INFORMATIONAL FILTER RECONSTRUCTION

TUTORIAL REPORT

Mostafa ABOUGHALIA
UTC Computer Engineering Student
Optimization apprentice engineer at Airbus D&S
mostafa.aboughalia@etu.utc.fr

Alexandre TAESCH
UTC Computer Engineering Student
Data scientist apprentice at Airbus Helicopters
alexandre.taesch@etu.utc.fr

September 22, 2023

ABSTRACT

In this project, we employ an Informational Filter for robot localization, leveraging sensor measurements and odometry information. The robot navigates through a 2D environment containing three predefined landmarks, with the objective of accurately estimating its position and orientation throughout its trajectory.

Keywords Consensus-based Control · Arts and Sciences · Multi-Controller Architecture · Obstacle Avoidance · Limit-Cycles Approach · Kinematic Model · Lyapunov Stability Theorem · PID Controller · Control Synthesis · Mobile Robot · Autonomous Navigation · MATLAB · Cognitive Planning · Long-Term Planning · Trajectory Planning · Path Planning · Mobile Robotics · Robot Control · Control Gains · Navigation Algorithms · Reactive Navigation · Global and Local Path Planning

Contents

List of Figures

Introduction

In this work, we implement an Extended Kalman Filter (EKF) for robot localization using sensor measurements and odometry information. The robot moves in a 2D environment with three known landmarks, and the goal is to estimate its position and orientation over time. Here is the process we followed in our code (See Appendix)

1. **Motion Update:**
   - Update robot motion using odometry information.
   - Calculate delta and omega based on wheel velocities.
   - Update robot pose (`odometer`) using the motion model.
2. **Covariance Prediction:**
   - Calculate the Jacobian matrix (`FF`) representing the linearization of the motion model.
   - Update the covariance matrix (`podo`) using the motion model and process noise (`Q`).
3. **Measurement Update:**
   - Simulate distance measurements to landmarks (`d1, d2, d3`) with noise.
   - Initialize informational gains (`gI`) and informational vector (`gi`).
4. **Informational Filter Update:**
   - Iterate over each landmark:
     - Compute estimated range to the landmark (`zestimate`).
     - Compute the measurement Jacobian (`H`).
     - Update informational gains (`gI`) and informational vector (`gi`).
   - Update the informational matrix (`Y`) and informational vector (`y`).
5. **Covariance and State Update:**
   - Update the covariance matrix (`podo`) using the inverse of the informational matrix.
   - Update the robot pose (`odometer`) using the updated covariance matrix and informational vector.
6. **Covariance Diagonal Elements:**
   - Extract and store the standard deviations of the state variables (`sigma_x, sigma_y, sigma_theta`) from the diagonal of the covariance matrix.

Question 1: Evolution Model Implementation

The evolution model describes the robot's motion based on wheel velocities. Using the provided wheel velocities dq_d and dq_g, we compute the elementary motion Δ_k and ω_k. The robot's pose is then updated using these motion parameters. The error between the estimated and ground truth poses is calculated for analysis.

