

Name: Abijith J. Kamath
 Student Id: 17788

E1 244: Detection and Estimation

February-May 2021

Solution – Homework 1

Analysis and Algorithms for Time-Difference-of-Arrival (TDOA) Localisation

Part A: Derivation and Modelling

Let the 2D position of the target be $\mathbf{x} = [x \ y]^\top$, and let $\{\mathbf{x}_i\}_{i=1}^M$ be the position of the M sensors used to locate the target. The TDOA system measures difference in ranges using the time-delay of a test signal and the measurements are modelled as:

$$r_i = cT_0 + d_i + w_i, \quad i = 1, \dots, M, \quad (1)$$

where T_0 is the time at which the target emits the beacon, c is the speed of propagation of the wave, $d_i = \|\mathbf{x} - \mathbf{x}_i\|_2$ is the distance between the sensors and the target and w_i is additive white Gaussian noise with uniform variance σ^2 . In the case of this example, the number of sensors $M = 4$. In TDOA, to eliminate T_0 , the difference in the ranges is used, i.e.,

$$r_{ij} = d_i - d_j + n_{ij}, \quad i, j = 1, 2, 3, 4, \quad (2)$$

where $n_{ij} = w_i - w_j$. The differences in ranges are used for estimation of the unknown location of the target \mathbf{x} .

To estimate the 2D position of the target, three range measurements are sufficient. This implies three range-difference measurements are sufficient. We consider the following measurements for estimation:

$$\underbrace{\begin{bmatrix} r_{12} \\ r_{23} \\ r_{34} \end{bmatrix}}_{\mathbf{r}} = \underbrace{\begin{bmatrix} d_1 - d_2 \\ d_2 - d_3 \\ d_3 - d_4 \end{bmatrix}}_{\mathbf{d}(\mathbf{x})} + \underbrace{\begin{bmatrix} n_{12} \\ n_{23} \\ n_{34} \end{bmatrix}}_{\mathbf{n}}. \quad (3)$$

The noise \mathbf{n} in (3) is a linear transformation of the i.i.d Gaussian random vector with zero mean and variance σ^2 . The transformation is given by:

$$\mathbf{n} = \begin{bmatrix} w_1 - w_2 \\ w_2 - w_3 \\ w_3 - w_4 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix}. \quad (4)$$

Therefore, distribution of \mathbf{n} is Gaussian with mean $\mathbb{E}[\mathbf{n}] = \mathbf{A}\mathbb{E}[\mathbf{w}] = \mathbf{0}$, and covariance matrix $\Sigma = \mathbb{E}[\mathbf{n}\mathbf{n}^\top] = \mathbf{A}\mathbb{E}[\mathbf{w}\mathbf{w}^\top]\mathbf{A} = \sigma^2\mathbf{A}\mathbf{A}^\top$. Therefore, the joint data-distribution of the measurements \mathbf{r} is also Gaussian, i.e., $\mathbf{r} \sim \mathcal{N}(\mathbf{d}(\mathbf{x}), \Sigma)$.

Derivation of Cramér-Rao Lower Bound

The likelihood function using the joint data-distribution:

$$p(\mathbf{r}; \mathbf{x}) = \frac{1}{(2\pi)^{3/2}(\det \boldsymbol{\Sigma})^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{r} - \mathbf{d}(\mathbf{x}))^\top \boldsymbol{\Sigma}^{-1}(\mathbf{r} - \mathbf{d}(\mathbf{x}))\right), \quad (5)$$

$$\implies \ln p(\mathbf{r}; \mathbf{x}) = -\frac{3}{2} \ln(2\pi) - \frac{1}{2} \ln \det \boldsymbol{\Sigma} - \frac{1}{2}(\mathbf{r} - \mathbf{d}(\mathbf{x}))^\top \boldsymbol{\Sigma}^{-1}(\mathbf{r} - \mathbf{d}(\mathbf{x})),$$

and therefore the score function is given by:

$$\nabla_{\mathbf{x}} \ln p(\mathbf{r}; \mathbf{x}) = -\left(\frac{\partial \mathbf{d}(\mathbf{x})}{\partial \mathbf{x}}\right)^\top \boldsymbol{\Sigma}^{-1}(\mathbf{r} - \mathbf{d}(\mathbf{x})). \quad (6)$$

It can be seen that $\mathbb{E}[\nabla_{\mathbf{x}} \ln p(\mathbf{r}; \mathbf{x})] = \mathbf{0}$, i.e., the joint data-distribution satisfies regularity conditions. Therefore, the inverse Fisher information matrix (FIM) gives a lower bound on the covariance of any unbiased estimator for \mathbf{x} . The Fisher information matrix is given by:

