Progress Report: Residual-Aware RL Tuning of MPC on Data-Driven Models

Erinç Ada Ceylan
22101844

Instructor: Assoc. Prof. Yıldıray Yıldız

ME 418/518 – Data-Based Control Systems

1 Project Overview and Reminder

The original project proposal focuses on designing a **Residual-Aware RL–MPC** framework. The main idea is to use data driven prediction models together with Model Predictive Control (MPC), and then construct a Reinforcement Learning (RL) agent adapt MPC parameters based on the *prediction residuals* of the models.

The target pipeline is:

- Use linear data driven models (ARX, FIR, N4SID) to predict system evolution.
- Compute a one step prediction error (residual) at each time step,

$$r_t = \|x_{t+1}^{meas} - \hat{x}_{t+1}^{pred}\|,$$

  and treat this as a measure of model reliability.
- Run an MPC controller on top of the chosen model and treat the stage cost and horizon parameters (Q, R, N) as *tunable* for RL side.
- In the full version of the project, an RL agent will learn a policy that maps $[x_t, r_t]$ to discrete choices of (Q, R, N), in the aim of weights-varying MPC as in [1, 2, 3], but in a simpler, linear setting.

This progress report summarizes what has been done so far (mainly on the identification and MPC sides, and on the literature/code review), what failed (first simulation setup attempt), and what is planned next.

2 Work Completed Since the Proposal

2.1 Identification Models and Residual Definition

In class we covered Auto Regressive models with eXogenous input (ARX), Finite Impulse Response (FIR) models, and N4SID-type subspace identification. Following that structure, my current plan is to obtain prediction models from *one* of these families.

So far, I have:

- Reviewed and organized the **existing course code** for ARX, FIR models, and N4SID, including how input–output data and identification options are passed.
- Drafted a small set of **candidate prediction models**:
  - Different ARX orders (n_a, n_b) as alternative models,
  - N4SID models with varying state dimension (e.g., small, medium, large order) for the same input–output data.

Conceptually, each of these gives a discrete-time state-space model

$$x_{k+1} = Ax_k + Bu_k, \quad y_k = Cx_k + Du_k$$

suitable for MPC.

- Defined the **residual signal** as a scalar norm of the one-step error between the measured next state/output and the model prediction. For linear models, this will be computed as

$$r_k = \|y_{k+1}^{\text{meas}} - \hat{y}_{k+1}^{\text{model}}\|_2,$$

using either the ARX/FIR regression form or the N4SID state-space predictor.

The structure of the models and the residual definition are now clear and compatible with the MPC/RL pipeline.

2.2 MPC Baseline Structure

The next building block is a plain MPC controller that does *not* use RL yet. The baseline design is:

- **Prediction model:** one of the identified linear models (initially ARX-derived, later possibly an N4SID model).
- **Cost function:**

$$J = \sum_{i=0}^{N-1} \left(x_{k+i}^\top Q x_{k+i} + u_{k+i}^\top R u_{k+i} \right),$$

with a small terminal weight or constraint if needed.

- **Constraints:** simple box constraints on states and inputs to keep the first version manageable.

The main focus has been to understand how to *connect* the identification models to the MPC problem in a way that later allows plugging in an RL based scheduler.

2.3 Literature and Code Review on Residual-Aware RL–MPC

Because the project idea is inspired by weights-varying MPC using RL, I studied both the paper and the open-source framework by Zarrouki and co-authors.

Safe RL-driven Weights-Varying MPC. The work in [1] proposes a safe RL-driven weights-varying Model Predictive Control scheme for autonomous vehicle motion control. The controller adjusts MPC weights online using a deep RL agent while respecting safety constraints, built on top of nonlinear MPC with a single-track vehicle model and a detailed tire model.

I focused on the following aspects:

- How they separate the *plant model* (vehicle dynamics) from the *controller* and from the *RL layer*,
- How the RL agent's action is restricted to a safe catalog of MPC weight combinations,
- How performance and safety metrics are defined in the closed-loop evaluation.

TUM-CONTROL Simulation Framework. I reviewed the associated TUM-CONTROL framework [5], which is a modular Python simulation environment for nonlinear MPC and its RL-augmented variants. The repository clearly separates:

- vehicle dynamics models,
- MPC formulations (ACADOS-based),
- trajectory planner emulator,
- safe RL-driven weights-varying controllers (under `Learning_To_Adapt/SafeRL_WMPC`).

From this code review, I learned how a realistic pipeline can be structured and what is *overkill* for my project. My project will stay with linear MPC, standard Gym-like environments, and much simpler models, but the architecture of TUM-CONTROL is a useful reference for separating modules and logging performance.

