Robot Localization and Bayesian Filtering

**Assignment Instructions:**

**Submission Process:** Submit your reports as a PDF document
For your reports, do not submit Word documents, raw text, or hardcopies etc. Make sure to generate and submit a PDF instead. On the first page of the PDF indicate your name, whether you are taking Failure to follow these rules will result in a lower grade in the assignment. If the assignment involves coding, you also need to submit a compressed file via Sakai, which contains your results and/or code as requested by the assignment.
**Late Submissions:** No late submissions are allowed. You will be awarded 0 points for late assignments.

**Precision:** Try to be precise in your description and thorough in your evaluation. Have in mind that you are trying to convince skeptical evaluators (i.e., computer scientists...) that your answers are correct.

**Problem 1. 10 points - Bayes filter** A vacuum cleaning robot is equipped with a cleaning unit to clean the floor. The robot has a binary sensor to detect whether a floor tile is clean or dirty. Neither the cleaning unit nor the sensor are perfect. From previous experience, you know that the robot succeeds in cleaning a dirty floor tile with a probability of

$$P(x_{t+1} = clean | x_t = dirty, u_t = vacuum\_clean) = 0.7$$

where $x_t$ is the state of a floor tile at time $t$, $u_t$ is the control action applied at $t$ and $x_{t+1}$ is the resulting state after the action has been applied. Also you know that vacuum-cleaning the tile when it is clean does not make it dirty.

The probability that the sensor indicates that the floor tile is clean although it is dirty is given by $p(z = clean | x = dirty) = 0.3$, and the probability that the sensor correctly detects a clean tile is given by $p(z = clean | x = clean) = 0.9$.

Unfortunately, you have no knowledge about the current state of the floor tile. However, after cleaning the tile, the sensor of the robot indicates that it is clean. Compute the probability that the floor tile is clean after the robot has vacuum-cleaned it. Assume a prior distribution for the state at time $t$ as $p(x_t = clean) = c$, where $0 \leq c \leq 1$.

**Problem 2. 15 points - Sampling from a Distribution** Use sampling and python (or any other programmatic way you see fit) to generate $N$ samples for the following cumulative distribution function in the range $x \in [-5, 5]$:

$$\Phi(x) = \frac{1}{2} + \frac{sign(x)}{2}\sqrt{1 - e^{-\frac{2x^2}{\pi}}}.$$

where $sign(x)$ is the sign of $x$, e.g., $sign(-1.6) = -1$, $sign(5.3) = 1$, and $sign(0) = 0$. Do not generates samples where $|x| > 5$. Plot a histogram of your data from $-5$ to 5 with $0.2$ increments (i.e., you should have 50 bins). Do this for $N = 100, 200, 500$, and 1000. You should submit four figures, one for each value of $N$.

**Problem 3. 45 points for CS460 - 25 points for CS560 students - Kalman Filter** If you are a CS560 student, you are encouraged to skip this problem and focus first on the implementation of the Extended Kalman Filter instead in Problem 4, which provides more points for you.

Suppose that there is a point mass moving in 2D with the system's state update equation being:

$$q_k = \begin{pmatrix} x_k \\ y_k \end{pmatrix} = \begin{pmatrix} x_{k-1} \\ y_{k-1} \end{pmatrix} + \begin{pmatrix} u_{1,k-1} \\ u_{2,k-1} \end{pmatrix} + \omega_{k-1},$$

where $q_k$ are the point mass coordinates, $u_k$ are the changes in the point mass coordinates given controls and $\omega_k$ is a noise vector. Furthermore, assume that it is also possible to collect observations $z_k$ regarding the robot's state $q_k$ with some noise $v_k$:

$$z_k = \begin{pmatrix} z_{x,k} \\ z_{y,k} \end{pmatrix} = \begin{pmatrix} x_k \\ y_k \end{pmatrix} + \nu_k.$$

Assume that the system noise $\omega$ is a zero mean Gaussian with the covariance matrix:

$$Q = \begin{pmatrix} 10^{-4} & 2 \times 10^{-5} \\ 2 \times 10^{-5} & 10^{-4} \end{pmatrix}$$

and the measurement noise is a zero mean Gaussian with the covariance matrix

$$R = \begin{pmatrix} 10^{-2} & 5 \times 10^{-3} \\ 5 \times 10^{-3} & 2 \times 10^{-2} \end{pmatrix}.$$

