Imitation learning

Salah EL Din SEKAR

August 22, 2024

Contents

Abstract

This report represents my internship at Verimag, Grenoble, under the supervision of Dr. Thao Dang. The project focuses on using imitation learning to train a neural network controller that integrates the advantageous properties of conventional controllers specified using Signal Temporal Logic (STL). The implementation is applied to robotic systems utilizing the Robot Operating System (ROS 2).

The core of the project involves generating behavioral traces through ROS 2 simulations, which provide diverse and comprehensive data. These traces are then used to train the neural network in MATLAB. To ensure the quality and reliability of the training data, the Breach tool is employed for falsification, identifying and rectifying deviations from expected behaviors. This approach aims to improve the accuracy, robustness, and safety of neural network controllers, making them suitable for deployment in complex, real-world robotic applications.

Introduction

Artificial intelligence (AI) and data sciences are rapidly transforming our society, leading to groundbreaking advancements across various domains. These technological innovations necessitate a twofold evolution in systems design. On one hand, new theories and methodologies must be developed to accommodate and leverage these AI advancements. On the other hand, systems design can harness the power of AI and data sciences to significantly enhance the efficiency, dependability, and security of control systems.

One of the major challenges in designing the next generation of control systems lies in the inherent unpredictability of AI techniques. Unlike traditional control methods, AI lacks a formal framework to provide robust safety guarantees, making it difficult to predict and manage potential risks. Addressing this challenge is crucial for the successful integration of AI into control systems, particularly in critical applications such as robotics.

This project aims to bridge this gap by focusing on the formal design of neural network controllers for robotic systems using imitation learning. The primary objective is to create a framework that ensures both the robustness and reliability of AI-driven controllers in real-world scenarios.

Glossary

Your glossary text.

Chapter 1

Imitation Learning

1.1 Context

Imitation learning, also known as Learning from Demonstration (LFD), is a method in machine learning that involves mimicking the behavior of an expert, often a human, to achieve optimal system performance. This technique relies on using the expert as a reference point to compare results and replicate the expert's behavior.

How It Works

1. **Data Collection:**
    - The first step involves gathering data from an expert performing a task. For instance, if the task involves controlling a PID water tank, the inputs and states of the system are recorded. The expert could also be operating a Model Predictive Control (MPC) system, where the control inputs and corresponding system states are logged.
2. **Learning:**
    - The collected data is then used to train a machine learning model, such as a neural network. This model learns to map the system states and inputs to the desired control actions by imitating the expert's behavior.
3. **Evaluation:**
    - After training, the model's performance is evaluated to validate its accuracy. This involves testing the neural network controller in various scenarios to ensure it can reliably replicate the expert's behavior. If the model's performance is not satisfactory, further improvements and refinements are made to the training process.

Applications

Imitation learning is widely used where defining reward criteria is difficult:

- **Autonomous Vehicles:** Train self-driving cars by learning from human drivers, capturing complex maneuvers and real-world driving behavior.
- **Robotics:** Teach robots tasks like cooking or folding clothes, which are easy for humans but hard to program.
- **Game Playing:** Train AI agents to play video and board games at a human level by learning from skilled players.
- **Healthcare:** Assist in surgical robotics by learning from expert surgeons to perform delicate operations.

1.2 My work

1.3 Results

Chapter 2

Implementation

2.1 Neural network

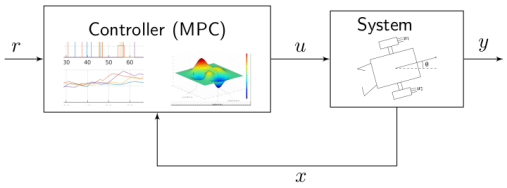

Figure 2.1: Neural Network Architecture for Imitation Learning

for exemple MPC soles a NL optimization problem at each step and it costly for online use

2.1.1 Issues with MPC Controller

Model Predictive Control (MPC) is an advanced control strategy that solves a nonlinear optimization problem at each control step. While MPC is highly effective for handling multi-variable control problems and constraints, it has several drawbacks:

- **Computational Cost:** Solving a nonlinear optimization problem at each time step is computationally intensive, making it costly for online use, especially in real-time systems.
- **Scalability:** The computational burden increases with the complexity and size of the system, which can be impractical for large-scale applications.

