

E1. Localization

Define all the imports

Import all the dependencies required to launch the notebook

```
In [334... # Import main dependencies
import sys
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import time
```

```
In [335... # Change to main repo folder for the imports
_, dir = os.path.split(os.getcwd())
if dir == 'notebooks':
    os.chdir('..')
    sys.path.append(os.getcwd())
# Ignore warnings from pandas
pd.set_option('mode.chained_assignment', None)
```

```
In [336... # Import methods from our local library
from src.localization.PF import ParticleFilter
from src.localization.EKF import ExtendedKalmanFilter
```

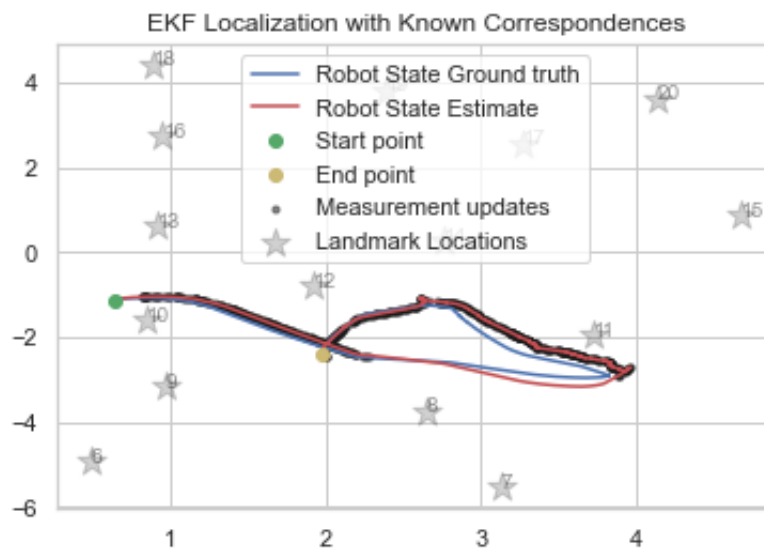
Set dataset

```
In [337... # Define the dataset to be used for the following tests.
dataset = "data/MRCLAM_Dataset4" # Dataset
end_frame = 10000 # Extension of the dataset
robot = 'Robot5' # Robot
```

T1. Review EKF localization

Review the ExtendedKalmanFilter() method, execute the EKF localization of the next code block, and answer the questions below:

```
In [338... # Build Extended Kalman Filter object
R = np.diagflat(np.array([1.0, 1.0, 10.0]))** 2
Q = np.diagflat(np.array([300, 300, 1e16]))** 2
ekf = ExtendedKalmanFilter(dataset, robot, end_frame, R, Q)
```



T11 The EKF localisation algorithm iteratively executes the functions `motion_update()` and `measurement_update()` to propagate the robot's state estimate. Could you specify, respectively for each of the two functions, when are they called and what is their purpose?

(Answer here max 150 words)

`motion_update()`

- called whenever there is odometry data available. This happens when the robot moves, and it gets data on its motion (e.g. forward movement or rotation)
- Purpose: predict the robot's next state based on the current state and the motion data

`measurement_update()`

- is called whenever there is sensor measurement data available. This happens when the robot receives information from its sensors about the environment
- Purpose: to correct the robot's predicted state (from the motion update) based on the new sensor information

T12 The `ExtendedKalmanFilter()` method inputs the R and Q matrices. Could you describe what are they used for and how they affect to the resulting localization estimation?

(Answer here max 150 words)

R Matrix: Process Noise Covariance Matrix

- the uncertainty in the robot's motion model during the prediction (motion update) step
- is added to the state covariance matrix after each motion update
- larger values in R: indicate a higher uncertainty in the robot's motion model --> cause the EKF to place less trust in the predicted state, leading to a larger

predicted state covariance

- smaller values in R: indicate robot's motion model is more reliable and accurate -
-> EKF will place more trust in the predicted state, leading to a smaller predicted state covariance

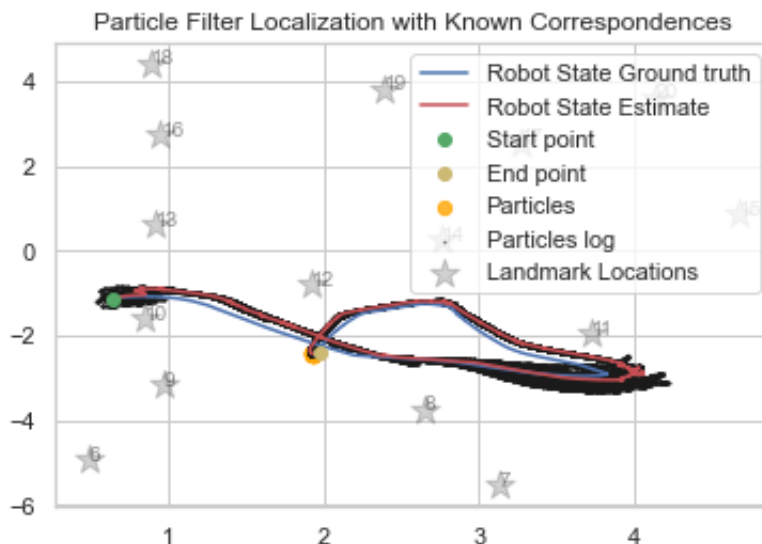
Q Matrix: Measurement Noise Covariance Matrix

- accounts for the uncertainty in the robot's sensor model during the correction (measurement update) step
- quantifies how much we trust the sensor measurements
- larger values in Q: indicate a higher uncertainty in the sensor measurements --> EKF will place less trust in the measurements, relying more on the predicted state from the motion model
- smaller values in Q: indicate that the sensor measurements are more accurate and reliable --> The EKF will place more trust in the sensor data --> smaller state covariance after correction

T2. Review PF localization

Review the ParticleFilter() method and execute the PF localization of the next code block. Then answer the questions below:

```
In [339... # Build Particle Filter object
num_particles = 20
motion_noise = np.array([0.1, 0.1, 0.1, 0.2, 0.2]) # Motion model noise [
measurement_noise = np.array([0.1, 0.1]) # Measurement model noise (in me
pf = ParticleFilter(dataset, robot, end_frame, num_particles, motion_nois
```



T2.1 The PF localisation algorithm iteratively executes the functions `motion_update()`, `measurement_update()`, `importance_sampling()` and `state_update()`. Could you specify, the goal of each of the functions?

(Answer here max 150 words)

motion_update()

- goal: To propagate the particles based on the robot's motion. It applies the motion model to each particle, predicting where the robot might be after a time step given the control inputs (odometry data).

measurement_update()

- goal: To evaluate how well each particle matches the observed sensor measurements. It computes the likelihood (weight) of each particle based on how closely the predicted sensor reading (based on the particle's state) matches the actual sensor reading.

importance_sampling()

- goal: To resample the particles according to their weights. Particles with higher weights (those that are more consistent with the actual sensor data) are more likely to be selected, while particles with lower weights may be discarded.

state_update()

- goal: To compute the robot's estimated state based on the current set of particles. It does this by taking the mean of all the particles.

