

Localization and Positioning Exercitation

Veronica Campana

Recap Positioning Problem

Positioning

Recalling the model for the measurement:

$$b_i = h_i(\mathbf{q}) + w_i.$$

Assuming $w_i \sim \mathcal{N}(0, \sigma_i^2)$, the model can be given by:

$$h_i(\mathbf{q}) = \rho_i = \sqrt{(\mathbf{q} - \mathbf{s}_i)^T(\mathbf{q} - \mathbf{s}_i)},$$

where q is the position to be found, s_i is the position of the i -th anchor, and ρ_i is the distance between the i -th anchor and the tag.

$$\mathbf{q} = \begin{bmatrix} x \\ y \end{bmatrix}, \quad \mathbf{s}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix}.$$

Positioning

Since we have m base stations, we have the vectorial representation:

$$\mathbf{b}_m = h(q) + \mathbf{w}_m,$$

where, as in the previous cases,

$$\mathbf{b}_m = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}, \quad h(\mathbf{q}) = \rho = \begin{bmatrix} h_1(\mathbf{q}) \\ h_2(\mathbf{q}) \\ \vdots \\ h_m(\mathbf{q}) \end{bmatrix}.$$

For **range uncertainties** we have:

$$\mathbb{E}[\mathbf{w}_m] = \mathbf{0}, \quad \text{Cov}[\mathbf{w}_m] = \begin{bmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_m^2 \end{bmatrix}.$$

Static Estimator

The objective of an estimator is to retrieve the estimate \hat{q} and possibly a measure of its uncertainty. For the ToA approach, static estimators can be:

- **Nonlinear Weighted Least Squares** minimizing:

$$J(\hat{q}) = \sum_{i=1}^m \frac{(b_i - \sqrt{(\hat{x} - x_i)^2 + (\hat{y} - y_i)^2})^2}{\sigma_i^2};$$

- **Maximum Likelihood**, which turns out to be a Nonlinear Weighted Least Squares in the case of Gaussian noises;
- **Least Squares** on a linearized version of the measurements.

Static Estimator

For the latter case, let us consider the squares of the measurements:

$$b_i^2 = (\rho_i + w_i)^2 = (\hat{x} - x_i)^2 + (\hat{y} - y_i)^2 + w_i^2 + 2w_i\rho_i = (\hat{x} - x_i)^2 + (\hat{y} - y_i)^2 + \eta_i.$$

where

$$\eta_i = w_i^2 + 2w_i\rho_i.$$

Expanding the squares, we have:

$$b_i^2 = \hat{x}^2 - 2x_i\hat{x} + x_i^2 + \hat{y}^2 - 2y_i\hat{y} + y_i^2 + \eta_i.$$

Static Estimator

By defining $r = x^2 + y^2$, we can rewrite the previous expression as:

$$b_i^2 - x_i^2 - y_i^2 = r - 2xx_i - 2yy_i + \eta_i.$$

By defining:

$$\mathbf{b}_m^* = \begin{bmatrix} b_1^2 - x_1^2 - y_1^2 \\ b_2^2 - x_2^2 - y_2^2 \\ \vdots \\ b_m^2 - x_m^2 - y_m^2 \end{bmatrix}, \quad \theta = \begin{bmatrix} x \\ y \\ r \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} -2x_1 & -2y_1 & 1 \\ -2x_2 & -2y_2 & 1 \\ \vdots & \vdots & \vdots \\ -2x_m & -2y_m & 1 \end{bmatrix},$$

m is the number of anchor

we can rewrite the previous expression as:

$$\mathbf{b}_m^* = \mathbf{H}\theta. \quad \text{Matrix form for the measurement model}$$

Measurement Uncertainty

The measurement uncertainty is now not only related to the noise of the measurements but also to the distance between the target and the i th anchor. The noise is given by:

$$\eta_i = w_i^2 + 2w_i\rho_i.$$

In this case, if we compute the expected value of the noise we see that $\mathbb{E}[\eta_i] = w_i^2$, meaning that a bias is present. However, assuming the noise small enough we have $w_i^2 \approx 0$, the noise is given by $\eta_i = 2w_i\rho_i$ which yields to $\mathbb{E}[\eta_i] \approx 0$.

Note that since the noise is a function of the distance, it can cause problems when the target moves.

Static Estimator

Notice that in the previous example both the estimates $\hat{\mathbf{q}}$ and \hat{r} are derived. Since $\hat{r} = \hat{\mathbf{q}}^T \hat{\mathbf{q}}$, this constraint should be enforced and hence **constrained Least Squares** solutions should be applied.

