

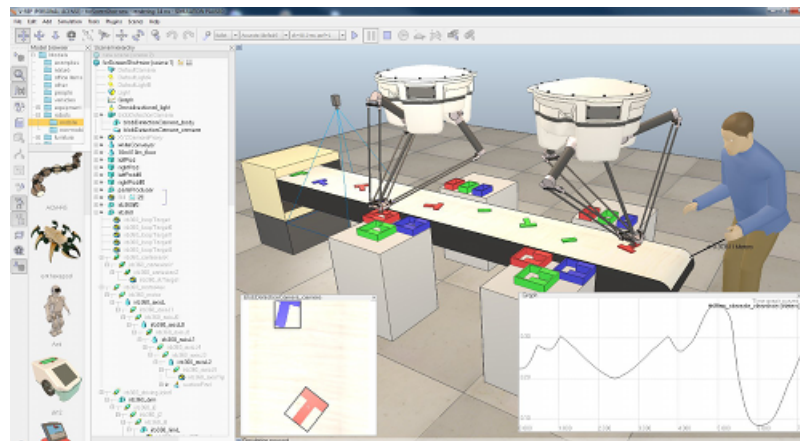
Simultaneous Localization and Mapping Workshop

During this workshop you will learn the basics of:

1. SLAM Algorithm for scene reconstruction
2. Using the V-Rep robot simulation software
3. Writing python code for controlling a robot in a virtual environment

V-Rep

V-rep is a robot simulator that has various premade robot models, ability to create custom environments, functionality for robotics related tasks such as Inverse Kinematics, path planning, sensor integration (lidars, cameras, force sensors, collision processing). It is used frequently by the scientific community and it also has a very extensive programming interface that you can use with 6 different programming languages (Python, C++, Matlab, Lua, etc.). You are highly advised to download it on your machines (available on all platforms) and experiment with it, extensive tutorials can be found [here](#).



Setting up:

1. Open a terminal in the `../SLAM` directory or navigate to it using `cd`.
2. Create and activate the python virtual environment:

```
virtualenv env
source env/bin/activate
```

3. Download the required packages (numpy, opencv, matplotlib):

```
pip install -r requirements.txt
```

4. Start V-Rep with the custom scene loaded by running:

```
python start.py
```

Task

You can see a simple indoor scene cluttered with every-day objects and a stationary robot that has a LIDAR sensor attached to it.

This particular LIDAR has the following configuration:

- **Scan angle:** 270 degrees.
- **Number of scan points** 270 (1 point per degree)

Your goal is to create a simple python script that will make the wheeled robot move from its starting position towards the ending point (marked by the red circle) by utilizing the LIDAR information.

The file *run.py* contains a function *loop* that you must modify in order to make the robot to perform actions in the environment.

To run the simulation:

```
python run.py
```

You can see the display window that shows the reconstruction of the scene from the lidar data.

You can obtain the information by using the class variables and methods of the object *agent*. Here is a short documentation of the functions that you have available:

Motors:

```
1. agent.change_velocity([ speed_left, speed_right ])
```

Set the target angular velocities of left
and right motors with a LIST of values:
e.g. [1, 1] in radians/s.

Values in range [-5:5] (above these
values the control accuracy decreases)

```
1. agent.current_velocity()
```

Returns a LIST of current angular velocities
of the motors

```

[ speed_left, speed_right ] in radians/s.

Lidar:
1. agent.read_lidar()
----
    A list that you can use to access lidar data
    Default length of the list -> 270

    Basic configuration of the lidar:
    Angle: [-135:135] Starting with the
    leftmost lidar point -> clockwise

Agent:
1. agent.pos
----
    Current x,y position of the agent (derived from
    SLAM data)

1. agent.position_history
----
    A deque containing N last positions of the agent
    (200 by default, can be changed in settings.py)

```

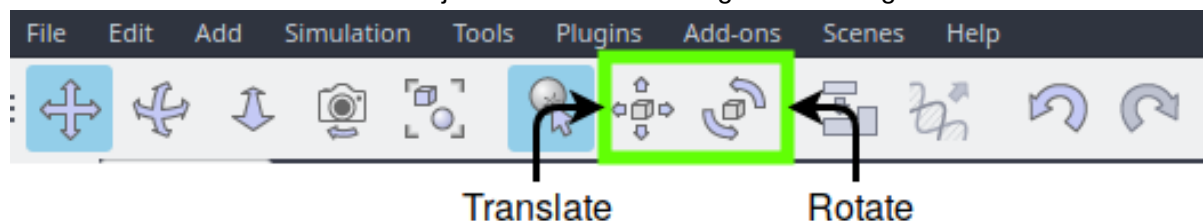
Using these methods and variables design a viable room traversing strategy using a combination of logic statements.

You also have access to *settings.py* file that contains various parameters of the environment. You are free to modify this file, although it is not necessary to obtain the desired outcome. In case something breaks there is a commented out section at the bottom of the *settings.py* to restore the original configuration.

Interacting with the simulation

You can freely interact with the *V-Rep* scene in various ways when the simulation is not running.

- You can move and rotate all the objects in the scene using the following buttons:



Just click on an object to select it and try dragging it around.

- Once you make some progress in order not to run the same route, you can just move the robot and test the behaviours out in different spots of the environment. Having said that, your goal is still to make the agent traverse all 4 rooms, starting at the center one.

Testing in a dynamic scene (optional)

Once you have a well performing navigation behaviour you can test it out in a dynamic environment. To do that:

1. Close the current simulation
2. Run

```
python start.py --test
```

This will open a modified scene that contains people who walk randomly around the scene. See how your navigation algorithm operates when there are dynamic objects in the scene. You can see the computational overhead introduced by the dynamic objects.

Remarks: having dynamic objects in the scene also complicates the probabilistic estimation of the surroundings that the SLAM algorithm performs. You can see a lot more artifacts in the scene, but this can be reduced by increasing the number of iterations per SLAM step (in *settings.py*, variable *max_iters* in the SLAM section.) This does, however, slow the simulation down even more.