Lab 3: Positioning, Odometry and Sensor Fusion

This laboratory requires the following equipment:

- Webots
- C compiler
- MATLAB

Depending on your programming skills, the laboratory duration might require an effort of up to five hours although assistance will be provided in the computer rooms during a time window of four hours (three hours during Week 4 and one hour during Week 5). Although this laboratory is not graded, we encourage you to take your own personal notes as the exam might leverage results acquired during this laboratory session. For any questions, please contact us at dista@groupes.epfl.ch.

Information

In the following text you will find several exercises and questions. The type of question is indicated between parentheses:

- The notation **S** means that the question can be solved using only additional simulation.
- The notation **Q** means that the question can be answered theoretically, without any simulation.
- The notation **I** means that the problem has to be solved by implementing a piece of code and performing a simulation.
- The notation **B** means that the question is optional and should be answered if you have enough time at your disposal.

1.1 Getting Started

Download lab03.zip available on moodle in your personal directory. Now, extract the lab archive. In a part of this lab, you will continue working with the simulated e-puck robot which we introduced in the previous lab.

2. Localization with positioning systems

Localization is the process of determining where a mobile robot is located with respect to its environment. It is essential for almost all robot applications. In this section, you will familiarize yourself with systems that provide absolute position.

Global Navigation Satellite System (GNSS, GPS is a kind of GNSS) are widely used for outdoor positioning. A Motion Capture System (MCS) is an example of an indoor localization system. Remember that such absolute positioning systems may not always be available.

In simple terms, a GNSS receiver works by calculating distances to various satellites, whose positions are known. It uses that information to calculate its own position by trilateration. In reality, there is much more complexity that is out of the scope of this lab. Typically, GNSS receiver modules are connected to robots via some communication interface and report latitude and longitude angles, which can be converted to absolute position in meters in a chosen reference frame.

In Webots, we will emulate a GPS receiver that directly provides 3D position in meters. To start with, we will assume that there is no noise in GNSS positions.

- Launch Webots. On Windows or MacOS you can launch Webots as any other software. On Ubuntu, you need to enter the following command in a terminal:
  `webots --mode=pause &`.
- From the menu, select *File->Open World*, and choose the *lab03.wbt* file from the *lab_03/webots/worlds* directory.
- From the menu tree (left side of the window), select *DEF ROBOT1 Robot > controller > controller*. Then click on `edit` to open the c file in the text editor.
- Click on the `clean` button and then on the `build` button.
- Again, click on the `clean` button and then on the `build` button.

2.1 Simulated GNSS and reference frame

1. **(Q)**: How many independent variables are required to properly define the pose, i.e. position and orientation, of a body (e.g., that of a wheeled robot) in a 2D world?

2. **(Q)**: What about a body (e.g., that of an aerial robot) in the 3D world?

3. **(Q)**: Based on your previous answers, propose a possible set of states to define the pose of your robot in a 2D world and in a 3D world. *Hint: Try to remember your physics courses on gyroscope and Euler angles.*

4. **(S)**: Compile the controller file and run it. You can control the robot (in 2D, on a plane) using keyboard commands. The GPS position values are available **once in 100 ms** and are printed in the terminal. Here are the keyboard commands to move the robot. Note that there is **no noise** in the GPS positions.

Keyboard	Simulation actions
R	Start moving
S	Stop moving
U	Increase speed
D	Decreases speed
↑↓	Move forward/backward
← →	Turn right/left

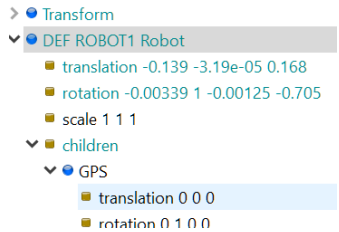
5. **(Q)**: The GPS positions do not directly provide information about the orientation of the robot. By looking at the position values in the console, can you deduce the orientation of the robot? How? What happens if the robot is moving backwards? What happens if the robot changes direction in between GPS updates?

6. **(Q):** Assume now that the GPS is noisy. Will you still be able to get a reliable value of orientation? Why or why not?

7. **(Q):**
   (a) How can you get the position of the robot in between GPS updates (say 50 ms after the last GPS update), without using any additional sensor?
   (b) How can you get the position of the robot in the event that the GPS is unavailable (for example, if the robot enters a tunnel)?

