Prof. Dr. Maren Bennewitz
M.Sc. Padmaja Kulkarni

**Bonn, 11 April 2019**

# Humanoid Robots

## Exercise Sheet 2 - Odometry Calibration

**Note: Due to changes in the schedule of the lecture, you have two weeks to complete this exercise sheet. Deadline: April 25.**

### Exercise 3 (20 points)

Mobile robots typically execute motion commands only inaccurately due to slippage on the ground, uneven terrain, or hardware issues. For example, if one of the knee joints of a humanoid robot is weaker than the other one, then it will walk on a circular trajectory although it intends to walk straight ahead. In order to account for such systematic errors, odometry calibration can be applied to learn and correct the drift.

In the lecture a system was introduced for calibrating odometry using a least squares approach.

The repository contains the following files:

- `src/03_odometry_calibration/data/calib.dat` contains odometry data recorded by a real robot. The first three columns contain the odometry data $(u_x^\star, u_y^\star, u_\theta^\star)$ measured with an external tracking system and the last three columns contain the odometry $(u_x, u_y, u_\theta)$ measured by the robot. $\theta$ is measured in radians counterclockwise.
- `src/03_odometry_calibration/src/OdometryCalibration.cpp` contains the functions that you have to implement.
- `src/03_odometry_calibration/include/odometry_calibration/CalibrationData.h` contains the data structures for the odometry, calibration data, and 2D poses.

The `main` function already loads the data file into the `MeasurementData` data structure (i.e., both the ground truth odometry data observed externally, and the robot's observed odometry).

### Exercise steps:

a) Implement `errorFunction(groundTruth,observation,calibrationMatrix)`, which calcuates the error $\mathbf{e}_i(x)$ between the ground truth odometry $\mathbf{u}_i^\star$ and the corrected observed odometry (which is corrected by the current estimate of the calibration matrix).

b) Calculate the Jacobian $\mathbf{J}_i$ of the error function for a given odometry measurement by implementing `jacobian(observation)`. The Jacobian is defined as

$$\mathbf{J}_i := \frac{\partial \mathbf{e}_i\left(\mathbf{x}\right)}{\partial \mathbf{x}}.$$

c) Calculate the calibration matrix by implementing `calibrateOdometry(measurements)`. The function uses the loaded data passed as a parameter and should return the calibration matrix. In our case, the weight matrix $\mathbf{\Omega}_i$ is the identity matrix. One iteration is sufficient, because the error function is linear.

d) Use the calibration matrix (computed in the previous step) to correct the robot's odometry observations in `applyOdometryCorrection(uncalibratedOdometry,calibrationMatrix)`.

e) Compute the robot's trajectory based on the corrected odometry observations by implementing `calculateTrajectory(calibratedOdometry)`:

   1) Assume that the robot starts at the position $(x, y, \theta) = (0, 0, 0)$.
   2) Transform the robot's pose using the corrected odometry by implementing `odometryToAffineTransformation(odometry)` (Check the lecture slides 34–35 for more information about affine transformations).
   3) Chain the affine transformation to get the next pose. (See lecture slides 36–38).
   4) Convert the chained affine transformation back to a robot pose by implementing `affineTransformationToPose(transformation)`.
   5) Store the pose in the trajectory vector.

When you push your code to the server, the server will plot a figure and save it to the result page in the Wiki. The figure shows the calibrated trajectory computed by your program in blue. Compare it to the ground truth trajectory (in red). The trajectories should match approximately, but there will still be a small error accumulating over time.

If you have Gnuplot installed, you can generate the same figure on your computer using the `scripts/plot.gp` script (see the Wiki for instructions).

**Deadline: 25 April 2019, 11:59 am**