Alternatively, we can get rid of r by computing $b_i^2 - b_j^2$, i.e.,

$$b_i^2 - b_j^2 - x_i^2 + x_j^2 - y_i^2 + y_j^2 = -2(x_i - x_j)x - 2(y_i - y_j)y + \eta_i - \eta_j.$$

As in the previous case, we can derive the matrix formulation for this problem as well, which, again, requires just the **Least Squares** to be computed.

Solution using Linear Algebra

Using linear algebra is possible to find the solution of the **Least Square** inverting the matrix H

Linear measurement model

$$\mathbf{b} = \mathbf{H}\mathbf{p} \rightarrow \mathbf{p} = \mathbf{H}^{-1}\mathbf{b}.$$

It is important to consider the shape of the matrix H

This is true if the matrix \mathbf{H} is square and full rank. If the matrix is rectangular, the pseudo-inverse should be computed instead of the inverse with the Moore-Penrose formula $\mathbf{H}^+ = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T$.

Using the trick of the previous slide we can rewrite the matrix b and H for the case with 3 anchors as

$$\bar{\mathbf{b}} = \begin{bmatrix} b_1^2 - b_2^2 - x_1^2 + x_2^2 - y_1^2 + y_2^2 \\ b_2^2 - b_3^2 - x_2^2 + x_3^2 - y_2^2 + y_3^2 \end{bmatrix}, \quad \bar{\mathbf{H}} = \begin{bmatrix} -2x_1 + 2x_2 & -2y_1 + 2y_2 \\ -2x_2 + 2x_3 & -2y_2 + 2y_3 \end{bmatrix}.$$

So the solution can be computed as:

$$\bar{\mathbf{b}} = \bar{\mathbf{H}}\mathbf{p} \rightarrow \mathbf{p} = \bar{\mathbf{H}}^{-1}\bar{\mathbf{b}}.$$

Noise Covariance Matrix

The vector of noises corresponding to the solution presented before is

$$\bar{\mathbf{w}}_m = \begin{bmatrix} \eta_1 - \eta_2 \\ \eta_2 - \eta_3 \\ \vdots \\ \eta_{m-1} - \eta_m \end{bmatrix}.$$

The uncertainties are

$$\mathbb{E}[\bar{\mathbf{w}}_m] = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \text{Cov}[\bar{\mathbf{w}}_m] = \begin{bmatrix} 4\rho_1^2\sigma_1^2 + 4\rho_2^2\sigma_2^2 & -4\rho_2^2\sigma_2^2 & 0 & \dots & 0 \\ -4\rho_2^2\sigma_2^2 & 4\rho_2^2\sigma_2^2 + 4\rho_3^2\sigma_3^2 & -4\rho_3^2\sigma_3^2 & \dots & 0 \\ 0 & -4\rho_3^2\sigma_3^2 & 4\rho_3^2\sigma_3^2 + 4\rho_4^2\sigma_4^2 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 4\rho_{m-1}^2\sigma_{m-1}^2 + 4\rho_m^2\sigma_m^2 \end{bmatrix}.$$

Covariance Matrix of w_1, w_2, w_3

Definitions

Let:

- $w_1 = \eta_1 - \eta_2$
- $w_2 = \eta_2 - \eta_3$
- $w_3 = \eta_3 - \eta_4$

Assume η_i are independent random variables with variances σ_i^2 .

Variances (Diagonal Elements)

$$\text{Var}(w_1) = \sigma_1^2 + \sigma_2^2$$

$$\text{Var}(w_2) = \sigma_2^2 + \sigma_3^2$$

$$\text{Var}(w_3) = \sigma_3^2 + \sigma_4^2$$

Covariances (Off-Diagonal Elements)

1. $\text{Cov}(w_1, w_2)$:

$$\text{Cov}(w_1, w_2) = \mathbb{E}[(\eta_1 - \eta_2)(\eta_2 - \eta_3)] = \mathbb{E}[\eta_1\eta_2] - \mathbb{E}[\eta_2^2] - \mathbb{E}[\eta_1\eta_3] + \mathbb{E}[\eta_2\eta_3] = -\mathbb{E}[\eta_2^2] = -\sigma_2^2$$

2. $\text{Cov}(w_2, w_3)$:

$$\text{Cov}(w_2, w_3) = -\sigma_3^2$$

3. $\text{Cov}(w_1, w_3)$:

$$\text{Cov}(w_1, w_3) = 0$$

Covariance Matrix

The covariance matrix \mathbf{w} is:

$$\mathbf{w} = \begin{bmatrix} \sigma_1^2 + \sigma_2^2 & -\sigma_2^2 & 0 \\ -\sigma_2^2 & \sigma_2^2 + \sigma_3^2 & -\sigma_3^2 \\ 0 & -\sigma_3^2 & \sigma_3^2 + \sigma_4^2 \end{bmatrix}.$$

Interpretation

- Diagonal entries: Variances of w_1, w_2, w_3 .
- Off-diagonal entries: Covariances due to shared terms.
- Non-adjacent w_i : Uncorrelated because η_i are independent.

