

complex_system_results

December 20, 2025

```
[ ]: from pathlib import Path
from scipy.io import loadmat
import sys
import os

dataset_path = Path('data') / 'data.mat'
if not dataset_path.exists():
    alt = Path.cwd().parent / 'data' / 'data.mat'
    if alt.exists():
        dataset_path = alt
    else:
        raise FileNotFoundError(f"data.mat not found under {Path.cwd()} or its parent")
notebook_path = os.getcwd()
print (f"Current notebook path: {notebook_path}")
project_root = os.path.dirname(notebook_path)
if project_root not in sys.path:
    sys.path.insert(0, project_root)
print (f"Added {project_root} to sys.path")

mat_data = loadmat(dataset_path)
print(mat_data.keys())
```

```
[ ]: import torch
import matplotlib.pyplot as plt
from utils import trainer
from utils import utils
from Systems import DynamicSystem
import Filters
import torch.nn.functional as F
from torch.utils.data import TensorDataset, DataLoader
import numpy as np
from scipy.io import loadmat
from scipy.interpolate import RegularGridInterpolator
import random
```

```

torch.manual_seed(42)
np.random.seed(42)
random.seed(42)
if torch.cuda.is_available():
    torch.cuda.manual_seed_all(42)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
DEVICE = device # For backward compatibility
print(f"device: {device}")

```

```

[ ]: import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import TensorDataset, DataLoader
import numpy as np
import matplotlib.pyplot as plt
import os
import Systems

# Parametry sekvencí
TRAIN_SEQ_LEN = 50          # Délka sekvence pro trénink (např. 100 kroků = 100 sekund při 1Hz)
VAL_SEQ_LEN = 100
TEST_SEQ_LEN = 500           # Délka sekvence pro testování (delší sekvence pro stabilnější vyhodnocení)
STRIDE = 1                   # Posun okna (překryv) pro data augmentation
BATCH_SIZE = 256
DATA_PATH = 'data/processed'
print(f"Běží na zařízení: {device}")

```

```

[4]: import torch
from torch.utils.data import TensorDataset, DataLoader
import os

def prepare_sequences(dataset_list, seq_len, stride, mode='train'):
    """
    Zpracuje list trajektorií na sekvence pro trénink dle článku.

    Nový formát dle [Song et al., 2024]:
    - Vstup u (4D): [v_left, v_right, theta_imu, omega_imu]
    - Cíl x (6D): [px, py, vx, vy, theta, omega]
    """
    X_seq_list = [] # Ground Truth (Cíl)
    Y_seq_list = [] # GPS Měření (Vstup do korekce)
    U_seq_list = [] # Control Input (IMU/Odo)

```

```

print(f"Zpracovávám {len(dataset_list)} trajektorií pro {mode}...")

for traj in dataset_list:
    # 1. Extrahuje data
    # GT z preprocessingu je [px, py, theta]
    gt = traj['ground_truth'].float()

    # GPS: [x, y] (obsahuje NaN!)
    gps = traj['filtered_gps'].float()

    # IMU: [ax, ay, theta, omega]
    imu = traj['imu'].float()
    theta_imu = imu[:, 2] # Orientace z IMU
    omega_imu = imu[:, 3] # Úhlová rychlosť z IMU

    # ODO: [v_left, v_right]
    odo = traj['filtered_wheel'].float()

    # Fix NaN v odometrii (nahradíme nulou)
    v_left = torch.nan_to_num(odo[:, 0], nan=0.0)
    v_right = torch.nan_to_num(odo[:, 1], nan=0.0)

    # 2. Sestavení vstupu  $u = [v_l, v_r, \theta_{imu}, \omega_{imu}]$  (4D)
    # Toto odpovídá "State Model" definovanému v článku (sekce II.C.2)
    u = torch.stack((v_left, v_right, theta_imu, omega_imu), dim=1)

    # 3. Sestavení cíle  $x$  (6D) pro state vector [px, py, vx, vy, theta, omega]
    # Vyplníme to, co máme z Ground Truth (px, py, theta).
    # Rychlosti (vx, vy, omega) v GT implicitně nemáme (nebo je složité je derivovať/presne),
    # ale pro trénink Loss funkce budeme stejně porovnávat primárně pozici.
    T = gt.shape[0]
    x_target = torch.zeros(T, 6)
    x_target[:, 0] = gt[:, 0] # px
    x_target[:, 1] = gt[:, 1] # py
    x_target[:, 4] = gt[:, 2] # theta
    # Ostatní (vx, vy, omega) zůstávají 0, protože v Loss funkci budeme maskovat nebo brát jen pozici.

    # 4. Sliding Window (Rozsekání na sekvence)
    num_samples = gt.shape[0]
    current_stride = stride if mode == 'train' else seq_len # U testu bez překryvu
    for i in range(0, num_samples - seq_len + 1, current_stride):