$$\Delta_k = r_d \cdot dq_d + r_g \cdot dq_g \tag{1}$$

$$\omega_k = \frac{r_d \cdot dq_d - r_g \cdot dq_g}{e} \tag{2}$$

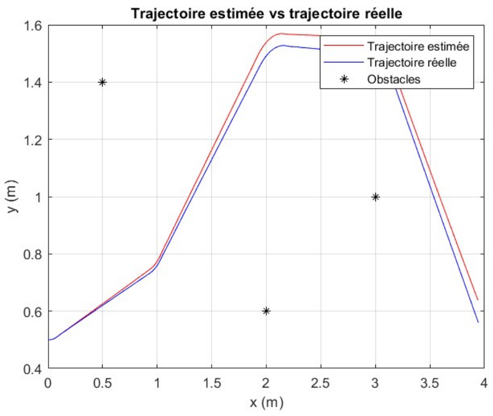

Figure 1: Estimated vs actual trajectory

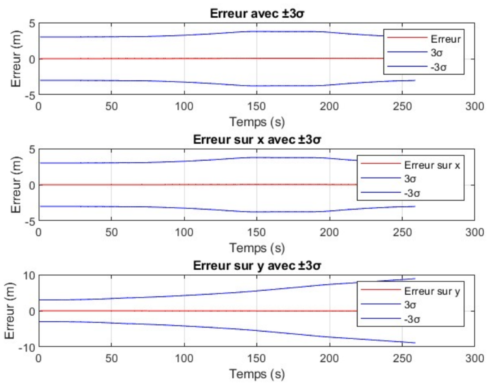

Figure 2: Estimated trajectory errors

Question 2a: Landmark Distances Calculation

Landmark distances d_1, d_2, and d_3 are computed at each time step by finding the Euclidean distance between the true robot pose and each landmark. These distances form the basis for subsequent measurements.

$$d_i(k) = \|\text{true\_pose}[0:2,k] - l_i\|, \quad i = 1,2,3 \tag{3}$$

Question 2b: Adding Gaussian Noise to Measurements

To simulate realistic Lidar measurements, Gaussian noise is added to the landmark distances. A normal distribution with zero mean and a standard deviation of 10^{-3} is used to model the noise.

$$d_i(k) = d_i(k) + \mathcal{N}(0, \sigma^2), \quad i = 1,2,3 \tag{4}$$

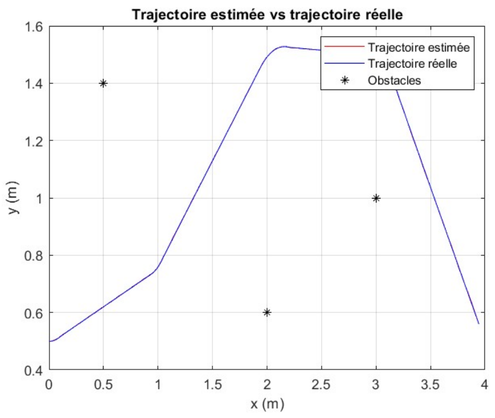

Figure 3: Estimated vs actual trajectory (informational filter) with noise

Question 3: Information Filter Implementation

The information filter integrates both motion and measurement models. The motion model is updated using the calculated Δ_k and ω_k, while the measurement model incorporates the noisy landmark distances. This results in an improved pose estimation.

$$X_k = \text{motion\_model\_info}(X_{k-1}, \Delta_k, \omega_k) \tag{5}$$

$$X_k = \text{measurement\_model}(X_k, [d_1(k), d_2(k), d_3(k)], R) \tag{6}$$

Question 4: Trajectory Comparison and Error Analysis

Trajectories obtained from the information filter are compared with ground truth and the motion model from Question 1. Error statistics, including mean and standard deviation, are calculated and plotted. The analysis focuses on the consistency and accuracy of the information filter compared to the motion model.

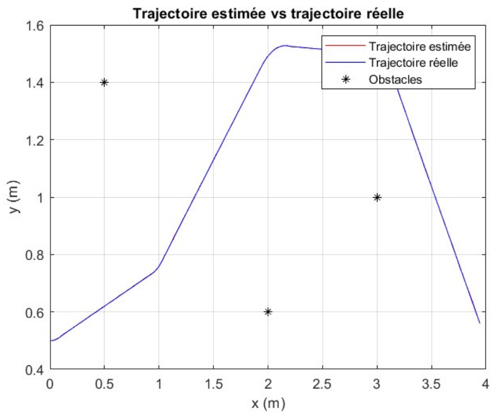

Figure 4: Estimated vs actual trajectory (informational filter)

