or the Ackerman.
Both the `tf` and the `odom_std` topic make use of the `nav_msgs/Odometry`, so the pose has been published using `x` and `y` coordinates, whereas for the rotation the quaternion system has been used.
The `tf tree` has two nodes, one called `car_odometry` which represents the origin and a child node called `car` which represents the pose of the car.

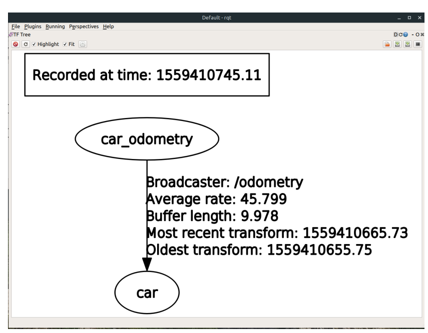

Figure 1: tf tree structure

2.2 Dynamically configuration

To dynamically reconfigure the parameteres it is possible to use the following commands:

```
$ rosrun dynamic_reconfigure dynparam set /odometry x VALUE
```

to set the `x` coordinate to a specified VALUE

```
$ rosrun dynamic_reconfigure dynparam set /odometry y VALUE
```

to set the `y` coordinate to a specified VALUE

```
$ rosrun dynamic_reconfigure dynparam set /odometry diff_acker CONDITION
```

set the `CONDITION` to `true` for Differential Drive Kinematics or `false` for Ackerman model.

3 How it works

The initial idea was to read directly from the bag file but later we decided to subscribe to the bag's topic and retrieve the data like this. So the node works as subscriber of the bag's topic. The next step was to read the data, at the beginning we tried to read the `speed_R`, `speed_L` and `steer` topics, but we had problems with synchronizing the three topics, so the alternative was to subscribe to the `speedR_stamped`, `speedL_stamped` and `steer_stmped` topics. For doing this we needed a custom message that included the timestamp in the Header and a `float` variable.
After this we wrote functions to calculate the fomulae for both models and to publish from the same node, so every time the bag published the speed of the wheels and the steer the node could publish the pose from the same node. There are three publishers: one for the `odom_custom` message, one for the `odom_std` message and one for the `tf`.
Finally we added functions to dynamically reconfigure the parameters.

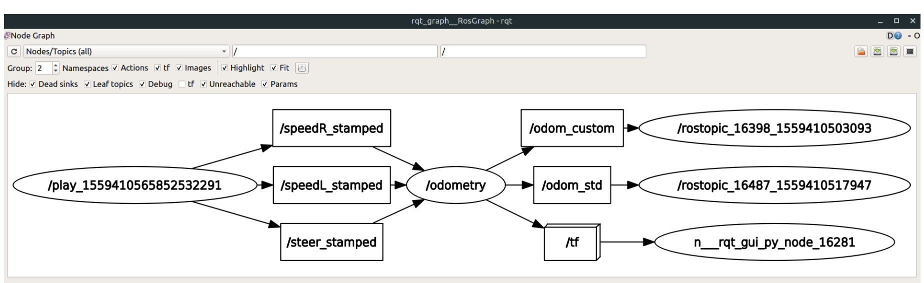

Figure 2: Subscribed and published topics

4 Tests

To check if the odometry was correctly calculated, we made use of `rviz`. We could observe that in both Differential Drive and Ackerman model the car was moving in circles.
Using the play and stop mode of `rosbag play` the pose is not precise. This is due to the time that is not taken from the bag's timestamp, but the node uses the ROS function `ros::Time::now()` instead. So when the playing bag is temporary stopped, the delta of the time increases and the formula does not work anymore.
Other tools we used were `rqt_graph` and `rqt_plot` to check the topics and the pose of the car. Moreover we used the `rqt_reconfigure` for gui based reconfiguration of the parameters. Lastly we used the `ROS_INFO` function that allowed us to debug the formulae and see in real time if every part of the code was working correctly.