$$\begin{aligned} \mathbf{I}(\mathbf{x}) &= \mathbb{E}\left[(\nabla_{\mathbf{x}} \ln p(\mathbf{r}; \mathbf{x})) (\nabla_{\mathbf{x}} \ln p(\mathbf{r}; \mathbf{x}))^\top\right], \\ &= \mathbb{E}\left[\left(\frac{\partial \mathbf{d}(\mathbf{x})}{\partial \mathbf{x}}\right)^\top \boldsymbol{\Sigma}^{-1}(\mathbf{r} - \mathbf{d}(\mathbf{x})) (\mathbf{r} - \mathbf{d}(\mathbf{x}))^\top \boldsymbol{\Sigma}^{-1}\left(\frac{\partial \mathbf{d}(\mathbf{x})}{\partial \mathbf{x}}\right)\right], \\ &= \left(\frac{\partial \mathbf{d}(\mathbf{x})}{\partial \mathbf{x}}\right)^\top \boldsymbol{\Sigma}^{-1} \underbrace{\mathbb{E}\left[(\mathbf{r} - \mathbf{d}(\mathbf{x})) (\mathbf{r} - \mathbf{d}(\mathbf{x}))^\top\right]}_{\boldsymbol{\Sigma}} \boldsymbol{\Sigma}^{-1} \left(\frac{\partial \mathbf{d}(\mathbf{x})}{\partial \mathbf{x}}\right), \\ &= \left(\frac{\partial \mathbf{d}(\mathbf{x})}{\partial \mathbf{x}}\right)^\top \boldsymbol{\Sigma}^{-1} \left(\frac{\partial \mathbf{d}(\mathbf{x})}{\partial \mathbf{x}}\right), \end{aligned} \quad (7)$$

where the derivative $\frac{\partial \mathbf{d}(\mathbf{x})}{\partial \mathbf{x}}$ is the matrix,

$$\frac{\partial \mathbf{d}(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial}{\partial x}(d_1 - d_2) & \frac{\partial}{\partial y}(d_1 - d_2) \\ \frac{\partial}{\partial x}(d_2 - d_3) & \frac{\partial}{\partial y}(d_2 - d_3) \\ \frac{\partial}{\partial x}(d_3 - d_4) & \frac{\partial}{\partial y}(d_3 - d_4) \end{bmatrix}, \quad (8)$$

and the entries are computed as $\frac{\partial}{\partial x}(d_i - d_j) = \frac{x - x_i}{d_i} - \frac{x - x_j}{d_j}$, and similarly with respect to y . Let $\hat{\mathbf{x}}$ be any unbiased estimator for \mathbf{x} . Then, using Cramér-Rao lower bound (CRLB) theorem:

$$\text{cov}(\hat{\mathbf{x}}) \geq \mathbf{I}(\mathbf{x})^{-1}, \quad (9)$$

and therefore the variance of the estimates for the coordinates, $\text{var}(\hat{x}) \geq [\mathbf{I}(\mathbf{x})]_{11}^{-1}$ and $\text{var}(\hat{y}) \geq [\mathbf{I}(\mathbf{x})]_{22}^{-1}$.

Maximum-Likelihood Estimator (MLE)

The maximum-likelihood estimator (MLE) for the parameter \mathbf{x} is given by:

$$\begin{aligned} \hat{\mathbf{x}}_{MLE} &= \arg \max_{\mathbf{x} \in \mathbb{R}^2} p(\mathbf{r}; \mathbf{x}), \\ &= \arg \max_{\mathbf{x} \in \mathbb{R}^2} \ln p(\mathbf{r}; \mathbf{x}), \\ &= \arg \min_{\mathbf{x} \in \mathbb{R}^2} J(\mathbf{x}) \doteq \frac{1}{2}(\mathbf{r} - \mathbf{d}(\mathbf{x}))^\top \boldsymbol{\Sigma}^{-1}(\mathbf{r} - \mathbf{d}(\mathbf{x})). \end{aligned} \quad (10)$$

The MLE is given by the solution to the unconstrained optimisation programme, and we choose gradient descent with fixed step-size to solve the optimisation programme. The gradient descent updates with constant step-size $\alpha > 0$ are given by:

$$\begin{aligned}\hat{\mathbf{x}}^{(k+1)} &= \hat{\mathbf{x}}^{(k)} - \alpha \nabla_{\mathbf{x}} J(\mathbf{x}^{(k)}), \\ &= \hat{\mathbf{x}}^{(k)} + \alpha \left(\frac{\partial \mathbf{d}(\mathbf{x}^{(k)})}{\partial \mathbf{x}} \right)^T \Sigma^{-1} (\mathbf{r} - \mathbf{d}(\mathbf{x}^{(k)})),\end{aligned}\quad (11)$$

where the gradient of the objective function is given identical to (6). The iterations are run until the error between successive updates is below tolerance or the maximum number of iterations is reached. The estimates of the coordinates of the target position from the MLE iterations in (11) is given directly by the entries of the vector.

