Machine Learning Assignment

Online Dynamics Learning for Predictive Control with an Application to Aerial Robots

Rishi Raj Sahoo

Subhankar Mishra Lab

Oct 7, 2024

Outline

Introduction

Abstract

- **Goal** : Improving the accuracy of dynamic models for **Model Predictive Control (MPC)** in an online setting.
- In offline learning:
  - Training data is collected.
  - A prediction model is learned via an elaborate training procedure.
  - However, the model does not adapt to **disturbances** or **model errors** observed during deployment.
- This adopt **knowledge-based neural ordinary differential equations (KNODE)** as the dynamic models.
  - Techniques inspired by **transfer learning** are used to improve model accuracy continually.
- Demonstrated with a **quadrotor**:
  - This verify the framework through simulations and physical experiments.
  - Results show that the approach can account for **time-varying disturbances** while maintaining good **trajectory tracking performance**.

Context

- MPC:
  - Is an **optimization-based** approach using prediction models.
  - Leverages physics models or accurate data-driven models for good closed-loop performance.
- Challenge:
  - **Reliance on accurate dynamic models** makes it hard for the controller to adapt to **system changes** or **environmental uncertainties**.
  - If robot dynamics change or disturbances occur during deployment, the controller must update its dynamic model to maintain performance.
- Recent advancements in **deep learning** have shown potential in modeling dynamical systems.
  - **Neural networks** offer:
    - A bypass to the **bottom-up construction** of dynamics that requires expert knowledge or physical intuition.
    - Faster optimization due to modern optimization algorithms.

Problem Formulation

- The robot dynamics are given by:

$$\boxed{\dot{x} = f(x, u)}$$

  - where:
    - $\dot{x}$: **State derivative** (rate of change of the state).
    - f: **True dynamics** of the robot.
    - x: **State** of the robot.
    - u: **Control input** to the robot.
- The updated dynamics model is represented by:

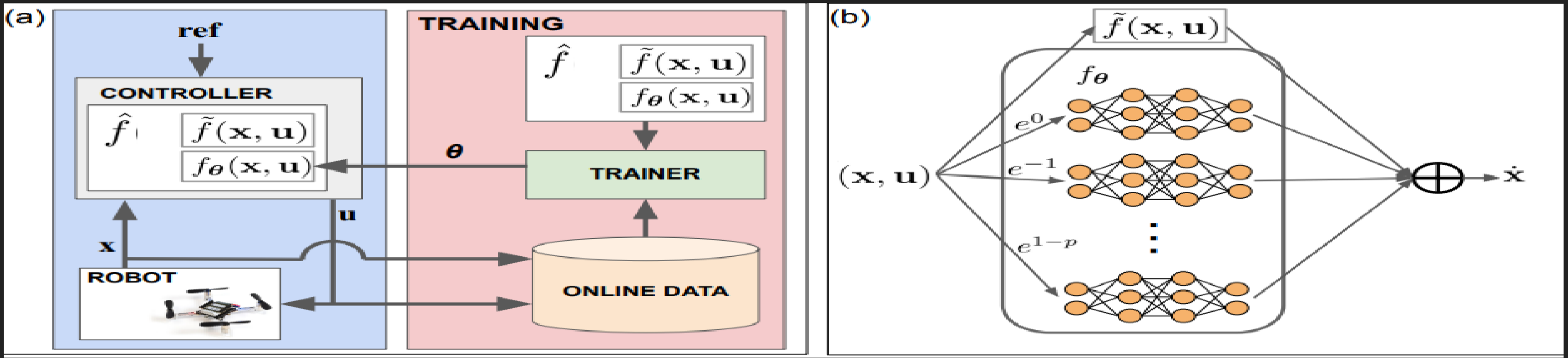
$$\boxed{\dot{x} = \hat{f}(x, u)}$$

  - where:
    - $\hat{f}$: **Updated estimate** of the dynamics model.
    - x: **State** of the robot.
    - u: **Control input** to the robot.

