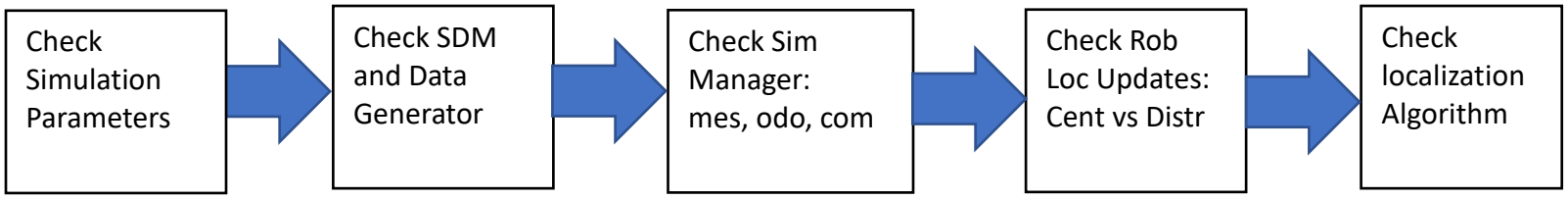
Sagar Doshi

Colo-AT Simulated Data Debug Report

The purpose of this document is to aid future users in debugging their usage of Colo-AT, and to acclimate new users to its implementation.

General Debugging Structure



The first step in running/debugging Colo-AT using simulated data is to create a test script.

1. Import the necessary python packages, Colo-AT Modules, and Localization Algorithms. This Report focuses on use of the simulated dataset manager, highlighted in red.

```python
# Import necessary Libraries
import os, sys
import numpy as np
from math import pi, sqrt

# Import other Colo-AT Modules
from dataset_manager.realworld_dataset_manager import RW_Dataset_Manager
from dataset_manager.simulated_dataset_manager import SimulatedDataSetManager
from simulation_process.sim_manager import  SimulationManager
from robots.robot_system import RobotSystem
from simulation_process.state_recorder import StatesRecorder
from data_analysis.data_analyzer import Analyzer
from data_analysis.realtime_plot import animate_plot

# Import Localization Algorithms
sys.path.append(os.path.join(os.path.dirname(__file__), "localization_algos"))

# LS -> centralized system.
from centralized_ekf import Centralized_EKF
from ekf_ls_bda import EKF_LS_BDA
from ekf_ls_ci import EKF_LS_CI
# GS -> distributed system
from ekf_gs_bound import EKF_GS_BOUND
from ekf_gs_ci2 import EKF_GS_CI2
from ekf_gs_sci2 import EKF_GS_SCI2
```

If you run into import errors here are some quick fixes/notes:

- If you add elements to the test script, don't forget to import additional libraries
- Add *__init__.py* file to additional folders created w/modules used in the test script.
- A lot of python linters mark non-existent import errors (like in the sample code above). See linter documentation for getting rid of this.
- Depending on if you moved folders/files in the repository, you may have to add/remove elements from the system path. Google documentation for how Python searches for different modules in a repository for more info on this\how to handle on different OS.
- Google the exact import error you are getting (most import errors can be quickly resolved this way)