For your solution, you should provide: 1) in your report: the Kalman filter update equations for this system, 2) A python file named `kalman2d.py` that you need to implement to process the given data (a skeleton file will be provided). The

python program takes four additional arguments. The first argument is the data file name. The second and third arguments correspond to the initialization of the point mass' coordinates $\hat{x}_0$ and $\hat{y}_0$, respectively. The last argument is a value $\lambda$ that will be used to initialize the state covariance matrix $\Sigma_0$ as $\Sigma_0 = \lambda \cdot I$ in which $I$ is the identity matrix. After getting the predicted $(x_k, y_k)$ values, plot them as a set of 2D points and connect them sequentially using line segments to form a path. In the same plot, also visualize the observations (i.e., $(z_{x,k}, z_{y,k})$) using a different color.

We will provide a data file corresponding to Table 1. Your program should work for data different from this specific set.

Table 1: Sample control input and observations for running the Kalman filter.

| $k$ | $u_{1,k-1}$ | $u_{2,k-1}$ | $z_{x,k}$ | $z_{y,k}$ |
|---|---|---|---|---|
| 1 | 0.258286754 | -0.323660615 | 2.275352754 | 0.973676385 |
| 2 | 0.167658082 | -0.173943802 | 2.230101836 | 0.701036583 |
| 3 | 0.213711214 | -0.096513636 | 2.77765305 | 0.664235947 |
| 4 | 0.257826742 | -0.153303882 | 2.929903792 | 0.725217065 |
| 5 | 0.227367085 | -0.226875214 | 3.192038877 | 0.604513851 |
| 6 | 0.29707499 | -0.125012662 | 3.347010866 | 0.278565189 |
| 7 | 0.253380049 | 0.040957103 | 3.900306915 | 0.333608292 |
| 8 | 0.434690544 | -0.179539834 | 3.95920546 | 0.266202459 |
| 9 | 0.233358557 | -0.110426773 | 4.513029017 | 0.264158685 |
| 10 | 0.309397654 | -0.039241497 | 4.681395671 | 0.128679188 |
| 11 | 0.268100576 | -0.052630663 | 4.944550247 | -0.183817474 |
| 12 | 0.336295529 | 0.050916321 | 5.380023776 | 0.314923847 |
| 13 | 0.383568488 | -0.263521311 | 5.816818264 | -0.115412464 |
| 14 | 0.271740175 | -0.316833884 | 5.892258439 | -0.677581348 |
| 15 | 0.37813998 | -0.100109434 | 6.35130042 | -0.551171782 |
| 16 | 0.32708519 | -0.124288625 | 6.50389161 | -0.955727407 |
| 17 | 0.318597789 | -0.216869042 | 6.992389399 | -0.951967449 |
| 18 | 0.215326213 | -0.334744463 | 7.043205612 | -1.221487912 |
| 19 | -0.035040957 | -0.077543457 | 7.006712655 | -1.476399369 |
| 20 | 0.170197008 | -0.062307673 | 7.076019663 | -1.702334042 |
| 21 | 0.325171499 | -0.008000277 | 7.534015162 | -1.481437319 |
| 22 | 0.414100752 | -0.043273351 | 8.001067914 | -1.49148567 |
| 23 | 0.197609976 | -0.391999135 | 8.211005889 | -2.015420805 |
| 24 | 0.409390148 | -0.114261647 | 8.649643037 | -2.122464452 |
| 25 | 0.334974442 | -0.065947082 | 8.925195479 | -2.247646534 |

**Problem 4. 90 points - Non-Linear Landmark-based Localization** This problem asks you to localize a robot, which is moving in a planar environment with identifiable landmarks. The state $X_t$ of the robot is it's 2D position and orientation $< x_t, y_t, \theta_t >$. The input regarding the motion of the robot corresponds to the odometry-based motion model, i.e., we have access to odometry measurements $o =< \delta_{rot1}, \delta_{trans}, \delta_{rot2} >$. We assume that there are landmarks present in the robot's environment, which are always visible. The robot observes the bearing $b^l$ to each landmark $l$ in the workspace. You are asked to implement the following steps:

**A. [5 points] Setup the landmark maps and your visualization.** The workspace is a rectangular region $100 \times 100$ that contains the robot and $N$ point landmarks (consider $N$ among the values of 4,7,10). You should be able to visualize the robot, including showing its orientation (e.g., as a vector or as a rectangle of the size of your choice - the geometry is not too important for this project as we will not care about collisions), and the point landmarks. Feel free to reuse any visualization code you have from Assignment 2 or use supportive external libraries for visualization purposes. Provide examples of your visualization in your report and briefly decribe the implementation choices you made. Furthermore, generate and submit at least 3 "landmark_ID" data files, where each file has: i) on the first row, the number $N$ of landmarks in the environment, and ii) on each row afterwards, the coordinates of each landmark separated by space. The ID in the filename is just an index (i.e., 0, 1 or 2) to differentiate the landmark maps. Use different values $N$ for each landmark data file. Submit a file with the function that generates these landmark maps and indicate in your report how to call it. This function receives as input the number of landmarks $N$.