Matlab Implementation

Trilateration Function

As seen previously the positioning problem can be solved with the linear algebra. We can implement a function that given the positions of the anchors and the distances to the target, it returns the estimated position of the target. Complete the code below where you read **TODO**.

```
% Define the positions of the anchors (x, y)
anchors = [0, 0; 10, 0; 5, 8.66]; % Example positions for 3 anchors

% Define the true position of the target (our ground truth, which is not
  ⇨ known and it's the position that we want to estimate)
master_true_position = [4, 2];

% Estimate position using 3 anchors : CALL THE trilateration() FUNCTION,
  ⇨ WHICH ARE THE INPUT AND THE OUTPUT OF THE FUNCTION?

% Check if the matrix S is invertible: TODO

estimated_position_3 = ;
disp('Estimated position with 3 anchors:');
disp(estimated_position_3);

% trilateration function: TODO
```

```

function = trilateration(anchors, distances)
    % Number of anchors
    n = size(anchors, 1);

    % Initialize matrices
    H = zeros(n-1, 2);
    b = zeros(n-1, 1);

    % Iterate over all anchors
    for i = 1:n-1
        % TODO
    end
end
end

```

Increase the number of anchors

```

% Add more anchors for multiple anchor localization
anchors = [anchors; 2, 7; 8, 3]; % Adding more anchors

% Estimate position using multiple anchors : CALL THE trilateration()
↪ FUNCTION
%TODO

% Moore-Penrose pseudoinverse: TODO

estimated_position_multi = ;
disp('Estimated position with multiple anchors:');
disp(estimated_position_multi);

% trilateration function: TODO
function = trilateration(anchors, distances)
    % Number of anchors
    n = size(anchors, 1);

    % Initialize matrices
    H = zeros(n-1, 2);
    b = zeros(n-1, 1);

    % Iterate over all anchors
    for i = 1:n-1

```

```

        % TODO
    end
end

```

Effect of Noise on the Estimation

We start seeing which is the effect of the noise on the estimation. In the first part, we will see the effect of the noise on the estimation of the target position. Complete the code below where you read TODO.

```

clear;
clc;

% Define the positions of the anchors (x, y)
anchors = [2, 0; 7, 1; 5, 4]; % Example positions for 3 anchors
n_anchor = size(anchors, 1);
% Define the true position of the target (our ground truth, which is not
↳ known and it's the position that we want to estimate)
master_true_position = [4, 2];

% Calculate distances from the target to each anchor
distances = % TODO

% What happens if we add noise to the distances?
% We add zero-mean Gaussian noise with standard deviation of 0.1 for 5 times
↳ to the distances

% Number of noisy measurements
n_mes = 5;

% Initialize the noisy distances
distances_noisy = zeros(n_mes, n_anchor);
for i = 1:n_mes
    % Add noise to the distances
    % TODO
end

% Compute the estimated position
estimated_position = zeros(n_mes, 2);
for i = 1:n_mes
    % Compute the estimated position using the noisy distances

```

```

    % TODO
end

% Plot the estimated positions
figure;
hold on;
plot(anchors(:,1), anchors(:,2), 'ro', 'MarkerSize', 10, 'DisplayName',
     ↪ 'Anchors');
plot(estimated_position(:,1), estimated_position(:,2), 'g+', 'MarkerSize',
     ↪ 10, 'DisplayName', 'Estimated Position');
plot(master_true_position(1), master_true_position(2), 'bx', 'MarkerSize',
     ↪ 10, 'DisplayName', 'True Position');
legend;
xlabel('X Position');
ylabel('Y Position');
title('2D Localization with Anchors');
grid on;
hold off;

```

Effect of the increasing Noise

Now we will see the effect of the increasing noise on the estimation of the target position. Complete the code below where you read TODO. At the end we will compare the estimation with low noise and high noise.

```

% Let's see what happens if the standard deviation of the noise is increased
↪ to 0.5

% Initialize the noisy distances
distances_noisy_2 = zeros(n_mes, n_anchor);
for i = 1:n_mes
    % Add noise to the distances
    % TODO
end

% Compute the estimated position
estimated_position_2 = zeros(n_mes, 2);
for i = 1:n_mes
    % Compute the estimated position using the noisy distances
    % TODO
end

