Robotics Project AA 2016/2017

Emanuele Ghelfi
Emiliano Gagliardi

May 31, 2017

1 Package structure

The ros-turtlebot-pilot package contains:

- The src folder containing the following file:
  - controller.cpp: this is the first version of the controller that drives the turtlebot to the goal.
  - improved_controller.cpp: this is the second and improved version of the controller that drives the turtlebot to the goal.
  - joypad_simulator.cpp: a node that receive input from keyboard and publish a sensor_msgs/Joy message on the topic used by the joy_node node.
  - joy_cmd.cpp: a node that receives messages from the topic "joy" and publishes them on the "cmd_joy" topic.
  - goal_reader.cpp: a node that receives the goal from the keyboard and publishes it on the topic "goal".
  - mux.cpp: a node that manages the switch between automatic mode and pilot mode.
  - improved_odometry.cpp: a simple nodes that implements various types of odometry using simple integration (Euler method), Runge-Kutta integration and Exact integration.
- The launch folder containing the pilot.launch file for starting the simulation with the improved controller and the pilot_old.launch for starting the simulation with the old controller.
- The srv folder containing the SetMode.srv file for the specification of the request of the server and the GetMode.srv.

2 Node Description

Joypad Simulator

The joypad simulator simulates a joypad and provides support for the two main axes and for different buttons.

With the arrows it's only possible to give one velocity (linear or angular) at a time.

In order to give two velocities at a time it's possible to use the buttons written in the instruction in the terminal.

The node initializes the terminal with some utility functions of ncurses in order to get the raw input from the keyboard without waiting the enter command.

When receiving a button it waits before sending it as message, in this way it can send multiple buttons in the same message.

Joy Cmd

The joy cmd node receives messages from the topic joy and writes them to the topic cmd_joy.

It also implements some logic in order to give more smoothness to the drive.

For the purpose of smoothness it increases the velocity contained in the message to send by a little increment in the desired direction until the maximum (or zero) velocity has been reached.

The drive experience has improved a lot with this little "trick".

Controller

The controller node implements the control logic needed to reach the goal. This node subscribes to the topic "goal" in order to get the position of the goal and to the topic "odom" specified in the launch file (improved_odom, or odom). This nodes interacts also with the PID controller by using the correct topics.

When it receives a goal it changes state from IDLE to INITIAL ROTATION. In this state the controller acts on the angular velocity until the robot has reached the correct orientation with respect to the goal. This is done by setting the orientation as setpoint in the topic of the pid and received the control effort based on the PID parameters.

When the orientation in near enough to the orientation of the goal the controller changes state to MOVING FORWARD. In this state the controller acts on the linear velocity until the robot is near enough the goal. This is done by setting the desired position as setpoint in the topic of the pid.

When the robot is near enough to the goal the controller changes state to FINAL ROTATION, this state is pretty similar to the INITIAL ROTATION state but here the robot should reach the final orientation specified in the goal.

Using this method there can be problems if the goal is far from the robot. If the robot starts moving forward with a small gap with respect to the correct orientation it can happen that the direction has to be correct during the navigation.

This controller implements a "lazy" trajectory correction in the sense that it changes the trajectory when it understand that the robot is going away from the goal. This is done by monitoring the distance from the goal at each simulation step. When the distance starts to increase it means that the robot should change trajectory and the controller changes state to INITIAL ROTATION to change the direction.

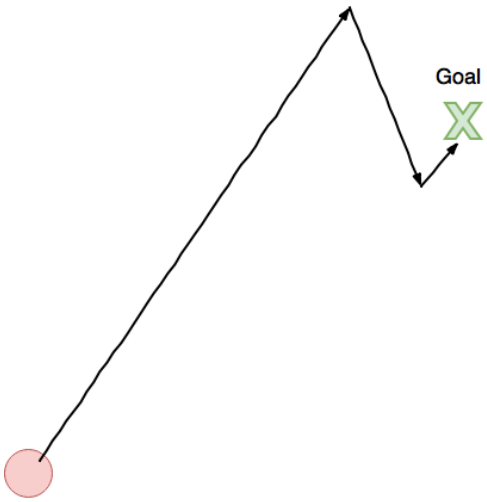

Figure 1: Example of trajectory of the turtlebot using the Controller node. The red circle is the robot. The trajectory is not smooth, the gap between the first direction and the goal is due to a little imprecision in the initial rotation.

Improved Controller

This is the second version of the controller. This is similar to the previous controller except that it controls the two linear and angular velocities simultaneously in the MOVING FORWARD phase.

The INITIAL ROTATION and FINAL ROTATION are exactly equals to the state of the previous controller.

This controller implements an "early" trajectory correction in the sense that it keeps changing the direction also during the MOVING FORWARD phase. This ensures a smoother and faster trajectory with respect to the previous one. This type of trajectory is reached using two different PID acting simultaneously on the direction and on the orientation.