**B. [5 points] Setup the "ground truth" robot trajectories.** Implement a process for generating and saving "ground truth" robot trajectories $X =< X_0, X_1, \ldots, X_K >$ inside your workspace, i.e., the sequence of states $X_i =< x_i, u_i, \theta_i >$

that the robot trully visits. These trajectories should be fully contained inside the $100 \times 100$ workspace region you have defined in your visualization. Decompose these trajectories into a set of $K + 1$ states. Aim for $K = 100$ (if you face computational issues with your project, you can select lower values and explain this in your report).

You are free to choose "ground truth" trajectories of your choise. For instance, you can consider circular or rectangular trajectories of different dimensions, where the robot starts and ends at the same location and the $K$ states are equally spaced. Pick the robot's orientation along these trajectories in a reasonable way (e.g., for a circular trajectory, the robot's forward direction is tangent to the circle). Or you can generate random trajectories, where each new state $X_{i+1}$ is generated from the previous one $X_i$ given some random translations and rotations. For instance, given the random rotations and traslations for a state transition:

$$\delta_{rot1} = random(-max\_rotation, max\_rotation),$$

$$\delta_{trans} = random(0, max\_translation),$$

$$\delta_{rot2} = random(-max\_rotation, max\_rotation),$$

where you can select the maximum values for these translations and rotations. You can update the next state according to:

$$x_{i+1} = x_i + \delta_{trans} \cdot cos(\theta + \delta_{rot1}),$$

$$y_{i+1} = y_i + \delta_{trans} \cdot sin(\theta + \delta_{rot1}),$$

$$\theta_{i+1} = \theta_i + \delta_{rot1} + \delta_{rot2}.$$

If you generate random trajectories, make sure to ensure that the resulting states remain inside your 100x100 region, so you can visualize them properly.

Describe in your report the process you followed for generating these ground truth robot trajectories. Generate and submit at least 3 "ground_truth_ID" data files, where each one of them has: i) on the first row, the number $K + 1$ of robot states in the ground truth trahjectory and ii) on each consecutive row the coordinates $x_i$ $u_i$ $\theta_i$ of the robot's state $X_i$ along the ground trajectory separated by space. The ID in the filename is just an index (i.e., 0, 1 or 2) to differentiate the trajectories. Provide in your report visualizations of your ground truth trajectories generated by your software. The visualizations can correspond to a sequence of arrows, each one of them representing a robot state, or a visualization of the robot's geometry at each state. Submit a file with the function that generates these ground truth trajectories and indicate in your report how to call it. This function can receive as inputs either the parameters of predefined trajectories (e.g., the radiud of a circular trajectory) or the parameters of random trajectories (e.g., the $max\_rotation$ and $max\_translation$ parameters from the description above). Indicate in your report the interface of your function.

**C. [10 points] Define your input data for your robot localization challenges.** This involves a process for computing noisy "odometry" measurements and "observations" given your "ground truth" robot trajectories and landmark maps.

In particular, for each state transition $X_i$ to $X_{i+1}$ from a ground truth trajectory, compute the "ground truth" odometry as follows:

$$\delta_{rot1} = atan2(y_{i+1} - y_i, x_{i+1} - x_i) - \theta_i$$

$$\delta_{trans} = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$$

$$\delta_{rot2} = \theta_{i+1} - \theta_i - \delta_{rot1}$$

Given these ground truth odometry values, then compute the noisy odometry measurement $o_{i+1} =< \hat{\delta}_{rot1}, \hat{\delta}_{trans}, \hat{\delta}_{rot2} >$:

$$\hat{\delta}_{rot1} = sample\_normal\_distribution(\delta_{rot1}, \sigma^2_{rot1})$$

$$\hat{\delta}_{trans} = sample\_normal\_distribution(\delta_{trans}, \sigma^2_{trans})$$

$$\hat{\delta}_{rot2} = sample\_normal\_distribution(\delta_{rot2}, \sigma^2_{rot2})$$

Where the "sample_normal_distribution( $\mu, \sigma^2$ )" is a function that samples according to a Gaussian distribution with mean $\mu$ and variance $\sigma^2$. You can experiment with different values for $\sigma^2_{rot1}$ and $\sigma^2_{trans}$ (reasonable values will depend on the spacing between your robot states in the ground truth trajectories) but initial suggestions to consider are: $\sigma^2_{rot1} = \sigma^2_{rot2} = 0.05^2$ and $\sigma^2_{trans} = 0.1^2$.

