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 $\mathbf{x} = [x \ y]^\mathsf{T}$, 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:

$$r_i = cT_0 + d_i + w_i, \ i = 1, \cdots, M, \tag{1}$$

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.,

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

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:

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

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:

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

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

Derivation of Cramér-Rao Lower Bound

The likelihood function using the joint data-distribution:

$$\begin{aligned} p(\mathbf{r};\mathbf{x}) &= \frac{1}{(2\pi)^{3/2}(\det\boldsymbol{\Sigma})^{1/2}}\exp\left(-\frac{1}{2}(\mathbf{r}-\mathbf{d}(\mathbf{x}))^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{r}-\mathbf{d}(\mathbf{x}))\right), \\ \implies \ln p(\mathbf{r};\mathbf{x}) &= -\frac{3}{2}\ln(2\pi) - \frac{1}{2}\ln\det\boldsymbol{\Sigma} - \frac{1}{2}(\mathbf{r}-\mathbf{d}(\mathbf{x}))^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{r}-\mathbf{d}(\mathbf{x})), \end{aligned} \tag{5}$$

and therefore the score function is given by:

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

It can be seen that $\mathbb{E}\left[\nabla_{\mathbf{x}} \ln p(\mathbf{r};\mathbf{x})\right] = \mathbf{0}$, 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:

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

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

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

and the entries are computed as $\dfrac{\partial}{\partial x}(d_i-d_j) = \dfrac{x-x_i}{d_i} - \dfrac{x-x_j}{d_j}$, 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:

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

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

Maximum-Likelihood Estimator (MLE)

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

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

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:

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

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:

$$r_{ij}^2 + d_j^2 + 2 r_{ij} d_j = d_i^2 + e_{ij}, \tag{12}$$

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:

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

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

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

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 $\mathbb{E}[e_{ij}] = \mathbb{E}[(w_i - w_j)^2] = \mathrm{var}(w_i^2) + \mathrm{var}(w_j^2) = 2\sigma^2$. The mean subtracted model $\boldsymbol{\gamma} - 2\sigma^2 \mathbb{1} = \mathbf{H}\boldsymbol{\theta} + \bar{\mathbf{e}}$, where $\bar{\mathbf{e}} = \mathbf{e} - 2\sigma^2 \mathbb{1}$ is a zero mean random vector, and $\mathbb{1}$ is the all-ones vector. The covariance matrix of the random vector $\mathbf{C} = \mathbb{E}[\bar{\mathbf{e}}\bar{\mathbf{e}}^{\mathsf{T}}] = \mathbb{E}[\mathbf{e}\mathbf{e}^{\mathsf{T}}] - 4\sigma^4 \mathbf{I}$. 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:

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

Then, the BLUE estimate for the parameter vector $\boldsymbol{\theta}$ is given by $\hat{\boldsymbol{\theta}} = \left(\mathbf{H}^{\mathsf{T}} \mathbf{C}^{-1} \mathbf{H} \right)^{-1} \mathbf{H}^{\mathsf{T}} \mathbf{C}^{-1} \left(\boldsymbol{\gamma} - 2\sigma^2 \mathbb{1} \right)$. The estimates of the coordinates of the target position from the BLUE estimate for $\boldsymbol{\theta}$ is then $\hat{x}_{BLUE} = [\hat{\boldsymbol{\theta}}]_1$ and $\hat{y}_{BLUE} = [\hat{\boldsymbol{\theta}}]_2$.