2. Set-Up the simulation to use generated data. This template file is *test_simulation_sagar.py*

```
# Template code for running/debugging simulated data

# Set a random seed for replication
np.random.seed(1)

# Initialize landmark map, landmark IDs must be larger than 5
# Format {LandmarkID: [x,y]}
landmarks = {11: [1,0], 12: [0,1], 13: [-1,0], 14: [0,-1]}

# Initalize Robots, can be 1-5
robot_labels = [1,2,3]

# Specify simulation parameters:
# Units are [m], [s], and [rad]
duration = 120
delta_t = 0.2

v_noise = 0.05
w_noise = 0.0001
r_noise = 0.05
phi_noise = sqrt(2)*pi/180
fov = 2*pi

v = 0.1
sigma_v = 0.0

# Instantiate Simulated Dataset Manager
testing_dataset = SimulatedDataSetManager('name', landmarks, duration, robot_labels,
velocity_noise=v_noise, angular_velocity_noise=w_noise, measurement_range_noise=r_noise, bearing_noise=phi_noise,
robot_fov=fov, delta_t=delta_t)

# Generate the data
start_time, starting_states, dataset_data, time_arr = testing_dataset.simulate_dataset('random', test=True, velocity=v, velocity_spread=sqrt(sigma_v))

# Create Data Analyzer
analyzer = Analyzer('analyzer', robot_labels)

# Specify Loc Algo and create RobotSystem
loc_algo =  Centralized_EKF('Centralized_EKF')
robot = RobotSystem('robot', robot_labels, loc_algo, distr_sys = False)

# Create the simulation manager and state recorder
sim = SimulationManager('sim')
state_recorder = StatesRecorder('Centralized_EKF', robot_labels)

# Run the simulation
end_time = sim.sim_process_native(robot_labels, testing_dataset, robot, state_recorder, simple_plot = True, comm=False, simulated_comm = False)

# Analyze the data
loc_err_per_run, state_err_per_run, trace_per_run, time_arr = analyzer.calculate_loc_err_and_trace_state_variance_per_run(state_recorder, plot_graphs = False)

# View an animated display of robot movement and loc algo performance
animate_plot(robot_labels, state_recorder, analyzer, testing_dataset.get_landmark_map())
```

See the Colo-AT manual on the GitLab repository to see the parameters and I/O of all the modules. Make sure to check that the simulation parameters are reasonable. "Reasonable" means to check for issues such as:

- Negative quantities passed as arguments
- The noise/spread parameters are std. deviations, **not** variances
- Check decimal errors (ex: error of 0.1 vs 0.01)
- Check units: [m], [s], [rad]

3. Run the test script w/desired parameters and localization algorithm. Check if the results are reasonable using a few techniques.

```
start_time, starting_states, dataset_data, time_arr = testing_dataset.simulate_dataset('path_type', test=True, velocity=v, velocity_spread=sqrt(sigma_v))
```

(a) When generating the data, there is pre-built test function that displays informative distributions describing the generated data. An example is shown below.