```

```

% Plot the estimated positions
figure;
hold on;
plot(anchors(:,1), anchors(:,2), 'ro', 'MarkerSize', 10, 'DisplayName',
     ↪ 'Anchors');
plot(estimated_position_2(:,1), estimated_position_2(:,2), 'g+',
     ↪ 'MarkerSize', 10, 'DisplayName', 'Estimated Position');
plot(master_true_position(1), master_true_position(2), 'bx', 'MarkerSize',
     ↪ 10, 'DisplayName', 'True Position');
legend;
xlabel('X Position');
ylabel('Y Position');
title('2D Localization with Anchors');
grid on;
hold off;

% Now compare the estimated positions with the different noise levels
figure;
hold on;
plot(anchors(:,1), anchors(:,2), 'ro', 'MarkerSize', 10, 'DisplayName',
     ↪ 'Anchors');
plot(estimated_position(:,1), estimated_position(:,2), 'g+', 'MarkerSize',
     ↪ 10, 'DisplayName', 'Estimated Position (0.1 std)');
plot(estimated_position_2(:,1), estimated_position_2(:,2), 'b+',
     ↪ 'MarkerSize', 10, 'DisplayName', 'Estimated Position (0.5 std)');
plot(master_true_position(1), master_true_position(2), 'rx', 'MarkerSize',
     ↪ 10, 'DisplayName', 'True Position');
legend;
xlabel('X Position');
ylabel('Y Position');
title('2D Localization with Anchors');
grid on;
hold off;

```

Effect of the Number of Anchors

We will investigate the effect of increasing the number of anchors on the estimation of the target position. Complete the code below where you read TODO.

```

% What happen is the number of anchors increases?
% Let's add a 3 anchors to the system

% Define the positions of the anchors (x, y)
anchors_2 = [2, 0; 7, 1; 5, 4; 3, 1; 1, 4; 2, -1; 3, 3]; % Example positions
↪ for 4 anchors

n_anchor_2 = size(anchors_2, 1);
% Define the true position of the target (our ground truth, which is not
↪ known and it's the position that we want to estimate)
master_true_position = [4, 2];

% Calculate distances from the target to each anchor
distances_2 = % TODO

% What happens if we add noise to the distances?
% We add zero-mean Gaussian noise with standard deviation of 0.1 for 5 times
↪ to the distances

% Initialize the noisy distances
distances_noisy_2 = zeros(n_mes, n_anchor_2);
for i = 1:n_mes
    % Add noise to the distances
    % TODO
end

% Compute the estimated position
estimated_position_2 = zeros(n_mes, 2);
for i = 1:n_mes
    % TODO
end

% Plot the estimated positions
figure;
hold on;
plot(anchors_2(:,1), anchors_2(:,2), 'co', 'MarkerSize', 10, 'DisplayName',
↪ 'Anchors');
plot(estimated_position(:,1), estimated_position(:,2), 'r.', 'MarkerSize', 5,
↪ 'DisplayName', 'Estimated Position 3');
plot(estimated_position_2(:,1), estimated_position_2(:,2), 'g.',
↪ 'MarkerSize', 5, 'DisplayName', 'Estimated Position 6');
plot(master_true_position(1), master_true_position(2), 'bo', 'MarkerSize',
↪ 10, 'DisplayName', 'True Position');

```

```

legend;
xlabel('X Position');
ylabel('Y Position');
title('2D Localization with Anchors');
grid on;
hold off;

% Compare the mean square error of the estimated positions
mse_3 = % TODO
mse_6 = % TODO

disp('Mean Square Error for 3 anchors:');
disp(mse_3);

disp('Mean Square Error for 6 anchors:');
disp(mse_6);

```

Recursive Least Squares

Least Squares Algorithm

Let's define again the measurement model for multiple measurements:

$$\mathbf{z}(i) = \mathbf{H}(i)\mathbf{x} + \varepsilon(i), \quad i = 1, 2, \dots, k$$

is the time varying set of the measurements collected at time i . The following aggregating vectors are

$$\mathbf{Z}^k = \begin{bmatrix} \mathbf{z}(1) \\ \mathbf{z}(2) \\ \dots \\ \mathbf{z}(k) \end{bmatrix}, \quad \mathbf{H}^k = \begin{bmatrix} \mathbf{H}(1) \\ \mathbf{H}(2) \\ \dots \\ \mathbf{H}(k) \end{bmatrix}, \quad \varepsilon = \begin{bmatrix} \varepsilon(1) \\ \varepsilon(2) \\ \dots \\ \varepsilon(k) \end{bmatrix}.$$