T2.1 The ParticleFilter() method inputs three different configuration parameters: num_particles, motion_noise, measurement_noise. Could you describe purpose and how they affect to the resulting estimation?

(Answer here max 150 words)

num_particles

- determines the number of particles used to represent the distribution of possible robot states (position and orientation). Each particle is a hypothesis about where the robot might be.
- Affects the accuracy and computational cost. More particles improve accuracy but the computational cost is higher

motion_noise

- represents the uncertainty in the robot's motion model. It describes how much noise is present in the robot's motion due to imperfect control.
- Higher motion noise causes the particles to spread out more
- lower motion noise keeps them closer to the predicted state

measurement_noise

- represents the uncertainty in the robot's sensor measurements. It describes how much error is present in the data.

- Affects how much trust is placed in the sensor data. Higher measurement noise reduces the influence of sensor data
- lower measurement noise makes the filter more responsive to sensor readings

T3. Benchmark

Compare the EKF and the PF algorithms using different configurations for each algorithm and different datasets. For the comparison use the error metric introduced in previous lab sessions. You may also want to use the measurement density metric from the first lab to characterize the datasets complexity.

The objective of this task is to be able to determine the localization algorithm and configuration that provides the best performance. In order to decide which model and configuration works best follow the next steps:

1. Extract the benchmark metrics for each dataset and model configuration.
2. Represent the results in different types of plots (for instance catplot or correlation matrix).
3. Analyze the results and extract conclusions.

Extract error metrics

```
In [340... from src.data.reader import Reader
# Code from Lab0
def build_timeseries(data, cols):
    timeseries = pd.DataFrame(data, columns=cols)
    timeseries['stamp'] = pd.to_datetime(timeseries['stamp'], unit='s')
    timeseries = timeseries.set_index('stamp')
    return timeseries

def filter_static_landmarks(lm, barcodes):
    for L, l in dict(barcodes).items(): # Translate barcode num to landmark
        lm[lm==l]=L
    lm = lm[lm.type > 5] # Keep only static landmarks
    return lm
```

```
In [341... datasets = ['./data/MRCLAM_Dataset1',
                './data/MRCLAM_Dataset2',
                './data/MRCLAM_Dataset3',
                './data/MRCLAM_Dataset4']

robots = ['Robot1',
          'Robot2',
          'Robot3',
          'Robot4']

errors = pd.DataFrame(columns=['dataset', 'robot', 'model_name'])

# I tried out more than these configurations,
# but because of the long computational time I focused on these
ekf_configs = [
```

```

(np.diag([1, 1, 1.0])**2, np.diag([300, 300, 1e16])**2),
(np.diag([1, 1, 10.0])**2, np.diag([1000, 1000, 1e20])**2),
(np.diag([0.1, 0.1, 10.0])**2, np.diag([500, 500, 1e15])**2),
(np.diag([100, 100, 100.0])**2, np.diag([50, 50, 1e15])**2),
]

pf_configs = [
    (1, np.array([0.1, 0.1, 0.1, 0.2, 0.2]), np.array([0.1, 0.1])),
    (10, np.array([0.2, 0.2, 0.2, 0.2, 0.2]), np.array([1, 1])),
    (10, np.array([0.2, 0.2, 0.2, 0.2, 0.2]), np.array([0.01, 0.01])),
    (10, np.array([0.2, 0.2, 0.2, 0.2, 0.2]), np.array([0.1, 0.1])),
    (10, np.array([1, 1, 1, 1, 1]), np.array([0.1, 0.1])),
    (50, np.array([0.1, 0.1, 0.1, 0.2, 0.2]), np.array([0.1, 0.1]))
]

def compute_error_metrics(data):
    # Calculate Euclidean distance between (x, y) and (x_gt, y_gt)
    eucl_dist = np.sqrt((data['x'] - data['x_gt'])**2 + (data['y'] - data

    # mean distance error
    mean_error = np.mean(eucl_dist)

    # maximum error
    max_error = np.max(eucl_dist)

    return mean_error, max_error

def convert_to_dataframe(states, gt):
    df_states = pd.DataFrame(states, columns=['stamp', 'x', 'y', 'theta'])
    df_gt = pd.DataFrame(gt, columns=['stamp', 'x_gt', 'y_gt', 'theta_gt'])
    merged_data = df_states.merge(df_gt, on='stamp', how='inner')

    return merged_data

def compute_measurement_density(model, landmarks):
    #Mean number of measures per unit of distance
    traveled_distance = np.sum(
        np.sqrt(
            (model.groundtruth_data[1:, 1] - model.groundtruth_data[:-1,
            (model.groundtruth_data[1:, 2] - model.groundtruth_data[:-1,
        )
    )
    # to avoid division by zero
    if traveled_distance > 0:
        number_landmarks = landmarks.shape[0]
        return number_landmarks / traveled_distance
    else:
        return 0

errors_list = []

for ds in datasets:
    for rob in robots:

```

```

#load data
self = Reader(ds, rob, end_frame)
measurements = build_timeseries(self.data, cols=['stamp', 'type', 'x', 'y', 'z'])
landmarks = measurements[measurements.type != -1]
landmarks = filter_static_landmarks(landmarks, self.barcodes_data)

# Compute models for EKF
for i, (R, Q) in enumerate(ekf_configs):
    start_time = time.time()
    ekf_model = ExtendedKalmanFilter(dataset=ds, robot=rob, end_frame=end_frame, R=R, Q=Q)

    data_ekf = convert_to_dataframe(ekf_model.states, ekf_model.ground_truth)
    # Compute ATE for EKF
    ekf_mean_error, ekf_max_error = compute_error_metrics(data_ekf)

    end_time = time.time()
    computation_time_ekf = end_time - start_time

    measurement_density_ekf = compute_measurement_density(ekf_model.ground_truth, ekf_model.states)

    errors_list.append({
        'dataset': ds,
        'robot': rob,
        'model_name': f'EKF_Config_{i+1}',
        'mean_error': ekf_mean_error,
        'max_error': ekf_max_error,
        'computation_time': computation_time_ekf,
        'measurement_density': measurement_density_ekf
    })

# Compute models for PF
for i, (num_particles, motion_noise, measurement_noise) in enumerate(pf_configs):
    start_time = time.time()
    pf_model = ParticleFilter(dataset=ds, robot=rob, end_frame=end_frame, num_particles=num_particles,
                              motion_noise=motion_noise, measurement_noise=measurement_noise)

    # Convert numpy arrays (states and ground truth) to pandas DataFrames
    data_pf = convert_to_dataframe(pf_model.states, pf_model.ground_truth)

    # Compute ATE for PF
    pf_mean_error, pf_max_error = compute_error_metrics(data_pf)

    end_time = time.time()
    computation_time_pf = end_time - start_time
    measurement_density_pf = compute_measurement_density(pf_model.ground_truth, pf_model.states)

    errors_list.append({
        'dataset': ds,
        'robot': rob,
        'model_name': f'PF_Config_{i+1}',
        'mean_error': pf_mean_error,
        'max_error': pf_max_error,
        'computation_time': computation_time_pf,
        'measurement_density': measurement_density_pf
    })