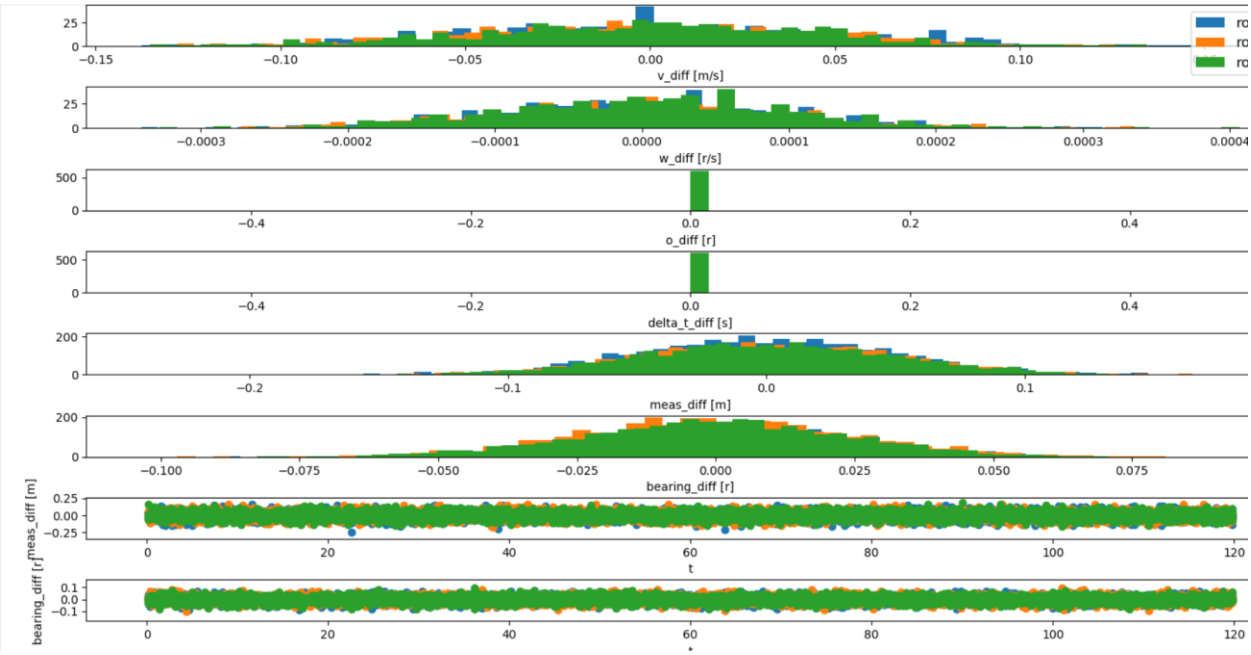


Figure 1 – Data Verifications Subplots. From top to bottom, the plots show the following:

- Distribution of $v_{diff} = v_{actual} - v_{recorded} \sim N(0, \sigma_v)$
- Distribution of $\omega_{diff} = \omega_{actual} - \omega_{recorded} \sim N(0, \sigma_\omega)$
- Distribution of θ_{diff} *and* t_{diff}. These should be 0 given that there is no noise with respect to groundtruth orientation or time.
- Distribution of $r_{diff} = r_{actual} - r_{measured} \sim N(0, \sigma_r)$ (Compared using Euclidean distance formula)
- Distribution of $\phi_{diff} = \phi_{acutal} - \phi_{measured} \sim N(0, \sigma_\phi)$
- Plots of r_{diff} and ϕ_{diff} vs. time. These can be used to spot outliers (if necessary). This example does not show much in terms of significant outliers.

(b) Assuming the generated data meets expectations, then the trace of the covariance matrix and a plot of the RMS error can be used to determine correctness/reasonability of simulation results.

```
end_time, diffs = sim.sim_process_native(robot_labels, testing_dataset, robot, state_recorder, simple_plot = True, comm=False, simulated_comm = False)
```

