# 8. Assignment, Introduction to Robotics WS17/18 - Ver. 1.01

Prof. Daniel Göhring, Martin Dreher, Jakob Krause
Institut für Informatik, Freie Universität Berlin
Submission: online until Tuesday, 19 Dec 2017, 11:55 a.m.

**Please summarize your results (images and descriptions) in a pdf-document and name it, e.g., "RO-08-<surnames of the students - group name>.pdf".**
**Submit your python code and video**
**Only one member of the group must submit the necessary files.**
**Do not copy solutions to other groups.**
**Every group must contain two people, unless granted differently.**
**Only submissions via KVV will be accepted.**

## 1 - Kalman filter - Combine odometry and visual gps data) (10 points)

Use a Kalman filter and combine the odometry data and GPS data to smoothen your position over time. delta x, delta y is the change of the car in x,y world coordinates. theta is the car's orientation in world coordinates. v is the car's velocity which can be calculated from the last two odometry positions and the time difference delta t - but please make sure that v becomes negative when the car is driving backwards.
You can use the simple model of the car in the prediction step.

delta x = v cos(theta) * delta t

delta y = v sin(theta) * delta t

delta_theta = the difference of orientations of the last two odometry angles (the orientation angle (as long as it is theta) of one odometry topic is calculated as arccos(quaternion.w) * 2 * sign(quaternion.z)

x_t^{prior} = x_{t-1}^{posterior} + delta_x

y_t^{prior} = y_{t-1}^{posterior} + delta_y

theta_t^{prior} = theta_{t-1}^{posterior} + delta_theta

This prediction needs to be updated.

a) (5 points) A very simple update method (which assumes that x,y,theta) are independent of each other would just take a weighted average from the predicted position and the measured (GPS-)position, e.g., for k = 0.5:

updatedPosition_x = k * measuredGPSPosition_x + (1-k) * predicted position_x,

you can do the calculations for y and theta, accordingly.

You can also use different values for k, especially a different one for x,y and one for theta and see how this affects your filtered position results. *Plot the raw data of your GPS, the odometry data and your filtered positions. Put a link to your source code (Kalman-Filter-Version1) in your Pdf.*

*Hint for debugging: When k = 0, you should get the same position results as for not using any sensory update at all. For k=1 you should get the raw GPS-positions as a result.*

b) (5 points) In general we do not want to use the same weighted average for each dimension of our state space, further we want to be able to handle cases where we receive odometry data and sensory data with different frequencies, e.g., two odom topics for each GPS-position. Therefore, now apply the Kalman-filter-formula as in the lecture slides, page 29. But to keep it simple you can implement 3 one-dimensional Kalman filters (one for x, one for y, one for theta). Assume that the values of the vectors are in meters for x,y, and in radian for theta, the values of the matrices are in meters^2 or radian^2 . For the prediction step, do as before, the a-priori P-matrix in the prediction step will just be the old a-posteriori P-matrix plus process noise matrix Q.
Make some assumptions for the process noise Q-matrices (for x,y,theta) and the sensor noise matrix R. For example, one could assume that the process noise (standard deviation) for component y and for a delta time of 0.01 s is 0.0001m, so in that case, Q would be (0.0001m)^2 . Make some assumptions for R (for x,y,theta) as well.
Assume that matrix H is the identity matrix, i.e., 1.

*Write down, which values you were using.* An initial value for the P-matrix for x and for y could be, e.g., (3m)^2, for theta it could be (pi/2)^2, your initial values for x,y,theta could be 0 .

*Hint: When you have done everything correctly, your K-matrix values (in this case one scalar for each dimension x,y,theta) should converge towards a value between 0 and 1.*

At the end, publish your estimated position as odom_gps (the message type: nav_msgs/Odometry). Move the car in a circle around the room, and *visualize the GPS-position, the odometry-position and your Kalman-filtered position* using rviz.

Commit the source code to your catkin_ws_user git repository. *Put a link to your source code (Kalman-Filter-Version2)* in your final Pdf.

*Hint: You can plot the odometry in rviz by adding the odometry topic, then click on that topic. You can now tell rviz how to represent that topic, i.e., how thick the trace should be and how many points of the past shall be drawn, e.g., 9999 .*