Best-Linear-Unbiased Estimator (BLUE)

The linearisation of the range measurements in (2) gives:

$$r_{ij}^2 + d_j^2 + 2r_{ij}d_j = d_i^2 + e_{ij}, \quad (12)$$

where the noise is approximates as $e_{ij} = n_{ij}^2 = (w_i - w_j)^2$. Using $d_i^2 = \|\mathbf{x} - \mathbf{x}_i\|_2^2$ in (12), we get:

$$r_{ij}^2 - \|\mathbf{x}_i\|_2^2 + \|\mathbf{x}_j\|_2^2 = -2(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{x} - 2r_{ij}d_j + e_{ij}, \quad (13)$$

which is linear in the parameter \mathbf{x} . In matrix-vector form, and taking the parameter vector $\boldsymbol{\theta} = [x \ y \ d_2 \ d_3 \ d_4]^T$, we have:

$$\underbrace{\begin{bmatrix} r_{12}^2 - \|\mathbf{x}_1\|_2^2 + \|\mathbf{x}_2\|_2^2 \\ r_{13}^2 - \|\mathbf{x}_1\|_2^2 + \|\mathbf{x}_3\|_2^2 \\ r_{14}^2 - \|\mathbf{x}_1\|_2^2 + \|\mathbf{x}_4\|_2^2 \\ r_{23}^2 - \|\mathbf{x}_2\|_2^2 + \|\mathbf{x}_3\|_2^2 \\ r_{24}^2 - \|\mathbf{x}_2\|_2^2 + \|\mathbf{x}_4\|_2^2 \\ r_{34}^2 - \|\mathbf{x}_3\|_2^2 + \|\mathbf{x}_4\|_2^2 \end{bmatrix}}_{\boldsymbol{\gamma}} = \underbrace{\begin{bmatrix} -2(x_1 - x_2) & -2(y_1 - y_2) & -2r_{12} & 0 & 0 \\ -2(x_1 - x_3) & -2(y_1 - y_3) & 0 & -2r_{13} & 0 \\ -2(x_1 - x_4) & -2(y_1 - y_4) & 0 & 0 & -2r_{14} \\ -2(x_2 - x_3) & -2(y_2 - y_3) & 0 & -2r_{23} & 0 \\ -2(x_2 - x_4) & -2(y_2 - y_4) & 0 & 0 & -2r_{24} \\ -2(x_3 - x_4) & -2(y_3 - y_4) & 0 & 0 & -2r_{34} \end{bmatrix}}_{\mathbf{H}} \underbrace{\begin{bmatrix} x \\ y \\ d_2 \\ d_3 \\ d_4 \end{bmatrix}}_{\boldsymbol{\theta}} + \underbrace{\begin{bmatrix} e_{12} \\ e_{13} \\ e_{14} \\ d_{23} \\ e_{24} \\ e_{34} \end{bmatrix}}_{\mathbf{e}}. \quad (14)$$

The complete distribution of the random vector \mathbf{e} is not important for estimation, and only mean and covariance of the distribution are sufficient. The mean of the entries $\mathbb{E}[e_{ij}] = \mathbb{E}[(w_i - w_j)^2] = \text{var}(w_i^2) + \text{var}(w_j^2) = 2\sigma^2$. The mean subtracted model $\boldsymbol{\gamma} - 2\sigma^2 \mathbf{1} = \mathbf{H}\boldsymbol{\theta} + \bar{\mathbf{e}}$, where $\bar{\mathbf{e}} = \mathbf{e} - 2\sigma^2 \mathbf{1}$ is a zero mean random vector, and $\mathbf{1}$ is the all-ones vector. The covariance matrix of the random vector $\mathbf{C} = \mathbb{E}[\bar{\mathbf{e}}\bar{\mathbf{e}}^T] = \mathbb{E}[\mathbf{e}\mathbf{e}^T] - 4\sigma^4 \mathbf{I}$. The entries are computed as: $[\mathbf{C}]_{11} = \mathbb{E}[(w_1 - w_2)^2(w_1 - w_2)^2] - 4\sigma^4 = 3\sigma^4 + 3\sigma^4 + 6\sigma^4 - 4\sigma^4 = 8\sigma^4$, $[\mathbf{C}]_{13} = [\mathbf{C}]_{14} = [\mathbf{C}]_{15} = \mathbb{E}[(w_1 - w_2)^2(w_1 - w_4)^2] - 4\sigma^4 = 2\sigma^4$, and $[\mathbf{C}]_{16} = \mathbb{E}[(w_1 - w_2)^2(w_3 - w_4)^2] = 0$. Similarly, we see that the diagonal entries are all equal to $[\mathbf{C}]_{11}$ and all the anti-diagonal entries are equal to zero. All other entries are equal to $[\mathbf{C}]_{12}$. Therefore the covariance matrix is equal to:

$$\mathbf{C} = 2\sigma^4 \begin{bmatrix} 4 & 1 & 1 & 1 & 1 & 0 \\ 1 & 4 & 1 & 1 & 0 & 1 \\ 1 & 1 & 4 & 0 & 1 & 1 \\ 1 & 1 & 0 & 4 & 1 & 1 \\ 1 & 0 & 1 & 1 & 4 & 1 \\ 0 & 1 & 1 & 1 & 1 & 4 \end{bmatrix}. \quad (15)$$

Then, the BLUE estimate for the parameter vector $\boldsymbol{\theta}$ is given by $\hat{\boldsymbol{\theta}} = (\mathbf{H}^T \mathbf{C}^{-1} \mathbf{H})^{-1} \mathbf{H}^T \mathbf{C}^{-1} (\boldsymbol{\gamma} - 2\sigma^2 \mathbf{1})$.