- The sequence of collected data samples is denoted by:

$$S := [(x(t_0), u(t_0)), (x(t_1), u(t_1)), \dots]$$

  - where:
    - S: **Sequence of data samples** consisting of states and control inputs.
    - $x(t_0), x(t_1), \dots$: **States** of the robot at different timestamps.
    - $u(t_0), u(t_1), \dots$: **Control inputs** at different timestamps.
    - $t_0, t_1, \dots$: **Timestamps** at which the data is collected.
- f_θ: **Neural Network parametrised with** θ
- $\tilde{f}$:**Physics Knowldege**



Online Dynamics Learning

KNODE

- Three aspects of KNODE:
  - It requires less data for training.(Improving adaptiveness)
  - It is a continuous-time dynamic model.(Compactability)
  - Many robotics systems have readily available physics mpdel that can be used as knowledge.
- State and control concatenated and represented as:

$$\boxed{z = [x^T, u^T]^T}$$

- The dynamics is expressed as:

$$\boxed{\hat{f}(z,t) = M_\psi(\tilde{f}(z,t), f_\theta(z,t))}$$

where

  - M_ψ = Selection Matrix parametrized with ψ(which couples neural network with knowledge)

- The loss function is defined as:

$$\boxed{L(\theta,\psi)=\frac{1}{m-1}\sum_{i=1}^{m-1}\int_{t_i}^{t_{i+1}}\delta(t_s-\tau)\|\hat{x}(\tau)-x(\tau)\|^2 d\tau+R(\theta,\psi)}$$

- where:
  - m: Number of points in the **training trajectory**.
  - δ: **Dirac delta function**.
  - $t_s \in T$: Any **sampling time** in set T.
  - $R(\theta,\psi)$: **Regularization term** on the neural network and coupling matrix parameters.
  - $\hat{x}(\tau)$: The **estimated state** at time τ.
  - $x(\tau)$: The **ground truth state** at time τ.
  - $\|\hat{x}(\tau)-x(\tau)\|^2$: **Squared error** between the estimated and true states.

- The estimated state $\hat{x}(\tau)$ comes from the state $\hat{z}(\tau)$, which is given by:

$$\boxed{\hat{z}(\tau) = z(t_i) + \int_{t_i}^{\tau} \hat{f}(z(\omega), \omega) d\omega}$$

  - where:
    - $\hat{z}(\tau)$: The **state at time** τ generated using the model $\hat{f}$.
    - $z(t_i)$: The **initial condition** at time t_i.
    - $\hat{f}(z(\omega), \omega)$: The **updated dynamics** model.
- The **optimization** of the neural network parameters in KNODE can be done using:
  - **Backpropagation**.
  - **Adjoint sensitivity method**, a memory-efficient alternative to backpropagation (used in this work, similar to [6]).

Online Data Collection and Learning

Algorithm 1 Data collection and model updates

Initialize the current time, last save time, total duration, and the collection interval as t_i, t_s, t_N, and t_{col}
$t_i \leftarrow 0$
OnlineData $\leftarrow$ []
while $t_i < t_N$ **do**
    if New model is available **then**
        Controller updates new model
        $t_s \leftarrow t_i$
    end if
    if t_i is not 0 and $t_i - t_s == t_{col}$ **then**
        Save OnlineData
        $t_s \leftarrow t_i$
        OnlineData $\leftarrow$ []
    end if
    Robot updates state using control input
    Append new robot state and control input to OnlineData
    $t_i \leftarrow$ current time
end while

Algorithm 2 Online dynamics learning

Initialize the current time and total duration as t_i and t_N
$t_i \leftarrow 0$
while $t_i < t_N$ **do**
    while No new data available **do**
        Wait
    end while
    Train a new model with the newest data
    Save the trained model
    $t_i \leftarrow$ current time
end while

$$\boxed{f^{(i+1)} = M\psi_{(i+1)}\left(\hat{f}^{(i)}, e^{i+1-p} f_{\theta(i+1)}\right)} \text{for } i < p,$$

with the initial condition:

$$\hat{f}^{(0)} = \tilde{f}$$

Applying learned Models in MPC

- **Objective**: Solve the following **constrained optimization problem** in a receding horizon manner:

$$\boxed{\min_{x_0,\ldots,x_N,u_0,\ldots,u_{N-1}} \sum_{i=1}^{N-1} x_i^T Q x_i + u_i^T R u_i + x_N^T P x_N}$$

