CMPE 565 AUTONOMOUS ROBOTS
Assignment 3: Odometry Calibration

Deadline: 01.03.2018

In this assignment, you will make odometry calibration for an omni-directional mobile robot. Update your repositories:

- If you have any uncommitted changes in your repository, first run the command below for both of your repositories (ros-ws and simulation), otherwise skip this command.
  `git stash`
- Update your repository inside your ros workspace to get recent changes for the homework.
  `git pull http://robot.cmpe.boun.edu.tr/gitlab/cmpe-565-2018/ros-ws.git`
- Update also your simulation repository. `git pull http://robot.cmpe.boun.edu.tr/gitlab/cmpe-565-2018/simulation.git`
- If you ran the git stash command, now run the command below for both of your repositories (ros-ws and simulation), otherwise skip it. `git stash apply`

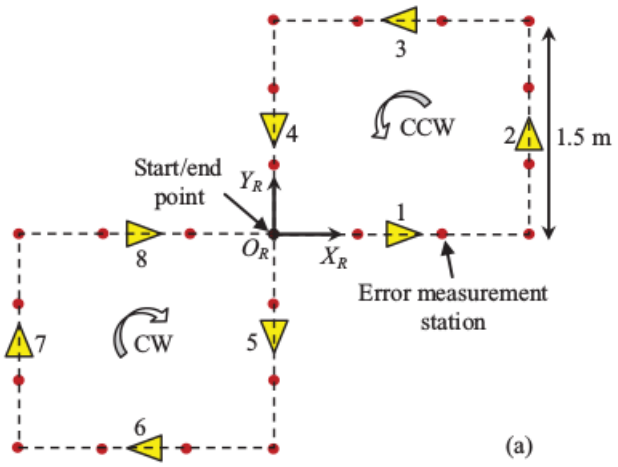
Things to do:

- Build new packages or changes with `catkin_make` in ros-ws folder.
- Download and read about the calibration procedure. (pages 969-976)
  `https://www.researchgate.net/publication/259434997_Calibration_of_omnidirectional_wheeled_mobile_robots_Method_and_experiments`
- Execute `roscore` command
- Run V-REP with `./vrep.sh` command in the V-REP folder. Make sure ROS plugins are loaded by checking the command line.
- Open `assignment3_calibrated.ttt` scene in V-REP.
- From the menu click ”Simulation”, and then select ”Using the ODE physics engine”. Run the simulation. Pay attention to the beginning position of the robot.
- Execute `rosrun robotino_odometry robotino_odometry`. The robot will start to move. Pay attention to the end position of the robot when it stops.

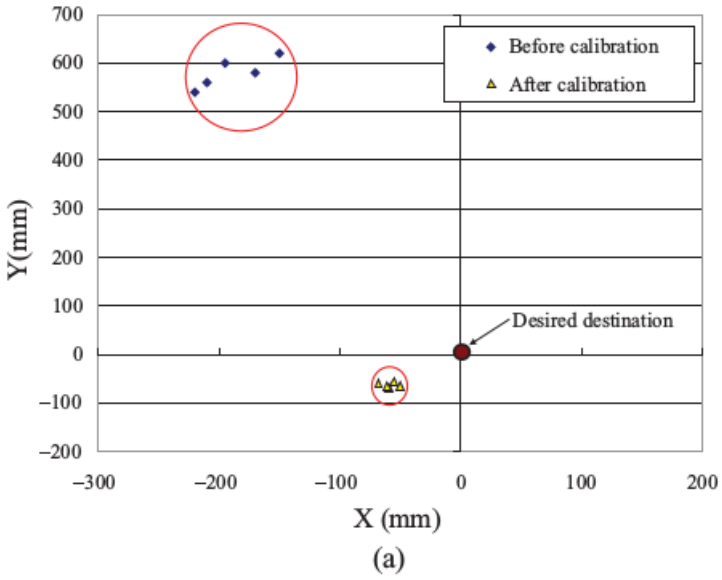
- Now open `assignment3_robotino_<your last name>.ttt` scene in V-REP, which has the uncalibrated robot.
- Don't forget to check your physics engine to be ODE each time you run a simulation. "Using the ODE physics engine". Run the simulation. Pay attention to the beginning position of the robot.
- Execute `rosrun robotino_odometry robotino_odometry`. The robot will start to move. Pay attention to the end position of the robot when it stops.
- Please also watch this video to understand it better. `https://youtu.be/ZGSU3Bc6acQ`
- You see that the end positions of the robot in assignment3_calibrated scene and assignment3_robotino_<your last name> scenes differ from each other. Your task is calibrating the robot in assignment3_robotino_<your last name> scene to make the robot move correctly.
- We provided the necessary information about the robotino base (angles, lengths, etc.) in the robotino_odometry.cpp. We also calculated the inverse Jacobian matrix(J^{-1}). Given the velocity commands $\vec{\dot{\mu}} = [x, y, \theta]^T$, we can find the velocity for each wheel ($\dot{\phi}_i$) by