From the "ground truth" robot states and the landmark maps you can compute the observations $Z =< z_1, \ldots, z_k >$, which correspond to the noisy bearings from each robot "ground truth" pose to each landmark. The first observation per trajectory is collected from state $X_1$, i.e., the second state in the trajectory. The "ground truth" bearing observation $b^j_i$ of landmark $l^j =< x^j_l, y^j_l >$ from state $X_i =< x_i, y_i, \theta_i >$ corresponds to:

$$b^j_i = atan2(y^j_l - y_i, x^j_l - x_i) - \theta_i.$$

Assume then a noise model for the bearing observations of landmarks with variance parameter $\sigma^2_{land}$, i.e., the actual, noisy bearing observation $\hat{b_i^j}$ corresponds to:

$$\hat{b_i^j} = sample\_normal\_distribution(b_i^j, \sigma^2_{land}).$$

You can experiment with different values for $\sigma^2_{land}$ but initial suggestions to consider are: $3^2$ in degrees (or $0.0523599^2$ in radian). Note to normalize angles in the $[0, 2\pi]$ range when you perform the above operations. Each landmark is unique and they are all visible from every state. Then, your observation is $z_i =< b_i^1, \ldots, b_i^L >$, i.e., the list of bearings to all $L$ landmarks as observed from state $X_i$.

For each pair of "landmark_ID" and "ground_truth_ID" data files with the same ID, your process should generate a "measurements_ID" data file, which contains the following information: i) at the top row, the initial pose of the robot $X_0$ (i.e., space separated $x_0$ $y_0$ $\theta_0$ coordinates); ii) the second row indicates the number $K-1$ corresponding to the available odometry measurements and landmark observations; iii) then, the rows alternate between reporting the odometry measurement $o_i$ (the odometry readings $\hat{\delta}_{rot1}$ $\hat{\delta}_{trans}$ $\hat{\delta}_{rot2}$ separated by space) and landmark observations $z_i$ (the bearings $\hat{b}_i^0$ $\hat{b}_i^1$ $\ldots \hat{b}_i^L$ separated by space). Generate and submit at least 3 such files that constitute the input to your localization challenges. Submit your script with the function that generates the measurement files when provided as input the landmark map and the ground truth trajectory files. Indicate in your report the name of the corresponding function and how to call it. In your report, plot for each localization input the difference between the "ground truth" odometry measurements $\delta_{rot1}, \delta_{trans}, \delta_{rot2}$ and the noisy ones $(\hat{\delta}_{rot1}, \hat{\delta}_{trans}, \hat{\delta}_{rot2})$ for each robot state. Similarly, plot the average difference between "ground truth" landmark bearings $b_i^j$ and the noisy ones $\hat{b}_i^j$ for each robot state $X_i$.

**D. [35 points] Solve the robot localization problem with a Particle Filter**. For each combination of a "landmark_ID" map and a "measurements_ID" data file, use a particle filter process to generate a "PF_estimate_ID" file containing the estimations made by the particle filter regarding the robot's state. The "PF_estimate_ID" data file has the same format as the "ground_truth_ID" data file, i.e., it is a $K+1$ sequence of robot states starting with the ground truth $X_0$ followed with the estimated states by the particle filter. Submit the corresponding estimation files. Submit your script with the function that generates the estimation files when provided as input the landmark map and the measurement files. Indicate in your report the name of the corresponding function and how to call it. Visualize the best estimate generated by the particle filter, together with the population of particles (just in terms of $x, y$ coordinates) and present them in comparison to the ground truth robot states. Execute the particle filter algorithm for different number of particles and plot the resulting error between the best estimate and the ground truth robot state. Discuss the resulting error in your report and the impact of different problem parameters (e.g., number of particles, number of landmarks, variance of the motion and observation models, etc.). You should not use an external library that already implements a Particle Filter.

