Sensor-based control lab

Labs 3: Multi-sensor control with constraints

1 Content of this lab

The goal is this lab is to control a mobile robot equipped with a pan-tilt camera (see Fig. 1).

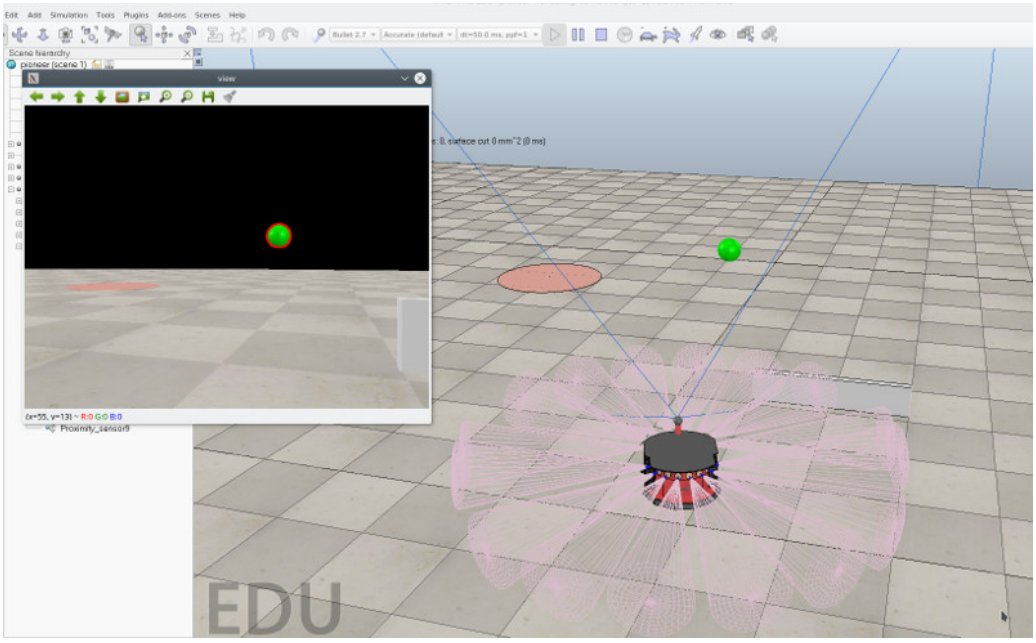

Figure 1: V-REP simulator with the mobile robot, image from the onboard camera and US sensors.

The robot is simulated with V-REP[1]. In the simulation are placed a green sphere and a target. The goal of the robot is to move to the target position while maintaining visibility on the sphere. The robot control will be performed in C++ and will rely on:

- The ROS framework to handle communication between the simulator and the control program.
- The ViSP library[2] to manipulate vectors and matrices and to perform linear algebra.

The actual classes that are used are detailed in Appendix A.

[1]Virtual robot experimentation platform, http://www.coppeliarobotics.com/
[2]Visual Servoing Platform, http://visp.inria.fr

1.1 Environment setup

To use this lab (and for some other ROS labs) you need to initialize the ROS environment in order to have access to the ViSP library. To do so, download the following file and execute it:

```
wget http://www.irccyn.ec-nantes.fr/~kermorga/files/ros_user_setup.sh
sh ros_user_setup.sh
```

In order to use an IDE like Qt Creator, you should open a new terminal and launch the IDE from the command line.

This lab is available on GitHub as a ROS package called `ecn_sensorbased`. In order to download it, you should first go in the `ros/src` directory. The package can then be downloaded through git:

```
git clone https://github.com/oKermorgant/ecn_sensorbased.git
```

1.2 Structure of the `ecn_sensorbased` ROS package

The package has the classical ROS structure:

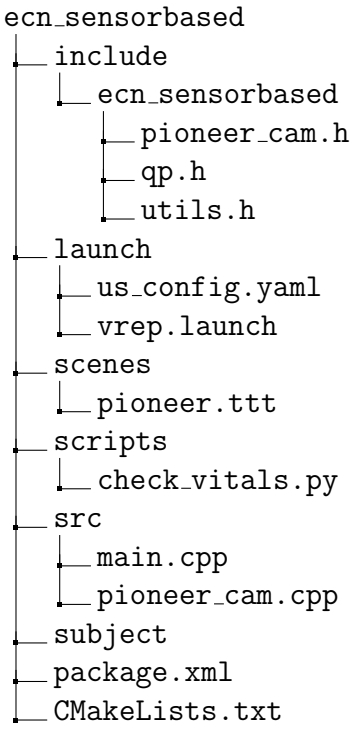

Figure 2: Files used by the package

The only file to be modified is `main.cpp`, which is the main file for the C++ program. The simulation can be launched with: `roslaunch ecn_sensorbased vrep.launch`. It will be run and stopped automatically from the main control.