2.1.2 Replacing MPC with a Neural Network

To address these issues, we propose using a neural network to emulate the MPC controller. The process involves the following steps:

1. **Data Collection:** Collect a dataset D of input-output pairs from the MPC controller. This dataset includes system states x and corresponding control inputs u generated by the MPC: $D = \{(x, u) \mid u = \text{MPC}(x)\}$.
2. **Training:** Use the collected dataset to train the neural network. The neural network learns to map the system states x to the control inputs u, effectively replicating the behavior of the MPC controller.
3. **Evaluation:** Validate the performance of the trained neural network to ensure it accurately replicates the MPC's control actions.

2.2 Signal Temporal Logic (STL)

2.2.1 Temporal Logic Background

Temporal logics are used to specify patterns that the timed behaviors of systems may or may not satisfy. These logics include basic operators along with temporal operators that allow for the expression of time-dependent behaviors.

Temporal Operators

- **NEXT (X):** Specifies that a condition will hold in the next time step.
- **Always (G):** Specifies that a condition will always hold.
- **Eventually (F):** Specifies that a condition will hold at some point in the future.
- **Until (U):** Specifies that one condition will hold until another condition becomes true.

Types of Temporal Logic

Different types of temporal logics are used based on the nature of the predicates and time:

- **Linear Temporal Logic (LTL):**

  $$G(r \rightarrow Fg)$$

  Boolean predicates, discrete-time.

- **Metric Temporal Logic (MTL):**

  $$G(r \rightarrow F_{[0,0.5s]}g)$$

  Boolean predicates, real-time.

- **Signal Temporal Logic (STL):**

$$G(x[t] > 0 \rightarrow F_{[0,0.5s]} y[t] > 0)$$

  Predicates over real values, real-time.

2.2.2 Example

Consider the following STL specification:

$$\varphi := G(x[t] > 0.5 \rightarrow F_{[0,0.6]}(G_{[0,1.5]} x[t] < 0.5))$$

This formula specifies that whenever $|x|$ is greater than 0.5, within 0.6 seconds, $|x|$ should settle under 0.5 and remain so for at least 1.5 seconds.

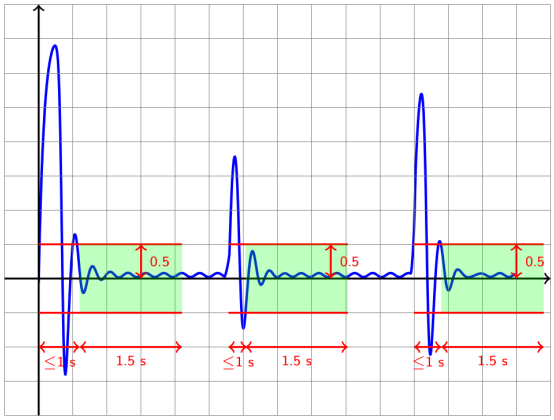

Figure 2.2: STL

2.3 Parametric Signal Temporal Logic (PSTL)

PSTL extends STL by allowing the replacement of numeric constants with symbolic variables or parameters.

Parameters: Parameters can be adjusted or learned. For instance, $\varphi = \square_{[0,\tau]}(\|y(t)\| < s)$ uses parameters τ and s.

Valuation: A PSTL formula is concretized into an STL formula through a valuation (mapping symbolic parameters to real numbers). For example, $p : \{\tau \rightarrow 2, s \rightarrow 10\}$ makes $\varphi(p) = \square_{[0,2]}(\|y(t)\| < 10)$.

Monotonicity: Some PSTL formulas can be monotonic, meaning their satisfaction increases with certain parameter values.

Use: Suitable for scenarios where the specific numeric thresholds or timing constraints are not known *a priori* and need to be inferred or optimized.

2.4 Breach Tool

Breach is a MATLAB/C++ toolbox designed for time series analysis and simulation-based analysis of dynamical, Cyber-Physical Systems (CPS), and hybrid systems [1]. This tool plays a core role in our study, including checking formal requirements on data and dynamical system behaviors using STL formula, conducting extensive testing, and ultimately falsifying an STL requirement using various optimization algorithms. Consequently, the Breach tool is essential to our testing of neural networks, as it allows testing the system with a neural network controller in closed loop.