A particle filter represents the robot's state distribution via a set of samples/particles. Refer to the lectures for a detailed explanation. Below is a summary of the approach given the notation of the assignment.

Each particle is an estimate of the robot's state. For each discrete time step $t$, the particle filter takes as input the set of particles from the previous time $p_t$ and first uses the noisy odometry measurements $o_{t+1}$ and the motion model to propagate these particles forward to generate candidate particles for the next step $\bar{p}_{t+1}$. In particular, given the $j$-th particle $p_t^j =< x_t^j, y_t^j, \theta_t^j >$ and the odometry measurement $o_{t+1} =< \hat{\delta}_{rot1}, \hat{\delta}_{trans}, \hat{\delta}_{rot2} >$, use the motion model to generate the particle $\bar{p}_{t+1}^j$, i.e., first sample guesses for the true odometry by sampling in the vicinity of the odometry measurements:

$$\bar{\delta}_{rot1} = sample\_normal\_distribution(\hat{\delta}_{rot1}, \sigma^2_{rot1})$$

$$\bar{\delta}_{trans} = sample\_normal\_distribution(\hat{\delta}_{trans}, \sigma^2_{trans})$$

$$\bar{\delta}_{rot2} = sample\_normal\_distribution(\hat{\delta}_{rot2}, \sigma^2_{rot2})$$

Then, use these guesses to define the state estimate of the new particle:

$$\bar{x}_{i+1}^j = x_i^j + \bar{\delta}_{trans} \cdot cos(\theta_i^j + \bar{\delta}_{rot1})$$

$$\bar{y}_{i+1}^j = y_i^j + \bar{\delta}_{trans} \cdot sin(\theta_i^j + \bar{\delta}_{rot1})$$

$$\bar{\theta}_{i+1}^j = \theta_i^j + \bar{\delta}_{rot1} + \bar{\delta}_{rot2}$$

These particles are then weighted based on the likelihood of the observed measurements $z_{t+1}$, i.e., for each particle $\bar{p}_{t+1}^j$ you have to estimate the probability $P(z_{t+1}|\bar{p}_{t+1}^j) = P(\hat{b}_{t+1}^0, \hat{b}_{t+1}^1, \ldots \hat{b}_{t+1}^L \,|\bar{p}_{t+1}^j)$. You can assume that the landmark observations are conditionally independent one from each other given an estimate of the robot's state, so the above probability becomes $\Pi_l P(\hat{b}_{t+1}^l|\bar{p}_{t+1}^j)$. To compute each probability per landmark bearing, you can first compute the expected landmark bearing $\bar{b}_{t+1}^{l,j}$ assuming the particle was the true robot state:

$$\bar{b}_{t+1}^{l,j} = atan2(y_l - \bar{y}_{t+1}^j, x_l - \bar{x}_{t+1}^j) - \bar{\theta}_{t+1}^j.$$

Then, $P(\hat{b}_{t+1}^l | \bar{p}_{t+1}^j)$ is equal to the probability of observing $\hat{b}_{t+1}^l$ assuming that the ground truth observation is $\bar{b}_{t+1}^{l,j}$ and the observation model follows a Gaussian distribution with variance $\sigma_{land}^2$.

Once the weight of each particle has been generated, a new set of particles are sampled from the weighted set. See the implementation of the resampling procedure from the lectures. These particles are used as the estimates for the state distribution at the new step $p_{t+1}$. Use the mean value of these particles as the best estimate at each step $t+1$.

**E. [35 points] Solve the robot localization problem with an Extended Kalman Filter**. For each combination of a "landmark_ID" map and a "measurements_ID" data file, use the Extended Kalman Filter (EKF) to generate an "EKF_estimate_ID" file containing the estimations made by the EKF regarding the robot's state. The "EKF_estimate_ID" data file has the same format as the "ground_truth_ID" data file, i.e., it is a $K+1$ sequence of robot states starting with the ground truth $X_0$ followed with the estimated states by the EKF. Submit the corresponding estimation files. Submit your script with the function that generates the estimation files when provided as input the landmark map and the measurement files. Indicate in your report the name of the corresponding function and how to call it. Visualize the best estimate generated by the EKF, together with the corresponding uncertainty as an ellipsoid (just in terms of $x, y$ coordinates) and present them in comparison to the ground truth robot states. Plot the resulting error between the best estimate and the ground truth robot state. Discuss the resulting error in your report and the impact of different problem parameters (e.g., number of landmarks, variance of the motion and observation models, etc.). Compare the quality of the result achieved by the EKF against your particle filter implementation as well as the computational cost of each approach. You should not use an external library that already implements an Extended Kalman Filter.

