AIMS Mobile Robotics 2020: Introductory ROS task

March 2020

Implement the code for this task in the aims_teleop package.

1 Launching the Simulator

To launch the simulator, you can use either of the following commands:

- roslaunch aims_jackal_sim jackal_sim_empty.launch
- roslaunch aims_jackal_sim jackal_sim.launch

The latter spawns a simulation of the environment we will test the robot in. If you need to use the joystick, add 'joystick:=true' to the end of the command. For example: 'roslaunch aims_jackal_sim jackal_sim.launch joystick:=true'

2 Obstacle detector

Create a node which subscribes to the laser (topic: 'front/scan'). This node should publish a boolean message to another topic (eg. 'obstacle_detected') to describe when there is, and isn't an obstacle in front of the robot.

To understand the format for the laser messages, use 'rostopic info /front/scan' to see information about the laser topic. To understand the message format, you can use 'rosmsg show LaserScan' or google the documentation for this message type online.

Optional: It may be useful for debugging purposes to publish a message to visualise the objects detected.

3 Text to speech

Create either an action server or service which receives a string and plays a text to speech readout of that string through the speakers of the laptop. We recommend using the 'pyttsx' package to do this, for which examples can be found online. The Jackal does not have a speaker, so the node for your action server/service should always be run on the laptop.

4 Simple safety controller

Modify the safety_controller.py file in aims_teleop to implement a safety controller which prevents the robot from hitting obstacles. The safety controller should only be activated if the 'x' button on the PS4 controller is pressed (a "dead man's" switch).

If an obstacle is detected and the 'x' button on the PS4 controller is pressed, this node should stop the robot from moving regardless of the velocity command given by the PS4 controller. The commands from the PS4 controller are published to 'bluetooth_teleop/cmd_vel'. To override these values, you may publish zero velocity to the topic 'joy_teleop/cmd_vel' which is given higher priority. Use the skeleton provided in safety_controller.py so that the velocity is only overridden if the 'x' button is pressed.

If the velocity is overidden by the safety controller, the text to speech on the laptop should periodically say an appropriate message.

Test your code in simulation first, and then test it on the robot.

5 Improved Safety Controller

Modify your safety controller, so that instead of publishing zero velocity, the controller publishes a velocity which moves the robot towards the direction given by the controller, but avoids obstacles.

The resulting controller should have the following properties:

- For a small obstacle (such as a person or pole) the robot should move around the obstacle, even if the PS4 controller is directing the robot towards that obstacle.
- For a large obstacle such as a wall, the controller should stop the robot hitting the wall.
- For a gap like a door, the robot should still be able to pass through the door.

5.1 Artificial Potential Fields

One approach to implementing the safety controller is to use artificial potential fields. In this approach, the 'goal' has an attractive force, and obstacles have repulsive forces. By summing the attractive and repulsive forces, we can compute an appropriate direction for the robot to move. For more details and justification of the equations below see "Introduction to Autonomous Mobile Robots" by Roland Siegwart.

A formula for the attractive force is:

$$F_{att} = k_{att} \cdot \vec{r}_{goal} \tag{1}$$

where $\vec{r}_{goal}$ is a vector from the robot towards the goal. The repulsive force should grow quickly as we move close to obstacles. A possible formula for the repulsive force is:

$$F_{rep} = \begin{cases} -k_{rep} \cdot \left(\frac{1}{|\vec{r}_{ob}|} - \frac{1}{\rho_0} \right) \cdot \frac{\vec{r}_{ob}}{|\vec{r}_{ob}|^3} & \text{if } \vec{r}_{ob} < \rho_0 \\ 0 & \text{otherwise.} \end{cases}$$

where $\vec{r}_{ob}$ is the vector to an obstacle, and ρ_0 is a threshold distance at which we start considering an obstacle.

Note 1: in our case, the goal is represented by the input vector given by the PS4 controller. To calculate a sensible vector, it might be useful to look at the messages published to 'bluetooth_teleop/joy'.

Note 2: the potential field calculation gives a vector for the direction the robot should move. You will need to use this vector to compute appropriate linear and angular velocity commands.