Moreover, Breach facilitates the computation and analysis of trajectories in hybrid systems, providing a set of simulation-based techniques for analyzing deterministic models. Its primary feature is its ability to compute and analyze the properties of large sets of trajectories efficiently. This is achieved through an efficient numerical solver for ordinary differential equations, which also provides sensitivity analysis with respect to parameter variations. Additionally, it supports the evaluation of STL formulas, allowing for detailed specification and monitoring of system properties over time.

2.5 Learning algorithms

the process of the trainng is based on several parts. first we tart by accumulating the data given by the real pid over simulation with different parameters, those data are calles traces, they should be defined oer a period of time, we use those data o estimate the false domain.

Chapter 3

ROS2

ROS2 is a middleware...

we had the breach toolbox that is capable to train a neural network and capable to test the robustness of our neural network by using the stl criteria. we also have ROS2 as a powerfull middleware for the robot interface so we got inspired by the following figure to apply our work:

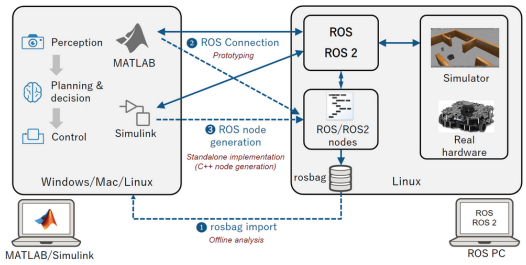

Figure 3.1: STL

the main objectif was to make the connection between ros2 and this toolbox. getting information in real time was not necessary since the algorithm wirks offline. we need o get traces of data that can be fed to the neural network and then after we itain the falsification we go back to the real simulation if a counter example is found.

the nodes structure is a simple structure giving the fact that the oal is just to make the onnection between ros2 and the breach tool. we have node for the simulation and other one for the control. we obtain data using "rosbag" the main issue wih this method was the speed of the simulation . since we are doing a replica to the simulation on matlab that is required to have more than simulation we had to make the simulation fast , and by that we had to have a high frquency for the communication between the nodes, and circumstanes of this is the potential loss of the messsage for control

3.1 work done

creating the enviroment where we have 2 nodes one for the simulation and the other one for the control:

3.1.1 data format

after collecting the data using the rsobag we had the information under file format .db3 we wanted to see if we should directly use the information in the algorithm in matlab but since the training of the neural network is offline it was easier. because we are not obliged to obtain the information directly in matlab, this would be another issue if we supposed to do it online. the solution was that after each simulation we transfer the file format to '.csv' that is readable in matlab

The code file used during this stage was `db3_to_csv.py`.

3.1.2 automatisation

to launch the simulation in ROS2 usualy you use a python script that launch the nodes '`Nodes_launch.py`' and normally to launch this with rosbag ou have a line that is responsible to record but due to a problem with obtaining the data because we wanted to automates the launch of different simulation there was an error to recover the data so i was obliged to manually stop the record by creating another script that will execute the command for the bag record and the simulation launch '`automatisation.py`'

and you have the choice to choose the parametters by including the parameter either in `.yaml` file or directly in the command responsible to execute the launch file. due to several technical issues and after testing lots of options it was the most optimal method to use the parameters in the command line in the terminal.

Before we introduce the ROS2 part to the project. the algorithm that was used to simulate the watertank generates several parameters for the initial state to obtain a variety of traces with valuable information. and ia had to maintain this set of parameters so i had to send those parameters to the simulation that is now in ROS2

3.1.3 downsampling

another matlab script was required to have the data sampled in way that was supposed to have jus the valuable information. in the main algorithm we needed the following informations: `Tf=20 Ts=0.1`

in my ROS2 i tried diiferent time step with different integration step so i had to downsample the information obtained so i can use just the valuable infromations. in the matlab algorithm the integration step was already been used in the ode45 function. so now for example if i had 100 integration over an time steop of 0.1 i had to obtain just the final integration number. adding on

that . the neural network wants not just the actual syaye but also the previous state and the one before. so i had to make sure that we are not taking some information in the middle of the integration.

Chapter 4

Combining PIDs

In this chapter, we explore an innovative approach to controller design by combining two distinct PID controllers with different characteristics to create a more optimal control strategy. This hybrid control scheme serves as the basis for training a neural network to mimic the combined behavior of these controllers, aiming to achieve a balance between speed and precision in the control process.