```

```

errors = pd.DataFrame(errors_list)
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)

print(errors)

```

	dataset	robot	model_name	mean_error	max_error
0	./data/MRCLAM_Dataset1	Robot1	EKF_Config_1	0.593929	1.375293
1	./data/MRCLAM_Dataset1	Robot1	EKF_Config_2	0.624860	1.562661
2	./data/MRCLAM_Dataset1	Robot1	EKF_Config_3	0.668923	1.678101
3	./data/MRCLAM_Dataset1	Robot1	EKF_Config_4	0.712387	3.543481
4	./data/MRCLAM_Dataset1	Robot1	PF_Config_1	2.173366	3.911661
5	./data/MRCLAM_Dataset1	Robot1	PF_Config_2	0.330158	0.548427
6	./data/MRCLAM_Dataset1	Robot1	PF_Config_3	0.104285	0.323130
7	./data/MRCLAM_Dataset1	Robot1	PF_Config_4	0.175197	0.451667
8	./data/MRCLAM_Dataset1	Robot1	PF_Config_5	0.426659	2.829524
9	./data/MRCLAM_Dataset1	Robot1	PF_Config_6	0.187858	0.520795
10	./data/MRCLAM_Dataset1	Robot2	EKF_Config_1	0.140073	0.455359
11	./data/MRCLAM_Dataset1	Robot2	EKF_Config_2	0.166784	0.645132
12	./data/MRCLAM_Dataset1	Robot2	EKF_Config_3	0.213673	0.625268
13	./data/MRCLAM_Dataset1	Robot2	EKF_Config_4	0.449107	2.589557
14	./data/MRCLAM_Dataset1	Robot2	PF_Config_1	0.486207	0.656377
15	./data/MRCLAM_Dataset1	Robot2	PF_Config_2	0.468984	0.764351
16	./data/MRCLAM_Dataset1	Robot2	PF_Config_3	0.136321	0.364094
17	./data/MRCLAM_Dataset1	Robot2	PF_Config_4	0.120500	0.352707
18	./data/MRCLAM_Dataset1	Robot2	PF_Config_5	0.232884	0.676438
19	./data/MRCLAM_Dataset1	Robot2	PF_Config_6	0.097774	0.320493
20	./data/MRCLAM_Dataset1	Robot3	EKF_Config_1	0.176818	0.680936
21	./data/MRCLAM_Dataset1	Robot3	EKF_Config_2	0.111109	0.296930
22	./data/MRCLAM_Dataset1	Robot3	EKF_Config_3	0.127264	0.326093
23	./data/MRCLAM_Dataset1	Robot3	EKF_Config_4	0.760793	6.504405
24	./data/MRCLAM_Dataset1	Robot3	PF_Config_1	1.139988	1.611789
25	./data/MRCLAM_Dataset1	Robot3	PF_Config_2	0.449264	0.599046
26	./data/MRCLAM_Dataset1	Robot3	PF_Config_3	0.156883	0.331050
27	./data/MRCLAM_Dataset1	Robot3	PF_Config_4	0.104274	0.285561
28	./data/MRCLAM_Dataset1	Robot3	PF_Config_5	0.323673	1.209050
29	./data/MRCLAM_Dataset1	Robot3	PF_Config_6	0.083090	0.218620
30	./data/MRCLAM_Dataset1	Robot4	EKF_Config_1	0.432464	0.929490
31	./data/MRCLAM_Dataset1	Robot4	EKF_Config_2	0.839798	2.701074
32	./data/MRCLAM_Dataset1	Robot4	EKF_Config_3	1.477167	4.870908
33	./data/MRCLAM_Dataset1	Robot4	EKF_Config_4	1.267313	3.343338
34	./data/MRCLAM_Dataset1	Robot4	PF_Config_1	0.716269	1.216564
35	./data/MRCLAM_Dataset1	Robot4	PF_Config_2	0.736966	1.124371
36	./data/MRCLAM_Dataset1	Robot4	PF_Config_3	0.147223	0.307223
37	./data/MRCLAM_Dataset1	Robot4	PF_Config_4	0.310326	0.699661
38	./data/MRCLAM_Dataset1	Robot4	PF_Config_5	0.483408	1.552654
39	./data/MRCLAM_Dataset1	Robot4	PF_Config_6	0.375250	1.213469
40	./data/MRCLAM_Dataset2	Robot1	EKF_Config_1	0.137301	0.340443
41	./data/MRCLAM_Dataset2	Robot1	EKF_Config_2	0.138592	0.293084
42	./data/MRCLAM_Dataset2	Robot1	EKF_Config_3	0.118793	0.245622
43	./data/MRCLAM_Dataset2	Robot1	EKF_Config_4	0.296233	0.789894
44	./data/MRCLAM_Dataset2	Robot1	PF_Config_1	0.952484	1.728932
45	./data/MRCLAM_Dataset2	Robot1	PF_Config_2	0.994134	1.662412
46	./data/MRCLAM_Dataset2	Robot1	PF_Config_3	0.081862	0.249111
47	./data/MRCLAM_Dataset2	Robot1	PF_Config_4	0.115228	0.271097
48	./data/MRCLAM_Dataset2	Robot1	PF_Config_5	0.388087	0.830045