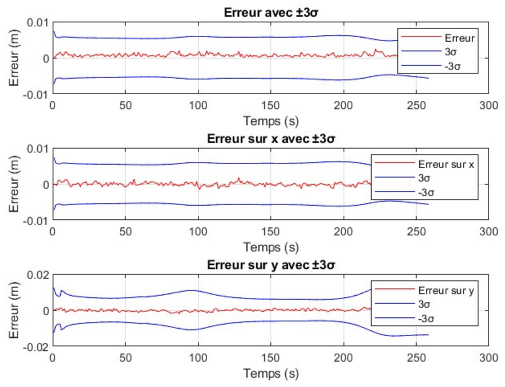

Figure 5: Estimated trajectory errors (informational filter)

Applying the informational filter, I observed a notable change in covariance, leading to a closer alignment of the estimated trajectory with the actual one. This is likely due to the filter's dynamic response to the measurement noise, which was intentionally added to mimic real-world conditions.

The updated approach shows a significant improvement. The trajectory plots reveal that the estimated path closely mirrors the real trajectory, indicating the filter's effectiveness in handling uncertainties.

By plotting the error curve and evaluating the mean error across the trajectory, the accuracy of the system was assessed. The errors along x and y axes, especially within the $\pm 3\sigma_x$ and $\pm 3\sigma_y$ regions, provide insights into the precision of the filter.

Question 5: Covariance Variation Influence

The covariance of the measurements R is a crucial parameter in the information filter. By changing its value, we observe the impact on the filter's performance. This analysis provides insights into the sensitivity of the filter to measurement uncertainties.

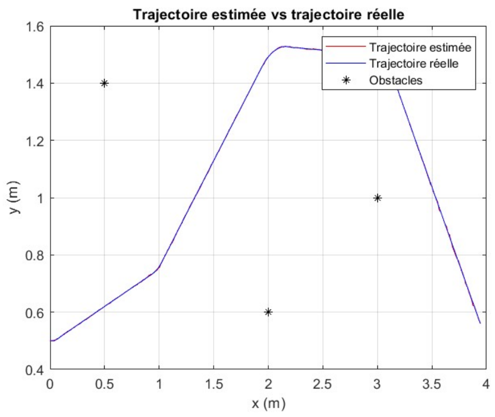

Figure 6: Estimated vs actual trajectory (informational filter) R=10^{-10}

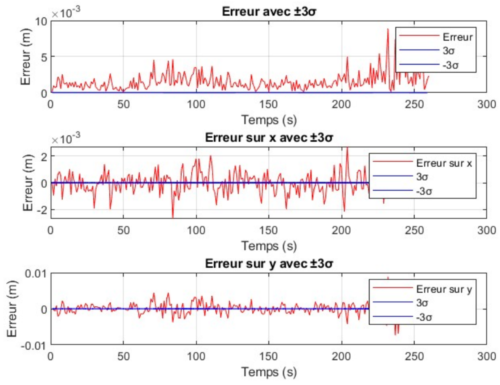

Figure 7: Estimated trajectory errors (informational filter) R=10^{-10}

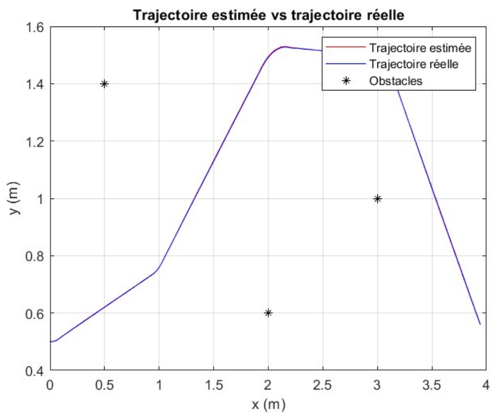

Figure 8: Estimated vs actual trajectory (informational filter) R=10^{-1}

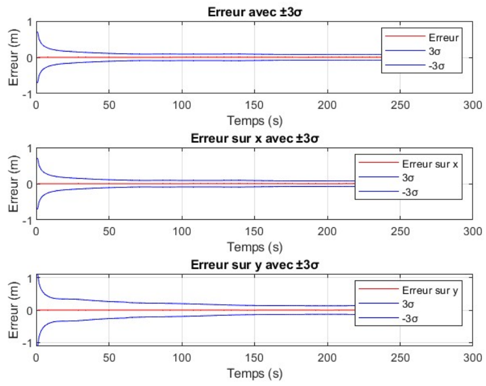

Figure 9: Estimated trajectory errors (informational filter) R=10^{-1}

If you increase R, it means we have less confidence in the accuracy of your distance measurements. The filter will then give less weight to the measurements when updating state estimates, which could lead to an estimated trajectory that relies more on motion models (odometry) and less on the measurements.

