Lab 9: Solutions

2 Odometry with non-deterministic uncertainties

When dead reckoning is properly calibrated, the e-puck can track its initial position really well as you saw in Lab 08. Here we will now add non-deterministic sensor noise as is present on real robots.

1. (Q): Open the robot's controller *e-puck_feature.c* and try to understand the code. Explain in your own words what happens. Try to come up with possible ways to improve the accuracy of the motion under non-deterministic sensor noise conditions (do not implement them). What are their downsides?
   Averaging → induces a time delay.
   Filtering → needs multiple sensors
   Buy a better sensor for the real hardware → costs money :)

2. (I): One possibility to correct the accumulating errors is feature-based navigation. We now want to use the front wall as a feature to correct the estimation of robot position. We will do this by developing a function that measures the distance to the wall and use it as the feature to adjust the robot's position. Open the file *e-puck_feature.c* and implement the above mentioned strategy, by supposing that the front wall is at the position (1, 0) and the robot start at position (0,0) in the function *feature_update*.

   Cf. code

3 Sensor fusion with Kalman Filter in Matlab

The feature-based correction of the 1D odometry implemented in Part 2 is a very basic sensor fusion. Indeed, we used the information of both the wheel encoders as well as a distance sensor, effectively resulting in a fusion of the information. This implementation, however, is a relatively crude one, since it assumes that the detection of the front wall is free of non-deterministic noise, which is not realistic. In this section we will gradually study a more general approach to sensor fusion leveraging a well-known estimation technique called Kalman filter. For this, open the file *Kalman1d.m* in Matlab.

3. (Q) Try to understand what this script does.
   Similar simulation as before in webots.

4. (Q) Using linear propagation of covariance (c.f. slides week 10), calculate the noise on the position estimate (*est_odo_x*) after 5 timesteps (position noise at T=1 is equal to 0). What is the noise after 200 steps? Compute this on paper.
   Hint: if $Q = a + b$ *then* $\delta Q = \sqrt{\delta a^2 + \delta b^2}$
   $\sigma = 0.05$ *(the relative error corresponds directly to the standard deviation, see https://ch.mathworks.com/help/matlab/math/random-numbers-with-specific-mean-and-variance.html).*
   *5 timesteps: sqrt(5*0.05^2)*
   *200 timesteps: sqrt(200*0.05^2)*

5. (S) What happens if you change the noise level *localization_noise*? Is this factor taken into account in the position estimation?

The position is set to a very noisy measure. The noise is not taken into account.

6. (I) Since for most real applications the noise level is known, it is suboptimal to ignore this information. Applying the central limit theorem, a very simple option to take into account the noise levels of two variables when combining them, has been proposed by Fraser and Potter in 1969 [1]:

$$x_{fused} = \frac{(\sigma_1^{-2} x_1 + \sigma_2^{-2} x_2)}{\sigma_1^{-2} + \sigma_2^{-2}}$$

For *est_fp_x*, replace the simple resetting of the position with this formula using the noise level calculated in Question 4 for the odometry estimate (200 steps correspond to the interval between features). Use σ=*localization_noise* for the measurement noise.
Run the script and compare the resulting trajectories. What do you observe?
Cf., code

7. (S) What happens if the measurement noise is significantly higher than the odometry noise and vice-versa? What is the issue of this sensor fusion algorithm?
If one noise is significantly higher, then this measure will not have any major impact. The sensor fusion algorithm doesn't have an 'issue' per se. One has to keep track manually of the noise level of the 'odometry measure' in order for it to be accurate, plus the idea of an 'optimal' gain to reduce noise as in Kalman filtering is not there.

8. (Q) Another method to fuse measurements is the Kalman filter. Try to identify what each line (step) of the algorithm does. Which steps of the Kalman filter did we already implement previously in this lab (maybe in a slightly different way)?
The first step (estimation based on motion model) is basically what we did for odometry, while tracking the accumulated estimation variance along time. The second step (Sensor and Motion model Fusion) is similar (though not exactly the same) to the fraser and poter algorithm, basically both measurements are fused depending on the noise levels, with the fused state estimation variance updated as well.

Implement the different steps of the Kalman filter:

9. (I) Create the new state variable μ = [x]. Implement the prediction step for μ such that the position value of μ increases by the odometry measure every timestep (i.e. every time the prediction is called). Run your code and verify (through printing) that μ increases as expected.

   Cf., code

10. (I) Create the noise state variable Σ = [0]. By initializing Σ in this way, what are we assuming? Implement the noise prediction update step.
Cf., code we assume that initially the position is perfectly known.

11. (I) Implement now the correction step. This step is called every time a new measure is acquired to correct or update the prediction. Set the measurement noise covariance matrix (C) to *localization_noise*^{2}. Uncomment the plotting function to plot μ, run your code and compare the result with what you obtained before.
Cf., code

12. (Q) What are the differences between the Fraser Potter algorithm and the Kalman filter in the 1D case?

They are the same thing. One can easily show this by starting from the fusion equation of the state and measurement in the Kalman filter and rearrange the terms to find the Fraser Potter formulation.

4 Sensor Fusion with 2D Kalman Filter

13. (I) What are μ, Σ, u and z? What is the purpose of K?

    μ: state vector
    Σ: covariance matrix
    u: 'external'/controller impact on the 'plant'
    z: measurements
    K: optimal Kalman gain to reduce maximally the Σ

1: **Algorithm Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)**:**
2: $\bar{\mu}_t = A_t\ \mu_{t-1} + B_t\ u_t$
3: $\bar{\Sigma}_t = A_t\ \Sigma_{t-1}\ A_t^T + R_t$
4: $K_t = \bar{\Sigma}_t\ C_t^T (C_t\ \bar{\Sigma}_t\ C_t^T + Q_t)^{-1}$
5: $\mu_t = \bar{\mu}_t + K_t(z_t - C_t\ \bar{\mu}_t)$
6: $\Sigma_t = (I - K_t\ C_t)\ \bar{\Sigma}_t$
7: return μ_t, Σ_t

Figure 1: Kalman filter. Note that steps 2 and 3 (called prediction) can be done independently from steps 4 to 6 (update step).

14. (B) We shall now consider the case where the robot speed v is considered as part of the state. Start by commenting out your previously implemented Kalman filter.
    Then create the new state variable μ = [x; v] with the noise state variable Σ = [0, 0; 0, 0]. Define A and B such that the position value of μ increases by mov every timestep. Adapt R for your code to execute and run your code and verify that μ and Σ increase as expected.
    Hint: Be careful about the dimensions of your matrices.
    Cf., code

15. (B) Copy your previously implemented correction step and adapt it to the new two-variable state. Run your code and observe the result. Is there a major difference to the previous Kalman filter result? (Uncomment your previous implementation if you are not sure)
    *Hint: the correction happens only on x, leaving the speed part of μ untouched. This means that both z and C*μ are 1 dimensional, whereas K needs to be in such a way that K*[1 dim] results in a vector of the same dimension as μ.*
    Cf., code

16. (B) Based on the changes you did to include the robot speed v in the state space, what changes would be necessary to include x, y, vx and vy?
    Changing from now 2 to 4 variables requires precisely the same matrices to change as you changed in Q14 and 15.

5 References

[1] D. Fraser and J. Potter, "The optimum linear smoother as a combination of two optimum linear filters," in IEEE Transactions on Automatic Control, vol. 14, no. 4, pp. 387-390, August 1969, doi: 10.1109/TAC.1969.1099196., url: https://ieeexplore.ieee.org/document/1099196

[2] Maybeck, Peter S. “Stochastic models, estimation, and control”. Academic press, 1982.