The Extended Kalman Filter estimates a Gaussian approximation of the robot state at each time $N(\mu_{t+1}, \Sigma_{t+1})$, based on the distribution at the previous time $N(\mu_t, \Sigma_t)$, the measured odometry $(o_{t+1})$ and the observation $(z_{t+1})$. Refer to the lectures for a detailed explanation. Below is a summary of the approach given the notation of the assignment.

First generate a prediction of the next mean $\bar{\mu}_{t+1} =< \bar{x}_{i+1}, \bar{y}_{i+1}, \bar{\theta}_{i+1} >$ by using the motion model, i.e., use the odometry readings $< \hat{\delta}_{rot1}, \hat{\delta}_{trans}, \hat{\delta}_{rot2} >$ to generate the predicted mean:

$$\bar{x}_{i+1} = x_i + \hat{\delta}_{trans} \cdot cos(\theta_i + \hat{\delta}_{rot1})$$

$$\bar{y}_{i+1} = y_i + \hat{\delta}_{trans} \cdot sin(\theta_i + \hat{\delta}_{rot1})$$

$$\bar{\theta}_{i+1} = \theta_i + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$$

Then, generate the predicted covariance matrix $\bar{\Sigma}_{t+1}$ according to $\bar{\Sigma}_{t+1} = G_{t+1}\Sigma_t G_{t+1}^T + V_{t+1}M_t V_{t+1}^T$, where:

- $\Sigma_t$ is the covariance matrix computed by the previous iteration of the EKF;
- $G_{t+1}$ is the state Jacobian, i.e., each row $i$ and column $j$ of this matrix expresses the partial derivative of how the $i$-th state variable at the next step depends on the $j$-th state variable at the previous step according to the above motion model;
- $M_t$ is the covariance matrix of the motion model's noise, i.e., a diagonal matrix, which stores the variance terms $\sigma_{rot1}^2, \sigma_{trans}^2, \sigma_{rot2}^2$ of the motion model on the diagonal.
- and $V_{t+1}$ is the noise Jacobian, i.e., each row $i$ and column $j$ of this matrix expresses the partial derivative of how the $i$-th state variable at the next step depends on the $j$-th noise term according to the above motion model.

Once you have the predicted distribution $N(\bar{\mu}_{t+1}, \bar{\Sigma}_{t+1})$, the next step is to use the landmark observations. First, you compute the expected observation $\bar{z}_{t+1}$ from the predicted mean state $\bar{\mu}_{t+1}$, i.e., for each landmark $l$, you can computed the expected bearing:

$$\bar{b}_{t+1} = atan2(y_l - \bar{y}_{t+1}, x_l - \bar{x}_{t+1}) - \bar{\theta}_{t+1}.$$

You can then compute, the next state mean according to: $\mu_{t+1} = \bar{\mu}_{t+1} + K_{t+1}(z_{t+1} - \bar{z}_{t+1})$, where $K_{t+1}$ is the Kalman gain matrix regulating how much the next mean should depend on the predicted mean $\bar{\mu}_{t+1}$ versus being influenced by the difference between the true observation $z_{t+1}$ and the expected observation $\bar{z}_{t+1}$. The expression for the Kalman gain is: $K_{t+1} = \bar{\Sigma}_{t+1}H_{t+1}^T(H_{t+1}\bar{\Sigma}_{t+1}H_{t+1}^T + Q_{t+1})$, where:

- $\bar{\Sigma}_{t+1}$ is the predicted covariance;
- $H_{t+1}$ is the observation Jacobian, i.e., each row $i$ and column $j$ of this matrix expresses the partial derivative of how the $i$-th observation variable depends on the $j$-th state variable;
- $Q_t$ is the covariance matrix of the observation model, i.e., a diagonal matrix, which stores the variance terms $\sigma_{land}^2$ along the diagonal.

Given the Kalman gain, it is also possible to compute the next state co-variance as $\Sigma_{t+1} = (I - K_{t+1}H_{t+1})\bar{\Sigma}_{t+1}$. You can use the mean $\mu_{t+1}$ as the best estimate of the Kalman filter at each step $t+1$ but you will keep track of both the mean and the covariance matrix at each step.