Solution Assesment Task: Four-Wheeled Robot

Frederik Zwilling

March 21, 2017

Assessment: You have been assigned the task of developing a simple controller for a planar four-wheeled mobile robot that would enable it to autonomously navigate around its environment. In order to accomplish the forenamed objective, the following constituent sub-tasks have been assigned:

1 Configuration Kinematic Model

Task: Derive the system's configuration kinematic model assuming that the robot's centre of mass coincides with its axis of symmetry.

According to [Campion and Chung, 2008] the *configuration kinematic model (CKM)* is a formula

$$\dot{q} = S(q)\eta \ ,$$

which describes the derivative of the configuration coordinates

$$q = \begin{pmatrix} X \\ Y \\ \theta \end{pmatrix} \ .$$

X and Y are the coordinates of the *center of mass (COM)* of the robot in a fixed inertial frame with orientation θ. η is the control input. However, the derived formulas in [Campion and Chung, 2008] are not applicable for our four-wheeled robot because the derivation is based on the *nonslip condition* stating that the velocity of each wheel is parallel to the wheel plane. Because our robot only has fixed wheels, it uses *skid-steering* like a tank where wheels are sliding sideways while turning in place.

A kinematic model for skid-steering is given by [Kozłowski and Pazderski, 2004] and can be simplified by using the fact that the COM coincides with the axis of the wheel symmetry[1]. This yields

$$\dot{q} = \begin{pmatrix} cos\theta & 0 \\ sin\theta & 0 \\ 0 & 1 \end{pmatrix} \eta \ , \text{ with } \eta = \begin{pmatrix} v_x \\ \omega \end{pmatrix} = r \begin{pmatrix} \frac{\omega_L + \omega_R}{2} \\ \frac{-\omega_L + \omega_R}{2c} \end{pmatrix} \ , \text{ where}$$

v_x is the relative forward velocity of the robot, ω is the rotational velocity, r is the wheel radius, ω_L and ω_R are the angular velocities of the left and right wheels, and $2c$ is the width of the robot.

[1] x_{ICR} and v_y can be set to 0.

2 Motor Input Commands

Task: Formulate an expression for the wheel motor input commands, based on the configuration kinematic model that was previously computed, assuming that the robot is controlled in velocity mode.

The expression for wheel motor input commands (as angular velocities) can be obtained by transforming the equation for η from Section 1:

$$\begin{pmatrix}\omega_L \\ \omega_R\end{pmatrix} = \frac{1}{r}\begin{pmatrix}1 & c \\ 1 & -c\end{pmatrix}\begin{pmatrix}v_x \\ \omega\end{pmatrix}$$

This assumes that the motors are mounted or controlled in a way so that a positive angular velocity of a wheel moves the robot forward.

3 Velocity Controller PWM function

Task: Implement the velocity controller by means of a PWM function.

To implement the velocity controller, we have to supply each wheel motor with the appropriate average voltage to achieve the intended angular velocity. As approximation we assume that the velocity linearly depends on the voltage. We implement the PWM function on a Micro-controller by using interrupts to ensure accurate timing. Listing 1 shows the PWM function. It takes a velocity between -1.0 and 1.0 as input and allows negative velocities by using a PWM with the voltages $5V$ and $-5V$.

4 ROS-based Mapping System

Task: Provide a snippet of code enabling the use of a simple ROS-based mapping system.

For this and the following two tasks, I created a small ROS package and used it on a robot simulated in Gazebo as shown in Figure 1. Here I only provide small code snippets. You can find the full package on `https://github.com/zwilling/4-wheeled-ros-robot` with some instructions how to run it. I also provided a demo video here: `https://youtu.be/NZHOTxBaLyU`
When you try it out yourself, don't expect too much from the simulation because the skid-steering is inaccurately simulated. However this could be fixed by tuning physics parameters in the simulation or by simulating the robot motion on a higher abstraction level.