49	./data/MRCLAM_Dataset2	Robot1	PF_Config_6	0.097768	0.239239
50	./data/MRCLAM_Dataset2	Robot2	EKF_Config_1	0.121794	0.285447
51	./data/MRCLAM_Dataset2	Robot2	EKF_Config_2	0.212436	0.615069
52	./data/MRCLAM_Dataset2	Robot2	EKF_Config_3	0.114199	0.308803
53	./data/MRCLAM_Dataset2	Robot2	EKF_Config_4	0.890747	2.954012
54	./data/MRCLAM_Dataset2	Robot2	PF_Config_1	0.643850	1.186549
55	./data/MRCLAM_Dataset2	Robot2	PF_Config_2	0.272991	0.617018
56	./data/MRCLAM_Dataset2	Robot2	PF_Config_3	0.193507	0.371624
57	./data/MRCLAM_Dataset2	Robot2	PF_Config_4	0.119061	0.319096
58	./data/MRCLAM_Dataset2	Robot2	PF_Config_5	0.619284	1.111638
59	./data/MRCLAM_Dataset2	Robot2	PF_Config_6	0.124524	0.353208
60	./data/MRCLAM_Dataset2	Robot3	EKF_Config_1	0.164281	0.415841
61	./data/MRCLAM_Dataset2	Robot3	EKF_Config_2	0.156735	0.348344
62	./data/MRCLAM_Dataset2	Robot3	EKF_Config_3	0.143138	0.318459
63	./data/MRCLAM_Dataset2	Robot3	EKF_Config_4	0.371933	1.524660
64	./data/MRCLAM_Dataset2	Robot3	PF_Config_1	0.867739	1.146713
65	./data/MRCLAM_Dataset2	Robot3	PF_Config_2	1.067544	1.821358
66	./data/MRCLAM_Dataset2	Robot3	PF_Config_3	0.163241	0.499292
67	./data/MRCLAM_Dataset2	Robot3	PF_Config_4	0.149897	0.412110
68	./data/MRCLAM_Dataset2	Robot3	PF_Config_5	0.355936	1.147783
69	./data/MRCLAM_Dataset2	Robot3	PF_Config_6	0.134443	0.438393
70	./data/MRCLAM_Dataset2	Robot4	EKF_Config_1	0.153877	0.404233
71	./data/MRCLAM_Dataset2	Robot4	EKF_Config_2	0.109125	0.321109
72	./data/MRCLAM_Dataset2	Robot4	EKF_Config_3	0.106440	0.333787
73	./data/MRCLAM_Dataset2	Robot4	EKF_Config_4	0.538884	2.220551
74	./data/MRCLAM_Dataset2	Robot4	PF_Config_1	1.471695	2.747947
75	./data/MRCLAM_Dataset2	Robot4	PF_Config_2	0.421875	0.718831
76	./data/MRCLAM_Dataset2	Robot4	PF_Config_3	0.163899	0.458694
77	./data/MRCLAM_Dataset2	Robot4	PF_Config_4	0.178469	0.488897
78	./data/MRCLAM_Dataset2	Robot4	PF_Config_5	0.340437	1.388878
79	./data/MRCLAM_Dataset2	Robot4	PF_Config_6	0.155868	0.448710
80	./data/MRCLAM_Dataset3	Robot1	EKF_Config_1	0.060016	0.148360
81	./data/MRCLAM_Dataset3	Robot1	EKF_Config_2	0.054269	0.146111
82	./data/MRCLAM_Dataset3	Robot1	EKF_Config_3	0.057996	0.150810
83	./data/MRCLAM_Dataset3	Robot1	EKF_Config_4	0.162305	1.151828
84	./data/MRCLAM_Dataset3	Robot1	PF_Config_1	1.031452	1.551347
85	./data/MRCLAM_Dataset3	Robot1	PF_Config_2	0.920112	1.776991
86	./data/MRCLAM_Dataset3	Robot1	PF_Config_3	0.081568	0.269785
87	./data/MRCLAM_Dataset3	Robot1	PF_Config_4	0.068807	0.265267
88	./data/MRCLAM_Dataset3	Robot1	PF_Config_5	0.396429	2.048252
89	./data/MRCLAM_Dataset3	Robot1	PF_Config_6	0.075288	0.238635
90	./data/MRCLAM_Dataset3	Robot2	EKF_Config_1	0.573486	1.740874
91	./data/MRCLAM_Dataset3	Robot2	EKF_Config_2	0.727863	2.183723
92	./data/MRCLAM_Dataset3	Robot2	EKF_Config_3	0.519045	1.397040
93	./data/MRCLAM_Dataset3	Robot2	EKF_Config_4	0.594950	12.045612
94	./data/MRCLAM_Dataset3	Robot2	PF_Config_1	0.300480	0.501518
95	./data/MRCLAM_Dataset3	Robot2	PF_Config_2	0.961190	1.721198
96	./data/MRCLAM_Dataset3	Robot2	PF_Config_3	0.169167	0.358901
97	./data/MRCLAM_Dataset3	Robot2	PF_Config_4	0.111066	0.223454
98	./data/MRCLAM_Dataset3	Robot2	PF_Config_5	0.147350	0.617876
99	./data/MRCLAM_Dataset3	Robot2	PF_Config_6	0.145387	0.340138
100	./data/MRCLAM_Dataset3	Robot3	EKF_Config_1	0.125507	0.485521
101	./data/MRCLAM_Dataset3	Robot3	EKF_Config_2	0.122977	0.397526
102	./data/MRCLAM_Dataset3	Robot3	EKF_Config_3	0.118661	0.354881
103	./data/MRCLAM_Dataset3	Robot3	EKF_Config_4	0.206961	0.994697
104	./data/MRCLAM_Dataset3	Robot3	PF_Config_1	1.085305	2.319639
105	./data/MRCLAM_Dataset3	Robot3	PF_Config_2	0.353094	0.622334

