Lab 9: Sensor Fusion for Localization

This laboratory requires the following equipment:

- C development tools (gcc, make, etc.)
- Webots
- Matlab

The laboratory duration is approximately 3 hours. Although this laboratory is not graded, we encourage you to take your own personal notes as the homework and the final exam might leverage results acquired during this laboratory session. For any questions, please contact us at **sis-ta@groupes.epfl.ch.**

1.1 Information

In this assignment, you will find several exercises and questions.

- The notation **Q** means that the question can be answered theoretically or with simple commands in the Linux operating system, without implementing or running any code.
- The notation **S** means that the question can be solved only by compiling and running a piece of code or an additional simulation.
- The notation **I** means that the problem has to be solved by implementing, possibly compiling, and running a piece of code.
- The notation **B** means that the question is optional (bonus) and should be answered if you have enough time at your disposal.

1.2 Getting Started (Short reminder)

To start with this lab, you will need to download the material available on Moodle. Download *lab09.zip* in your personal directory and extract the lab archive.

1.3 General remarks and documentation

In this lab, you will continue working on odometry with the e-puck robot, extending the techniques introduced in Lab 8. Here you will understand the importance of sensor fusion to obtain an accurate interpretation of the data, and how noise on these data can be considered in this process. The first part demonstrates how the e-puck can navigate using 1D dead reckoning under non-deterministic uncertainties. The second part shows different techniques to fusion odometry with feature-based localization in Matlab. The last part demonstrates how to leverage a Kalman filter in Webots to fuse odometry with feature-based localization.

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 8. Here we will now add non-deterministic sensor noise, since this is present on real robots.

1. **(Q)**: Open the world *featurenavigation.wbt*, compile the controllers and run the simulation. 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 behaviour under these conditions (do not implement them). What are their downsides? *Note: run the simulation for about 3h of simulation time to fully see the degradation in performance of odometry.*

2. **(I)**: One possibility to correct the accumulating errors is feature-based navigation. We now want to use the front wall as a feature each time we are next to it. We will do this by developing a function that measures the distance to the wall (i.e., the feature) and adjust the robot's position such that it is the same every time the robot finishes going back and forth. Open the file *e-puck_feature.c* and look for the already defined function *feature_update*. Fill the contents of function *feature_update*, so that it does the described behavior.
   Hint: The front wall is situated at 1 m from the initial position.

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, it assumes for example that the measurement of the back wall is free of non-deterministic noise which is obviously not possible. 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 *Kalman.m* in Matlab.

3. **(Q)** Try to understand what this script does.
   Note: This lab is not very guided on purpose. Take the time to reflect on each step and refer back to lecture slides to understand what you are implementing and why.

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* $\delta Q = \sqrt{\delta a^2 + \delta b^2}$
   Hint: you can refer to this link https://ch.mathworks.com/help/matlab/math/random-numbers-with-specific-mean-and-variance.html for the probabilistic distribution of random noise.

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

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], [2]:

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

Implement the position estimate using the Fraser Potter equation (*est_fp_x* in the code). *Hint: 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?

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?

8. **(Q)** Another method to fuse measurements is the Kalman filter. Take a look at the formula (seen in class) in Figure 1. What are μ, Σ, u and z? What is the purpose of K?

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).

9. **(Q)** 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)?

Implement the different steps of the Kalman filter. The final position estimate will be stored in the variable *est_kal_x*:

10. **(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. *Hint: Since for now we don't have a robot model A we can set A to 1, without external input our estimation remains as it is. The odometry measure is added through u.*

11. **(I)** Create the noise state variable Σ = [0]. By initializing Σ in this way, what are we assuming?

12. **(I)** Implement the noise prediction update step. What is the purpose of R?
Hint: line 3 of Figure 1

13. **(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 (Q) to (*localization_noise*)^2. Run your code to plot μ and compare the result with what you

obtained before. You can run your code several times to see how each approach (odometry, Fraser-Potter and Kalman filter) performs on average, the random noise introduced in the system will give you slightly different results each time.

14. **(I)** We shall now consider the case where the robot speed v is considered as part of the state. Start by setting kalman_dimension to 2 (which deactivates your previously implemented Kalman filter).
    Then create the new state variable μ = [x; v] with the noise state variable Σ = [0, 0; 0, 0]. To implement the prediction step, define A and B such that the position value of μ increases by odo every timestep. Adapt R for your code to execute and run your code and verify that μ and Σ fluctuate as expected.
    Hint: Be careful about the dimensions of your matrices.

15. **(I)** Implement the 2D case of the Kalman filter by inspiring yourself from your previously implemented correction step and adapting 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.*

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?

4 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.