



SATURNE advanced use and maintenance

Following predefined GPS points autonomously and additional sensors functionalities

Using a laptop, connect to **Robot_4_roues** Wi-Fi network, with **Robot4roues** key and **192.168.1.185/24** IP address.

The embedded computer (hostname: NUC-4ROUES) can be accessed using **ssh** saturne@192.168.1.200 command and **Robot4roues** password.

Then:

```
cd Robot_4_roues
```

Update waypoints (rectangle area to scan and number of rails) in **regul/src/mission.cpp** (**read()**) and ensure:

```
roboteq
```

```
serial
```

```
light
```

```
nmea_navsat_driver
```

```
regul
```

```
sbg_ros_driver
```

```
velodyne_pointcloud (optional)
```

```
velodyne_reader (optional)
```

```
rosbridge_server (optional)
```

ROS packages are on the robot (e.g. installed via **apt** when available, or as source in the workspace, see **Developer information** section below). Then:

```
source devel/setup.bash
```

```
export ROS_MASTER_URI=http://192.168.1.200:11311
```

```
export ROS_IP=192.168.1.200
```

```
roslaunch regul launcher.launch
```

On the laptop, optionally (to check):

```
cd Robot_4_roues/
```

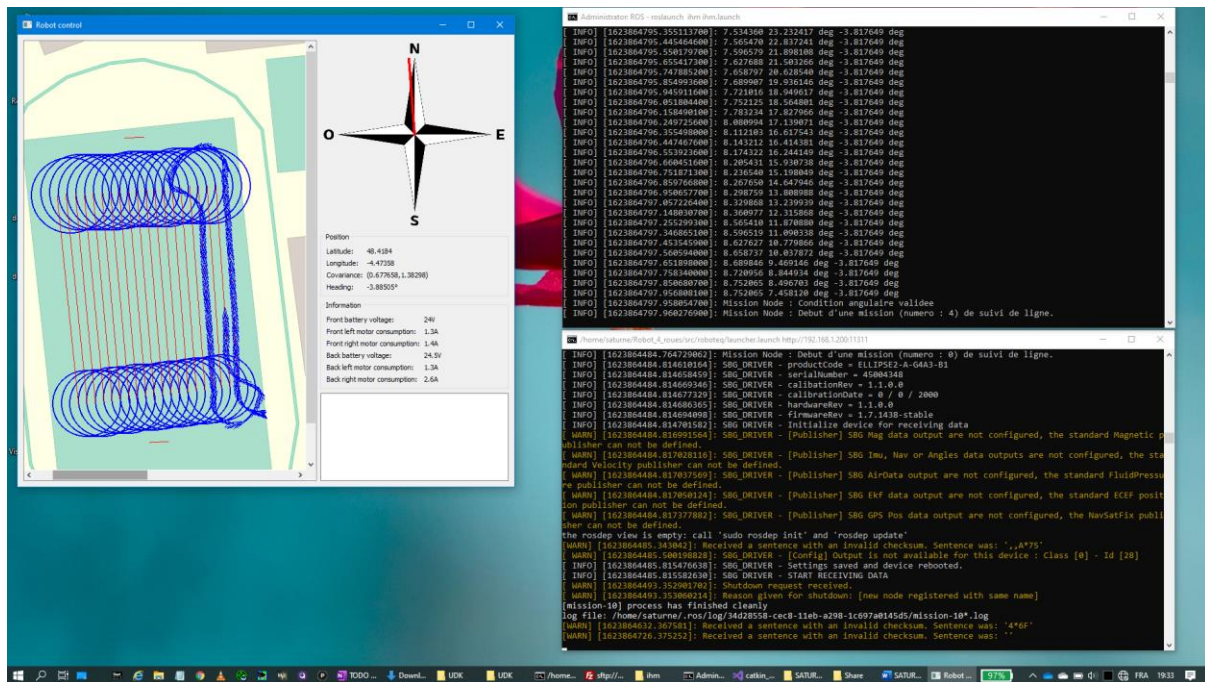
```
catkin_make
```

```
source devel/setup.bash
```

```
export ROS_MASTER_URI=http://192.168.1.200:11311
```

```
export ROS_IP=192.168.1.185
```

```
roslaunch ihm ihm.launch
```



The Velodyne configuration page can be accessed on <http://192.168.1.201>. It is physically connected to an Ethernet interface of the embedded computer (via an Ethernet switch) and should be configured to send data only to the embedded computer since the bandwidth would be too limited through Wi-Fi and a broadcast would also slow down the whole network due to its large data.

To test the obstacle detection, try `roslaunch velodyne_reader launcher.launch` on the embedded computer (the robot should stop and lights should turn on when an obstacle is too close).