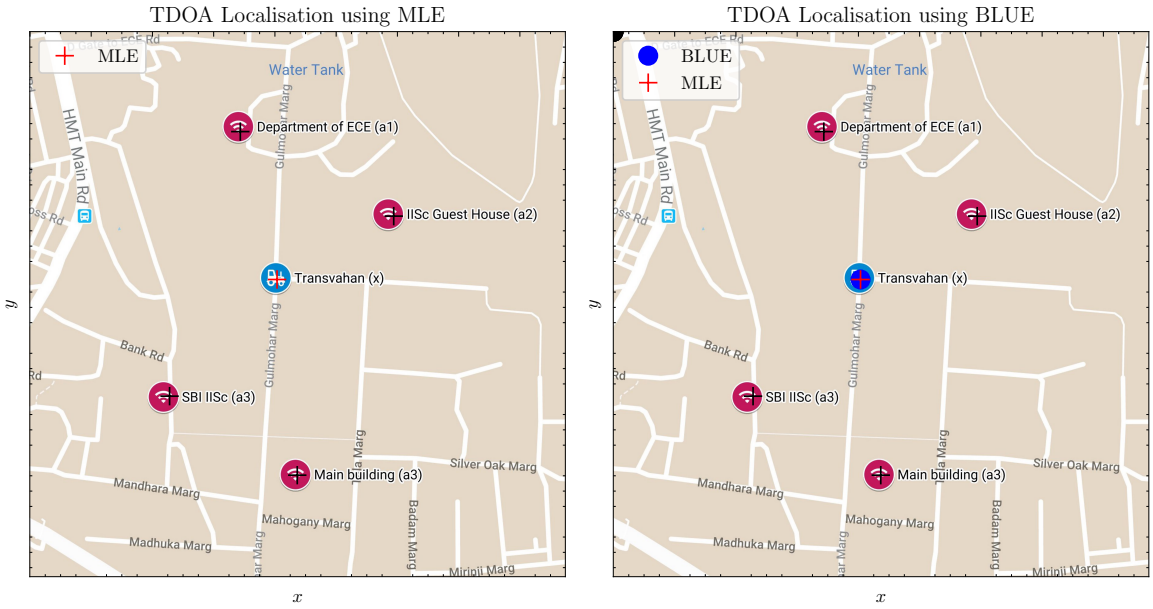
Part B: Implementation

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 $\hat{\mathbf{x}}_{MLE}$.

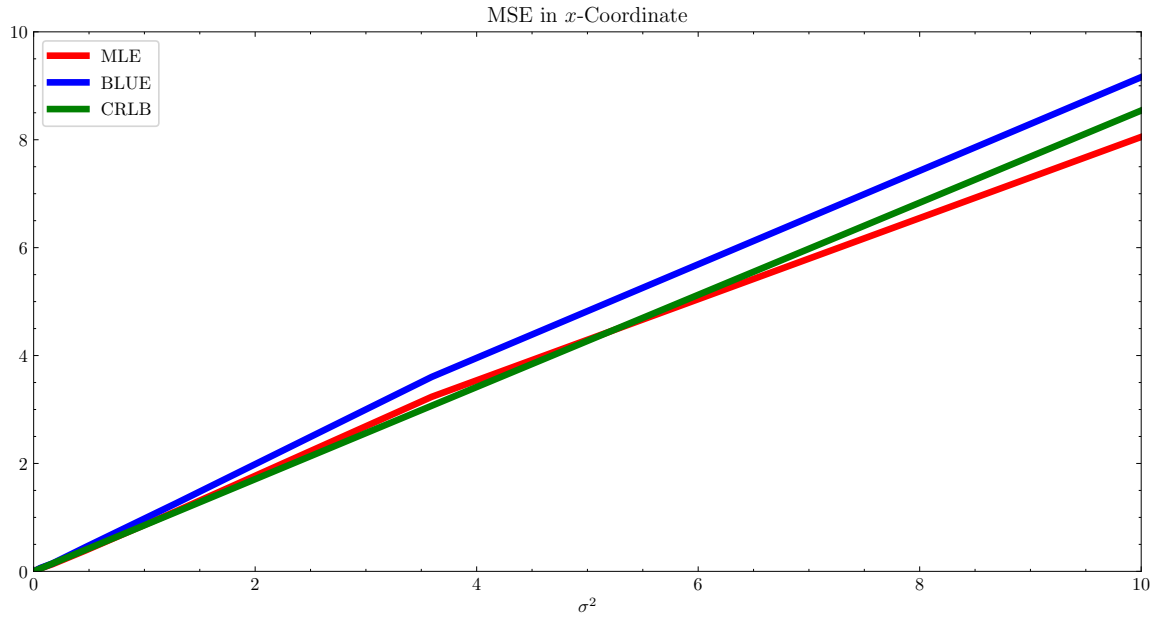


(a) (b)