The covariance matrix

$$\text{Cov}[\varepsilon] = \begin{bmatrix} \text{Cov}[\varepsilon(1)] & 0 & \dots & 0 \\ 0 & \text{Cov}[\varepsilon(2)] & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \text{Cov}[\varepsilon(k)] \end{bmatrix}.$$

Least Squares Algorithm

The estimation error is given by

$$\mathbf{e}^k = \mathbf{x} - \hat{\mathbf{x}}^{LS}.$$

The estimated position $\hat{\mathbf{x}}^{LS}$ is given by the solution of the following

$$\hat{\mathbf{x}}^{LS} = (\mathbf{H}^{kT} \mathbf{C}^{k-1} \mathbf{H}^k)^{-1} \mathbf{H}^{kT} \mathbf{C}^{k-1} \mathbf{Z}^k = \mathbf{P}(k) \mathbf{H}^{kT} \mathbf{C}^{k-1} \mathbf{Z}^k.$$

\mathbf{C} is the covariance matrix of the anchors

New Measurements

At time $k + 1$, new measurements are collected and the new matrices are

$$\mathbf{Z}^{k+1} = \begin{bmatrix} \mathbf{Z}^k \\ \mathbf{z}(k+1) \end{bmatrix}, \quad \mathbf{H}^{k+1} = \begin{bmatrix} \mathbf{H}^k \\ \mathbf{H}(k+1) \end{bmatrix}, \quad \varepsilon^{k+1} = \begin{bmatrix} \varepsilon^k \\ \varepsilon(k+1) \end{bmatrix}.$$

The covariance matrix is updated as

$$\mathbb{C}ov[\varepsilon^{k+1}] = \begin{bmatrix} \mathbb{C}ov[\varepsilon^k] & 0 \\ 0 & \mathbb{C}ov[\varepsilon(k+1)] \end{bmatrix}.$$

This problem will explode in terms of computational complexity as the number of measurements increases. For this reason, the **Recursive Least Squares** algorithm is used.

Recursive Least Squares Algorithm

The recursive solution is given by

$$\begin{aligned} \mathbf{S}(k+1) &= \mathbf{H}(k+1) \mathbf{P}(k) \mathbf{H}(k+1)^T + \mathbb{C}ov[\varepsilon(k+1)], \\ \mathbf{W}(k+1) &= \mathbf{P}(k) \mathbf{H}(k+1)^T \mathbf{S}(k+1)^{-1}, \\ \hat{\mathbf{x}}^{RLS}(k+1) &= \hat{\mathbf{x}}^{RLS}(k) + \mathbf{W}(k+1)(\mathbf{z}(k+1) - \mathbf{H}(k+1) \hat{\mathbf{x}}^{RLS}(k)), \\ \mathbf{P}(k+1) &= (\mathbf{I} - \mathbf{W}(k+1) \mathbf{H}(k+1)) \mathbf{P}(k). \end{aligned}$$

Matlab Implementation

Least Squares Algorithm

We need to modify the trilateration function implemented before to construct also the covariance matrix. Complete the code below where you read **TODO**.

```
% trilateration function
function [H,z,C] = trilateration(anchors, distances, noise_std)
    % Number of anchors
    n = size(anchors, 1);

    % Initialize matrices
    H = zeros(n-1, 2);
    z = zeros(n-1, 1);
    C = zeros(n-1);

    % Iterate over all anchors
    for i = 1:n-1
        % Fill the matrices
        % TODO
        % Fill the covariance matrix
        if i == 1
            % TODO
        elseif i < n-1
            % TODO
        else
            % TODO
        end
    end
end
```

Least Squares Algorithm

Now we can implement the recursive least squares algorithm. Complete the code below where you read **TODO**.

```
clear;
clc;

% Define the positions of the anchors (x, y)
```

```

anchors = [2, 0; 7, 1; 5, 4]; % Example positions for 3 anchors
n_anchor = size(anchors, 1);
% Define the true position of the target (not know, the position that
% we want to estimate)
master_true_position = [4, 2];

% Calculate distances from the target to each anchor
distances = % TODO

% What happens if we add noise to the distances?
% We add zero-mean Gaussian noise with standard deviation of 0.1 for 5 times
↪ to the distances

% Number of noisy measurements
k = 100;

% Initialize the noisy distances
distances_noisy = zeros(k, n_anchor);

for i = 1:k
    distances_noisy(i, :) = distances + 0.1 * randn(n_anchor, 1);
end

% Compute the problem matrices
H_k = zeros((n_anchor-1)*k, 2);
Z_k = zeros((n_anchor-1)*k, 1);
C_k = zeros((n_anchor-1)*k);

for i = 1:k
    l = 1 + (i-1)*(n_anchor - 1);
    % TODO = trilateration(...);
end

P_k = % TODO
x_ls = % TODO

% Plot the estimated position
figure;
hold on;
plot(anchors(:,1), anchors(:,2), 'ro', 'MarkerSize', 10, 'DisplayName',
↪ 'Anchors');
plot(x_ls(1), x_ls(2), 'g+', 'MarkerSize', 10, 'DisplayName', 'Estimated
↪ Position');