106	./data/MRCLAM_Dataset3	Robot3	PF_Config_3	0.118650	0.379710
107	./data/MRCLAM_Dataset3	Robot3	PF_Config_4	0.155477	0.556075
108	./data/MRCLAM_Dataset3	Robot3	PF_Config_5	0.185910	0.980233
109	./data/MRCLAM_Dataset3	Robot3	PF_Config_6	0.155064	0.378015
110	./data/MRCLAM_Dataset3	Robot4	EKF_Config_1	1.096087	1.797889
111	./data/MRCLAM_Dataset3	Robot4	EKF_Config_2	0.863528	1.764419
112	./data/MRCLAM_Dataset3	Robot4	EKF_Config_3	0.892935	1.499431
113	./data/MRCLAM_Dataset3	Robot4	EKF_Config_4	0.561277	3.210691
114	./data/MRCLAM_Dataset3	Robot4	PF_Config_1	0.216938	0.392048
115	./data/MRCLAM_Dataset3	Robot4	PF_Config_2	0.395131	0.760009
116	./data/MRCLAM_Dataset3	Robot4	PF_Config_3	0.097329	0.302837
117	./data/MRCLAM_Dataset3	Robot4	PF_Config_4	0.096915	0.348672
118	./data/MRCLAM_Dataset3	Robot4	PF_Config_5	0.487524	1.628254
119	./data/MRCLAM_Dataset3	Robot4	PF_Config_6	0.100015	0.290205
120	./data/MRCLAM_Dataset4	Robot1	EKF_Config_1	0.141117	0.373915
121	./data/MRCLAM_Dataset4	Robot1	EKF_Config_2	0.142389	0.318979
122	./data/MRCLAM_Dataset4	Robot1	EKF_Config_3	0.142112	0.284144
123	./data/MRCLAM_Dataset4	Robot1	EKF_Config_4	0.162202	2.109398
124	./data/MRCLAM_Dataset4	Robot1	PF_Config_1	0.532730	0.870583
125	./data/MRCLAM_Dataset4	Robot1	PF_Config_2	0.409286	0.678077
126	./data/MRCLAM_Dataset4	Robot1	PF_Config_3	0.098386	0.303862
127	./data/MRCLAM_Dataset4	Robot1	PF_Config_4	0.097794	0.309216
128	./data/MRCLAM_Dataset4	Robot1	PF_Config_5	0.185605	0.761174
129	./data/MRCLAM_Dataset4	Robot1	PF_Config_6	0.088680	0.317436
130	./data/MRCLAM_Dataset4	Robot2	EKF_Config_1	0.534286	2.158784
131	./data/MRCLAM_Dataset4	Robot2	EKF_Config_2	0.389710	2.150837
132	./data/MRCLAM_Dataset4	Robot2	EKF_Config_3	0.255545	1.056083
133	./data/MRCLAM_Dataset4	Robot2	EKF_Config_4	0.408033	2.052439
134	./data/MRCLAM_Dataset4	Robot2	PF_Config_1	1.615248	3.566938
135	./data/MRCLAM_Dataset4	Robot2	PF_Config_2	1.290038	2.696199
136	./data/MRCLAM_Dataset4	Robot2	PF_Config_3	0.779018	1.910918
137	./data/MRCLAM_Dataset4	Robot2	PF_Config_4	0.193474	0.376782
138	./data/MRCLAM_Dataset4	Robot2	PF_Config_5	0.160181	0.500697
139	./data/MRCLAM_Dataset4	Robot2	PF_Config_6	0.096199	0.306959
140	./data/MRCLAM_Dataset4	Robot3	EKF_Config_1	0.094832	0.214943
141	./data/MRCLAM_Dataset4	Robot3	EKF_Config_2	0.106556	0.179360
142	./data/MRCLAM_Dataset4	Robot3	EKF_Config_3	0.110371	0.164279
143	./data/MRCLAM_Dataset4	Robot3	EKF_Config_4	0.405707	1.036353
144	./data/MRCLAM_Dataset4	Robot3	PF_Config_1	0.215337	0.442552
145	./data/MRCLAM_Dataset4	Robot3	PF_Config_2	0.998820	1.716678
146	./data/MRCLAM_Dataset4	Robot3	PF_Config_3	0.106481	0.332402
147	./data/MRCLAM_Dataset4	Robot3	PF_Config_4	0.130771	0.392333
148	./data/MRCLAM_Dataset4	Robot3	PF_Config_5	0.174315	0.787246
149	./data/MRCLAM_Dataset4	Robot3	PF_Config_6	0.109737	0.213996
150	./data/MRCLAM_Dataset4	Robot4	EKF_Config_1	0.846335	1.781619
151	./data/MRCLAM_Dataset4	Robot4	EKF_Config_2	0.758953	2.048759
152	./data/MRCLAM_Dataset4	Robot4	EKF_Config_3	0.432417	1.196724
153	./data/MRCLAM_Dataset4	Robot4	EKF_Config_4	0.462486	2.782528
154	./data/MRCLAM_Dataset4	Robot4	PF_Config_1	2.147441	4.519269
155	./data/MRCLAM_Dataset4	Robot4	PF_Config_2	3.011044	6.166774
156	./data/MRCLAM_Dataset4	Robot4	PF_Config_3	0.151802	0.469950
157	./data/MRCLAM_Dataset4	Robot4	PF_Config_4	0.128840	0.261827
158	./data/MRCLAM_Dataset4	Robot4	PF_Config_5	0.544338	1.115304
159	./data/MRCLAM_Dataset4	Robot4	PF_Config_6	0.100656	0.325740

	computation_time	measurement_density
0	0.181671	31.845766

1	0.183641	31.845766
2	0.182109	31.845766
3	0.181681	31.845766
4	0.504333	31.845766
5	3.243894	31.845766
6	3.122105	31.845766
7	3.039611	31.845766
8	3.175705	31.845766
9	16.376620	31.845766
10	0.183339	46.673841
11	0.182333	46.673841
12	0.181483	46.673841
13	0.181334	46.673841
14	0.567718	46.673841
15	3.775423	46.673841
16	3.841459	46.673841
17	3.784988	46.673841
18	3.980019	46.673841
19	19.822130	46.673841
20	0.180438	71.867721
21	0.183270	71.867721
22	0.184029	71.867721
23	0.184618	71.867721
24	0.679276	71.867721
25	4.623589	71.867721
26	4.611233	71.867721
27	4.625869	71.867721
28	4.729079	71.867721
29	24.136126	71.867721
30	0.179831	25.964396
31	0.183408	25.964396
32	0.183253	25.964396
33	0.182616	25.964396
34	0.463460	25.964396
35	2.752431	25.964396
36	2.743137	25.964396
37	2.728240	25.964396
38	2.667508	25.964396
39	14.336040	25.964396
40	0.191305	38.089720
41	0.198215	38.089720
42	0.192459	38.089720
43	0.192697	38.089720
44	0.542028	38.089720
45	3.379306	38.089720
46	3.373251	38.089720
47	3.347999	38.089720
48	3.521992	38.089720
49	17.592122	38.089720
50	0.190987	57.343997
51	0.195212	57.343997
52	0.193796	57.343997
53	0.193391	57.343997
54	0.629165	57.343997
55	4.463396	57.343997
56	4.586025	57.343997
57	4.382213	57.343997

58	4.648090	57.343997
59	22.548807	57.343997
60	0.190797	86.217016
61	0.193003	86.217016
62	0.191745	86.217016
63	0.193559	86.217016
64	0.815436	86.217016
65	5.331750	86.217016
66	5.356739	86.217016
67	5.415821	86.217016
68	5.428111	86.217016
69	28.843465	86.217016
70	0.195943	29.359332
71	0.197220	29.359332
72	0.194453	29.359332
73	0.199646	29.359332
74	0.581091	29.359332
75	3.070745	29.359332
76	3.139898	29.359332
77	3.205196	29.359332
78	3.305148	29.359332
79	14.961803	29.359332
80	0.190174	85.199342
81	0.196155	85.199342
82	0.195571	85.199342
83	0.194363	85.199342
84	0.757364	85.199342
85	5.606836	85.199342
86	6.025625	85.199342
87	5.911160	85.199342
88	6.249794	85.199342
89	30.028907	85.199342
90	0.206858	77.575963
91	0.210034	77.575963
92	0.213538	77.575963
93	0.210530	77.575963
94	0.733445	77.575963
95	5.481492	77.575963
96	5.276574	77.575963
97	5.316530	77.575963
98	5.019636	77.575963
99	27.472380	77.575963
100	0.196631	74.832523
101	0.197948	74.832523
102	0.198745	74.832523
103	0.196852	74.832523
104	0.696785	74.832523
105	4.734859	74.832523
106	4.789931	74.832523
107	4.769971	74.832523
108	4.774875	74.832523
109	24.112040	74.832523
110	0.217788	106.316661
111	0.208462	106.316661
112	0.209969	106.316661
113	0.209422	106.316661
114	0.690879	106.316661