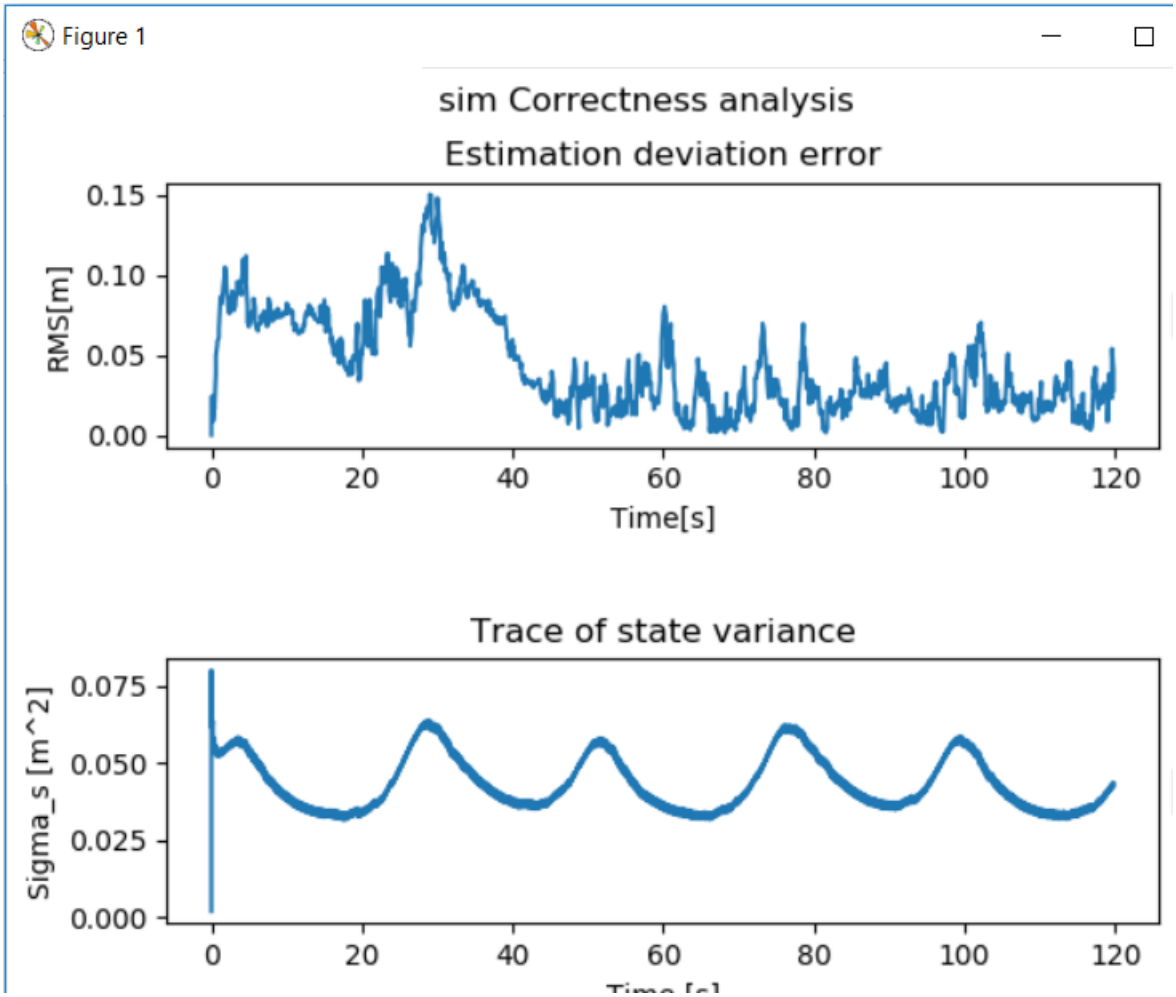


Figure 2 – Example trace for EKF_GS_CI_2 algorithm in Colo-AT.

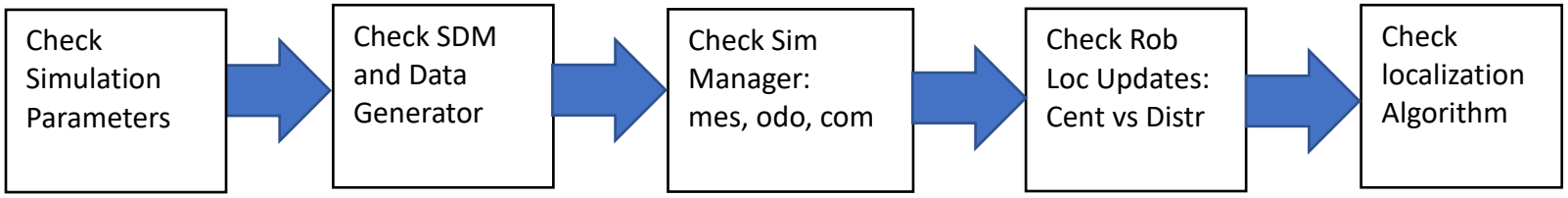
Looking at the mathematical implementation for this algorithm, the RMS (root-mean square of the location error) does NOT scale with distance, but the trace of the state variance σ^2 does. In this example there is a single landmark, and a robot with a $360°$ view is moving in a circle towards and away from the landmark. As the robot gets farther from the landmark, the RMS does not change, but σ^2 grows and recedes. This qualitatively verifies the simulation.

(c) Sometimes it is easier to visualize/figure out what's going on during the simulation with a live animation.

```
# View an animated display of robot movement and loc algo performance
animate_plot(robot_labels, state_recorder, analyzer, testing_dataset.get_landmark_map())
```

This function creates an animated plot of the robots' actual movements vs the algorithm estimation of the location. In addition, there is a live-updated view of the RMS and σ^2 similar to *figure 2.*

Techniques (a), (b), and (c) provide a quick way to confirm the results of the simulation with generated data. If the simulation results/generated data do not meet expectations, then here are some pointers for debugging/tracing through Colo-AT from my experience doing so.