4.1 Context and Motivation

The primary motivation for this approach is to leverage the strengths of different PID controllers to create a system that performs better than any single PID controller could. In control systems, there is often a trade-off between speed and stability. A fast PID controller may achieve quicker responses but at the cost of overshoot, which can lead to instability in certain systems. On the other hand, a slower PID controller may avoid overshoot and provide more stability but can be less responsive, leading to slower reaction times.

The goal of this approach is to develop a controller that can switch between these two PID controllers, taking advantage of the fast controller's speed when rapid response is required and utilizing the slow controller's stability when precision is needed. By alternating between these two controllers, we aim to create an optimal control strategy that combines the best features of both.

4.2 Implementation Strategy

To implement this approach, the following steps were taken:

1. **Designing Two PID Controllers**:
    - **Fast PID Controller**: This controller is tuned for a quick response, prioritizing speed. However, this comes with the drawback of potential overshoot, which may lead to instability in some situations.

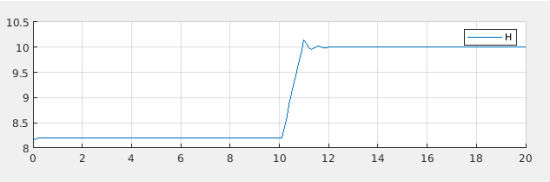

Figure 4.1: Slop of fast PID

- **Slow PID Controller**: This controller is designed to avoid overshoot, focusing on stability and precision. The trade-off here is a slower response time.

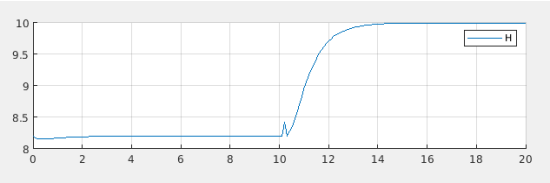

Figure 4.2: Slop of slow PID

2. **Alternating Between Controllers**:

   - A mechanism is devised to switch between the fast and slow PID controllers based on the current state of the system and the desired performance. This switching strategy could be based on criteria such as the magnitude of the error, the rate of change of the error, or specific system conditions that favor one controller over the other.
   - The switching logic is crucial and must be designed to avoid introducing instability or excessive oscillations due to frequent switching.

Algorithm 1 Controller Alternation Algorithm

if $ref_0 > H_0$ **then**
    $threshold_H \leftarrow ref_0 + Oshoot$
    $threshold_L \leftarrow -\infty$
else
    $threshold_L \leftarrow ref_0 - Oshoot$
    $threshold_H \leftarrow \infty$
end if
if Hf exceeds $threshold_H$ **or** Hf is below $threshold_L$ **then**
    $V \leftarrow Vs$
    $HOUT \leftarrow Hs$
else if Close to overshoot boundary then
    if $|Hf - ref| < |Hs - ref|$ **and** Difference is significant **then**
        $V \leftarrow Vf$
        $HOUT \leftarrow Hf$
    else
        $V \leftarrow Vs$
        $HOUT \leftarrow Hs$
    end if
else
    if $|Hf - ref| < |Hs - ref|$ **then**
        $V \leftarrow Vf$
        $HOUT \leftarrow Hf$
    else
        $V \leftarrow Vs$
        $HOUT \leftarrow Hs$
    end if
end if

3. **Training a Neural Network to Imitate the Combined Controller**:
    - With the two PID controllers operating in tandem, the behavior of the combined system is observed and used as the target for training a neural network.
    - The neural network is trained to imitate the combined behavior of the two PID controllers, effectively learning a control strategy that incorporates both speed and precision.
    - The training process involves using data from the combined PID system to adjust the neural network's parameters so that it can replicate the desired control actions.

4.3 Verification Using Signal Temporal Logic (STL)

Once the neural network has been trained to imitate the combined PID controller, the next step is to verify its robustness. Signal Temporal Logic (STL) is

used to formally verify the performance of the neural network-based controller. Specifically, we focus on monitoring overshoot and convergence time, which are critical aspects of control system performance.