To give access to raw sensors and actuators data to **another computer without ROS**, on the embedded computer (to check):

```
cd ~/Robot_4_roues
source devel/setup.bash
export ROS_MASTER_URI=http://192.168.1.200:11311
export ROS_IP=192.168.1.200
roslaunch regul raw.launch
```

Then it should be possible to use the Python package [roslibpy](#) on the other computer to e.g. control the motors through `/cmd_vel` ROS topic, and get sensors data through `/imu/data`, `/fix`, `/velodyne_points`, etc. (see [other/roslibpy_sample.py](#) and [Developer information](#) section below, especially the **Low-level nodes** subsection). Note however that the Velodyne data might be too large to be processed remotely (please comment all the lines mentioning velodyne in `raw.launch` if it is causing issues).

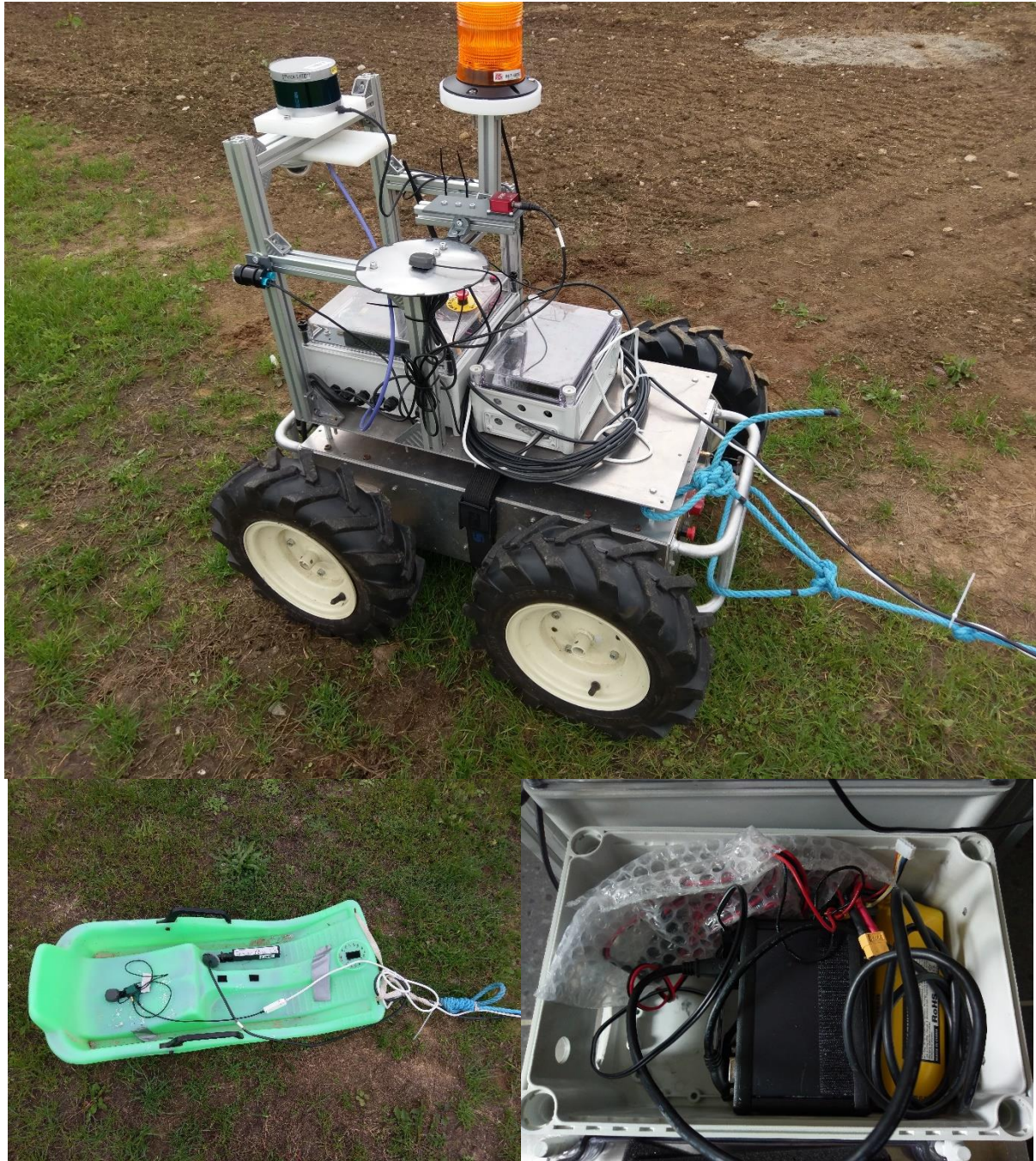
The camera is on <http://192.168.1.209> (user: **root**, password: **root**), the Wi-Fi AP on <http://192.168.1.202> (user: **ubnt**, password: **Robot4roues**).