The general outline is reproduced here, and the current step is the 2nd box.

The Simulated Dataset Manager and Data Generator are two editable classes whose I/O is described in the Colo-AT user manual.

Within the Simulated Dataset Manager:

- Double check initialization of starting states: this function can be modified to meet user requirements (at the risk of invalid starting states being provided).
- The general structure for the dataset_data array is:
  - {'odometry': [], 'measurement': [], 'groundtruth': []}
  - Each subarray within the top arrays represent data for on individual robot (in order of robot_labels provided to simulation)
- Trace through the following functions:
  - respond()
  - get_dataline()
  - find_measurement()

These functions are built case-wise, and this can be used to determine the potential source of error (especially if these have been modified to custom-fit user input).

Within the Data Generator:

- verify_generated_data() is the function that created the distribution plots to see if generated data was reasonable
- Comment out the noise distribution if you want to investigate the effect off noise on the data (then go through steps 3 (a)-(c) to see if this created the expected/desired change on the simulation results).
    - Similarly, you can eliminate noise and add a bias to see if this produces the expected effect

```
measurement_range = self.calc_distance(robot_loc, landmark_loc) + np.random.normal(loc=0.0,scale=self.measurement_range_noise)
```

- Modify some of the internal parameters such as radian_step in circular data generation to better fit desired simulation
- The same generate_odometry_data() and generate_measurement_data() functions are used for all data generation – the only thing that changes is groundtruth data
- The primary bugs/changes would be made within:
    - generate_circular_data()
    - generate_random_data()
    - generate_straight_line_data()
- recall that both orientation and bearing θ, ϕ are between $-\pi \; and \; \pi$
- ϕ is shifted to be within the local coordinate frame of the robot

If the error is beyond the Simulated DataSet Manager or DataGenerator, this means that something may be occurring with how data is being transferred throughout Colo-AT. It is then worthwhile to view the overall class structure of Colo-AT to see where the issue originates.

CoLo-AT Structure:

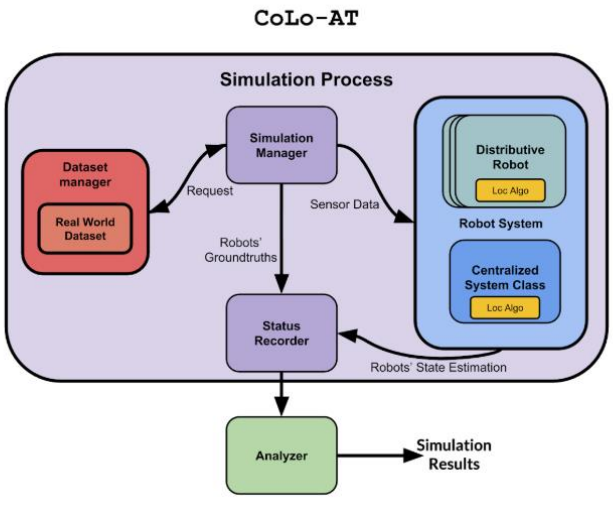


Figure 3 – Courtesy of William Chen from the Colo-AT Manual.

This figure directs a trace through of the sim manager → robot system → state recorder

The Simulated Dataset Manager communicates with the simulation manager via the request response class, so if invalid requests or invalid responses are sent, this ruins the entire simulation. Beyond this, the sim manager facilitates the running of the robot system and state recorder. It is the "highest-level" of the overall simulation process.

Delving into its functions, the simulated dataset manager currently only supports the mode of:

sim_process_native().

Within this function, there are two operations that happen.

- localization_update()
- communication

The localization update is in the robot_system, and the robot system also contains the initializations of the covariance matrices (which **should be modified by the user** to be close to the noise passed as simulation parameters).

