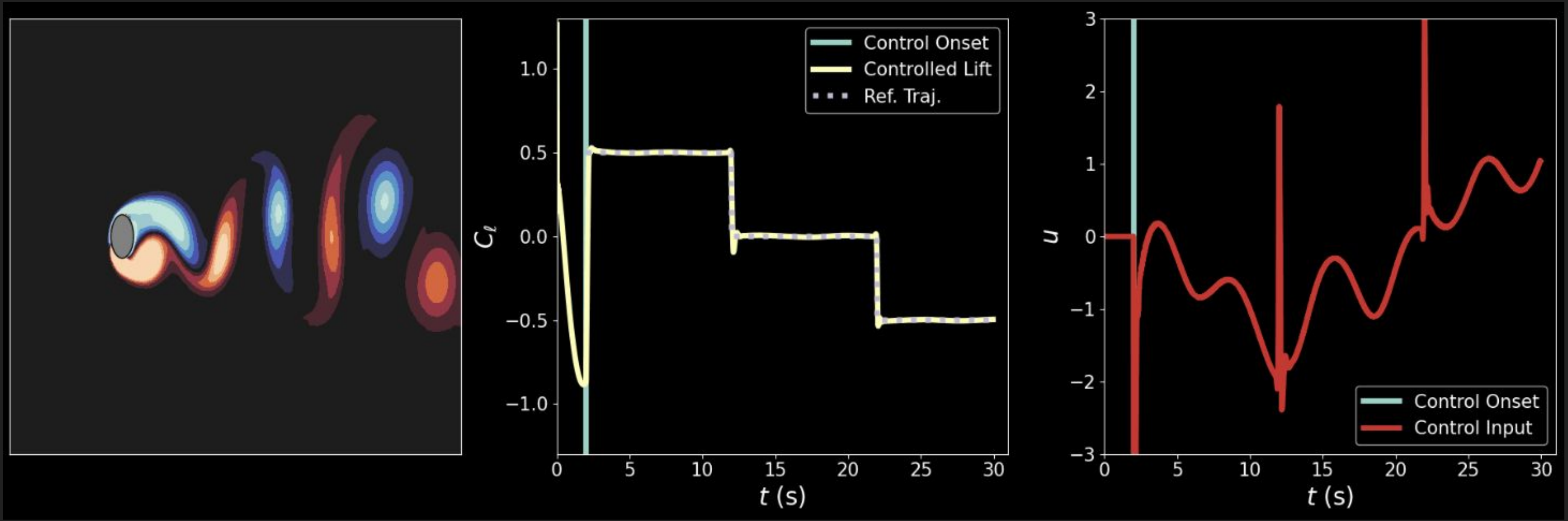
Recurrent Network Based Model Predictive Control

CSE 583: Project Presentation

Stewart Lamon, Jan Williams, and Deyang Zheng

Problem Statement

- Surrogate modeling with various recurrent neural network architectures
- Compute near-optimal control actions from the trained surrogate model



Project Objectives

- Flexible integrator class for user defined systems
  - Existing code requires the user to define a unique class for each system of interest
  - Stewart: Implementation of general ODE models

- Function to simulate a system with control given a trained surrogate model
  - Existing code requires user to write their own closed loop simulation
  - Deyang: Implementation of utils function to simulate ODE systems with control

- Implementation of Simplified Structured State Space Layer for Sequences (S5)
  - Jan: Develop modified S5 model for use as surrogate model in model predictive control

Human centered description of users/user cases

Researcher-Centric Design: The tool is tailored for researchers across various fields, including academia, industry, and education

Ease of Use: For users like graduate students or domain experts new to advanced machine learning frameworks, the project offers a beginner-friendly interface

Researcher (Judith)

- **User Story**: Judith, a postdoc specializing in recurrent neural network architectures, wants to evaluate and compare multiple control techniques for complex systems. Our project allows her to experiment with custom architectures. By leveraging the modular design of the tool, Judith adds her own RNN-based controllers, runs experiments, and obtains insights into their comparative performance

Technologies

Machine Learning Frameworks

- **PyTorch**: Used for building and training deep recurrent neural network models, enabling accurate and efficient predictions for the MPC framework

Numerical Libraries

- **NumPy**: Utilized for array manipulations, linear algebra, and numerical computations
- **SciPy**: Incorporated for solving differential equations and integrating models using scipy.integrate.solve_ivp
- **Neuromancer:** Utility functions for instantiating benchmark control systems

Major Challenges

Figuring out what a good test looks like. What we landed on:

- Type/Shape checking
- Error bounds for known solutions
- Smoke Tests

Moving back and forth from torch tensors to numpy arrays

- Different parts of the code need one or the other so conversion between the two needs to happen often. Docstrings help a lot!
- Default data types are different between the two, need to be explicit

S5 Models allow for parallel scan evaluation but PyTorch does not

- Models run slower than they need to which can be an issue in real-time control

Next Steps

- Develop integrator class to account for partial observations
  - We can’t always measure everything
- Expand functionality of closed loop simulation function to include external models
  - Not all systems of interest can be represented as ODEs!
- Efficient S5 implementation
  - Efficiency isn’t just nice, it’s necessary for online computation of control actions