  - subject to:

$$x_{i+1} = f(x_i, u_i), \quad \forall i = 0, \ldots, N-1$$

$$x_i \in X,\ u_i \in U, \quad \forall i = 0, \ldots, N-1$$

$$x_0 = x(t),\ x_N \in X_f$$

- **Variables**:
  - x_i: Predicted states.
  - u_i: Control inputs.
  - N: The **horizon** length.
  - X, U, X_f: The **state**, **control input**, and **terminal state constraint sets**.
  - $f(\cdot,\cdot)$: A **discretized version** of the learned **KNODE model**.

- **Purpose**:
  - The model $f(\cdot, \cdot)$ is used to **predict future states** within the horizon.
  - A **precise model** results in more **accurate predictions** of the states, which leads to more **effective control actions**.
- **Cost Function Weights**:
  - Q: Weighting matrix penalizing the **states**.
  - R: Weighting matrix penalizing the **control inputs**.
  - P: **Terminal state** cost matrix.
- **Initial Condition**:
  - $x(t)$: The state obtained at time step t, which acts as an **input** to the optimization problem.
- **Control Action**:
  - Upon solving the optimization problem, the first element of the **optimal control sequence** u_0^* is applied to the robot as the control action.

- **Receding Horizon Execution**:
  - The robot moves according to the control action and generates new state measurements $x(t+1)$.
  - These new measurements are used to **solve the optimization problem** at the next time step.
- **Implementation**:
  - The optimization problem is implemented and solved using **CasADi**
  - **IPOPT**, an interior-point method within the CasADi library, is used to solve the problem.
  - The solver is **warm-started** at each time step by providing an **initial guess** of the solution, based on the optimal solution from the previous time step (both **primal and dual variables**).

Simulation

Dynamics of a Quadrotor

- To apply the **KNODE-MPC-Online framework**, we first construct a **KNODE model** by combining a **nominal model** derived from physics with a **neural network**.
- **Nominal Model**: For the quadrotor, the nominal model is derived from its **equations of motion**:

$$\boxed{m\ddot{r} = mg + R\eta, \quad I\dot{\omega} = \tau - \omega \times I\omega}$$

  - where:
    - r: **Position** of the quadrotor.
    - ω: **Angular rates** of the quadrotor.
    - η: **Thrust** generated by the motors.
    - τ: **Moments** generated by the motors.
    - g: **Gravity vector**.
    - R: **Transformation matrix** mapping η to accelerations.
    - m: **Mass** of the quadrotor.
    - I: **Inertia matrix** of the quadrotor.

- **State and Control Input:**
  - Define the **state** as:

$$\boxed{x := [r^\top, \dot{r}^\top, q^\top, \omega^\top]^\top}$$

    - where q denotes the **quaternions** representing the orientation of the quadrotor.
  - Define the **control input** as:

$$\boxed{u := [\eta^\top, \tau^\top]^\top}$$

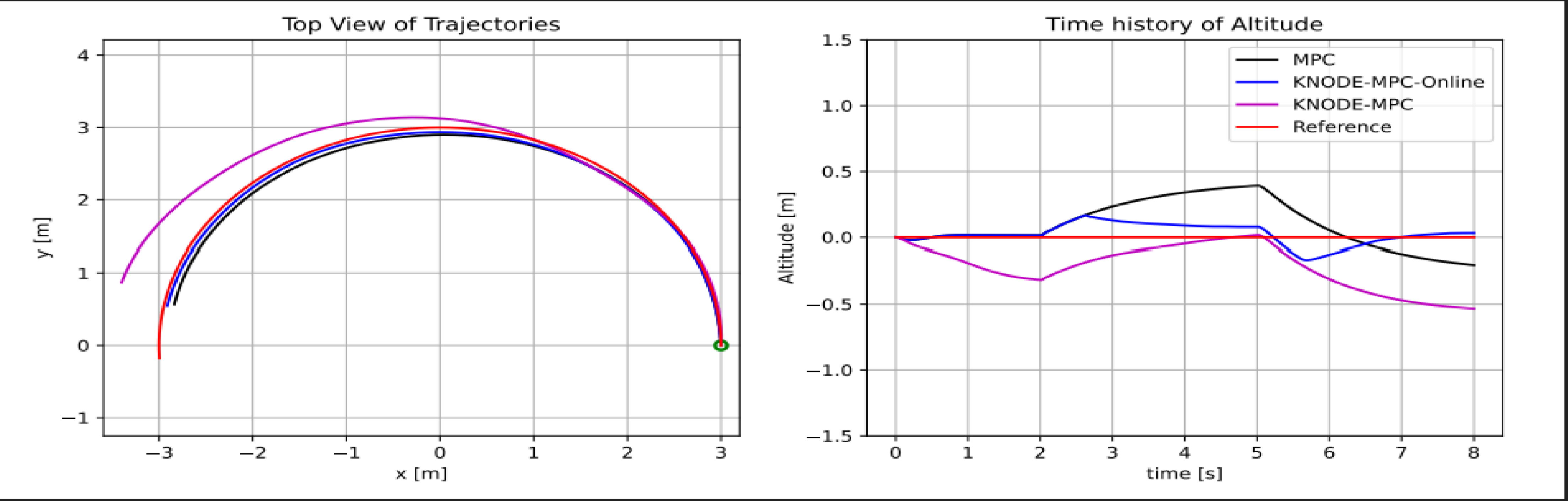
- **Nominal Component of KNODE:**
  - The nominal component of the KNODE model can be expressed as:

$$\tilde{f}(x, u)$$

  - where:
    - $\tilde{f}(x, u)$: The **nominal dynamics model** based on physics for the quadrotor.