A **note** about the communication protocol is that there must be at least 3 robots for simulated communication to function. Print statements can be inserted into the if comm: … portion of the sim manager to see if the simulated communication protocol is functioning as expected.

The sim manager can be used to print out various metrics/data fetched from:

- State recorder
- robot system
    - centralized
    - distributive
- localization_algorithm

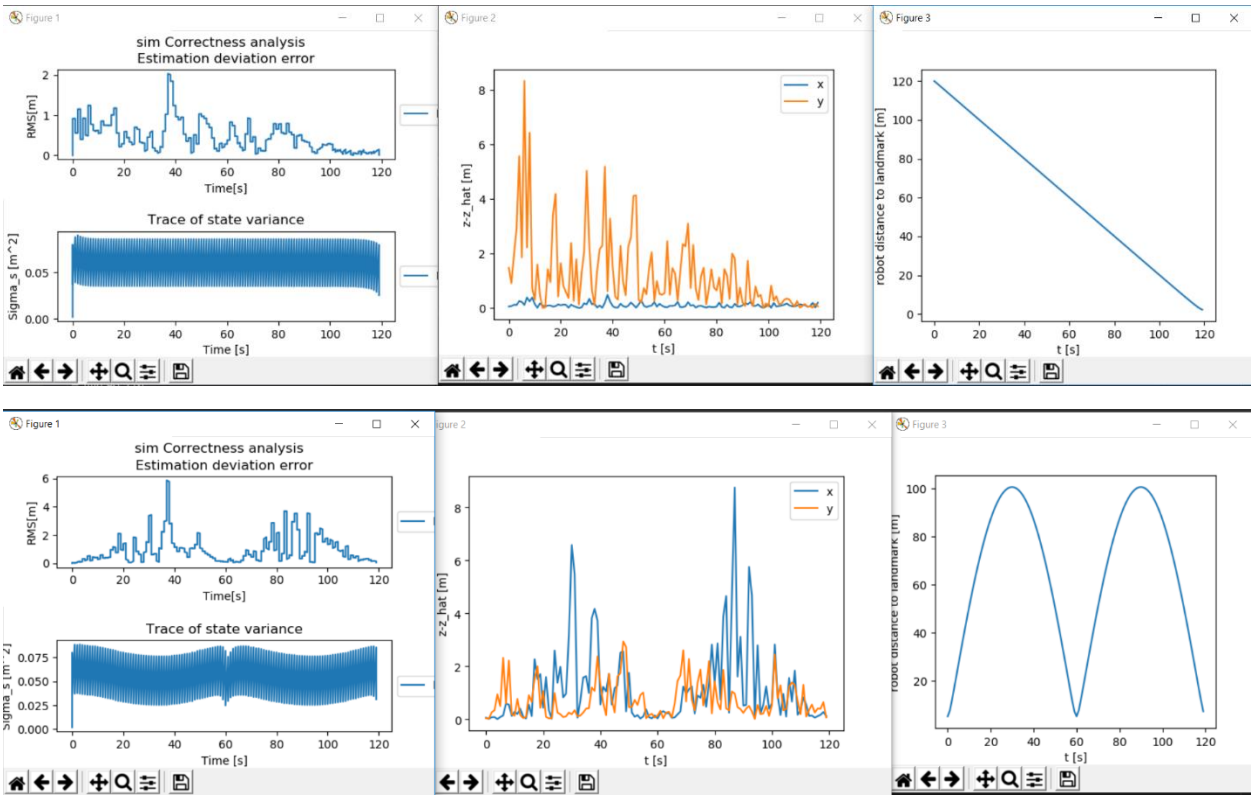
And these metrics/data include but are not limited to:

- Q, R Matrices
- z, $\hat{z}$
- covariance matrix
- robot state

This is the starting point of viewing what mathematical quantities are being incorrectly calculated, and this can lead to the source of the error.

The last part of this document focuses on a trace through that was used to pinpoint the source of some simulation discrepancy.

The discrepancy is shown in the following two figures.