$$\vec{\dot{\Phi}} = J^{-1}\vec{\dot{\mu}} \tag{1}$$

- Follow the calibration steps explained in the paper.
- You will only use assignment3_robotino_calibrated.ttt scene to move the robot from (x,y)=(0,0) to (x,y)=(1,0) (length of the desired trajectory, L, which will be 1 meters). Open the scene, run the robotino_odometry package after you edit 157th line of the code properly. When you change the limit for c, you change the time the robot moves. Find a proper limit value in order to move the robot 1 meter to the right.
- Find lateral corrective matrix (F_{lat}) and longitudinal corrective factor (F_{lon}) using assignment3_robotino_<your last name>.ttt scene to calculate the corrected inverse Jacobian matrix (J_c^{-1}). Do not forget to include them in your report.
- For going on a straight line, calculate the radial error (δr_e) and the mean error improvement index (δr_m) to create a table like Table 2 in the paper. (You do not need to calculate Kurtosis and Skewness)
- Modify the code in robotino_odometry.cpp to make the robot move in a double square motion (for counterclockwise followed by clockwise direction).



- You only need to do the double square path experiment as shown in the figure above (Figure 6.a in the paper). Do not worry about error measurement stations, since you will not draw the path of the robot on a graph.



- Compile and run the code again. Once the robot completes the double square, measure the difference between the starting position and the end position. (Do this 10 times total, 5 before calibration and 5 after calibration to draw a graph like the one above)

- Calculate the radial error (δr_e) before and after calibration. Then calculate the root mean square (RMS) improvement to create a table like Table 4 (first row) in the paper.

- In your report, include a plot of the errors you collected **before** and **after** the calibration (like Figure 8.a in the paper). You should also report lateral corrective matrix (F_{lat}), longitudinal corrective factor (F_{lon}) and the corrected inverse Jacobian matrix(J_c^{-1}). You should create a table similar to Table 2 (without Kurtosis and Skewness columns) in the paper, after the

straight line runs. You should also create another table similar to Table 4 (first row) in the paper, after your 10 runs (5 before calibration, 5 after calibration) of double-square path. Finally, include screenshots and videos of the uncorrected and corrected robot paths.

- Correction of the Equation 13 on the paper, there should be an equals sign after J^{-1}.

$$J_c^{-1} = F_{lon}F_{lat}J^{-1} = diag\left[\left(1 + \frac{\dot{\phi}_{e,1}}{\dot{\phi}_1}\right)\frac{1}{r_1 cos\gamma_1}...\left(1 + \frac{\dot{\phi}_{e,n}}{\dot{\phi}_n}\right)\frac{1}{r_n cos\gamma_n}\right] \times ... \quad (2)$$

  You do not need to worry about the expanded version of the equation. Only the first part $J_c^{-1} = F_{lon}F_{lat}J^{-1}$ is enough.