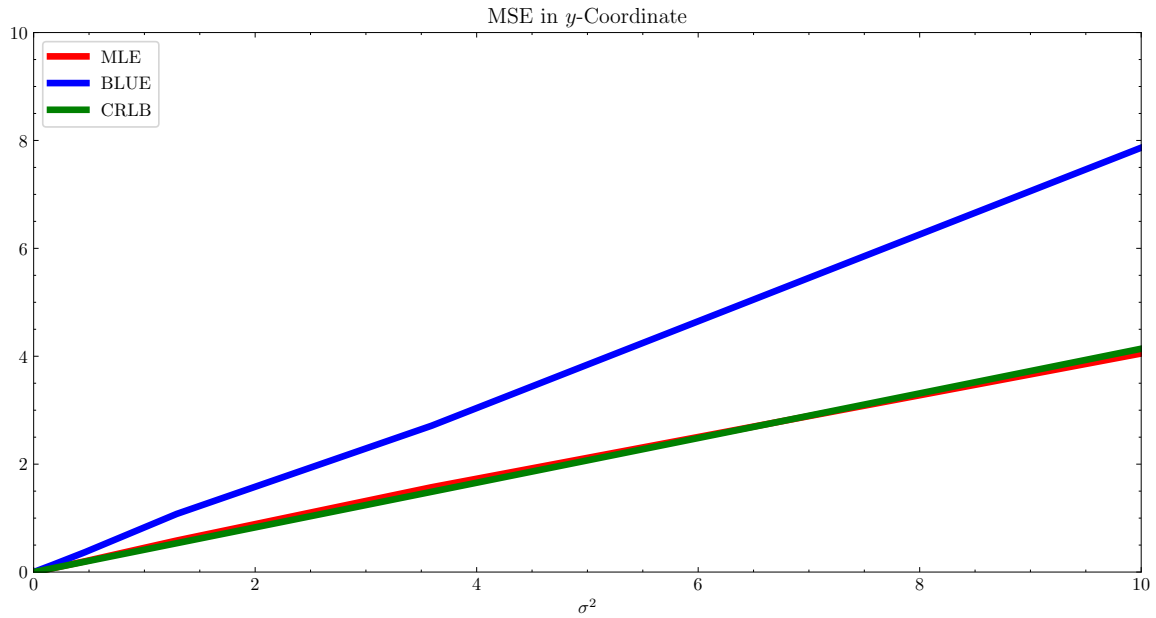
Figure 1: TDOA localisation using MLE and BLUE.

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.



(a)



(b)

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`.

```
'''

TIME DIFFERENCE OF ARRIVAL (TDOA) LOCALISATION
USING MAXIMUM LIKELIHOOD ESTIMATION

AUTHOR: ABIJITH J KAMATH
abijithj@iisc.ac.in

'''

# %% IMPORT PACKAGES
import os
import numpy as np

from skimage.io import imread
from scipy.io import loadmat

from matplotlib import pyplot as plt
from matplotlib import style
from matplotlib import rcParams

import utils

# %% PLOT SETTINGS

plt.style.use(['science','ieee'])

plt.rcParams.update({
    "font.family": "serif",
    "font.serif": ["cm"],
    "mathtext.fontset": "cm",
    "font.size": 11})

# %% LOAD DATA

img = imread('mapimage.jpeg')

data = loadmat('TDOA_data.mat')

anchor_location = data['anchor_location'].astype(np.float)
target_location = data['target_location'].astype(np.float)
true_distances = data['true_distances']

noisy_distances = data['noisy_distances']
var_vec = data['sigma2']

# %% ESTIMATE USING MLE

init_pos = np.array([0.0,0.0])[:,np.newaxis]
mle_pos = utils.mle_tdoa(noisy_distances[:,0,0], anchor_location, init_pos,
    var=0.001, step_size=.0005, max_iter=100)

# %% PLOT LOCATIONS

os.makedirs('./results', exist_ok=True)
path = './results/'

utils.plot_sensors(target=mle_pos, anchors=anchor_location, image=img,
```

