Practical Session 3: Data Association and Robot Relocation

The objective of this session is to program in MATLAB the most complex data association problem in SLAM: the vehicle relocation. It can be stated as follows:

- given a previously built stochastic map of the environment,
- and a set of observations taken from the vehicle sensors,
- without having any a priori vehicle estimation,
- determine the vehicle location.

For this purpose we will use the "Trees in the Park" dataset provided by José Guivant and Eduardo Nebot, form the University of Sydney (many thanks!). In a previous EKF-SLAM run, a stochastic map of a part of the park was built, using as features the tree trunk center points obtained by Sydney's segmentation algorithm. The mean value and variance of trunk radiuses and the covisibility of the trees were also computed during the map building process.

Take a look at the main program `relocation`. At the beginning, the stochastic map (without vehicle) is loaded in variable `features`. For a given `step`, the laser scan is segmented to obtain trees, and the result is stored in variable `observations.` Trunk center positions `observations.z` are already represented in Cartesian coordinates relative to the vehicle reference. The corresponding covariance matrix `observations.R` is block-diagonal (why?).

An estimation of the ground-true vehicle location for the first 2000 steps is given in variable `vehicle.ground`. This information is used by the main program to compare with the solution obtained by your relocation algorithm. You must not use it in your data association algorithm.

Your task is to program the function `relocation_GCBB` to obtain the correct data association hypothesis, using the GCBB algorithm. A hypothesis will be represented by a vector showing the map feature corresponding to each observation, where a zero represents a non matched observation.

To solve the problem you will also need to program functions `estimate_distance`, `binary` and `unary`. Some provided functions you may want to use are:

- `is_consistent`: computes the vehicle location using EKF and verifies global consistency using Joint Compatibility. A faster but less precise solution would be to compute robot location by least squares (see code).
- `point_rows`: gives the row numbers corresponding to a set of points in `features` or in `observations`. See the main program for an example of usage.

The `tools` directory contains functions for drawing and for computing vehicle location, that are used by the main program. You don't need to use them in your code.

You should follow these steps:

1. **Preliminary.** Try running the `relocation` program. Figure 1 shows the stochastic map with the true vehicle location in blue. Figure 2 shows the scan corresponding to the selected step and the trees found by the segmentation algorithm. After running the program, the solution obtained is drawn in red in figure 1: blue points are map features and red points are observations. The hypothesis found (a silly answer in this case) is shown by joining the matched trees with a red line.

2. **Stochastic distances.** We will use the distances between tree trunk centers as binary constraint. Given a vector with the Cartesian coordinates of two points: $x = [x_1, y_1, x_2, y_2]^T$ and its 4x4 covariance matrix P (remember, points in the map ARE correlated) write down the equations needed to compute the estimated distance between the two points and its variance, and code them in function `estimate_distance`

3. **Binary constraints.** Program the basic GCBB algorithm using binary constraints: for a given hypothesis, the distance between any pair of observed trees must be compatible with the distances between the corresponding trees in the map. Compatibility must be verified using a chi-squared test. To test your program use the `configuration.small_map` option and steps between 1 and 200, otherwise running time may be a little long.

4. **Unary constraints.** In this problem the trunk radius can be used as a unary constraint, that can also be coded as a chi-squared test. Add this constraint to your GCBB algorithm and verify the improvement in computing speed.

5. **Locality.** To solve the relocation problem inside the whole 145-feature map in reasonable time you need to implement the *locality* concept using the map covisibility matrix: when a map feature has been matched, the search for more pairings can be restricted to its covisible features. Add this idea to your code and verify the improvement in computing speed.

6. **Optional improvement.** To avoid repeating distance computations, you can add global variables to store the distances between the trees in the map and in the observations. When a distance is first needed it is computed and stored. If it is needed later on, it is simply retrieved. Does your program compute all possible distances? Why?