The estimates of the coordinates of the target position from the BLUE estimate for $\boldsymbol{\theta}$ is then $\hat{x}_{\text{BLUE}} = [\hat{\boldsymbol{\theta}}]_1$ and $\hat{y}_{\text{BLUE}} = [\hat{\boldsymbol{\theta}}]_2$.

Part B: Implementation

Maximum-Likelihood Estimator (MLE)

Figure 1(a) shows the TDOA localisation using MLE. The distance measurements have additive white Gaussian noise with variance $\sigma^2 = 0.001$. The estimated location of the target $\hat{\mathbf{x}}_{MLE}$ is exact.

Best-Linear-Unbiased Estimator (BLUE)

Figure 1(b) shows the TDOA localisation using BLUE. The distance measurements have additive white Gaussian noise with variance $\sigma^2 = 0.001$. The estimated location of the target $\hat{\mathbf{x}}_{BLUE}$ is exact and overlaps with $\hat{\mathbf{x}}_{MLE}$.

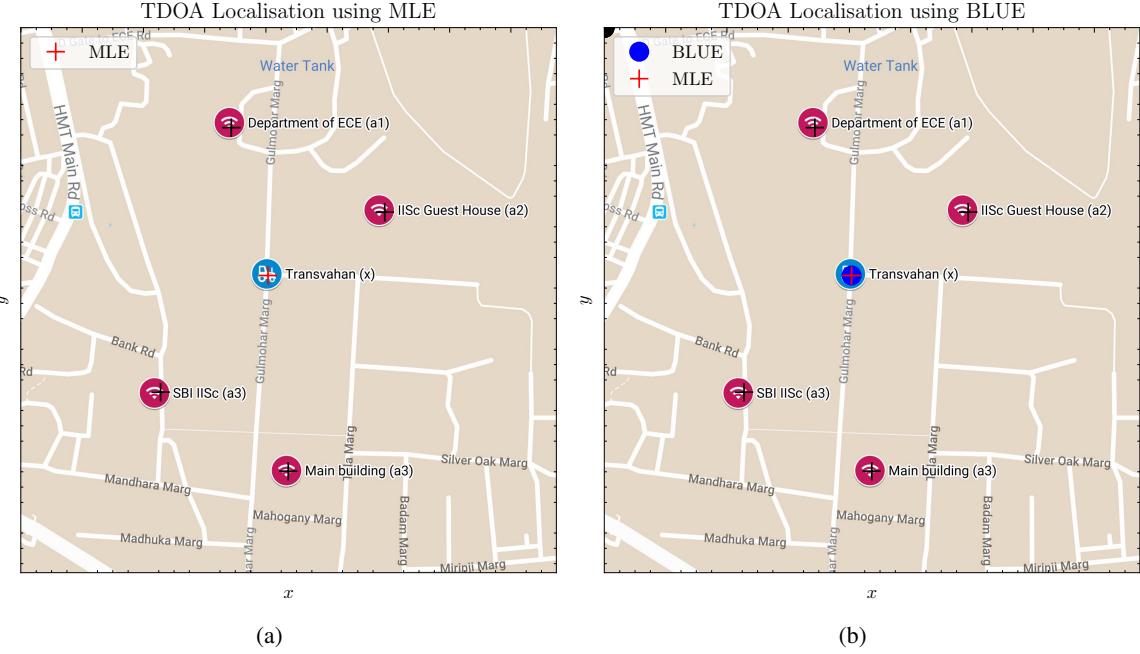


Figure 1: TDOA localisation using MLE and BLUE.

Noise Analysis

Figure 2(a) and 2(b) shows the mean-squared-error (MSE) between estimated target location and true target location for varying values of σ^2 , using MLE and BLUE respectively; along with their corresponding CRLB. It can be seen that the MLE and BLUE achieve CRLB as the σ^2 goes to zero. It can also be observed that the MLE estimate consistently has a lower MSE than the BLUE estimate. Further, the MLE estimate has MSE achieving the CRLB even for higher values of σ^2 , whereas the BLUE estimate is evidently suboptimal for higher values of σ^2 .