115	4.691147	106.316661
116	4.706104	106.316661
117	4.771819	106.316661
118	4.696930	106.316661
119	24.108201	106.316661
120	0.189488	44.978274
121	0.203086	44.978274
122	0.189848	44.978274
123	0.196439	44.978274
124	0.551335	44.978274
125	3.545456	44.978274
126	3.569172	44.978274
127	3.528491	44.978274
128	3.464267	44.978274
129	18.539824	44.978274
130	0.180611	40.115626
131	0.183208	40.115626
132	0.183588	40.115626
133	0.184998	40.115626
134	0.540165	40.115626
135	3.465827	40.115626
136	3.597343	40.115626
137	3.527378	40.115626
138	3.457432	40.115626
139	18.286040	40.115626
140	0.183270	76.866356
141	0.184355	76.866356
142	0.200498	76.866356
143	0.186265	76.866356
144	0.726609	76.866356
145	5.440377	76.866356
146	5.270970	76.866356
147	5.346415	76.866356
148	5.477219	76.866356
149	27.712073	76.866356
150	0.182396	72.254065
151	0.184892	72.254065
152	0.185906	72.254065
153	0.186136	72.254065
154	0.711827	72.254065
155	5.042334	72.254065
156	5.195897	72.254065
157	5.060289	72.254065
158	5.090719	72.254065
159	26.793130	72.254065

Represent metrics

For instance you could use the catplot of the seaborn library. Feel free to use another informative plot.

```
In [342]: # Plot measurement density for each robot and dataset

unique_density = errors[['dataset', 'robot', 'measurement_density']].drop

plt.figure(figsize=(12, 8))
```

```

sns.barplot(x='robot', y='measurement_density', hue='dataset', data=unique

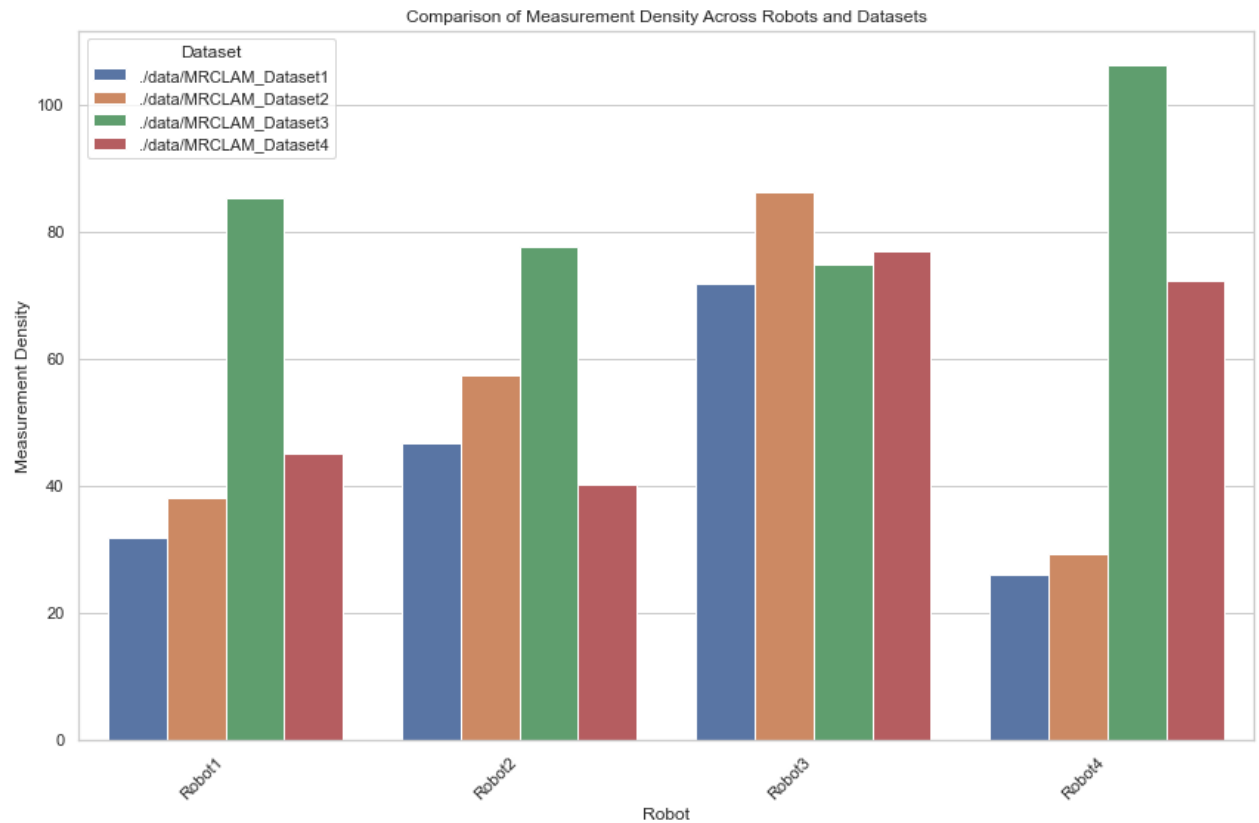
plt.xlabel('Robot')
plt.ylabel('Measurement Density')
plt.title('Comparison of Measurement Density Across Robots and Datasets')

plt.xticks(rotation=45, ha="right")

plt.legend(title='Dataset')

plt.tight_layout()
plt.show()

```



```

In [343... fig, axes = plt.subplots(2, 1, figsize=(10, 12))

# Max Error Plot
sns.barplot(
    data=errors,
    x="model_name", y="max_error", hue="robot",
    ax=axes[0]
)
axes[0].set_title("Max Error by Model Configuration and Robot")
axes[0].set_xlabel("Model Configuration")
axes[0].set_ylabel("Max Error")
axes[0].legend(title="Robot")
axes[0].set_xticklabels(axes[0].get_xticklabels(), rotation=45)

# Mean Error Plot
sns.barplot(
    data=errors,
    x="model_name", y="mean_error", hue="robot",
    ax=axes[1]
)