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

Implementation of TDOA Localisation using BLUE

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

```
'''

TIME DIFFERENCE OF ARRIVAL (TDOA) LOCALISATION
USING BEST LINEAR UNBIASED ESTIMATOR

AUTHOR: ABIJITH J KAMATH
abijithj@iisc.ac.in

'''

# %% IMPORT PACKAGES
import os
import numpy as np

from skimage.io import imread
from scipy.io import loadmat

from matplotlib import pyplot as plt
from matplotlib import style
from matplotlib import rcParams

import utils

# %% PLOT SETTINGS

plt.style.use(['science','ieee'])

plt.rcParams.update({
    "font.family": "serif",
    "font.serif": ["cm"],
    "mathtext.fontset": "cm",
    "font.size": 11})

# %% LOAD DATA

img = imread('mapimage.jpeg')

data = loadmat('TDOA_data.mat')

anchor_location = data['anchor_location'].astype(np.float)
target_location = data['target_location'].astype(np.float)
true_distances = data['true_distances']

noisy_distances = data['noisy_distances']
var_vec = data['sigma2']

# %% ESTIMATE USING BLUE

blue_pos = utils.blue_tdoa(noisy_distances[:,0,0], anchor_location, var=0.001)

# %% ESTIMATE USING MLE
```

```

init_pos = np.array([0.0,0.0])[:,np.newaxis]
mle_pos = utils.mle_tdoa(noisy_distances[:,0,0], anchor_location, init_pos,
    var=0.001, step_size=.0005, max_iter=100)

# %% PLOT LOCATIONS

os.makedirs('./results', exist_ok=True)
path = './results/'

plt.figure(figsize=(12,6))
ax = plt.gca()

utils.plot_sensors(target=blue_pos, anchors=anchor_location*0, image=img, ax=ax,
    xaxis_label=r'$x$', yaxis_label=r'$y$', marker='o', markersize=11,
    target_colour='blue', legend_label=r'BLUE', show=False)

utils.plot_sensors(target=mle_pos, anchors=anchor_location, image=None, ax=ax,
    xaxis_label=r'$x$', yaxis_label=r'$y$', title_text=r'TDOA Localisation using BLUE',
    marker='+', markersize=11, target_colour='red', legend_label=r'MLE',
    show=True, save=path+'blue')
   # %%
```

Noise Analysis of TDOA Localisation using MLE and BLUE

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

```
'''

MONTE-CARLO ANALYSIS OF TDOA LOCALISATION
COMPARING MLE, BLUE WITH CRLB

AUTHOR: ABIJITH J KAMATH
abijithj@iisc.ac.in

'''

# %% IMPORT PACKAGES
import os
import numpy as np

from tqdm import tqdm
from scipy.io import loadmat

from matplotlib import pyplot as plt
from matplotlib import style
from matplotlib import rcParams

import utils

# %% PLOT SETTINGS

plt.style.use(['science','ieee'])

plt.rcParams.update({
    "font.family": "serif",
    "font.serif": ["cm"],
    "mathtext.fontset": "cm",
    "font.size": 11})

```

```
# %% LOAD DATA

data = loadmat('TDOA_data.mat')

anchor_location = data['anchor_location'].astype(np.float)
target_location = data['target_location'].astype(np.float)
true_distances = data['true_distances']

noisy_distances = data['noisy_distances']
var_vec = data['sigma2']


# %%

_, num_iter, var_len = noisy_distances.shape
init_pos = np.array([0.0,0.0])[:,np.newaxis]

min_x_var = np.zeros(var_len)
min_y_var = np.zeros(var_len)

mle_x_mse = np.zeros(var_len)
mle_y_mse = np.zeros(var_len)

blue_x_mse = np.zeros(var_len)
blue_y_mse = np.zeros(var_len)

for noise_iter in tqdm(range(var_len)):

    noise = var_vec[noise_iter]

    # Compute CRLB
    fim = utils.crlb_tdoa(target_location, anchor_location, noise)
    cov_est = np.linalg.inv(fim)
    min_x_var[noise_iter] = cov_est[0,0]
    min_y_var[noise_iter] = cov_est[1,1]

    # Compute error in MLE and BLUE estimates
    for avg_iter in range(num_iter):
        mle_est = utils.mle_tdoa(noisy_distances[:,avg_iter,noise_iter],
            anchor_location, init_pos, var=noise, step_size=noise/2.)

        blue_est = utils.blue_tdoa(noisy_distances[:,avg_iter,noise_iter],
            anchor_location, var=noise)

        mle_x_mse[noise_iter] += (mle_est[0]-target_location[0])**2
        mle_y_mse[noise_iter] += (mle_est[1]-target_location[1])**2

        blue_x_mse[noise_iter] += (blue_est[0]-target_location[0])**2
        blue_y_mse[noise_iter] += (blue_est[1]-target_location[1])**2

    mle_x_mse[noise_iter] /= num_iter
    mle_y_mse[noise_iter] /= num_iter

    blue_x_mse[noise_iter] /= num_iter
    blue_y_mse[noise_iter] /= num_iter

# %% PLOT MSE

os.makedirs('./results', exist_ok=True)
path = './results/'

