CS-8803 Special Topic: Machine Learning for Robotics

Final Project: Demining the 3D World

Support: cedric.pradalier@georgiatech-metz.fr
Office hours: 10:00 to 18:00.

Setup

This project use the 3D scene: `/cs-share/pradalier/scenes/lakeExampleArm2.ttt`

Copy the package

`/cs-share/pradalier/cs8804_mlrobotics_students/src/vrep_vsv_driver` into your workspace and run catkin_make.

Note that the sensor placement on the vehicle we're controlling this time is a suggestion. You're welcome to edit the scene and displace the sensors if you think another localisation would be better. If you add sensors, you will have to modify the LUA script controlling the simulation. Don't hesitate to ask for support.

The robot we're controlling in this project is quite complex. It has the kinematic of a car (called Ackermann Steering), plus a 4 degrees of freedom robotic arm, equipped at the end with an orientable mine detector. To simplify your task, the inverse kinematics is already solved for you:

- The truck can be controlled with a twist input: this twist will be generated at the middle of the rear axle, and the truck will adjust its wheel speed and steering to achieve it. Only linear.x and angular.z are considered.
- The arm tip can be controlled with a twist command as well. Only linear.x, linear.y and linear.z are considered, and they correspond respectively to the longitudinal velocity of the arm (not so useful), the lateral velocity (away from the truck) and vertical velocity.
- The tool orientation can be controlled with a single Float32 topic defining its angle with respect to the vertical (-pi/2 keeps it parallel to the ground).

Check vrep_vsv_driver/nodes/teleop_geom.py for an example of use.

Try controlling the robot with the joystick after launching the vsv_geom.launch file from the vrep_vsv_driver. The left thumbpad controls the vehicle displacement, the right thumpad controls the arm tip lateral and vertical speed. The left/right buttons on the left cross control the forward motion of the arm. The red button parks the tool inside the vehicle, the green button deploy it for operation, and the blue one starts issuing twist commands.

The robot is equipped with four sensors:

- A camera located on the top of the cabin and looking down. The image is in /vrep/.../visionSensor
- A kinect located just in front of the tool and attache to it so it always looks down when the tool is parallel to the ground. This outputs an image in /vrep/.../visionSensor and a point-cloud in /vrep/.../depthSensor.
- A 2D-laser scanner outputing a smaller but more precise laser scanner, mounted on the cabin

and looking down slightly forward. This can be found in /vrep/hokuyoSensor. This generate the red points on the ground when the simulation is running.

- A mine proximity detector located inside the tool (yellow part) and outputting a single number in /vrep/metalDetector, returning a 0.0 when a mine is far away and 1.0 when it is close.

Project Objective 1

We know that any mine would have been installed at the interface between the flat gray squares and the brown earth, so we need to make sure that the sensor stays close to it while we're driving. To this end, we need to control the tool to stay above the interface. Adjust the shore_follower package to use the kinect image and teach the robot the correct behaviour using a deep CNN on 32x32 images.

In this objective, you should control only the arm tip. For the demonstration, the simplest is probably to either use 2 joysticks or one joystick with one person controlling the arm velocity based on the images and the other one controlling the truck around the dry lake. There is also a keyboard teleoperation that might be useful.

Once you've found a high peak of the metal detector, you can publish it using a ROS marker (similar to plane indicator we published from the linear regression).

You're free to chose your approach and to build your experiment to validate it.

Project Objective 2

We want to make sure the mine sensor stays as close as possible to the ground, but we cannot let it collide with obstacles.

Using data from the kinect, and generating vertical twist commands, add this behavior to the arm tip.

At the same time, record the metal detector output and use it to detect the position of the mines. Once you've found one, you can publish a marker (similar to the cylinder detector) to visualize it.

An ideal solution should use all the measurements related to a mine to precisely localise it.

Project Objective 3

Add a controller for the truck that will keep a target distance to the floor/dirt interface while driving around the area.

If you want to do it using tools from control theory, assuming you drive at constant speed and measuring a lateral error y, with an orientation error θ, a simple control such as:

- twist.linear.x = v
- twist.angular.z = k_1 y + k_2 θ

would work reasonably well but requires a solution to compute y and theta.

A trivial solution would be to have the arm follow autonomously the shore and control the truck steering to make sure the arm tip stays at a constant distance from the arm shoulder. This would work but would not be very good at anticipating turns.

A more complex solution is to use the image captured by the camera on the vehicle cabin. Should you

want to use this, there is package that handle projecting the camera image to the ground plane so as to associate a metric position with every pixel. Contact your instructor for details.

A real ML solution would be to train another CNN to achieve this task by demonstration, in this case, this is another adaptation of the shore_follower.

You're free to chose your approach and to build your experiment to validate it.

Deliverables

Submit the deliverables by email by Friday December 9th, 10:00 to cedric.pradalier@georgiatech-metz.fr (don't forget to mention you group name). The deliverables list is as follows:

- A video of your system in action (you can use VLC to record your desktop as a video stream).
- A demonstration during the exam week, during the normal class hours (exact time TBC). This demonstration should intend to “sell” the class the performance of your system. It should last approximately 10min and demonstrate your system autonomously demining the test environment. The demonstration may use small slide show if you need it better sell your system performance.