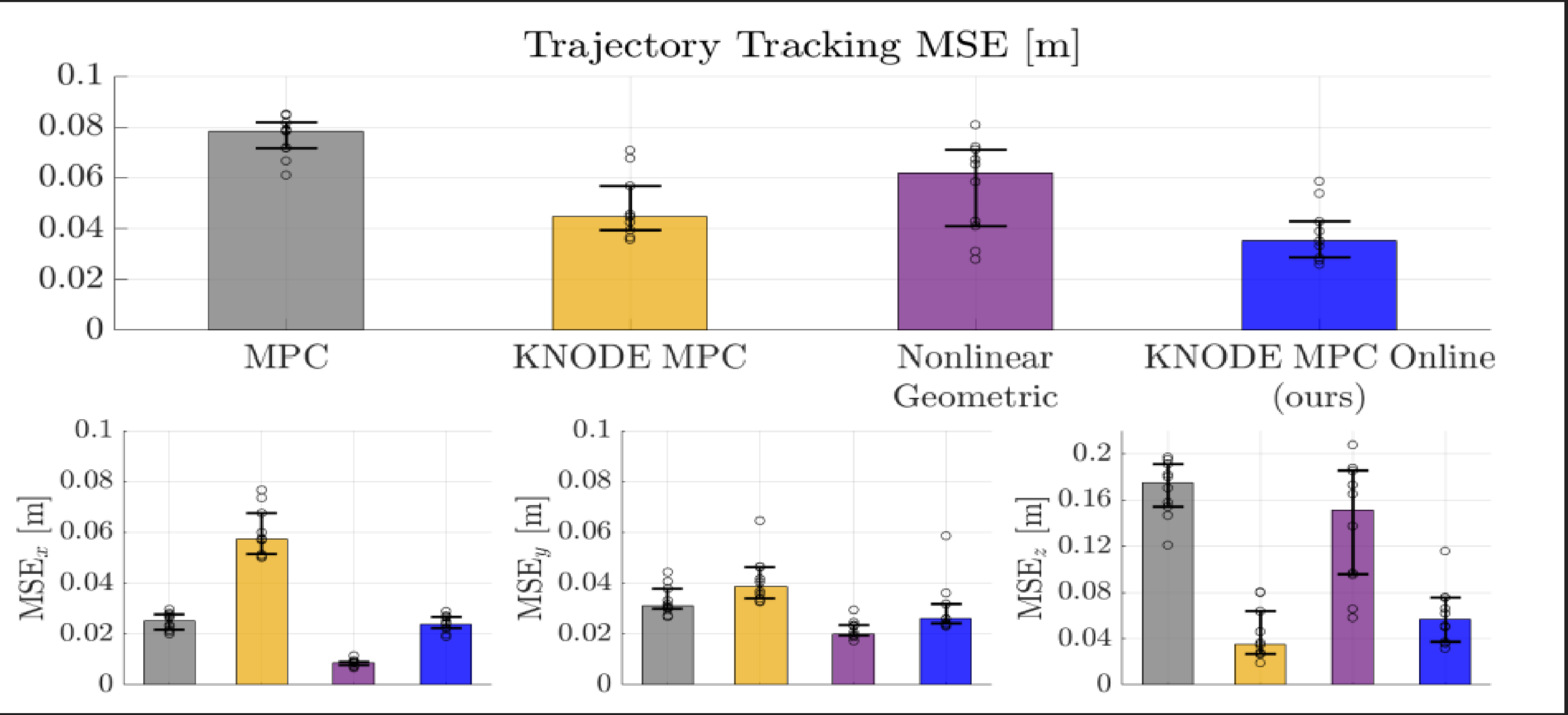
Simulation Setup and Results

Radius [m]	2.0			3.0			4.0		
Speed [m/s]	0.8	1.0	1.2	0.8	1.0	1.2	0.8	1.0	1.2
MPC	0.0904	0.1280	0.1705	0.0949	0.1371	0.1861	0.0967	0.1412	0.1937
KNODE-MPC [25]	0.1222	0.1945	0.2555	0.1974	0.1769	0.2098	0.5303	0.4175	0.3418
Geometric Control [34]	0.2168	0.2572	0.3253	0.2067	0.2267	0.2606	0.2046	0.2194	0.2416
KNODE-MPC-Online (ours)	**0.0660**	**0.1113**	**0.1678**	**0.0657**	**0.1043**	**0.1554**	**0.0709**	**0.1092**	**0.1571**



Physical Experiments

Performance of KNODE-MPC-Online



Conclusion

- **Proposed Framework:**
  - We introduce a novel and **sample-efficient framework** called **KNODE-MPC-Online**.
  - The framework learns the **dynamics of a quadrotor robot** in an **online setting**.
- **Application in MPC:**
  - The learned **KNODE model** is applied in a **Model Predictive Control (MPC)** scheme.
  - The **dynamic model** is **adaptively updated** during deployment to respond to changes.
- **Key Results:**
  - **Simulations and real-world experiments** demonstrate that:
    - The proposed framework enables the quadrotor to **adapt and compensate for uncertainty and disturbances** during flight.
    - It improves the **closed-loop trajectory tracking performance**.
- **Future Work:**
  - Applying this framework to **other robotic applications** where dynamic models can be learned to achieve **enhanced control performance**.

Limitations

- **Assumption**:
  - The framework assumes a **continuous-time** nature of system dynamics.
  - This limits its applicability to **stochastic systems**.
- **Potential Improvements**:
  - There are **variants of NODE** that model **stochastic differential equations**.
  - Future work will aim to extend the algorithm to **incorporate stochastic models** to broaden its applicability.

References

- Github repo link
- Main papers:
    1. Online Dynamics Learning for predictive Control with an Application to aerial Robots
    2. KNODE-MPC: A Knowledge-Based Data-Driven Predictive Control Framework for aerial Robots
- Images sources:
    - All images
- Video Sources:
    - Video zip

Questions?