```

```

# Cíl: 6D stav
x_seq = x_target[i : i+seq_len, :]

# Měření: GPS [px, py]
y_seq = gps[i : i+seq_len, :]

# Vstup: 4D control input
u_seq = u[i : i+seq_len, :]

X_seq_list.append(x_seq)
Y_seq_list.append(y_seq)
U_seq_list.append(u_seq)

# Stack do tenzorů
X_out = torch.stack(X_seq_list)
Y_out = torch.stack(Y_seq_list)
U_out = torch.stack(U_seq_list)

return X_out, Y_out, U_out

# === NAČTENÍ DAT ===
# Ujistíme se, že cesty a konstanty jsou definované (pokud nejsou, doplňte je u
# nahoře)
if 'DATA_PATH' not in locals(): DATA_PATH = 'data/processed'
if 'TRAIN_SEQ_LEN' not in locals(): TRAIN_SEQ_LEN = 100
if 'VAL_SEQ_LEN' not in locals(): VAL_SEQ_LEN = 200
if 'TEST_SEQ_LEN' not in locals(): TEST_SEQ_LEN = 500
if 'STRIDE' not in locals(): STRIDE = 20
if 'BATCH_SIZE' not in locals(): BATCH_SIZE = 256

train_data_raw = torch.load(os.path.join(DATA_PATH, 'train.pt'))
val_data_raw = torch.load(os.path.join(DATA_PATH, 'val.pt'))
test_data_raw = torch.load(os.path.join(DATA_PATH, 'test.pt'))

# === PRÍPRAVA SEKVENCÍ ===
print("--- Generuje trénovací data (Paper compatible) ---")
train_X, train_Y, train_U = prepare_sequences(train_data_raw, TRAIN_SEQ_LEN, u
# nahoře, 'train')

print("\n--- Generuje validační data ---")
val_X, val_Y, val_U = prepare_sequences(val_data_raw, VAL_SEQ_LEN, VAL_SEQ_LEN, u
# nahoře, 'val')

print("\n--- Generuje testovací data ---")
test_X, test_Y, test_U = prepare_sequences(test_data_raw, TEST_SEQ_LEN, TEST_SEQ_LEN, u
# nahoře, 'test')

```

```

# Vytvoření DataLoaderů
train_loader = DataLoader(TensorDataset(train_X, train_Y, train_U), □
    ↪batch_size=BATCH_SIZE, shuffle=True)
val_loader = DataLoader(TensorDataset(val_X, val_Y, val_U), □
    ↪batch_size=BATCH_SIZE, shuffle=False)
test_loader = DataLoader(TensorDataset(test_X, test_Y, test_U), □
    ↪batch_size=BATCH_SIZE, shuffle=False)

print(f"\n Data připravena.")
print(f"Train batches: {len(train_loader)}")
print(f"Shapes -> X: {train_X.shape} (6D State), U: {train_U.shape} (4D Input), □
    ↪Y: {train_Y.shape} (2D Meas)")

```

--- Generuje validační data ---
Zpracovávám 2 trajektorií pro val...

--- Generuje testovací data ---
Zpracovávám 3 trajektorií pro test...

 Data připravena.
Train batches: 430
Shapes -> X: torch.Size([109843, 50, 6]) (6D State), U: torch.Size([109843, 50, 4]) (4D Input), Y: torch.Size([109843, 50, 2]) (2D Meas)