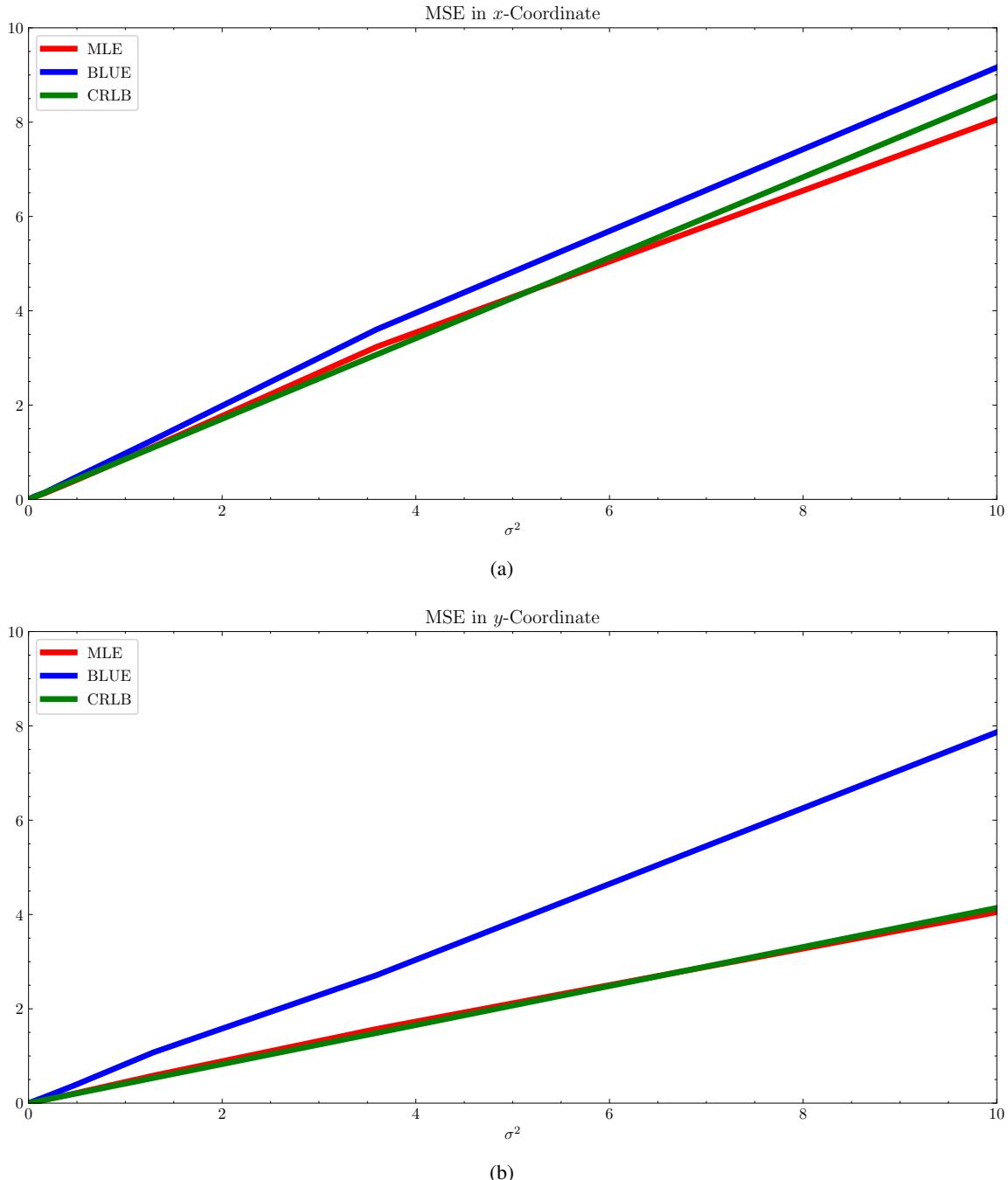


Figure 2: Noise analysis of TDOA localisation using MLE and BLUE estimates.

Scripts

The Python3 scripts to generate all figures can be downloaded from the GitHub repository https://github.com/kamath-abhijith/TDOA_Localisation. Use `requirements.txt` to install all dependencies. Also, see the following code snippets for reference.

Implementation of TDOA Localisation using MLE

The relevant function to implement MLE estimate mle_tdoa is in utils.py.

```
1 """
2
3 TIME DIFFERENCE OF ARRIVAL (TDOA) LOCALISATION
4 USING MAXIMUM LIKELIHOOD ESTIMATION
5
6 AUTHOR: ABIJITH J KAMATH
7 abijithj@iisc.ac.in
8
9 """
10
11 # %% IMPORT PACKAGES
12 import os
13 import numpy as np
14
15 from skimage.io import imread
16 from scipy.io import loadmat
17
18 from matplotlib import pyplot as plt
19 from matplotlib import style
20 from matplotlib import rcParams
21
22 import utils
23
24 # %% PLOT SETTINGS
25
26 plt.style.use(['science','ieee'])
27
28 plt.rcParams.update({
29     "font.family": "serif",
30     "font.serif": ["cm"],
31     "mathtext.fontset": "cm",
32     "font.size": 11})
33
34 # %% LOAD DATA
35
36 img = imread('mapimage.jpeg')
37
38 data = loadmat('TDOA_data.mat')
39
40 anchor_location = data['anchor_location'].astype(np.float)
41 target_location = data['target_location'].astype(np.float)
42 true_distances = data['true_distances']
43
44 noisy_distances = data['noisy_distances']
45 var_vec = data['sigma2']
46
47 # %% ESTIMATE USING MLE
48
49 init_pos = np.array([0.0,0.0])[:,np.newaxis]
50 mle_pos = utils.mle_tdoa(noisy_distances[:,0,0], anchor_location, init_pos,
51     var=0.001, step_size=.0005, max_iter=100)
52
53 # %% PLOT LOCATIONS
54
55 os.makedirs('./results', exist_ok=True)
56 path = './results/'
57
58 utils.plot_sensors(target=mle_pos, anchors=anchor_location, image=img,
```

```

59     xaxis_label=r'$x$', yaxis_label=r'$y$', title_text=r'TDOA Localisation using MLE',
60     marker='+', markersize=11, target_colour='red', legend_label=r'MLE',
61     show=True, save=path+'mle')
62 # %%

