CS-7630 Autonomous Robotics

# Homework – Probabilistic Localisation and Mapping

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

This homework assumes that you have completed the previous homeworks.

Submit the ros package (tar.gz or zip), by email by next Monday to cedric.pradalier@georgiatech-metz.fr

The goal of this homework is to implement the probabilistic localisation and mapping algorithms seen in the class, and use them to localise our 6 wheel rover and turtlebots.

Launch V-REP and open the scene lakeExampleAR.ttt. This environment contains rotating billboards with specialised markers that can be detected from the camera image. The ROS node responsible for the localisation of these tags is from the ar_track_alvar package. Its output is an array of 6-DoF pose of the detected tags in the frame of the camera.
In addition to the tags, the rover is equipped with a compass that outputs its absolute orientation in the world frame.

A set of ros packages is provided as a starting point (ar_mapping_base, ar_loc_base, ar_slam_base). They contains a structure that subscribes to the relevant topic and can call localisation and/or mapping filters. These filters are what we will be implementing in this homework.

The launch file provided in the package calls the rover_driver implemented during a former homework. If you want to use your files, make sure to remove the part related to computing the odometry (to save the computing time) and **remove the part where the transform from /odom to /rover/ground is published.** Leaving it would lead to an invalid transform tree.

The basic node rover_driver.py subscribe to this twist and to the relevant Vrep topics (each wheel publishes the states of its steering and drive motors as well as subscribe to commands for these drives). The steering motors can be controlled in position (Float64, radians) and the drive motors can be controlled in velocity (Float64, radians/seconds).

To plot your results, you can use rosbag record at run time, and then rostopic echo with the -b and -p option to generate files that can be plotted with Matlab or Octave.

You can also work on recorded data for the development: use rosbag record -a to record all the topics while driving around among the markers. Afterward, you can use rosbag play to repeat the same sequence and compare the performance of your filters in comparable situations.

**Important: the wheel radius is measured when the rover_driver.py node starts, assuming the**

**rover is starting on a flat ground.**

## *Step 1: Mapping with known localisation.*

The goal of this first step is to create a node that builds a map of the markers in the lakeExampleAR.ttt scene.

The base package is ar_mapping_base. In this package, you only need to modify the mapping_kf.py script (in the src directory) to do the following (check the label TODO):

- Decide when to initialise a new landmark or update an existing one (function update_ar)
- Implement the initialisation code for a given landmark (class Landmark, function __init__)
- Implement the update code for a given landmark (class Landmark, function update)

This package already subscribes to the correct topics and as an output it publishes the visualization markers that can be seen as MarkerArray in Rviz.

## *Step 2: Localisation with respect to a known map.*

The goal of this step is to implement the localisation equations in a Kalman filter. Source file: rover_kf.py. After each step, evaluate the localisation performance by driving around a loop and using Rviz to check how the published pose move with respect to the /rover/ground frame.

1.a: Modify the predict function to implement the prediction step of the Kalman filter. Only modify the part of the function marked with #TODO

1.b: Modify the ar_update function to implement the update step of the Kalman filter assuming landmarks are observed. The argument Z is a 2x1 vector of the measurement in the rover reference frame. L is the position of the observed landmark in the world frame as a 2x1 vector. Uncertainty is the uncertainty on the measurement process, given as a direction-less radius.

## *Step 3: Simultaneous Localisation and Mapping*

In this final step, we estimate simultaneously the localisation and the map of landmarks. The V-Rep scene is still lakeExampleAR.ttt.

The base package you can use is ar_slam_base, in which the parts to modify have been marked with the label TODO:

- Implement the prediction step of the kalman filter
- Implement the update based on observation of AR tags
- Implement the update based on the compass observations

You need to reuse the code from the previous step but adapt it to the new state.

This package already output a Pose topic and a MarkerArray topic, similar to the mapping package (step 2). Both of them will be visible in Rviz.

**H**ow can you deal with incorrect/spurious detections?

### *Step 4: Test your SLAM on the turtlebot*

The final goal of this homework is to implement your SLAM system on the turtlebot. There are two specific points to adapt:

- The markers are no longer rotating, so observing their orientation can be taken into account to improve the localisation.
- Use sticky tape to attach enough markers in GTL's second floor hallways. Because we have only 17 markers, we might need to account for re-occurring ids. This is fine as long as they are far apart. You could also generate more IDs. Check the alvar page.