```

```

plot(master_true_position(1), master_true_position(2), 'bx', 'MarkerSize',
     ↪ 10, 'DisplayName', 'True Position');
legend;
xlabel('X Position');
ylabel('Y Position');
title('2D Localization with Anchors');
grid on;
hold off;

```

Recursive Least Squares Algorithm

Implement the recursive least squares function. Complete the code below where you read TODO.

```

% Iterative solution for the recursive least squares
function [x_k_1, P_k_1] = recursive_wls(x_k, P_k, z_k_1, H_k_1, C_new)
    S_k_1 = % TODO
    W_k_1 = % TODO
    x_k_1 = % TODO
    P_k_1 = % TODO
end

```

Recursive Least Squares Algorithm

We see how to add a new measurement to the least squares algorithm. Complete the code below where you read TODO.

```

% New measurements
distances_noisy_2 = distances + 0.1 * randn(n_anchor, 1);

% Compute the problem matrices
% TODO = trilateration(...);
Z_new = % TODO
H_new = % TODO
C_new = % TODO
x_ls_new = % TODO use the least squares algorithm to find the new position

% Now we simulate the recursive least squares from the first measurement
% Initialize the noisy distances
distances_noisy = zeros(k, n_anchor);

```

```

std_dev = 0.5;
for i = 1:k
    distances_noisy(i, :) = % TODO
end

% Compute the problem matrices
% TODO = trilateration(...);

P = % TODO;
x_ls = % TODO;

% Plot the estimated position
figure;
hold on;
plot(anchors(:,1), anchors(:,2), 'ro', 'MarkerSize', 10, 'DisplayName',
     ↪ 'Anchors');
plot(x_ls(1), x_ls(2), 'g+', 'MarkerSize', 10, 'DisplayName', 'Estimated
     ↪ Position');
plot(master_true_position(1), master_true_position(2), 'bx', 'MarkerSize',
     ↪ 10, 'DisplayName', 'True Position');
legend;
xlabel('X Position');
ylabel('Y Position');
xlim([3.8, 4.2]);
ylim([1.8, 2.2]);
title('2D Localization with Anchors with IWSL');
grid on;

x_values = zeros(k, 2);
x_values(1,:) = x_ls;
P_values = P;

for i = 2:k
    % TODO = trilateration();
    [] = recursive_wls();
    plot(x_values(i,1), x_values(i,2), 'g+', 'MarkerSize',
         ↪ 10, 'HandleVisibility', 'off');
    drawnow;
    pause(0.1);
end

```

Moving object with Recursive Least Squares

We will see the effect of the moving target on the estimation of the target position. The system dynamic is given by $x_{k+1} = Ax_k$ where $A = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$. Complete the code below where you read **TODO**.

```
% Define the system dynamics
A = % TODO
state = % TODO
dt = 0.1; % Time step

% Calculate distances from the target to each anchor
distances = % TODO

% Add noise to the distances
distances_noisy = % TODO

% Initialize the recursive least squares
[] = trilateration();
P = % TODO
x_ls = % TODO

x_values = zeros(k, 2);
x_values(1,:) = x_ls;
P_values = P;

% Initialize the plot
figure;
hold on;
line_handle = plot(NaN, NaN, 'bo-', 'MarkerSize', 5);
plot(anchors(:,1), anchors(:,2), 'ro', 'MarkerSize', 10, 'DisplayName',
     ↪ 'Anchors');
line_handle_est = plot(NaN, NaN, 'g+', 'MarkerSize', 10, 'DisplayName',
     ↪ 'Estimated Position');
xlim([-2, 2]);
ylim([-2, 2]);
title('Real-Time Dynamical System Trajectory');
xlabel('x');
ylabel('y');
grid on;
hold off;
```

```

% Real-time update
x_data = [];
y_data = [];
for k = 2:200
    % Update the state
    state = % TODO
    x_data = [x_data, state(1)];
    y_data = [y_data, state(2)];

    % Calculate distances from the target to each anchor
    distances = %TODO;

    % Update the recursive least squares
    [H,z,C] = %TODO;
    [x_ls, P] = %TODO;
    x_values(k,:) = x_ls;

    % Update the plot
    set(line_handle_est, 'XData', x_values(1:k,1), 'YData', x_values(1:k,2));
    set(line_handle, 'XData', x_data, 'YData', y_data);
    drawnow;
    pause(0.05);
end

```

Dynamic Estimator

Unicycle Model

- The unicycle model represents a mobile robot on a plane.
- Typical example: differential robot with two lateral wheels.
- Nonholonomic constraint: movement is only allowed in the direction of the orientation.