When both the simulation and the control program run, the ROS graph looks like this:

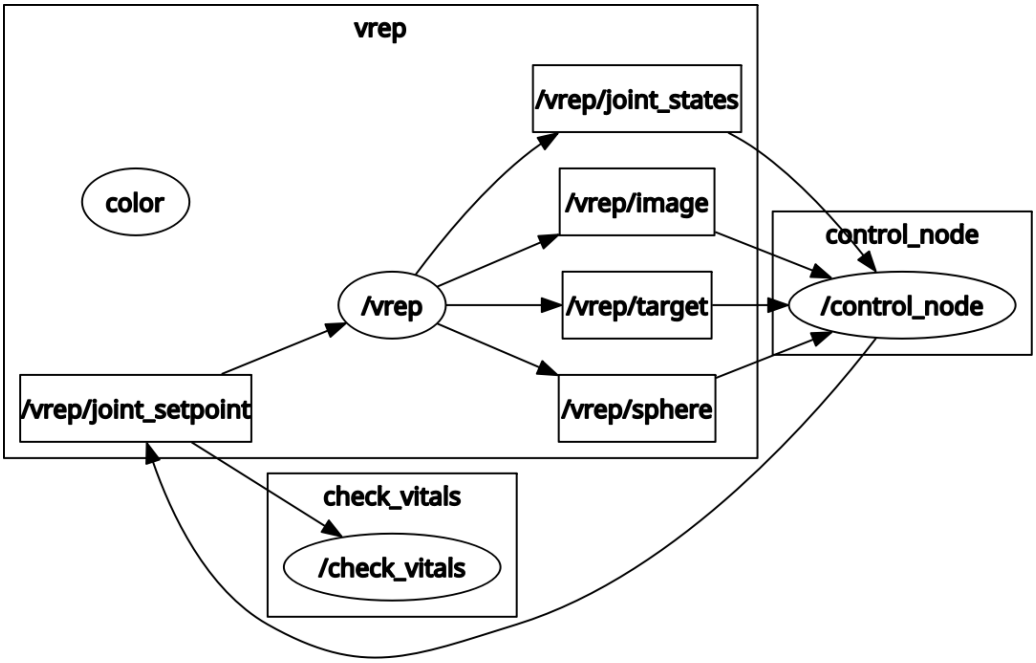

Figure 3: ROS graph showing nodes (ellipses) and communication topics (rectangles)

We can see that the simulator sends the joint states, camera image, target and sphere positions to the control node. In return, the control node sends the joint setpoints to the simulator. They are velocity setpoints for the wheels and the pan-tilt camera joints.
The `check_vitals` is a script that checks the control node is running and stops the simulation otherwise.

1.3 The PioneerCam robot

The simulated robot is a Pioneer P3dx. A pan-tilt camera is placed at the front of the robot. The robot is controlled in joint velocity with $\dot{\mathbf{q}} = (v, \omega, \dot{q}_p, \dot{q}_t)$, where (v, ω) are the linear and angular velocity of the mobile base considered as a unicycle, and (q_p, q_t) are the pan and tilt joint angles. In the initial state of the control node, the visibility is not taken into account and the robot follows a very simple control law:

$$\begin{cases} v &= \lambda_v(x - d) \\ \omega &= \lambda_\omega \text{atan2}(y, x) \end{cases} \tag{1}$$

where (x, y) is the coordinate of the target in the robot frame, d is the final distance to the target (set to 0.1 m), and $(\lambda_v, \lambda_\omega)$ are the control gains.

You should verify that this control runs fine, even if it raises several problems:

- The robot wheels have limited velocities, which is not taken into account with the current control law
- The sphere visibility is not ensured

2 Expected work

The goal is to modify the control law in order to take into account the maximum velocity of the wheels, and the visibility constraint.

1. The wheel kinematic model can be written as:

$$\begin{cases} v &= \frac{r}{2}(\omega_l + \omega_r) \\ \omega &= \frac{r}{2b}(\omega_r - \omega_l) \end{cases} \tag{2}$$

   where r is the wheel radius, b is the distance between the wheels and (ω_l, ω_r) are the left and right wheel velocities. The velocity limit applies to ω_l and ω_r, not on (v, ω).

2. The visibility constraint can be taken into account from the current position of the sphere and its Jacobian (see Appendix A.2 for how to get them).

3. Finally, we would like to have a motion that follows the basic one from (1). Do to so, a constraint on the ratio between v and ω should be added so that the resulting robot motion is proportional to the desired one.

A Main classes and tools

A.1 ViSP classes

This library includes many tools for linear algebra, especially for 3D transformations. The documentation is found here: `http://visp-doc.inria.fr/doxygen/visp-daily/classes.html`.
The main classes from ViSP (at least in this lab) are:

- `vpMatrix` represents a classical matrix, can then be transposed, inversed (or pseudo-inversed), multiplied with a vector, etc.
- `vpColVector` is a column vector with classical mathematical properties.

A.2 The PioneerCam class

The PioneerCam class hides all the ROS communication aspects in order to have a higher-level access. As this is not a lab on robot modeling, all Jacobians are already available - even if it could be a nice exercice to compute the camera Jacobian.

The main methods of the `PioneerCam` class are:

- `setVelocity(vpColVector)`: sends a $(v, \omega, \dot{q}_p, \dot{q}_t)$ velocity to the robot, wheels velocity limits will be ensured
- `getImagePoint`: gives the current position of the sphere in the image