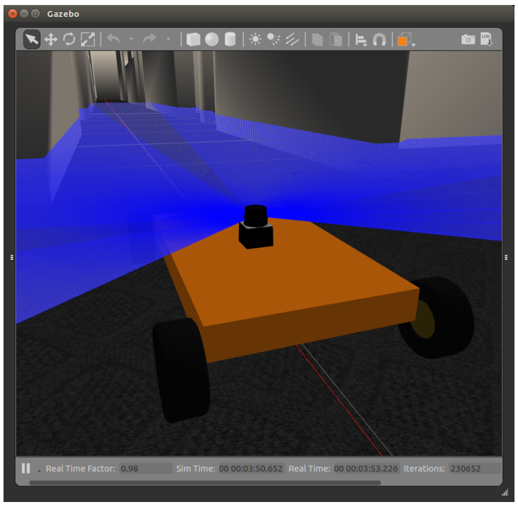
As ROS-based mapping system, I utilize `gmapping`. It builds a map and localizes the robot in it. It depends on laser data generated by a simulated Hokuyo sensor and on the odometry of the robot, which is provided by the `gazebo_ros_skid_steer_drive` plugin for Gazebo. This plugin also applies velocity commands to the simulated wheels. `gmapping` can be started with the ros launch file shown in Listing 2

5 Navigation

Task: Demonstrate via ROS libraries and C++ code, how the outputs of the mapping system could be broadcasted and subsequently utilised to navigate the robot to desired

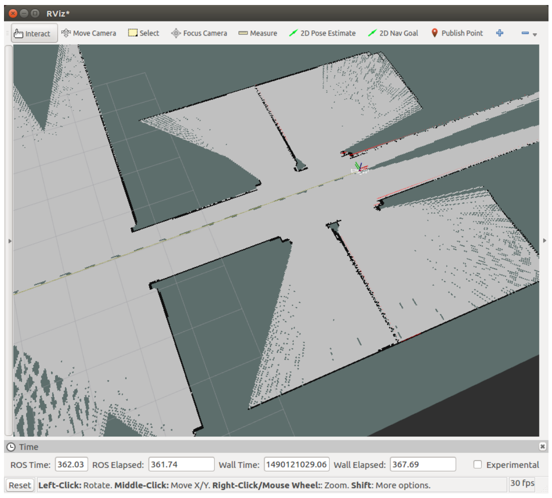
```c
unsigned char pwm_period = 255;
unsigned char pwm_duty;
unsigned char pwm_flag = 0;

void set_velocity(float vel)
{
  // cap velocity
  if(vel > 1.0 || vel < -1.0)
    vel = vel / abs(vel);
  // transform (possibly negative) velocity to duty time
  pwm_duty = (unsigned char) ((vel+1.0) * 128.0);
  //start timer
  EA = 1; // enable interrupts
  ET0 = 1; // enable timer0 interrupts
  TR0 = 1; // start timer
  TMOD = 0; // timer limit
}

// Timer 0 Interrupt service routine
void timer0() interrupt 1
{
  if (!pwm_flag) {
    // set high level, e.g., 5V
    PWMPIN = 1;
    // set timer
    TH0 = pwm_duty;
  } else {
    // set low level, e.g., -5V
    PWMPIN = 0;
    // set timer
    TH0 = pwm_period - pwm_duty;
  }
  // change pwm_flag and clear interrupt flag
  pwm_flag = 1-pwm_flag;
  TF0 = 0;
}
```

Listing 1: PWM function for the velocity controller



Real Time Factor: 0.98 Sim Time: 00 00:03:50.652 Real Time: 00 00:03:53.226 Iterations: 230652

(a) Robot with laser sensor in Gazebo



ROS Time: 362.03 ROS Elapsed: 361.74 Wall Time: 1490121029.06 Wall Elapsed: 367.69

Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.