State Coordinates and Control Variables

When trying to estimate the position of a moving target, the problems become more complex. An additional term should be considered in the model, i.e., the orientation of the target. The new coordinates are $\mathbf{q} = [x, y, \theta]^T$, where:

- x, y are the position of the target.
- θ is the orientation of the target.

The control variable of the system are:

- v the velocity of the target.
- ω the angular velocity of the target.

System Dynamics

$$\begin{aligned}\dot{x} &= v \cos(\theta), \\ \dot{y} &= v \sin(\theta), \\ \dot{\theta} &= \omega.\end{aligned}$$

Discretizing the Model

To simulate numerically, we discretize with time step Δt :

$$\begin{cases} x_{k+1} = x_k + v_k \cos \theta_k \Delta t \\ y_{k+1} = y_k + v_k \sin \theta_k \Delta t \\ \theta_{k+1} = \theta_k + \omega_k \Delta t \end{cases}$$

It is easy to see that the model is nonlinear. For this reason, the Extended Kalman Filter (EKF) is used. The model can be written as

$$\mathbf{x}_{k+1} = f_k(\mathbf{x}_k, \mathbf{u}_k)$$

Measurement Model

As in the previous cases, the measurements are the distances from the target to the anchors. The model to estimate the current x and y is the same linearized model as before. It is possible to use it without any modification,

$$\mathbf{z}_{k+1} = \mathbf{H}_{k+1} \mathbf{x}_{k+1} + \varepsilon_{k+1},$$

Extended Kalman Filter

Since the model is nonlinear, the Extended Kalman Filter (EKF) is used. The EKF is a recursive estimator that uses a linear approximation of the model to estimate the state of the system. The EKF is composed of two steps:

1. **Prediction:** the state of the system is predicted using the model.
2. **Update:** the predicted state is corrected using the measurements.

Prediction Step

The prediction step is written as

$$\begin{aligned}\hat{\mathbf{x}}_{k+1}^- &= f_k(\hat{\mathbf{x}}_k, \mathbf{u}_k, \mathbf{v}_k), \\ \mathbf{P}_{k+1}^- &= \mathbf{A}_k \mathbf{P}_k \mathbf{A}_k^T + \mathbf{G} \mathbf{Q}_k \mathbf{G}^T,\end{aligned}$$

where \mathbf{A}_k is the Jacobian of the model, \mathbf{G} is the Jacobian of the noise, and \mathbf{Q}_k is the covariance matrix of the model noise. The Jacobians are given by

$$\begin{aligned}\mathbf{A}_k &= \frac{f_k(\mathbf{x}, \mathbf{u}, \mathbf{v})}{d\mathbf{x}}, \\ \mathbf{G} &= \frac{f_k(\mathbf{x}, \mathbf{u}, \mathbf{v})}{d\mathbf{v}},\end{aligned}$$

computed for $\mathbf{x} = \hat{\mathbf{x}}_k$, $\mathbf{u} = \mathbf{u}_k$ and $\mathbf{v} = \mathbf{0}$.

Correction Step

The correction step is written as

$$\begin{aligned}\mathbf{S}_{k+1} &= \mathbf{H}_{k+1} \mathbf{P}_{k+1}^- \mathbf{H}_{k+1}^T + \mathbf{R}_{k+1}, \\ \mathbf{K}_{k+1} &= \mathbf{P}_{k+1}^- \mathbf{H}_{k+1}^T \mathbf{S}_{k+1}^{-1}, \\ \hat{\mathbf{x}}_{k+1} &= \hat{\mathbf{x}}_{k+1}^- + \mathbf{K}_{k+1} (\mathbf{z}_{k+1} - h_{k+1}(\hat{\mathbf{x}}_{k+1}^-)), \\ \mathbf{P}_{k+1} &= (\mathbf{I} - \mathbf{K}_{k+1} \mathbf{H}_{k+1}) \mathbf{P}_{k+1}^-.\end{aligned}$$

The Jacobian \mathbf{H}_{k+1} is given by

$$\mathbf{H}_{k+1} = \frac{h_{k+1}(\mathbf{x})}{d\mathbf{x}},$$

computed for $\mathbf{x} = \hat{\mathbf{x}}_{k+1}^-$ and $\varepsilon = 0$.