[5]: # === INICIALIZACE DYNAMICKÉHO MODELU (System Instance - Paper Version) ===

```

# 1. Parametry systému podle článku [Song et al., 2024]
# State (6D): [px, py, vx, vy, theta, omega]
# Referenční rovnice (5) v článku.
state_dim = 6
# Meas (2D): [gps_x, gps_y]
# Referenční rovnice (6) v článku.
obs_dim = 2
# Časový krok (z preprocessingu)
dt = 1.0

# 2. Definice Matice Q (Procesní šum / Model Uncertainty)
# Nyní máme 6 stavů. Musíme definovat nejistotu pro každý z nich.
# Hodnoty jsou nastaveny heuristicky (lze ladit):
# - Pozice (idx 0,1): 0.1
# - Rychlosť (idx 2,3): 0.1
# - Úhel/Omega (idx 4,5): 0.01 (IMU je v NCLT docela přesné, ale driftuje)
q_diag = torch.tensor([0.1, 0.1, 0.1, 0.1, 0.01, 0.01])
Q = torch.diag(q_diag)

# 3. Definice Matice R (Šum měření / Sensor Noise)

```

```

# GPS měří jen pozici (px, py).
# Nastavujeme 1.0 m^2. To odpovídá standardní odchylce 1m.
# Pokud je GPS v datasetu horší, KalmanNet se naučí "nedůvěřovat" vstupu y
# a spoléhat více na predikci z u (odometrie).
r_diag = torch.tensor([1.0, 1.0])
R = torch.diag(r_diag)

# 4. Počáteční podmínky (Prior)
# Ex0: Nulový vektor 6x1
Ex0 = torch.zeros(state_dim, 1)

# P0: Počáteční kovariance
# Autoři používají P k inicializaci EKF[cite: 700].
# Nastavíme rozumnou počáteční nejistotu.
P0 = torch.eye(state_dim) * 0.5

# 5. Vytvoření instance DynamicSystemNCLT
# Důležité: f=None zajistí, že se použije interní `f_paper_dynamics` (rovnice ↵5),
# která očekává 4D vstup (v_l, v_r, theta, omega).
sys_model = Systems.DynamicSystemNCLT(
    state_dim=state_dim,
    obs_dim=obs_dim,
    Q=Q,
    R=R,
    Ex0=Ex0,
    P0=P0,
    dt=dt,
    f=None, # None -> Použije se model z článku: px += vc*cos(theta_imu)...
    h=None, # None -> Použije se GPS model: y = [px, py]
    device=DEVICE
)

print(f" System Model NCLT inicializován (Paper Version).")
print(f" - State Dim: {sys_model.state_dim} [px, py, vx, vy, theta, omega]")
print(f" - Meas Dim: {sys_model.obs_dim} [gps_x, gps_y]")
print(f" - Input Dim: 4 [v_l, v_r, theta_imu, omega_imu]") # Implicitně v modelu
print(f" - Q Diag: {q_diag.tolist()}")

```

```

System Model NCLT inicializován (Paper Version).
- State Dim: 6 [px, py, vx, vy, theta, omega]
- Meas Dim: 2 [gps_x, gps_y]
- Input Dim: 4 [v_l, v_r, theta_imu, omega_imu]
- Q Diag: [0.10000000149011612, 0.10000000149011612, 0.10000000149011612,
0.10000000149011612, 0.009999999776482582, 0.009999999776482582]

```

```
[7]: import torch
import torch.optim as optim
import os
from state_NN_models import StateKalmanNet
from utils import trainer

# === 1. KONFIGURACE A INICIALIZACE MODELU ===

# Hyperparametry sítě
# State dim je 3. Multiplier 40 znamená hidden state velikosti 120.
# To je pro navigaci s nelinearitami (sin/cos) rozumná kapacita.
print("Inicializuji StateKalmanNet...")
state_knet = StateKalmanNet(
    system_model=sys_model,
    device=DEVICE,
    hidden_size_multiplier=12,           # Větší kapacita pro složitější dynamiku
    output_layer_multiplier=4,
    num_gru_layers=1,                   # 1 vrstva GRU obvykle stačí a je
    ↵stabilnější
    gru_hidden_dim_multiplier=10
).to(DEVICE)
print(state_knet)

# Počet trénovatelných parametrů
params_count = sum(p.numel() for p in state_knet.parameters() if p.
    ↵requires_grad)
print(f"Model má {params_count} trénovatelných parametrů.")

# === 2. NASTAVENÍ TRÉNINKU (TBPTT) ===

# Parametry pro Sliding Window trénink (TBPTT)
# NCLT sekvence jsou dlouhé (100 kroků), gradienty by mohly explodovat.
# Dělíme je na okna délky 20 a gradienty ořezáváme.
TBPTT_WINDOW = 20 # Délka okna (w)
TBPTT_STEP = 10   # Krok pro detach (k) - obvykle polovina w

EPOCHS = 100
LEARNING_RATE = 1e-3
WEIGHT_DECAY = 1e-5 # Jemná regularizace
CLIP_GRAD = 1.0    # Důležité: Ořezání gradientů pro stabilitu RNN

# === 3. SPUŠTĚNÍ TRÉNINKU ===
print("\n Spouštím tréninkovou smyčku...")

trained_knet = trainer.train_state_KalmanNet_sliding_windowNCLT(
    model=state_knet,
    train_loader=train_loader,
```

```

    val_loader=val_loader,
    device=DEVICE,
    epochs=EPOCHS,
    lr=LEARNING_RATE,
    weight_decay_=WEIGHT_DECAY,
    clip_grad=CLIP_GRAD,
    early_stopping_patience=20, # Zastaví, pokud se 20 epoch nezlepší loss
    tbptt_k=TBPTT_STEP,
    tbptt_w=TBPTT_WINDOW
)

# === 4. ULOŽENÍ MODELU ===
save_path = 'best_kalmannet_nclt_sensor_fusion.pth'
torch.save(trained_knet.state_dict(), save_path)
print(f"\n Trénink dokončen. Nejlepší model uložen do: {save_path}")