```

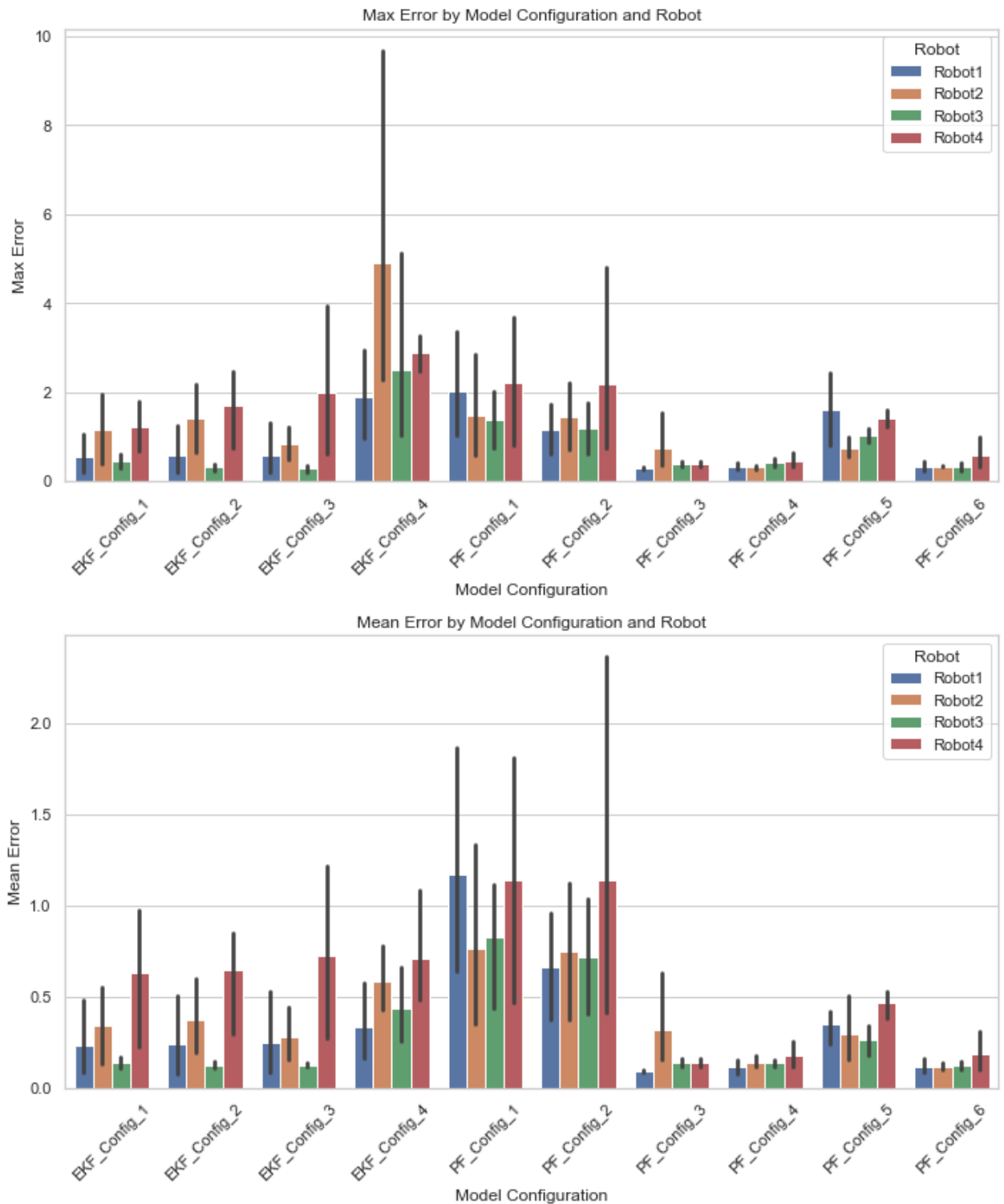
```

axes[1].set_title("Mean Error by Model Configuration and Robot")
axes[1].set_xlabel("Model Configuration")
axes[1].set_ylabel("Mean Error")
axes[1].legend(title="Robot")
axes[1].set_xticklabels(axes[1].get_xticklabels(), rotation=45)

plt.tight_layout()

plt.show()

```



```

In [344...] # Represent metrics for all experiments
catplot = sns.catplot(
    data=errors, kind="bar",
    x="model_name", y="max_error", hue="robot",

```

```

col="dataset", height=5, aspect=1.2
)

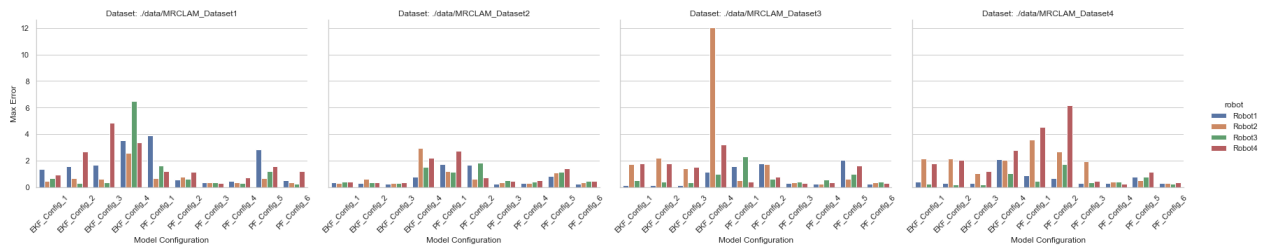
catplot.set_axis_labels("Model Configuration", "Max Error")
catplot.set_titles("Dataset: {col_name}")

for ax in catplot.axes.flat:
    ax.set_xticklabels(ax.get_xticklabels(), rotation=45)

catplot.tight_layout()

plt.show()

```



In [345... *#For a better overview I plot the data into two separate figures*

```

ekf_errors = errors[errors['model_name'].str.contains('EKF')]
pf_errors = errors[errors['model_name'].str.contains('PF')]

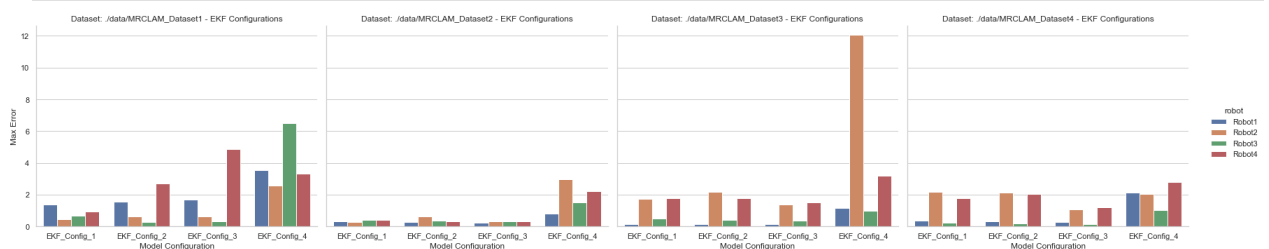
# Plot for EKF configurations
catplot_ekf = sns.catplot(
    data=ekf_errors, kind="bar",
    x="model_name", y="max_error", hue="robot",
    col="dataset", height=5, aspect=1.2
)
catplot_ekf.set_axis_labels("Model Configuration", "Max Error")
catplot_ekf.set_titles("Dataset: {col_name} - EKF Configurations")
catplot_ekf.tight_layout()

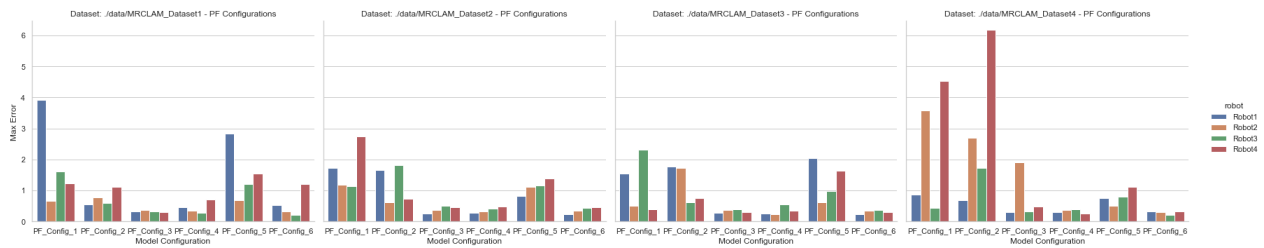
plt.show()

# Plot for PF configurations
catplot_pf = sns.catplot(
    data=pf_errors, kind="bar",
    x="model_name", y="max_error", hue="robot",
    col="dataset", height=5, aspect=1.2
)
catplot_pf.set_axis_labels("Model Configuration", "Max Error ")
catplot_pf.set_titles("Dataset: {col_name} - PF Configurations")
catplot_pf.tight_layout()

plt.show()