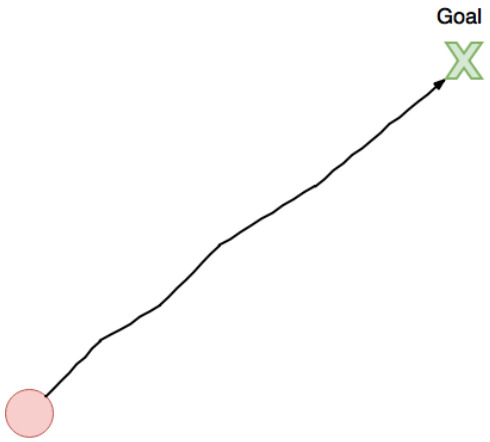

Figure 2: Example of trajectory of the turtlebot using the Improved Controller node. The red circle is the robot. The trajectory is very smooth and it's continuously corrected by the pid acting on the angular velocity

Improved Odometry

This node implements the computation of different types of Odometry (Euler, Runge-Kutta and Exact). This node reads from the launch file the required odometry to use and publishes it on the topic improved_odom. The other parameter of this node is data_num that is the number of measurements to average in order to get the current velocities. If this number is large the averaged velocity will have the expected value of the real velocity but the this node will be slow. If data_num is low the averaged velocity could have more variance (due to sensor noise) but the rate of this node will be very high.

This node accumulates measurements in the odometry callback, it uses the averaged linear and angular velocities to compute the new position using the integration method specified in the launch file.

3 Installation Instruction

Dependencies

In order to run the simulation these packages are required:

- libncurses-5-dev: for terminal utility
- ros-kinetic-pid: for the ros implementation of the PID controller
- ros-kinetic-turtlebot-simulator and ros-kinetic-turtlebot-gazebo for the turtlebot model and worlds.

In order to install these packages run these commands:

```
sudo apt-get install libncurses5-dev
sudo apt-get install ros-kinetic-pid
sudo apt-get install ros-kinetic-turtlebot-gazebo
sudo apt-get install ros-kinetic-turtlebot-simulator
```

Compilation

In order to compile the package copy it into the catkin workspace (here is assumed as catkin_ws) inside the src folder.

Then source the setup.bash file inside devel folder and the compile with catkin.

```
cd catkin_ws
source devel/setup.bash
catkin_make
```

Run

If everything is ok and nothing has failed run the simulation using:

```
roslaunch ros_turtlebot_pilot pilot.launch
```

In this way the simulation starts with the improved controller. In order to start the simulation with the first version of the controller:

```
roslaunch ros_turtlebot_pilot pilot_old.launch
```

4 Execution

Parameter Settings

In the launch file the following parameters can be changed:

- Odometry used by the controller:

  ```
  <node name="improved_controller" pkg="ros_turtlebot_pilot" type=
      "improved_controller" output="screen">
         <param name = "odom" value="improved_odom"/> <!--possible
            values: odom, improved_odom"-->
  </node>
  ```

  Or for the first version of the controller:

  ```
  <node name="controller" pkg="ros_turtlebot_pilot" type="
      controller" output="screen">
           <param name = "odom" value="improved_odom"/> <!--
              possible values: odom, improved_odom"-->
  </node>
  ```

  The parameter "odom" can have values "odom" or "improved_odom" and the controller reads the odometry published in the specified topic.

- Type of odometry used by the Improved odometry node:

```
<node name="improved_odometry" pkg="ros_turtlebot_pilot" type="
    improved_odometry" output="screen">
              <param name="odom_method" value="exact"/><!--
                 possible values: euler, runge-kutta, exact-->
              <param name="data_num" value="1"/> <!--number of
                 data read that are then averaged-->
</node>
```

The odom_method represents the type of odometry and the possible values are "euler", "runge-kutta" and "exact".

Autonomous Mode

The turtlebot robot starts with autonomous mode. In this mode the turtlebot reaches the goal provided in the topic "/goal" in an autonomous way.

In order to give a goal (position) to the robot start the goal_reader node with the following command:

```
rosrun ros_turlebot_pilot goal_reader
```

Type a position following the instructions on the terminal and watch the turtlebot reaching the goal. During the movement it prints on the terminal its state and when the goal is reached "Goal is reached successfully" will be printed on the terminal.

It's possible to type another goal while the turtlebot is moving and the robot changes goal dynamically.

Pilot Mode

In the pilot mode the pilot is driven by the keyboard.

In order to start the pilot mode start the joypad_simulator node with the following command:

```
rosrun ros_turtlebot_pilot joypad_simulator
```

The button "r" switches the mode. If the mode is autonomous is then changed to pilot.

Now it's possible to drive the turtlebot using the arrows or the buttons. Notice that if the mode changes from autonomous to pilot when the turtlebot is trying to reach a goal, it will try to reach the goal when the mode switches back to auto.

5 Reference

- Project repository: https://github.com/EmilianoGagliardiEmanueleGhelfi/ros_turtlebot_pilot