```

Inicializuje StateKalmanNet...

DEBUG: Layer 'output_final_linear.0' initialized near zero (Start K=0).

StateKalmanNet(
 (dnn): DNN_KalmanNet(
 (input_layer): Sequential(
 (0): Linear(in_features=16, out_features=768, bias=True)
 (1): ReLU()
)
 (gru): GRU(768, 400)
 (output_hidden_layer): Sequential(
 (0): Linear(in_features=400, out_features=48, bias=True)
 (1): ReLU()
)
 (output_final_linear): Sequential(
 (0): Linear(in_features=48, out_features=12, bias=True)
)
)
)

Model má 1436892 trénovatelných parametrů.

Spouštím tréninkovou smyčku...

INFO: Starting training with TBPTT(k=10, w=20)

INFO: Returns covariance: False

Epoch [1/100] | Train Loss: 2.5089 | Val Loss: 5.7570
 -> New best model saved! (Val Loss: 5.7570)

Epoch [2/100] | Train Loss: 2.1682 | Val Loss: 4.7777
 -> New best model saved! (Val Loss: 4.7777)

Epoch [3/100] | Train Loss: 2.1006 | Val Loss: 4.4130
 -> New best model saved! (Val Loss: 4.4130)

Epoch [4/100] | Train Loss: 2.0131 | Val Loss: 4.1254
 -> New best model saved! (Val Loss: 4.1254)

```
Epoch [5/100] | Train Loss: 1.9965 | Val Loss: 4.6585
Epoch [6/100] | Train Loss: 1.9455 | Val Loss: 5.5450
Epoch [7/100] | Train Loss: 1.9436 | Val Loss: 4.4762
Epoch [8/100] | Train Loss: 1.9200 | Val Loss: 4.8222
Epoch [9/100] | Train Loss: 1.8721 | Val Loss: 4.8345
Epoch [10/100] | Train Loss: 1.8435 | Val Loss: 3.7339
    -> New best model saved! (Val Loss: 3.7339)
Epoch [11/100] | Train Loss: 1.8118 | Val Loss: 4.1930
Epoch [12/100] | Train Loss: 1.7907 | Val Loss: 3.7686
Epoch [13/100] | Train Loss: 1.7689 | Val Loss: 3.9523
Epoch [14/100] | Train Loss: 1.7379 | Val Loss: 4.9595
Epoch [15/100] | Train Loss: 1.7445 | Val Loss: 4.4925
Epoch [16/100] | Train Loss: 1.7040 | Val Loss: 3.6586
    -> New best model saved! (Val Loss: 3.6586)
Epoch [17/100] | Train Loss: 1.6944 | Val Loss: 3.5547
    -> New best model saved! (Val Loss: 3.5547)
Epoch [18/100] | Train Loss: 1.6569 | Val Loss: 3.9977
Epoch [19/100] | Train Loss: 1.6516 | Val Loss: 3.7136
Epoch [20/100] | Train Loss: 1.6376 | Val Loss: 3.5399
    -> New best model saved! (Val Loss: 3.5399)
Epoch [21/100] | Train Loss: 1.6415 | Val Loss: 3.9294
Epoch [22/100] | Train Loss: 1.6139 | Val Loss: 3.8442
Epoch [23/100] | Train Loss: 1.5917 | Val Loss: 3.9282
Epoch [24/100] | Train Loss: 1.5885 | Val Loss: 3.7033
Epoch [25/100] | Train Loss: 1.5850 | Val Loss: 3.8076
Epoch [26/100] | Train Loss: 1.5777 | Val Loss: 4.0681
Epoch [27/100] | Train Loss: 1.5706 | Val Loss: 3.7046
Epoch [28/100] | Train Loss: 1.5592 | Val Loss: 3.8230
Epoch [29/100] | Train Loss: 1.5538 | Val Loss: 4.1050
Epoch [30/100] | Train Loss: 1.5316 | Val Loss: 3.8408
Epoch [31/100] | Train Loss: 1.5313 | Val Loss: 4.0893
Epoch [32/100] | Train Loss: 1.5236 | Val Loss: 3.8977
Epoch [33/100] | Train Loss: 1.5127 | Val Loss: 3.8361
Epoch [34/100] | Train Loss: 1.5091 | Val Loss: 3.9346
Epoch [35/100] | Train Loss: 1.4976 | Val Loss: 4.1879
Epoch [36/100] | Train Loss: 1.4941 | Val Loss: 4.5886
Epoch [37/100] | Train Loss: 1.4709 | Val Loss: 3.7761
Epoch [38/100] | Train Loss: 1.4872 | Val Loss: 4.2372
Epoch [39/100] | Train Loss: 1.4667 | Val Loss: 3.9888
Epoch [40/100] | Train Loss: 1.4665 | Val Loss: 3.7201
```