```





```
In [346... ekf_models = [name for name in errors['model_name'].unique() if "EKF" in name]
pf_models = [name for name in errors['model_name'].unique() if "PF" in name]

# Define two different color palettes
ekf_palette = sns.color_palette("Blues", len(ekf_models)) # Blue shades
pf_palette = sns.color_palette("Oranges", len(pf_models)) # Orange shades

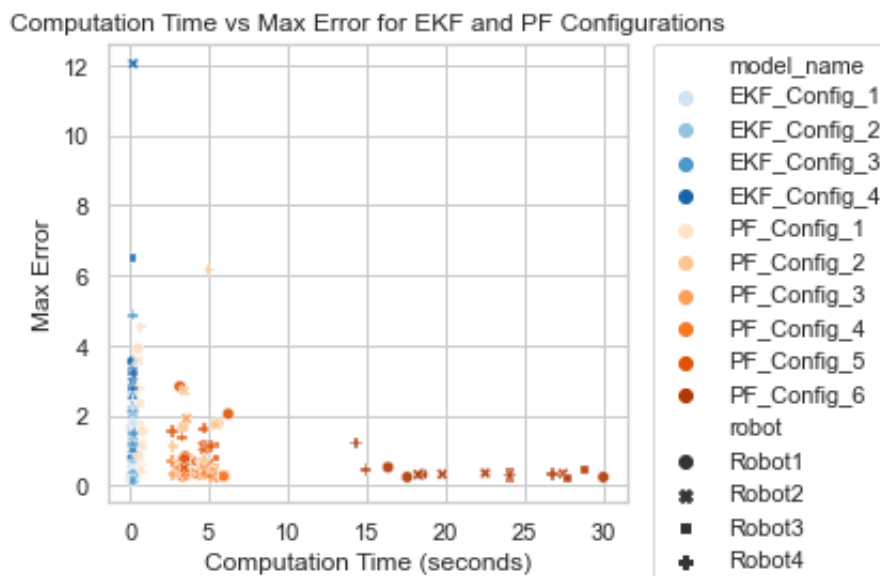
palette = dict(zip(ekf_models + pf_models, ekf_palette + pf_palette))

sns.scatterplot(
    data=errors,
    x='computation_time', y='max_error',
    hue='model_name', style='robot',
    palette=palette,
    s=30, alpha=0.9
)

plt.title("Computation Time vs Max Error for EKF and PF Configurations")
plt.xlabel("Computation Time (seconds)")
plt.ylabel("Max Error")

plt.legend(bbox_to_anchor=(1.05, 1), loc='best', borderaxespad=0.)

plt.tight_layout()
plt.show()
```



Analyze metrics

T3.1 Analyze the representations, and extract some conclusions regarding the

performance of each of the model configurations used. Which would be the best suited method for offline execution? Why?

(Answer here max 300 words)

EKF

- Low/Moderate process and measurement noise (EKF_Config_1) offers the most balanced and accurate performance. It consistently shows lower max error and mean error, while maintaining low computation times
- High process noise (EKF_Config_4) leads to significant errors --> make the filter less certain about the robot's own motion --> inaccurate predictions
- very high measurement noise (EKF_Config_2 with [1000, 1000, 1e20]) makes the filter more skeptical of the sensor measurements --> greater reliance on the motion model --> However, in cases where the motion model is inaccurate, this can lead to larger errors (can be seen in error spikes for certain datasets)
- low state noise with moderate measurement noise (EKF_Config_3) also performs well but slightly less consistently than EKF_Config_1

PF

Number of Particles:

- PF_Config_1, with only 1 particle, performs poorly
- Higher particle counts (PF_Config_6) lead to more accurate results (but have a higher computation time)

Motion Noise:

- PF_Config_5 with high motion noise, which leads to significant errors --> makes filter more uncertain about the robot's movement
- PF_Config_3 and PF_Config_6, with moderate noise are better --> filter can more accurately model the robot's true state without too much uncertainty

Measurement Noise:

- Lower measurement noise (PF_Config_3 and PF_Config_6) generally results in better performance, as the filter can correct for errors more effectively.
- High motion noise (PF_Config_5) degrades performance significantly

Best-suited Method:

For offline execution, Particle Filters (PF), especially configurations with higher particle counts, are generally better suited. While they require more computation time, they tend to provide higher accuracy in terms of both max error and mean error. Configurations like PF_Config_4 or PF_Config_6 balance accuracy and consistency across datasets, making them ideal for offline tasks where precision is crucial, and time is less .

T3.2 Which would be the best suited method for online execution? Try to add the computation time as an extra metric for the assessment.

(Answer here max 150 words)

Best-suited Method:

Extended Kalman Filters (EKF) are the best-suited method for online execution, because of the significantly lower computation time. Sometimes PF configurations can achieve better accuracy, but as you can see in the plot "Computation Time vs max error for EKF and PF Configurations" the EKF models are much faster and with the right configuration can still have a good accuracy. For example, EKF models like EKF_Config_1 and EKF_Config_3.




T3.3 BONUS. How would you modify the PF filter to allow filter initialization without prior state belief?

(Answer here max 150 words)

Without prior state belief, I would initialize the particles uniformly

- Therefore you need to define boundaries (e.g., possible ranges of positions)
- sample particles uniformly from the previously defined range

Delivery Instructions:

1.  Please download your work in both Notebook and Markdown formats. Simply navigate to:
 - **File > Download as > Notebook**
 - **File > Download as > Markdown**
2.  Once you have the necessary files and any associated figures, kindly compress them into a single .zip file. When naming your file, please use the format: **E1_FirstName_LastName.zip**.
3.  Finally, make sure to upload your .zip file to Aula Digital by the set deadline: **15/10/24**.

Best of luck with your submission, and reach out if you have any questions!