```

Implementation of TDOA Localisation using BLUE

The relevant function to implement BLUE estimate blue_tdoa is in utils.py.

```

1 """
2
3 TIME DIFFERENCE OF ARRIVAL (TDOA) LOCALISATION
4 USING BEST LINEAR UNBIASED ESTIMATOR
5
6 AUTHOR: ABIJITH J KAMATH
7 abijithj@iisc.ac.in
8
9 """
10
11 # %% IMPORT PACKAGES
12 import os
13 import numpy as np
14
15 from skimage.io import imread
16 from scipy.io import loadmat
17
18 from matplotlib import pyplot as plt
19 from matplotlib import style
20 from matplotlib import rcParams
21
22 import utils
23
24 # %% PLOT SETTINGS
25
26 plt.style.use(['science','ieee'])
27
28 plt.rcParams.update({
29     "font.family": "serif",
30     "font.serif": ["cm"],
31     "mathtext.fontset": "cm",
32     "font.size": 11})
33
34 # %% LOAD DATA
35
36 img = imread('mapimage.jpeg')
37
38 data = loadmat('TDOA_data.mat')
39
40 anchor_location = data['anchor_location'].astype(np.float)
41 target_location = data['target_location'].astype(np.float)
42 true_distances = data['true_distances']
43
44 noisy_distances = data['noisy_distances']
45 var_vec = data['sigma2']
46
47 # %% ESTIMATE USING BLUE
48
49 blue_pos = utils.blue_tdoa(noisy_distances[:,0,0], anchor_location, var=0.001)
50
51 # %% ESTIMATE USING MLE

```

```

52
53 init_pos = np.array([0.0,0.0])[:,np.newaxis]
54 mle_pos = utils.mle_tdoa(noisy_distances[:,0,0], anchor_location, init_pos,
55 var=0.001, step_size=.0005, max_iter=100)
56
57 # %% PLOT LOCATIONS
58
59 os.makedirs('./results', exist_ok=True)
60 path = './results/'
61
62 plt.figure(figsize=(12,6))
63 ax = plt.gca()
64
65 utils.plot_sensors(target=blue_pos, anchors=anchor_location*0, image=img, ax=ax,
66 xaxis_label=r'$x$', yaxis_label=r'$y$', marker='o', markersize=11,
67 target_colour='blue', legend_label=r'BLUE', show=False)
68
69 utils.plot_sensors(target=mle_pos, anchors=anchor_location, image=None, ax=ax,
70 xaxis_label=r'$x$', yaxis_label=r'$y$', title_text=r'TDOA Localisation using BLUE',
71 marker='+', markersize=11, target_colour='red', legend_label=r'MLE',
72 show=True, save=path+'blue')
73 # %%

```

Noise Analysis of TDOA Localisation using MLE and BLUE

The relevant function to compute the CRLB `crlb_tdoa` is in `utils.py`.

```

1 """
2
3 MONTE-CARLO ANALYSIS OF TDOA LOCALISATION
4 COMPARING MLE, BLUE WITH CRLB
5
6 AUTHOR: ABIJITH J KAMATH
7 abijithj@iisc.ac.in
8
9 """
10
11 # %% IMPORT PACKAGES
12 import os
13 import numpy as np
14
15 from tqdm import tqdm
16 from scipy.io import loadmat
17
18 from matplotlib import pyplot as plt
19 from matplotlib import style
20 from matplotlib import rcParams
21
22 import utils
23
24 # %% PLOT SETTINGS
25
26 plt.style.use(['science','ieee'])
27
28 plt.rcParams.update({
29     "font.family": "serif",
30     "font.serif": ["cm"],
31     "mathtext.fontset": "cm",
32     "font.size": 11})
33