Early stopping triggered after 40 epochs.

Training completed.

Loading best model with validation loss: 3.539865

Trénink dokončen. Nejlepší model uložen do:
best_kalmannet_nclt_sensor_fusion.pth

```
[8]: import torch
import torch.nn.functional as F
import numpy as np
import Filters # Tvůj modul pro klasické filtry
from utils import utils

# =====
# 0. KONFIGURACE A PŘÍPRAVA MODELŮ
# =====
# 1. KalmanNet (tvůj natrénovaný model)
try:
    trained_model_classic = state_knet
    trained_model_classic.eval()
    print("INFO: KalmanNet (state_knet) připraven k testování.")
except NameError:
    raise NameError("Chyba: Proměnná 'state_knet' neexistuje. Spusťte nejprve ↵ trénink.")

# 2. Klasické filtry (EKF, UKF)
# Poznámka: Aby EKF/UKF na NCLT fungovaly dobře, musí jejich implementace ↵ podporovat
# vstup 'u' (rychlosť/omega) v predikčním kroku  $f(x, u)$ .
# Pokud tvá třída Filters.py nepodporuje 'u', budou tyto filtry fungovat jen ↵ jako 'GPS smoother'.
print("Inicializuji EKF a UKF...")
ekf_filter = Filters.ExtendedKalmanFilter(sys_model)
ukf_filter = Filters.UnscentedKalmanFilter(sys_model)

# =====
# 1. VYHODNOCOVACÍ SMYČKA
# =====
mse_knet = []
mse_ekf, anees_ekf = [], []
mse_ukf, anees_ukf = [], []

traj_idx = 0
total_trajectories = len(test_loader.dataset)

print(f"\nVyhodnocuji {total_trajectories} sekvencí z testovací sady...")
print("POZNÁMKA: Startujeme z Ground Truth pozice.")

with torch.no_grad():
    # ZMĚNA: Unpacking 3 hodnot (x, y, u)
    for x_true_batch, y_meas_batch, u_input_batch in test_loader:

        batch_size = x_true_batch.shape[0]
```

```

for i in range(batch_size):
    traj_idx += 1

    # Příprava dat pro jednu trajektorii
    y_seq = y_meas_batch[i].to(DEVICE)      # [Seq_Len, 2]
    u_seq = u_input_batch[i].to(DEVICE)      # [Seq_Len, 2]
    x_true = x_true_batch[i].to(DEVICE)      # [Seq_Len, 3]

    seq_len = y_seq.shape[0]

    # --- PRÍPRAVA POČÁTEČNÍHO STAVU (GROUND TRUTH) ---
    # KalmanNet: [1, State_Dim]
    knet_init_state = x_true[0, :].unsqueeze(0)

    # Filtry: [State_Dim, 1]
    filter_init_state = x_true[0, :].unsqueeze(1)

    # --- A. KalmanNet (Neural Network) ---
    trained_model_classic.reset(batch_size=1,
↳initial_state=knet_init_state)

    knet_preds = [knet_init_state] # Startujeme z GT

    for t in range(1, seq_len):
        # ZMĚNA: Posíláme  $y_t$  i  $u_t$  (řízení)
        y_t = y_seq[t, :].unsqueeze(0)
        u_t = u_seq[t, :].unsqueeze(0)

        # KNet Step
        x_hat_t = trained_model_classic.step(y_t, u_t)
        knet_preds.append(x_hat_t)

    full_x_hat_knet = torch.cat(knet_preds, dim=0)

    # --- B. Extended Kalman Filter (EKF) ---
    # Pokusíme se použít standardní process_sequence.
    # Pokud Filters.py neumí 'u', tento odhad bude horší než KNet.
    # try:
    #     # Předpokládáme, že process_sequence umí bud' 'u' jako argument, nebo ho ignoruje.
    #     # Pokud ho ignoruje, EKF pojede jen na GPS modelu (ConstantPosition).
    #     # ekf_res = ekf_filter.process_sequence(y_seq,
    ↳Ex0=filter_init_state, P0=sys_model.P0)
    #     # full_x_hat_ekf = ekf_res['x_filtered']
    #     # full_P_hat_ekf = ekf_res['P_filtered']
    #     # except Exception as e:

```

```

#      # Fallback, pokud EKF selže (např. kvůli dimenzím)
#      full_x_hat_ekf = torch.zeros_like(x_true)
#      full_P_hat_ekf = torch.eye(3).unsqueeze(0).repeat(seq_len, 1, 1)
#      ↵1).to(DEVICE)

# --- C. Unscented Kalman Filter (UKF) ---
try:
    ukf_res = ukf_filter.process_sequence(y_seq, u_seq, ↵
                                          Ex0=filter_init_state, P0=sys_model.P0)
    full_x_hat_ukf = ukf_res['x_filtered']
    full_P_hat_ukf = ukf_res['P_filtered']
except Exception as e:
    full_x_hat_ukf = torch.zeros_like(x_true)
    full_P_hat_ukf = torch.eye(3).unsqueeze(0).repeat(seq_len, 1, 1)
#      ↵1).to(DEVICE)

# --- VÝPOČET METRIK ---
# MSE pro pozici (první 2 stavy: x, y)
# Ignorujeme theta, protože MSE na úhlech je ošemetoné (periodické)
mse_val_knet = F.mse_loss(full_x_hat_knet[1:, :2], x_true[1:, :2]).item()
#      mse_val_ekf = F.mse_loss(full_x_hat_ekf[1:, :2], x_true[1:, :2]).item()
mse_val_ukf = F.mse_loss(full_x_hat_ukf[1:, :2], x_true[1:, :2]).item()

mse_knet.append(mse_val_knet)
# mse_ekf.append(mse_val_ekf)
mse_ukf.append(mse_val_ukf)

# ANEES (Pokud filtry vrátily P)
# Funkce ANEES by měla zvládnout celou trajektorii najednou
# (Pozor na dimenze: [1, T, Dim])
def safe_anees(x_t, x_h, P_h):
    if torch.all(x_h == 0): return 0.0 # Skip failed filters
    return utils.calculate_anees_vectorized(
        x_t.unsqueeze(0).cpu(),
        x_h.unsqueeze(0).cpu(),
        P_h.unsqueeze(0).cpu()
    )

# anees_ekf.append(safe_anees(x_true, full_x_hat_ekf, ↵
#                               full_P_hat_ekf))
anees_ukf.append(safe_anees(x_true, full_x_hat_ukf, full_P_hat_ukf))

if traj_idx % 50 == 0:

```

```

        print(f"Zpracováno {traj_idx}/{total_trajectories} trajektorií..
        ↵.")

# =====
# 2. VÝPIS VÝSLEDKŮ
# =====

def avg(lst): return np.mean([l for l in lst if l is not None])

print("\n" + "="*80)
print(f"FINÁLNÍ VÝSLEDKY NA NCLT DATASETU (Test Set)")
print(f"Metrika: MSE Pozice [m^2] (Nižší je lepší)")
print("="*80)

# KalmanNet
print(f"{'KalmanNet (Trained)':<30} | MSE: {avg(mse_knet):.4f} | ANEES: N/A")

# EKF
# print(f"{'EKF (Standard)':<30} | MSE: {avg(mse_ekf):.4f} | ANEES:_
# ↵{avg(anees_ekf):.4f}")

# UKF
print(f"{'UKF (Standard)':<30} | MSE: {avg(mse_ukf):.4f} | ANEES:_
# ↵{avg(anees_ukf):.4f}")

# print("*80)
# if avg(mse_knet) < avg(mse_ekf):
#     print(" KalmanNet překonal EKF!")
# else:
#     print(" KalmanNet zatím nepřekonal EKF. Zkuste více epoch nebo ladit Q/R.
# ↵")

```

INFO: KalmanNet (state_knet) připraven k testování.