Matlab Implementation

Extended Kalman Filter Functions

We will implement the Extended Kalman Filter for the unicycle model. In this first part of the code the prediction and update functions have to be written. Remember that the system model is not linear while the measurement model is linear. Complete the code below where you read `TODO`.

```
% Predict step function for Kalman filter
function [x_pred, P_pred] = predict_step(x, P, A, G, Q, fun)
    x_pred = fun(x(1), x(2), x(3), x(4), x(5)); % Assuming x contains [x, y,
    ↪ theta, vel, omega]
    A_c = A(x(1), x(2), x(3), x(4), x(5));
    P_pred = % TODO
end

% Update step function for Kalman filter
function [x_k_1, P_k_1] = update_step(x_k, P_k, z_k_1, H_k_1, C_new)
    K = % TODO
    x_k_1 = % TODO
    P_k_1 = % TODO
end
```

System Function and Jacobians

Now we will implement the system function and the Jacobians. Complete the code below where you read `TODO`.

```
clc
clear all
close all

dT = 1/30;

% Define the system dynamics
fun = @(x, y, theta, vel, omega) % TODO

% Define the system Jacobian matrix
A = @(x, y, theta, vel, omega) % TODO
```

```

G = zeros(3, 3);

nu = [0;0;0];

Q = 0.1 * eye(3);

% Define the initial state
state = [4; 2; 0]; % Initial state

k = 300;

% Define anchors
anchors = [2, 0; 7, 1; 1, 1; 0, 5]; % Example positions for 3 anchors
n_anchor = size(anchors, 1);

% Calculate distances from the target to each anchor
distances = sqrt(sum((anchors - state(1:2)').^2, 2));

% Add noise to the distances
distances_noisy = distances + 0.1 * randn(n_anchor, 1);

% Initialize the first position which will be used to initialize the EKF
[H,z,C] = trilateration(anchors, distances_noisy, 0.1);
P = % TODO
x_ls = % TODO

```

Initialization of the Extended Kalman Filter

Here we initialize the covariance matrix P, the plot and the variables to store the estimated positions.

```

x_values = zeros(k, 3);
x_values(1,:) = [x_ls', 0];
P_values = [P, zeros(2, 1); zeros(1, 3)];
P_values(3,3) = 0.1;
P = P_values;

% Initialize the plot
figure;
hold on;
line_handle = plot(NaN, NaN, 'bo-', 'MarkerSize', 5, 'DisplayName', 'Real
↪ Trajectory');

```

```

plot(anchors(:,1), anchors(:,2), 'ro', 'MarkerSize', 10, 'DisplayName',
    ↪ 'Anchors');
line_handle_est = plot(NaN, NaN, 'g+', 'MarkerSize', 10, 'DisplayName',
    ↪ 'Estimated Position');
title('Real-Time Dynamical System Trajectory');
xlabel('x');
ylabel('y');
grid on;
legend;

% Real-time update
x_data = [];
y_data = [];
vel = 1;
omega = 0.4;
trace_P = zeros(200,1);
trace_P(1) = trace(P);

```

Real-Time Update of the Extended Kalman Filter

In this part of the code we will update the Extended Kalman Filter in real-time inside a loop. Note that looking at the trace of the covariance matrix is a good way to see how the estimation is behaving. Complete the code below where you read `TODO`.

```

for k = 2:200
    % Simulate the system
    state = % TODO

    % Predict step
    [x_pred, P_pred] = % TODO

    % Calculate distances from the target to each anchor
    distances = % TODO
    [H,z,C] = % TODO

    % Update step
    [x_values(k,:), P] = % TODO
    x_data = [x_data, state(1)];
    y_data = [y_data, state(2)];

    % Update the plot

```

```

        set(line_handle_est, 'XData', x_values(1:k,1), 'YData', x_values(1:k,2));
        set(line_handle, 'XData', x_data, 'YData', y_data);
        drawnow;
        pause(0.1);

        % Store the trace of the covariance matrix
        trace_P(k) = trace(P);
    end

    % Plot the results of the estimation
    figure;
    plot(trace_P)
    title('Trace of the covariance matrix');
    xlabel('Time step');
    ylabel('Trace of the covariance matrix');

```

$$H(1, :) = [2 \cdot x_2 - 2x_1]$$

$$H(2, :) = [2x_3 - 2x_2]$$

$$\text{distances} = \begin{bmatrix} \text{anchors} \cdot x \end{bmatrix}$$