```

```
# Plotting for x-coordinate
plt.figure(figsize=(12,6))
ax = plt.gca()

utils.plot_curve(var_vec, mle_x_mse, ax=ax, plot_colour='red',
    legend_label=r'MLE', line_width=4)
utils.plot_curve(var_vec, blue_x_mse, ax=ax, plot_colour='blue',
    legend_label=r'BLUE', line_width=4)
utils.plot_curve(var_vec, min_x_var, ax=ax, plot_colour='green',
    legend_label=r'CRLB', line_width=4, xaxis_label=r'$\sigma^2$',
    title_text=r'MSE in $x$-Coordinate', xlimits=[0,10], ylimits=[0,10],
    show=True, save=path+'crlb_x')

# Plotting for y-coordinate
plt.figure(figsize=(12,6))
ax = plt.gca()

utils.plot_curve(var_vec, mle_y_mse, ax=ax, plot_colour='red',
    legend_label=r'MLE', line_width=4)
utils.plot_curve(var_vec, blue_y_mse, ax=ax, plot_colour='blue',
    legend_label=r'BLUE', line_width=4)
utils.plot_curve(var_vec, min_y_var, ax=ax, plot_colour='green',
    legend_label=r'CRLB', line_width=4, xaxis_label=r'$\sigma^2$',
    title_text=r'MSE in $y$-Coordinate', xlimits=[0,10], ylimits=[0,10],
    show=True, save=path+'crlb_y')
```

`utils.py`

This script contains all the relevant functions and helpers.

```
'''

UTILITY FUNCTIONS FOR TDOA LOCALISATION

AUTHOR: ABIJITH J KAMATH
abijithj@iisc.ac.in

'''

# %% IMPORT LIBRARIES
import numpy as np

from matplotlib import pyplot as plt

# %% PLOTTING

def plot_sensors(target, anchors, image=None, ax=None, xaxis_label=None,
    yaxis_label=None, title_text=None, marker='+', markersize=4,
    target_colour='red', legend_label=None, legend_show=True,
    legend_loc='upper left', show=True, save=None):
    ''' Plots target and anchors on image '''
    if ax is None:
        fig = plt.figure(figsize=(12,6))
        ax = plt.gca()

    plt.plot(target[0,0], target[1,0], marker, color=target_colour,
        marker=marker, markersize=markersize, label=legend_label)
    plt.plot(anchors[0,:], anchors[1,:], marker, color='black',
        marker=marker, markersize=markersize)

```

```
    if image is not None:
        plt.imshow(image)

    if legend_label and legend_show:
        plt.legend(loc=legend_loc, frameon=True, framealpha=0.8, facecolor='white')

    if show:
        plt.xlabel(xaxis_label)
        plt.ylabel(yaxis_label)
        plt.title(title_text)
        plt.tick_params(left = False, right = False , labelleft = False ,
                labelbottom = False, bottom = False)
        if save:
            plt.savefig(save + '.pdf', format='pdf')
        plt.show()

    return

def plot_curve(x, y, ax=None, plot_colour='blue', xaxis_label=None,
    yaxis_label=None, title_text=None, legend_label=None, legend_show=True,
    legend_loc='upper left', line_style='-', line_width=None,
    show=False, xlimits=[0,1], ylimits=[-1,1], save=None):
    '''
    Plots signal with abscissa in x and ordinates in y

    '''
    if ax is None:
        fig = plt.figure(figsize=(12,6))
        ax = plt.gca()

    plt.plot(x, y, linestyle=line_style, linewidth=line_width, color=plot_colour,
        label=legend_label)
    if legend_label and legend_show:
        plt.legend(loc=legend_loc, frameon=True, framealpha=0.8, facecolor='white')
    plt.xlabel(xaxis_label)
    plt.ylabel(yaxis_label)

    if show:
        plt.xlim(xlimits)
        plt.ylim(ylimits)
        plt.title(title_text)
        if save:
            plt.savefig(save + '.pdf', format='pdf')
        plt.show()

    return

# %% OPERATORS

def covariance_mtx(var):
    ''' Returns covariance matrix of range differences '''
    return var * np.array([[2,-1,0], [-1,2,-1], [0,-1,2]])

def target_anchor_distance(x, anchors):
    ''' Returns distance between target and anchors '''
    return np.linalg.norm(x-anchors, axis=0)

def dist_diff(distances):
    ''' Returns distance differences '''
    return -1.0 * np.diff(distances)