Inicializuji EKF a UKF...

Vyhodnocuji 26 sekvencí z testovací sady...

POZNÁMKA: Startujeme z Ground Truth pozice.

=====

=====

=====

=====

=====

FINÁLNÍ VÝSLEDKY NA NCLT DATASETU (Test Set)

Metrika: MSE Pozice [m^2] (Nižší je lepší)

=====

KalmanNet (Trained)	MSE: 39.0307 ANEES: N/A
UKF (Standard)	MSE: 102782.1331 ANEES: 0.0000

[9]: import matplotlib.pyplot as plt

=== VIZUALIZACE POSLEDNÍ TESTOVACÍ TRAJEKTORIE ===

```

# 1. Převod na NumPy (pro matplotlib)
# Používáme data z poslední iterace smyčky (poslední trajektorie v test setu)
gt_np = x_true.cpu().numpy()
knet_np = full_x_hat_knet.cpu().numpy()
# ekf_np = full_x_hat_ekf.cpu().numpy()
ukf_np = full_x_hat_ukf.cpu().numpy()

# Názvy indexů pro lepší čitelnost
PX, PY = 0, 1

# Vytvoření subplotů (3 řádky, 1 sloupec)
fig, axs = plt.subplots(3, 1, figsize=(10, 18), constrained_layout=True)
fig.suptitle(f"Vizualizace odhadu polohy (Trajektorie č. {traj_idx})", fontsize=16)

# --- 1. Graf: KalmanNet ---
axs[0].plot(gt_np[:, PX], gt_np[:, PY], 'k-', linewidth=2, label='Ground Truth', alpha=0.6)
axs[0].plot(knet_np[:, PX], knet_np[:, PY], 'b-.', linewidth=2, label='KalmanNet (Trained)')
axs[0].set_title(f"KalmanNet (MSE: {F.mse_loss(full_x_hat_knet[:, :2], x_true[:, :2]):.4f})", fontsize=14)
axs[0].set_ylabel("Pozice Y [m]")
axs[0].legend()
axs[0].grid(True)
axs[0].axis('equal') # Aby mapa nebyla deformovaná

# --- 2. Graf: EKF ---
# axs[1].plot(gt_np[:, PX], gt_np[:, PY], 'k-', linewidth=2, label='Ground Truth', alpha=0.6)
# axs[1].plot(ekf_np[:, PX], ekf_np[:, PY], 'r--', linewidth=2, label='EKF (Standard)')
# axs[1].set_title(f"Extended Kalman Filter (MSE: {F.mse_loss(full_x_hat_ekf[:, :2], x_true[:, :2]):.4f})", fontsize=14)
# axs[1].set_ylabel("Pozice Y [m]")
# axs[1].legend()
# axs[1].grid(True)
# axs[1].axis('equal')

# --- 3. Graf: UKF ---
axs[2].plot(gt_np[:, PX], gt_np[:, PY], 'k-', linewidth=2, label='Ground Truth', alpha=0.6)
axs[2].plot(ukf_np[:, PX], ukf_np[:, PY], 'g:', linewidth=3, label='UKF (Standard)')