8. **(B):** Let us deviate from the topic of positioning and look into how Webots works. Consider this a tutorial, which will be helpful for your project.

   In Webots, all sensors and actuators are types of "devices". You can add / remove devices (and other objects) from the robot, or from the simulation in general using the scene tree on the left. Expand the Robot node to see the GPS device (see picture).



   (a) To add a new device, right click on "children" and click "add new".
   (b) Once you add a new device, you need to initialize it in your controller code. Look for the function `controller_init_gps()` in the code (around line 447) for initializing GPS, and `controller_get_gps()` for reading GPS values (around line 190).
   (c) Of course, you need to include the library that has the relevant functions, which in this case is `webots/gps.h`. See the inclusion of this file around line 9.

3. Odometry

In this part, we will perform localization using on-board sensors – namely accelerometer and wheel encoders. We will use the same Webots world and the controller used in the previous part. Keep the C file controller.c open.

3.1 Accelerometer

You must already be aware that given a time series of acceleration measurements, you can integrate over time to obtain velocity as a function of time. Similarly, you can integrate velocity to obtain position. Of course, you need to know the initial velocity and position.

9. (**S**): Set VERBOSE_ACC to true (at the top of controller.c) and compile the robot's controller. You may set VERBOSE_GPS to false to avoid too much information in the console.

   Start the simulation on Webots and do not move the robot. You should see the values of the accelerometer in the terminal. You will see that the values are not zero. What is the accelerometer measuring?

10. (**B**): Imagine that the e-puck is in a free fall. What values of acceleration do you think the accelerometer will measure?

11. (**S**): Uncomment, the lines 124 – 128 in the main. This will make the robot stay static for five seconds. This time is used to estimate the bias of the accelerometer. The resulting mean values for the acceleration are stored in *_meas.acc_mean* in file *controller.c*. Compile the robot controller and start the simulation.

12. **(Q)**: Let your robot move forward in the x-axis direction (of Webots) using the keyboard (press R). Stop it before it reaches the end of the arena (press S). Do not turn the robot.

    A file *lab03.csv* has been created inside the controller folder. This file contains the logs of the simulation.
    Open the Matlab code *lab03/matlab/plot_main.m.* **Run only Part A** (by pressing Ctrl+Enter) to plot the values of the accelerometer.

    What is the frame of the accelerometer measurements? Which indexes of the accelerometer array (*acc*) correspond to the x-, y- and z-axis of the Webots world? Be sure to inspect the position property of the Robot node in the scene tree.

13. (**I**): Implement your odometry in X dimension (in the Webots coordinate frame) using the accelerometer data, in the MATLAB function *odo_acc.m*. Use the code *plot_main.m* (**only part B**), to plot your odometry. Start by writing the equations along X axis for the motion of the robot.

    What happens if you remove the mean (bias) before the integration?

14. (**I**): Implement your equations for the odometry in Webots, within the function `odo_compute_acc()` of the file *odometry.c* (see controller folder). Run the simulation and let your robot run in a straight line. Analyze the resulting logs in MATLAB using *plot_main.m* **(only part C)**. Do you obtain similar results to those previously obtained in MATLAB? If not, try to correct your code.

15. (Q): What happens if there is noise in the acceleration measurements?

16. (B): Bonus: Implement odometry in both X and Y dimensions.

3.2 Wheel Encoders

Wheel encoders measure the motion of the wheel of the robots, usually by counting the number of "steps" of rotation (i.e., discretized rotation angle). The real e-puck robot is equipped with a stepper motor, where one can command the motor to turn a specified number of steps.

Note the difference here. On the one hand, the real e-puck uses stepper motors, characterized by 1000 steps (increments) per wheel revolution; this type of motor is controlled in open-loop and provides information on the wheel position by itself. On the other hand, the simulated e-puck reproduces wheel encoders, sensing devices that in reality are combined with DC

motors for closed-loop control; they are physically separated from the motors but anchored to the rotating shaft to obtain the same increment counting functionality.

In both cases, what is known is the number of steps of turn of each wheel during a certain period of time. By counting them, you can calculate the position of the robot.