```

```
def dist_derivative(x, anchors, distances):
    ''' Returns derivative of distance differences '''
    num_anchors = anchors.shape[1]
    return (((x - anchors)/distances)[:,:num_anchors-1] -
        ((x - anchors)/distances)[:,1:]).T

# %% ACCURACY ANALYSIS

def crlb_tdoa(target, anchors, var):
    '''
    Returns the Fisher information matrix of 2D TDOA

    :param target: 2D position of the target
    :param anchors: 2D position of the anchors
    :param var: Variance of the noise in range measurements

    :return fim: Fisher information matrix

    '''

    num_anchors = anchors.shape[1]

    # Noise statistics
    covW = covariance_mtx(var)
    presW = np.linalg.inv(covW)

    # Distance vector
    target_anchor_dist = target_anchor_distance(target, anchors)

    del_dist_vec = dist_derivative(target, anchors, target_anchor_dist)

    # Fisher information matrix
    fim = (del_dist_vec.T.dot(presW)).dot(del_dist_vec)

    return fim

# %% SOLVERS

def mle_tdoa(distances, anchors, init_pos, var=1., step_size=0.1, max_iter=100):
    '''
    Returns maximum-likelihood estimate (MLE) for TDOA localisation.
    Gradient descent with fixed step-size is used as the solver.

    :param distances: distance measurements
    :param anchors: 2D position of anchors
    :param init_pos: Initialisation for gradient descent
    :param step_size: Step size of gradient descent optimiser
    :param max_iter: Maximum iterations before exit
    :param var: variance of noise in distance measurements

    :returns: mle estimate for position

    '''

    covN = covariance_mtx(var)
    presN = np.linalg.inv(covN)

    range_diff = dist_diff(distances)
    mle_estimate = init_pos

    for _ in range(max_iter):
```

```
        dx = target_anchor_distance(mle_estimate, anchors)
        dx_vec = dist_diff(dx)
        del_dx = dist_derivative(mle_estimate, anchors, dx)
        mle_estimate += step_size * ((del_dx.T).dot(presN)).dot(range_diff -
            dx_vec)[:,np.newaxis]

    return mle_estimate

def blue_tdoa(distances, anchors, var=1.):
    '''
    Returns best linear unbiased estimate (BLUE) for TDOA localisation.

    :param distances: distance measurements
    :param anchors: 2D position of anchors
    :param var: variance of noise in distance measurements

    :returns: blue estimate for position

    '''

    num_anchors = anchors.shape[1]

    cov_mtx = 2.0*var**2 * np.array([[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]], dtype=np.float)
    pres_mtx = np.linalg.inv(cov_mtx)
    r = np.zeros(6)
    gamma_vec = np.zeros(6)
    H_mtx = np.zeros([6, 5])

    idx = 0
    for i in range(num_anchors):
        for j in range(i+1, num_anchors):
            r[idx] = distances[i]-distances[j]
            gamma_vec[idx] = (r[idx])**2 - np.linalg.norm(anchors[:,i])**2 + np.linalg.
    norm(anchors[:,j])**2 - 2*var
            idx = idx+1

    idx = 0
    for anchor_idx in range(num_anchors - 1):
        lst = list(range((anchor_idx), num_anchors - 1))
        for pos in lst:
            H_mtx[idx,0] = -2*(anchors[0,anchor_idx] - anchors[0,(pos+1)])
            H_mtx[idx,1] = -2*(anchors[1,anchor_idx] - anchors[1,(pos+1)])

            H_mtx[idx,(pos+2)] = -2*r[idx]
            idx = idx+1

    den = np.linalg.inv((((H_mtx.T).dot(pres_mtx)).dot(H_mtx)))
    num = ((H_mtx.T).dot(pres_mtx)).dot(gamma_vec-2*var)
    blue_est = den.dot(num)

    return (blue_est[:2])[:,np.newaxis]
```