2.4 First Simulation Environment Attempt and Lessons Learned

Initially, I tried to directly couple a **racecar_gym**-style environment with an MPC implementation in Python. The idea was to:

- Use the environment's dynamics as the "plant",
- Log state–input trajectories,
- Identify ARX/FIR/N4SID models,
- Then close the loop with MPC.

However, in practice this attempt failed mainly because of:

- Heavy and version-sensitive dependencies (Gym versions, PyTorch/TensorFlow, rendering backends),
- Time spent debugging installation and environment issues rather than the control problem itself,
- Limited documentation for integrating external controllers (e.g., replacing the default RL policy by a custom MPC in a clean way).

This experience made it clear that environment choice is critical. For the next iteration, I will deliberately look for:

- A *simpler* Gym-like environment with low dependency overhead,
- Clear access to continuous state and action spaces,
- Deterministic dynamics suitable for linear approximation (e.g., small planar robot, simple vehicle, or process control benchmark).

3 Current Status and Challenges

Overall, the conceptual pieces of the project are in place:

- Identification models: ARX/FIR/N4SID structure and residual definition are ready to be implemented on data.
- MPC baseline: cost structure, constraints, and connection to linear models are clear, following [7].
- Literature/code review: the connection to weights-varying MPC and its implementation style (TUM-CONTROL) has been studied.

The main challenges so far are:

- **Environment selection:** The first attempt with a racecar-type Gym environment highlighted that a too complex or fragile simulator can block progress.
- **Time and scope management:** To keep the project feasible, I need to prioritize a stable linear MPC + simple environment pipeline before adding RL.
- **RL implementation maturity:** We are still in the process of learning RL in the course, so it is reasonable to postpone the full RL integration to the later stages and treat it as an extension on top of a working MPC baseline.

4 Planned Figures and Final Evaluation Metrics

- **Identification quality:**
  - Predicted vs. true output trajectories for ARX/FIR/N4SID models,
  - Residual time series r_t under different models and operating conditions.
- **MPC performance:**
  - Step or tracking response plots (state, input) for different fixed (Q, R, N) settings,
  - Constraint satisfaction plots (e.g., input saturation, state limits).
- **Residual-aware adaptation (if RL part is reached):**
  - Comparison between residual-blind and residual-aware tuning in terms of tracking error, overshoot, and constraint violations,
  - Evolution of the selected weight/horizon index over time, correlated with r_t.

These figures will be used to evaluate:

- Tracking performance (ISE/IAE),
- Robustness to model mismatch (performance under different models),
- Constraint violation rates.

5 Next Steps

The concrete next steps are:

1. **Finalize environment choice:** Select a simpler Gym-like or custom Python environment with continuous states and inputs, and set up a minimal data-logging loop.
2. **Run identification on simulated data:** Fit ARX/FIR and N4SID models to one experiment, compare their prediction quality, and decide on a primary model family.
3. **Implement linear MPC baseline:** Using the chosen model, implement a standard MPC controller (either with an existing QP solver or a simple quadratic program) and demonstrate closed-loop tracking.
4. **Define residual-based tuning rule:** As an intermediate step before full RL, design a simple hand-crafted rule that switches between "aggressive" and "conservative" MPC settings based on thresholds on r_t.
5. **RL Agent:** Replace the hand-crafted rule by a small RL agent (e.g., tabular or simple DQN) whose action space is a finite catalog of (Q, R, N) combinations, similar in spirit to [1, 4].

References

[1] B. Zarrouki, M. Spanakakis, and J. Betz, "A Safe Reinforcement Learning driven Weights-varying Model Predictive Control for Autonomous Vehicle Motion Control," *IEEE Intelligent Vehicles Symposium (IV)*, 2024.

[2] B. Zarrouki, C. Wang, D. Schuurmans, and J. Betz, "Weights-varying MPC for autonomous vehicle guidance: a deep reinforcement learning approach," *European Control Conference (ECC)*, 2021.

[3] B. Zarrouki, C. Wang, and J. Betz, "Adaptive stochastic nonlinear model predictive control with look-ahead deep reinforcement learning for autonomous vehicle motion control," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024.

[4] F. Airaldi, "mpcrl: Reinforcement Learning with Model Predictive Control," TU Delft, 2024. [Online]. Available: `https://github.com/TUDelft-DataDrivenControl/mpcrl`

[5] B. Zarrouki et al., "TUM-CONTROL: A modular simulation framework for ultra-rapid prototyping of self-adaptive and robust Nonlinear Model Predictive Control," GitLab Repository, Autonomous Vehicle Systems Lab, TUM, 2024. [Online]. Available: `https://gitlab.lrz.de/bzarr/tum-control`

[6] P. Van Overschee and B. De Moor, "N4SID: Subspace algorithms for the identification of combined deterministic-stochastic systems," *Automatica*, vol. 30, no. 1, pp. 75–93, 1994.

[7] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*, 2nd ed., Nob Hill Publishing, 2018.