17. (**S**): Set VERBOSE_ENC (top of controller.c) to true and compile the robot's controller. Do a small run in the arena with the e-puck. What is the unit used by the wheel encoders and how to convert it to translation of the wheel in meters? Is it an absolute or a relative measurement of the wheel position?

18. (**Q**): You are given the incremental measurement of the encoders of both wheels at each time step, $\Delta_{enc,Left}$ and $\Delta_{enc,Right}$. (i.e., change in angle of the wheels between two time steps). Using this, write down the equations for forward velocity and rotational velocity of the robot, in frame B which is fixed to the robot.

    $v_B = \cdots$

    $\omega_B = \cdots$

Hint: The wheel encoders are expressed in radiants, use the WHEEL_RADIUS to obtain translation in meters. The distance between the two wheels is stored in WHEEL_AXIS. The rotational speed of the robot is positive when the right speed is higher than the left one.

19. (**Q**): Compute the rotation matrix from the body frame B to the Webots world frame A in order to obtain 2D velocity of the robot in the World frame. Try to complete the following equations.

$$\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} = R_B^A \begin{bmatrix} v_B \\ 0 \\ \omega_B \end{bmatrix}$$

$$R_B^A = \begin{bmatrix} \cdots & \cdots & 0 \\ \cdots & \cdots & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

    Hint: Use the orientation of the robot.

20. (**Q**): Using the previous equations, calculate the position of the robot (in world frame). Start by discretizing your equations and differentiating using the Euler Forward Method (T is the sampling time):

$$\dot{v}(t) = \frac{v_{t+1} - v_t}{T}$$

    $x_{t+1} = \cdots$

    $y_{t+1} = \cdots$

    $\theta_{t+1} = \cdots$

21. (**S**): Do a complete run around the square with the e-puck in order to collet some robot motion data.

22. (**I**): Implement your odometry equations in the MATLAB function `odo_enc.m`. Use the code *lab03/matlab/plot_main.m* **(only part D)**, to plot your odometry based on the wheel encoders.

23. (**S**): Try to reduce the deterministic errors by slightly changing the values of the *WHEEL_RADIUS* and *WHEEL_AXIS*. Your odometry curves should get closer to those obtained with the GPS.

24. (**I**): Implement your equations for the odometry in Webots in the function `odo_compute_encoders()` in file *odometry.c*. Run the simulation and again follow the square. Compare the resulting logs on MATLAB using *plot_main.m* (only part E). As before, you can change the values of the *WHEEL_RADIUS* and *WHEEL_AXIS* to improve your odometry.

25. **(Q):** Imagine that there is noise in the encoders, and that the wheels can slip. How will this affect the accuracy of the calculated odometry? Think about how you can account for such errors in your calculations.

4. Sensor Fusion with Kalman Filter

→ The lab session of next week will be a continuation of this exercise. You will have more time to do this part next week. Therefore, please make sure you have thoroughly finished all the previous questions

You have seen two different ways of localization – odometry and absolute positioning systems (GPS). You saw that GPS may not be available everywhere. You also saw that while odometry may work everywhere, noisy sensors and actuators result in errors accumulating up. How do you combine the benefits of both?

In this section, you will "fuse" both sources of information together. You will also see how to account for errors in sensors and actuators. We will add noise to both accelerometer and GPS to simulate reality.

In this exercise, you will implement a basic Kalman Filter in Matlab. However, you should be able to later implement the same in Webots.

26. **(S):** Open the file *lab03/matlab/kalman_filter.m* and run it. The file loads data from a robot run on a plane (i.e., 2D), containing GPS and accelerometer data. For convenience, both GPS and accelerometer data are in the world frame.

27. **(Q):** A refresher on uncertainty and Gaussian distributions:
    (a) Consider two 1D Gaussian random variable, x, y, with variance σ_x^2, σ_y^2. What is the variance in kx, where k is a constant? What about $x + y$? How about $kx + y$?
    (b) Consider an n-dimensional Gaussian random vector X with covariance matrix Σ_X. Let A be a constant matrix of dimension $n \times n$. What is the covariance of AX?

    Refresh your knowledge about uni- and multi-variate Gaussian distributions. Refer to the lecture about error propagation. We will use these concepts in the exercises below.