4.3.1 STL Formulas for Robustness Verification

To monitor overshoot and convergence time, the following STL formulas were employed:

- **Parameter Definitions**:

$$\begin{aligned} &\mathrm{dt} = 0.001, \\ &T_1 = 6, \\ &T_2 = 4, \\ &\mathrm{ov} = 0.02, \\ &d = 0.01 \end{aligned}$$

- **Stability Formula ($\phi_{\mathbf{stab}}$)**: This formula ensures that if the reference signal changes by more than a certain threshold d, the system's output $H[t]$ will converge to within a small margin of the reference signal $ref[t]$ within a specified time interval. The formula is defined as:

$$\phi_{\text{stab}} := \text{alw}_{[0,10]} \left((|\text{ref}[t + \text{dt}] - \text{ref}[t]| > d) \vee (\text{ref}[t] = 0) \right) \rightarrow \left(\text{ev}_{[0,4]} \left(\text{alw}_{[0,1]} \left(|\text{H}[t] - \text{ref}[t]| < 0.2 \right) \right) \right)$$

  Overshoot Formula ($\phi_{\mathbf{ov}}$): This formula checks that the system does not exceed a specified overshoot limit (ov) relative to the reference signal $ref[t]$ over time. The formula is written as:

$$\begin{aligned} \phi_{\text{ov}} := &\left(\text{ref}[0] - \text{H}[0] \geq 0\right) \rightarrow \text{alw}_{[T_1,T_2]} \left(\text{H}[t] - \text{ref}[t] < \text{ref}[t] \times \text{ov}\right) \\ &\wedge \left(\text{ref}[0] - \text{H}[0] < 0\right) \rightarrow \text{alw}_{[T_1,T_2]} \left(\text{ref}[t] - \text{H}[t] < \text{ref}[t] \times \text{ov}\right) \\ &\wedge \text{alw}_{[0,10-\text{dt}]} \left(\text{ref}[t + \text{dt}] - \text{ref}[t] > d\right) \rightarrow \text{alw}_{[T_1,T_2]} \left(\text{H}[t] - \text{ref}[t] < \text{ref}[t] \times \text{ov}\right) \\ &\wedge \text{alw}_{[0,10-\text{dt}]} \left(\text{ref}[t] - \text{ref}[t + \text{dt}] > d\right) \rightarrow \text{alw}_{[T_1,T_2]} \left(\text{ref}[t] - \text{H}[t] < \text{ref}[t] \times \text{ov}\right) \end{aligned}$$

- **Combined Formula (ϕ)**: The overall robustness of the system is tested by combining the stability and overshoot formulas:

$$\phi := \phi_{\text{stab}} \wedge \phi_{\text{ov}}$$

4.3.2 Verification Process

Using the STL formulas defined above, the robustness of the neural network-based controller was tested. The process involved:

1. **Baseline Verification**: Initially, the real system's robustness was verified using the combined formula ϕ, which served as a baseline for comparison.

2. **Neural Network Verification**: The same formula was then applied to the neural network-based controller to assess whether it meets or exceeds the robustness criteria established by the real system.

3. **Iterative Refinement**: If the neural network failed to satisfy the robustness properties, the training process was revisited, and the network was retrained until it met the desired performance standards.

This verification approach allowed us to rigorously evaluate the neural network's performance in imitating the combined PID controller, ensuring that it could effectively balance speed and stability while avoiding excessive overshoot.

- **Performance Gains**: The alternating controller strategy allows the system to respond quickly when needed, while still maintaining precision and stability in steady-state conditions.

- **Neural Network Effectiveness**: The neural network's ability to learn and replicate this strategy suggests that it can generalize well to similar control problems, offering a flexible solution to balancing competing control objectives.

Future work could explore the application of this approach to other types of controllers or more complex systems, as well as further refinement of the switching logic and STL verification methods.

4.4 Results and Discussion

The combined PID approach, when successfully implemented and mimicked by the neural network, has the potential to significantly enhance the performance of control systems. The neural network, trained to alternate between a fast and a slow PID controller, can achieve a balance between rapid response and stable, overshoot-free control.

Bibliography

[1] A. Donzé, "Breach, A Toolbox for Verification and Parameter Synthesis of Hybrid Systems," in *Proc. 22nd International Conference on Computer Aided Verification*, 2010, pp. 167-170.