```

```

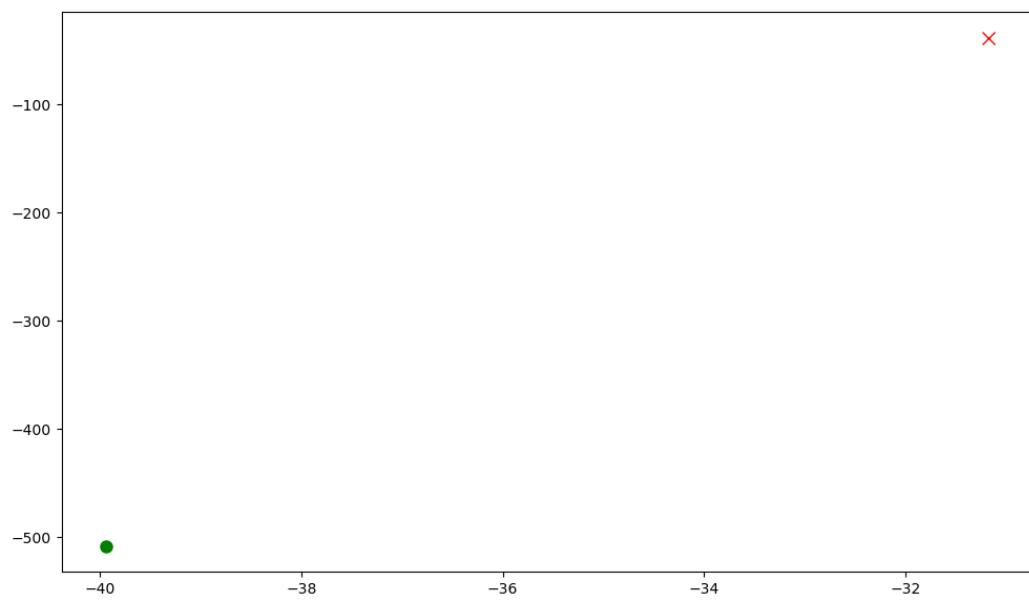
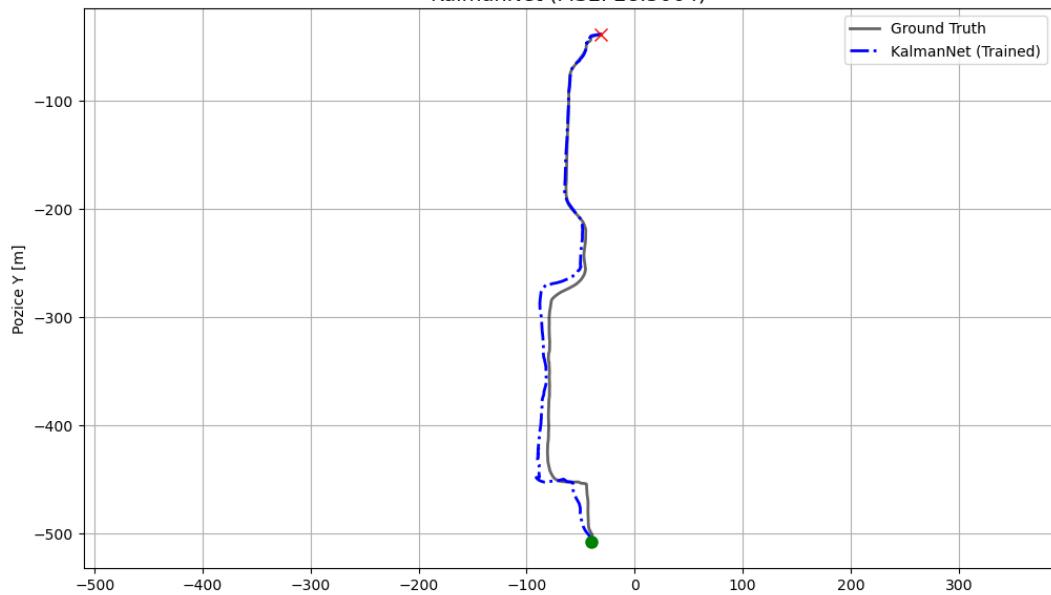
axs[2].set_title(f"Unscented Kalman Filter (MSE: {F.mse_loss(full_x_hat_ukf[:, :2], x_true[:, :2]):.4f})", fontsize=14)
axs[2].set_xlabel("Pozice X [m]")
axs[2].set_ylabel("Pozice Y [m]")
axs[2].legend()
axs[2].grid(True)
axs[2].axis('equal')

# Přidání start/cíl markerů do všech grafů
for ax in axs:
    ax.plot(gt_np[0, PX], gt_np[0, PY], 'go', markersize=8, label='Start')
    ax.plot(gt_np[-1, PX], gt_np[-1, PY], 'rx', markersize=8, label='Cíl')

plt.show()

```

Vizualizace odhadu polohy (Trajektorie č. 26)
KalmanNet (MSE: 28.5064)



Unscented Kalman Filter (MSE: 53606.1484)