To cleanly shutdown the computer:
`sudo shutdown now`

MagMap

Warning: always ensure the **magnetometer** is kept as **far away** as possible from **any potentially magnetic hardware**.

The embedded MagMap computer and the magnetometer interface box should be powered through 12 V voltage converters using any **Li-Po 3S, 4S or 6S battery** of sufficient capacity. They should be installed on the rear of the robot, while the magnetometer and the GPS should be installed on the towed sled using a **cable length of 8 m** (this is the length configured in the waypoints controller so that the sled follows the correct path).



The embedded MagMap computer (hostname: NUC-12) can be accessed using **RDP (mstsc /admin** command on Windows, on non-default port **443**) or **NoMachine** (on default port 4000) on the **Robot_4_roues** Wi-Fi network (see previous section), with **192.168.1.212** IP address (user: **Administrator**, password: **Workstation**).

Ensure the GPS is connected in USB, as well as the magnetometer interface box.

Start **config_map.vspe** (creates 2 bridged virtual serial ports COM15 and COM16 running at a baudrate of 921600).

Launch **FGM3D** app, ensure it is configured to output data on COM15 with a baudrate of 921600, click **Connect** (this might need to be retried several times until it succeeds) and then **Start** and **Live streaming\Start**.

Start Ubuntu 20.04 virtual machine (user: **robin**, pass: **Robot4roues**). On the top-right of VMware Player, ensure there is an indication about COM16 and ublox GNSS receiver connected to the VM (the GPS should appear as `/dev/ttyACM0` and will be opened by **simple_gnss_node** (or `nmea_navsat_driver`), which assumes that the GPS has been configured to send \$GPGGA NMEA sentences).

Then:

```
sudo chmod a+rw /dev/ttyS1
```

```
sudo chmod a+rw /dev/ttyACM0
```

```
cd ~/workspace_GNSS
```

```
source devel/setup.bash
```

```
roslaunch sensys launcher-sensys.launch
```

A **CSV** file should appear in `~/ros`. To check data, run in another tab:

```
rostopic echo /data
```

Optionally, in another tab:

```
rosbag record /data
```

Then the robot can be manually controlled to map the area, or follow predefined waypoints as in the previous section.

Finally, the magnetic map can be displayed by running **analyze_map.py** on the **CSV** file.

Developer information

See https://github.com/ENSTABretagneRobotics/SATURNE_simulation, especially **saturne** branch.

High-level nodes

ihm ROS package:

GUI displaying information from **regul** package.

regul ROS package:

mission node: specific mission of lines, waypoints, circles.

obj node: line, waypoint, circle following, communication with **ihm** node using custom messages.

ctrl node: yaw and translation speed control.

estim node: check GPS quality, fuse GPS data with AHRS-based model (without using motor inputs).

velodyne_reader ROS package:

Obstacle detection in Velodyne data.

Low-level nodes

roboteq (needs **serial**, available on <https://github.com/wjwwood/serial>) ROS package:

Low-level control of the motors. 2 instances of the node need to be launched, one for the front Roboteq board controlling the 2 front motors and the other for the rear Roboteq board controlling the 2 rear motors.

Important ROS topics: subscribes to **geometry_msgs::Twist /cmd_vel**, **roboteq::msg_cmd_esc cmd_esc**; publishes to **std_msgs::Float64 batt_volts, internal_volts, left_amps, right_amps**.

light (needs **serial**, available on <https://github.com/wjwwood/serial>) ROS package:
Control of the 2 front white lights and the orange beacon through a Pololu Maestro board.
Important ROS topics: subscribes to **std_msgs::Bool /pololu/lights_on, /pololu/blinking_on, /pololu/warning_on, /emergency**; publishes to: **std_msgs::Bool /pololu/button_0, /pololu/button_1, /pololu/button_2**.

sbg_driver (could be installed in the system using **sudo apt install ros-melodic-sbg-driver**) ROS package:

AHRS data.

Important ROS topics: publishes to **sensor_msgs::Imu /imu/data**.

nmea_navsat_driver (could be installed in the system using **sudo apt install ros-melodic-nmea-navsat-driver**) ROS package:

GPS data.

Important ROS topics: publishes to **sensor_msgs::NavSatFix /fix**.

velodyne_pointcloud (could be installed in the system using **sudo apt install ros-melodic-velodyne**)

ROS package:

Velodyne data.

Important ROS topics: publishes to **sensor_msgs::PointCloud2 /velodyne_points**.

rosbridge_server (could be installed in the system using **sudo apt install ros-melodic-rosbridge-suite**) ROS package:

Access to ROS topics from other computers without ROS.

Maintenance

Check that the tires are inflated.

Regularly check that all screws are properly tightened.