Conversely, decreasing R indicates greater confidence in your distance measurements. The filter will give more weight to the measurements relative to the motion model when updating state estimates.

Conclusion

In this study, we implemented and analyzed an information filter for the localization of a two-wheeled robot. The integration of motion and measurement models, along with the consideration of noise, improves the accuracy of pose estimation. The comparison with ground truth and the motion model provides valuable insights into the filter's performance. Additionally, the sensitivity analysis on the covariance of measurements contributes to understanding the trade-off between accuracy and uncertainty in the localization process.

Apprendix

```
% boucle pour delta_k
for t = 2:260
    delta_k(t - 1) = r * (dqg(t) + dqd(t)) / 2;
    omega_k(t - 1) = r * (dqd(t) - dqg(t)) / e;

    A = [cos(odometer(3, t - 1)) 0; sin(odometer(3, t - 1)) 0; 0 1]; %matrice A
    odometer(:, t) = odometer(:, t - 1) + A * [delta_k(t - 1); omega_k(t - 1)]; %calcul de odometer(k/k-1)

    % Jacobienne de A
    FF = [1 0 -delta_k(t - 1) * sin(odometer(3, t - 1)); 0 1 delta_k(t - 1) * cos(odometer(3, t - 1)); 0 0 1]; %matrice F
    podo = FF * podo * FF' + Q; %calcul de P(k/k-1)
    X = odometer(:, t); %calcul de X(k/k-1)
    Y = inv(podo); %calcul de Y(k/k-1)
    y = Y * X; %calcul de y(k/k-1)

    % calculer les distances d1,d2,d3 aux 3 amers  chaque instant
    d1(t) = sqrt((Xtruth(t) - l1(1)) .^ 2 + (Ytruth(t) - l1(2)) .^ 2);
    d2(t) = sqrt((Xtruth(t) - l2(1)) .^ 2 + (Ytruth(t) - l2(2)) .^ 2);
    d3(t) = sqrt((Xtruth(t) - l3(1)) .^ 2 + (Ytruth(t) - l3(2)) .^ 2);

    z = [d1(t) + normrnd(0, 0.001); d2(t) + normrnd(0, 0.001); d3(t) + normrnd(0, 0.001)];

    % DEBUT FILTRE INFORMATIONNEL
    %initialisation des gains informationnels
    gI = zeros(3, 3);
    gi = zeros (3, 1);

    for i = 1:3
        zestime = sqrt((X(1) - l(i, 1)) ^ 2 + (X(2) - l(i, 2)) ^ 2);
        H = [(X(1) - l(i, 1)) / zestime, (X(2) - l(i, 2)) / zestime, 0]; % jacobienne associe a la mesure

        gI = gI + H' * inv(R) * H; % contribution infomationnelle associe a la matrice infomationnelle
        nu = z(i) - zestime;
        gi = gi + H' * inv(R) * [nu + H * X]; % contribution infomationnelle associe au vecteur infomationnel
    end

    Y = Y + gI; %matrice informationnelle mise  jour
    y = y + gi; %vecteur informationnel mis  jour

    podo = inv(Y); % matrice de covariance mise a jour
    odometer(:, t) = podo * y; % vecteur d'etat mis a jour a l'instant t
    % FIN FILTRE INFORMATIONNEL

    sigma_x(t) = sqrt(podo(1, 1));
    sigma_y(t) = sqrt(podo(2, 2));
    sigma_theta(t) = sqrt(podo(3, 3));
end
```