Figures 4 & 5 – Displays of RMS/trace for one robot, Centralized_EKF

In figure 4, the robot is moving in a straight line towards a single landmark.

In figure 5, the robot is moving a circle towards and away from single landmark.

In both cases, the RMS clearly grows and subsides as the robot gets farther/closer to the landmark.

After verifying the Simulated Dataset Manager and Data Generator, print statements were used inside the sim manager to print out important metrics. Most notably, the localization algorithm was modified to also return $|z - \hat{z}|$ which is the middle plot in both figures 4 and 5.

There did not appear to be any discrepancies in the transfer in data, so taking a closer look at the algorithm implementation yielded an unintended noise dependency on distance. This resolved the issue.

Template code used to create the $|z - \hat{z}|$ plot is provided below. Note that the plotting code was in the test script, and the only modification to the other modules of Colo-AT was the "passing-up" of data/information.

```
# Investigation of RMS/trace corresponding to meas_range
x_diffs = diffs['x_diff']
y_diffs = diffs['y_diff']

times = np.arange(0, 120, delta_t)

x_diffs = list(filter(lambda a: a != -1, x_diffs))
x_diffs = [np.asscalar(x) for x in x_diffs]
y_diffs = list(filter(lambda a: a != -1, y_diffs))
y_diffs = [np.asscalar(y) for y in y_diffs]

robot_x = [gt['x_pos'] for gt in dataset_data['groundtruth'][0]]
robot_y = [gt['y_pos'] for gt in dataset_data['groundtruth'][0]]

robot_locs = [ [robot_x[i], robot_y[i] ] for i in range(0, len(robot_x))]
distances = [calc_distance(landmarks[11], robot_loc) for robot_loc in robot_locs]


fig = plt.figure(2)

plt.plot(times, x_diffs, times, y_diffs)

plt.xlabel('t [s]')
plt.ylabel('|z-z_hat| [m]')

plt.legend(['x', 'y'], loc='upper right')

fig = plt.figure(3)

plt.plot(times, distances)
plt.xlabel('t [s]')
plt.ylabel('robot distance to landmark [m]')

plt.show()
```

The diffs variable is $|z - \hat{z}|$, and this was passed from the localization algorithm to the robot system to the state recorder before finally reaching the simulation manager which would return the arrays of $|z - \hat{z}|$ for varying times to the test script.

Performing a similar trace through going through:

sim manager → robot system → state recorder

Before checking the localization algorithm eliminates the possibility of data transfer/code issues which may be quicker to resolve than reformulating the mathematics of an algorithm.

Some other tests that can be run may include tabulating the effect of varying certain simulation parameters. Some examples are listed below.

Base Simulation Parameters:

```
landmarks = {11: [1,0], 12: [0,1], 13: [-1,0], 14: [0,-1]}
robot_labels = [1]
duration = 120
delta_t = 0.2

v_noise = 0.05
w_noise = 0.0001
r_noise = 0.05
phi_noise = sqrt(2)*pi/180
fov = 2*pi
v = 0.1
sigma_v = 0.0
simulate_dataset('random', test=True, velocity=v, velocity_spread=sqrt(sigma_v))
loc_algo =  EKF_GS_CI2('EKF_GS_CI2')
```

Varying Measurement Range Noise

$\sigma_r\ [m]$	Avg Loc Err [m]	Avg trace of state var
0.05	0.0213	0.0114
0.10	0.0283	0.0114
0.30	0.0603	0.0113

Varying Velocity

$v\ [m/s\ ]$	Avg Loc Err [m]	Avg trace of state var
0.1	0.0213	0.0114
0.25	0.0273	0.0195
0.5	0.0302	0.0318

Varying Velocity Spread

$\sigma_v\ [m/s\ ]$	Avg Loc Err [m]	Avg trace of state var
0.01	0.0222	0.0115
0.05	0.0707	0.0117
0.1	0.108	0.0120

Overall, I hope this document is helpful for users trying to run Colo-AT, and I hope it aids in resolving errors that may occur when using the Simulated Dataset Manager & Data Generator.