```

```

34 # %%
35
36 data = loadmat('TDOA_data.mat')
37
38 anchor_location = data['anchor_location'].astype(np.float)
39 target_location = data['target_location'].astype(np.float)
40 true_distances = data['true_distances']
41
42 noisy_distances = data['noisy_distances']
43 var_vec = data['sigma2']
44
45
46 # %%
47
48 _, num_iter, var_len = noisy_distances.shape
49 init_pos = np.array([0.0,0.0])[:,np.newaxis]
50
51 min_x_var = np.zeros(var_len)
52 min_y_var = np.zeros(var_len)
53
54 mle_x_mse = np.zeros(var_len)
55 mle_y_mse = np.zeros(var_len)
56
57 blue_x_mse = np.zeros(var_len)
58 blue_y_mse = np.zeros(var_len)
59
60 for noise_iter in tqdm(range(var_len)):
61
62     noise = var_vec[noise_iter]
63
64     # Compute CRLB
65     fim = utils.crlb_tdoa(target_location, anchor_location, noise)
66     cov_est = np.linalg.inv(fim)
67     min_x_var[noise_iter] = cov_est[0,0]
68     min_y_var[noise_iter] = cov_est[1,1]
69
70     # Compute error in MLE and BLUE estimates
71     for avg_iter in range(num_iter):
72         mle_est = utils.mle_tdoa(noisy_distances[:,avg_iter,noise_iter],
73                                 anchor_location, init_pos, var=noise, step_size=noise/2.)
74
75         blue_est = utils.blue_tdoa(noisy_distances[:,avg_iter,noise_iter],
76                                 anchor_location, var=noise)
77
78         mle_x_mse[noise_iter] += (mle_est[0]-target_location[0])**2
79         mle_y_mse[noise_iter] += (mle_est[1]-target_location[1])**2
80
81         blue_x_mse[noise_iter] += (blue_est[0]-target_location[0])**2
82         blue_y_mse[noise_iter] += (blue_est[1]-target_location[1])**2
83
84         mle_x_mse[noise_iter] /= num_iter
85         mle_y_mse[noise_iter] /= num_iter
86
87         blue_x_mse[noise_iter] /= num_iter
88         blue_y_mse[noise_iter] /= num_iter
89
90 # %% PLOT MSE
91
92 os.makedirs('./results', exist_ok=True)
93 path = './results/'
94
```

```

95 # Plotting for x-coordinate
96 plt.figure(figsize=(12,6))
97 ax = plt.gca()
98
99 utils.plot_curve(var_vec, mle_x_mse, ax=ax, plot_colour='red',
100     legend_label=r'MLE', line_width=4)
101 utils.plot_curve(var_vec, blue_x_mse, ax=ax, plot_colour='blue',
102     legend_label=r'BLUE', line_width=4)
103 utils.plot_curve(var_vec, min_x_var, ax=ax, plot_colour='green',
104     legend_label=r'CRLB', line_width=4, xaxis_label=r'$\sigma^2$',
105     title_text=r'MSE in $x$-Coordinate', xlims=[0,10], ylims=[0,10],
106     show=True, save=path+'crlb_x')
107
108 # Plotting for y-coordinate
109 plt.figure(figsize=(12,6))
110 ax = plt.gca()
111
112 utils.plot_curve(var_vec, mle_y_mse, ax=ax, plot_colour='red',
113     legend_label=r'MLE', line_width=4)
114 utils.plot_curve(var_vec, blue_y_mse, ax=ax, plot_colour='blue',
115     legend_label=r'BLUE', line_width=4)
116 utils.plot_curve(var_vec, min_y_var, ax=ax, plot_colour='green',
117     legend_label=r'CRLB', line_width=4, xaxis_label=r'$\sigma^2$',
118     title_text=r'MSE in $y$-Coordinate', xlims=[0,10], ylims=[0,10],
119     show=True, save=path+'crlb_y')

```

utils.py

This script contains all the relevant functions and helpers.

```

1 """
2
3 UTILITY FUNCTIONS FOR TDOA LOCALISATION
4
5 AUTHOR: ABIJITH J KAMATH
6 abijithj@iisc.ac.in
7
8 """
9
10 # %% IMPORT LIBRARIES
11 import numpy as np
12
13 from matplotlib import pyplot as plt
14
15 # %% PLOTTING
16
17 def plot_sensors(target, anchors, image=None, ax=None, xaxis_label=None,
18     yaxis_label=None, title_text=None, marker='+', markersize=4,
19     target_colour='red', legend_label=None, legend_show=True,
20     legend_loc='upper left', show=True, save=None):
21     ''' Plots target and anchors on image '''
22     if ax is None:
23         fig = plt.figure(figsize=(12,6))
24         ax = plt.gca()
25
26     plt.plot(target[0,0], target[1,0], marker, color=target_colour,
27             marker=marker, markersize=markersize, label=legend_label)
28     plt.plot(anchors[0,:], anchors[1,:], marker, color='black',
29             marker=marker, markersize=markersize)
30

```

```

31     if image is not None:
32         plt.imshow(image)
33
34     if legend_label and legend_show:
35         plt.legend(loc=legend_loc, frameon=True, framealpha=0.8, facecolor='white')
36
37     if show:
38         plt.xlabel(xaxis_label)
39         plt.ylabel(yaxis_label)
40         plt.title(title_text)
41         plt.tick_params(left = False, right = False , labelleft = False ,
42                         labelbottom = False, bottom = False)
43     if save:
44         plt.savefig(save + '.pdf', format='pdf')
45     plt.show()
46
47     return
48
49 def plot_curve(x, y, ax=None, plot_colour='blue', xaxis_label=None,
50                 yaxis_label=None, title_text=None, legend_label=None, legend_show=True,
51                 legend_loc='upper left', line_style='-', line_width=None,
52                 show=False, xlims=[0,1], ylims=[-1,1], save=None):
53     """
54     Plots signal with abscissa in x and ordinates in y
55
56     """
57     if ax is None:
58         fig = plt.figure(figsize=(12,6))
59         ax = plt.gca()
60
61     plt.plot(x, y, linestyle=line_style, linewidth=line_width, color=plot_colour,
62               label=legend_label)
63     if legend_label and legend_show:
64         plt.legend(loc=legend_loc, frameon=True, framealpha=0.8, facecolor='white')
65     plt.xlabel(xaxis_label)
66     plt.ylabel(yaxis_label)
67
68     if show:
69         plt.xlim(xlims)
70         plt.ylim(ylims)
71         plt.title(title_text)
72     if save:
73         plt.savefig(save + '.pdf', format='pdf')
74     plt.show()
75
76     return
77
78 # %% OPERATORS
79
80 def covariance_mtx(var):
81     """ Returns covariance matrix of range differences """
82     return var * np.array([[2,-1,0], [-1,2,-1], [0,-1,2]])
83
84 def target_anchor_distance(x, anchors):
85     """ Returns distance between target and anchors """
86     return np.linalg.norm(x-anchors, axis=0)
87
88 def dist_diff(distances):
89     """ Returns distance differences """
90     return -1.0 * np.diff(distances)
91