(b) Already mapped environment in Rviz

Figure 1: Simulation of four-wheeled robot with SLAM and navigation

```
<launch>
  <node pkg="tf" type="static_transform_publisher" name="hokuyo_link_broadcaster"
        args="0 0 0.08 0 0 0 1 base_link base_laser 100" />
  <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" output="screen">
    <rosparam>
      odom_frame: odom
      map_update_interval: 1.0
      maxUrange: 10.0
      base_frame: base_link
      [...]
    </rosparam>
    <remap from="/scan" to="/qbot/laser/scan" />
  </node>
</launch>
```

Listing 2: Launch file for `gmapping`

```
<node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
  <rosparam file="params/costmap_common.yaml" command="load" ns="global_costmap" />
  <rosparam file="params/costmap_common.yaml" command="load" ns="local_costmap" />
  <rosparam file="params/local_costmap.yaml" command="load" />
  <rosparam file="params/global_costmap.yaml" command="load" />
  <rosparam file="params/base_local_planner.yaml" command="load" />
  <remap from="/cmd_vel" to="/qbot/cmd_vel" />
</node>
```

Listing 3: `move_base` node for the navigation launch file

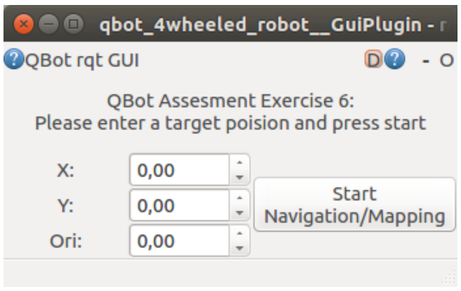
locations in Cartesian/world coordinates.

The `gmapping` package used in the previous section periodically publishes the transform between the robot and the map frame and the updated map on the `/map` topic. This can be used in conjunction with the laser data and the velocity controller to navigate the robot around with collision avoidance. A ROS package providing this is `move_base`. It generates costmaps from map and laser data and handles motion planning to navigate to a goal specified on the `/move_base_simple/goal` topic. The node that needs to be added to the launch file is shown in Listing 3. There are multiple configuration files specifying collision avoidance parameters and locomotion capabilities of the robot.

6 ROS-based Graphical User Interface

Task: Append a button/checkbox to a custom ROS-based graphical user interface, whose toggling/checking activates the previously-designed controller (C++ code).

This task is solved with a C++ `rqt` plugin, which is shown in Figure 2. This GUI plugin can be added to a `rqt` window and sends a `geometry_msgs::PoseStamped` message on the `/move_base_simple/goal` topic to start the navigation. The target coordinates and orientation are read from the corresponding fields in the GUI. Listing 4 shows how this message is sent in the callback function of the button being pressed.


qbot_4wheeled_robot__GuiPlugin - r

QBot rqt GUI

QBot Assesment Exercise 6:
Please enter a target poision and press start

X: 0,00
Y: 0,00
Ori: 0,00

Figure 2: `rqt` plugin to send navigation commands

References

[Campion and Chung, 2008] Campion, G. and Chung, W. (2008). Wheeled robots. In *Springer Handbook of Robotics*, pages 391–410. Springer.

[Kozłowski and Pazderski, 2004] Kozłowski, K. and Pazderski, D. (2004). Modeling and control of a 4-wheel skid-steering mobile robot. *Int. J. Appl. Math. Comput. Sci*, 14(4):477–496.

```
void GuiPlugin::on_start_button_push()
{
  //button was pushed
  //build navigation goal message from values in SpinBoxes
  geometry_msgs::PoseStamped msg;
  msg.pose.position.x = ui_.doubleSpinBox_x->value();
  msg.pose.position.y = ui_.doubleSpinBox_y->value();
  //get Quaternion from yaw to write it into the msg
  tf::Quaternion q;
  q.setRPY(0.0 , 0.0, ui_.doubleSpinBox_ori->value());
  tf::quaternionTFToMsg(q, msg.pose.orientation);
  msg.header.frame_id = "map";
  msg.header.stamp = ros::Time::now();

  //send
  navigation_pub_.publish(msg);
}
```

Listing 4: `rqt` function starting the navigation