Kalman Filter is a framework for estimating an unknown variable (called "state") using a series of measurements containing statistical noise. It assumes that the relationship between the states and measurements is linear, and all errors and uncertainties follow a Gaussian distribution. It is the best possible linear-estimator.

This Linear-Gaussian assumption may not be perfectly true in real-world systems, but with some mathematical tricks and further assumptions, the Kalman Filter works well in practice. Following is the set of Kalman Filter Equations:

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

Note that μ is the state variable, Σ is the state uncertainty, A is the motion model or the process model, and C is the measurement model. R is the covariance representing motion/actuator noise. B is called the control matrix and u is the control or process input.

Each iteration of the Kalman filter consists of two steps – Process step (lines 2,3) and measurement step (lines 4,5,6). In this exercise, in the process step, you will use the odometry to compute the new robot state at a given time. Then, you will use measurements at that time to perform the measurement step to get the updated state.

28. **(Q)**: We wish to estimate the position and velocity of the robot. List down all the variables to be estimated and formulate a state variable for Kalman filter. What is the dimension of this state variable μ? What will be the dimension of the corresponding covariance matrix Σ?

Hint: Your state should contain position along x and y axes, as well as velocity along x and y axes. i.e., x, y, vx, vy.

29. (Q): Process model.
    (a) You know the state and acceleration measurement at time step k. Using this, calculate the individual elements of the state variables (position, velocity) at time step $k+1$. Answer in terms of dt, the duration of each time step.
    (b) You will use the process input u to represent acceleration a_k. Based on this, formulate the matrices A, B such that $\bar{\mu}_{k+1} = A\mu_k + Ba_k$.
    (c) Make sure the dimension of A you obtained is 4×4 and that of B is 4×2

30. (Q): What are the sources of process noise (R) in the model that we have built so far?

    We will take a highly simplified approach to incorporating the process noise in our Kalman Filter equations. We will assume that

$$R = dt \cdot \begin{bmatrix} 0.05 & 0 & 0 & 0 \\ 0 & 0.05 & 0 & 0 \\ 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0.01 \end{bmatrix}$$

    (Note multiplication with *dt*. Why did we do this?). In reality, this will be a function of the accelerometer noise and the duration of time step.

31. (Q): Measurement model. Your measurement consists of GPS position expressed as a 2x1 vector.

$$z = \begin{bmatrix} x_{GPS} \\ y_{GPS} \end{bmatrix}$$

    (a) Formulate the measurement matrix C such that you can "predict" the measurement using the state variable μ_k. $z_{predict} = C\mu_k$. What is the dimension of C?
    (b) Q is the covariance matrix representing measurement noise. If the GPS has independent error of variance 1 m in each direction, write down the value of Q.

32. (Q): Now that you calculated the various quantities, study the set of Kalman Filter Equations again. In particular, pay attention to computation of Kalman Gain, K in line 4, and its use in line 5. In essence, K functions as an adaptive weight, which will balance its "trust" in your process model vs. your measurements.

    (a) Without any actual calculations, can you tell what will happen if Q is extremely high? Will there be a big or a small change in the state variable in line 5?
    (b) What if Q is nearly zero?

    Hint: For easier intuition, assume all quantities are 1-dimensional, instead of vectors and matrices.

33. (I): In section A of the Matlab script kalman_filter.c (around lines 45-60), implement the process equations (corresponding to lines 2,3 of the Kalman Filter algorithm above). The boilerplate code has been provided to you. Run the script to generate the plots. Is this same as performing odometry?
    (a) Is this same as performing odometry?

(b) Why does the "disagreement" between true and estimated trajectory keep increasing?
(c) Does this increasing disagreement correspond to the values of the covariance matrix Σ? The first two diagonal elements of Σ are plotted. If not, why and what can you do about it?

34. (I): In Section B of the Matlab script kalman_filter.c (around lines 65-80), Implement the measurement equations (lines 4,5,6 of the algorithm above) in Matlab.
(a) If your implementation is correct, your estimated trajectory must be closer to the true one. What happens to the plot of the diagonal elements of the covariance matrix?
(b) You considered the situation of GPS outage in the odometry section earlier. With this sensor fusion implementation, what will be the difference?