```

```

92 def dist_derivative(x, anchors, distances):
93     ''' Returns derivative of distance differences '''
94     num_anchors = anchors.shape[1]
95     return (((x - anchors)/distances)[:, :num_anchors-1] -
96             ((x - anchors)/distances)[:, 1:]).T
97
98 # %% ACCURACY ANALYSIS
99
100 def crlb_tdoa(target, anchors, var):
101     """
102     Returns the Fisher information matrix of 2D TDOA
103
104     :param target: 2D position of the target
105     :param anchors: 2D position of the anchors
106     :param var: Variance of the noise in range measurements
107
108     :return fim: Fisher information matrix
109
110     """
111
112     num_anchors = anchors.shape[1]
113
114     # Noise statistics
115     covW = covariance_mtx(var)
116     presW = np.linalg.inv(covW)
117
118     # Distance vector
119     target_anchor_dist = target_anchor_distance(target, anchors)
120
121     del_dist_vec = dist_derivative(target, anchors, target_anchor_dist)
122
123     # Fisher information matrix
124     fim = (del_dist_vec.T.dot(presW)).dot(del_dist_vec)
125
126     return fim
127
128 # %% SOLVERS
129
130 def mle_tdoa(distances, anchors, init_pos, var=1., step_size=0.1, max_iter=100):
131     """
132     Returns maximum-likelihood estimate (MLE) for TDOA localisation.
133     Gradient descent with fixed step-size is used as the solver.
134
135     :param distances: distance measurements
136     :param anchors: 2D position of anchors
137     :param init_pos: Initialisation for gradient descent
138     :param step_size: Step size of gradient descent optimiser
139     :param max_iter: Maximum iterations before exit
140     :param var: variance of noise in distance measurements
141
142     :returns: mle estimate for position
143
144     """
145
146     covN = covariance_mtx(var)
147     presN = np.linalg.inv(covN)
148
149     range_diff = dist_diff(distances)
150     mle_estimate = init_pos
151
152     for _ in range(max_iter):

```

```

153     dx = target_anchor_distance(mle_estimate, anchors)
154     dx_vec = dist_diff(dx)
155     del_dx = dist_derivative(mle_estimate, anchors, dx)
156     mle_estimate += step_size * ((del_dx.T).dot(presN)).dot(range_diff -
157         dx_vec)[:, np.newaxis]
158
159     return mle_estimate
160
161 def blue_tdoa(distances, anchors, var=1.):
162     """
163     Returns best linear unbiased estimate (BLUE) for TDOA localisation.
164
165     :param distances: distance measurements
166     :param anchors: 2D position of anchors
167     :param var: variance of noise in distance measurements
168
169     :returns: blue estimate for position
170
171     """
172
173     num_anchors = anchors.shape[1]
174
175     cov_mtx = 2.0*var**2 * np.array([[4, 1, 1, 1, 1, 0],
176                                     [1, 4, 1, 1, 0, 1],
177                                     [1, 1, 4, 0, 1, 1],
178                                     [1, 1, 0, 4, 1, 1],
179                                     [1, 0, 1, 1, 4, 1],
180                                     [0, 1, 1, 1, 1, 4]], dtype=np.float)
181     pres_mtx = np.linalg.inv(cov_mtx)
182     r = np.zeros(6)
183     gamma_vec = np.zeros(6)
184     H_mtx = np.zeros([6, 5])
185
186     idx = 0
187     for i in range(num_anchors):
188         for j in range(i+1, num_anchors):
189             r[idx] = distances[i]-distances[j]
190             gamma_vec[idx] = (r[idx])**2 - np.linalg.norm(anchors[:,i])**2 + np.linalg.
191             norm(anchors[:,j])**2 - 2*var
192             idx = idx+1
193
194     idx = 0
195     for anchor_idx in range(num_anchors - 1):
196         lst = list(range((anchor_idx), num_anchors - 1))
197         for pos in lst:
198             H_mtx[idx, 0] = -2*(anchors[0, anchor_idx] - anchors[0, (pos+1)])
199             H_mtx[idx, 1] = -2*(anchors[1, anchor_idx] - anchors[1, (pos+1)])
200
201             H_mtx[idx, (pos+2)] = -2*r[idx]
202             idx = idx+1
203
204     den = np.linalg.inv(((H_mtx.T).dot(pres_mtx)).dot(H_mtx))
205     num = ((H_mtx.T).dot(pres_mtx)).dot(gamma_vec-2*var)
206     blue_est = den.dot(num)
207
208     return (blue_est[:2])[:, np.newaxis]

```