REINFORCEMENT LEARNING BASED HYBRID CONTROLLER FOR ROBOTIC MANIPULATOR

N.G.L.S Jayathilake

218527X

Master of Science in Artificial Intelligence

Department of Computational Mathematics

Faculty of Information Technology

University of Moratuwa

Sri Lanka

April 2024

REINFORCEMENT LEARNING BASED HYBRID CONTROLLER FOR ROBOTIC MANIPULATOR

N.G.L.S Jayathilake

218527X

Thesis/Dissertation submitted in partial fulfillment of the requirements for the degree

Master of Science in Artificial Intelligence

Department of Computational Mathematics

Faculty of Information Technology

University of Moratuwa

Sri Lanka

April 2024

Declaration

I declare that this is my own work and this thesis/dissertation does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other University or Institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text. I retain the right to use this content in whole or part in future works (such as articles or books).

27/04/2024

Signature: Date:

The above candidate has carried out research for the PhD/MPhil/Masters thesis/dissertation under my supervision. I confirm that the declaration made above by the student is true and correct.

Name of Supervisor: Dr. A. T. P. Silva

Signature of the Supervisor: Date:28/04/2024

DEDICATION

I wish to dedicate this thesis to my family, whose steadfast support and encouragement have been my guiding force throughout this academic journey. Your sacrifices, understanding, and unwavering love have ignited my ambition and provided me with strength during every obstacle.

This thesis is dedicated to each and every one of you, with heartfelt gratitude and appreciation for your unwavering support, encouragement, and belief in me. Thank you for being a part of this journey and for helping me to realize my dreams.

ACKNOWLEDGEMENT

I am deeply and humbly thankful to my supervisors, Dr. Thushari Silva, Head of the Department of Computational Mathematics, and Dr. Sagara Sumathipala, Senior Lecturer in the Department of Computational Mathematics at the Faculty of Information Technology, University of Moratuwa. Their invaluable guidance, steadfast support, and profound insights have been instrumental throughout my master's thesis journey. With their expertise in robotics and machine learning, their patience, and their mentorship, they have significantly influenced this research and encouraged me to surpass my perceived limitations. Their unwavering commitment to academic excellence and their dedication to fostering intellectual curiosity. I am incredibly thankful for the opportunity to learn under their mentorship.

I extend my sincere appreciation to Dr. Subha Fernando, Senior Lecturer in Computational Mathematics at the Faculty of Information Technology, University of Moratuwa, for her invaluable support, encouragement, and scholarly guidance throughout this academic journey. Dr. Fernando's expertise in the field of Reinforcement learning has enriched my research and provided invaluable insights that have contributed significantly to the development of this thesis. I am deeply grateful for her generosity in sharing her knowledge and expertise, and for her unwavering commitment to fostering a culture of excellence in academia.

I would like to extend my heartfelt thanks to my colleagues at Zebra Technologies and Sigma Connectivity WSI for their support and collaboration throughout this research endeavor. Their insights, feedback, and technical expertise have been invaluable in shaping the direction of this thesis and in overcoming various challenges encountered along the way. I am deeply grateful for the opportunity to collaborate with such talented individuals and for their willingness to share their knowledge and expertise.

Lastly, I extend my heartfelt appreciation to my family for their enduring love, support, and motivation throughout this endeavor. Their unwavering belief in me, words of encouragement, and steadfast support have served as a consistent source of inspiration and drive. I am profoundly grateful for their presence in my life.

ABSTRACT

Reinforcement learning provides robotics with a framework and a set of tools to develop complex and challenging behaviors that are difficult to engineer. Although reinforcement learning showed a significant improvement of applying into the robotics field, we could find only very few industrial applications. The intrinsic challenge of data inefficiency represents a significant obstacle that restricts the application of reinforcement learning algorithms in addressing practical problems in robotics. Looking back at the development of reinforcement learning over the past, it could be identified that applying it alone is not sufficient for real industrial application in robotics. Therefore, the proposed solution is trying to apply reinforcement learning in more innovative way by introducing it to hybrid the existing controller by taking the inspiration from muscle memory concept in human.

This thesis reports on the development of hybrid solution to robotic manipulator with the use of reinforcement learning alone with classical deterministic controller. Here, we utilize both PILCO, a practical and data-efficient model-based policy search method in reinforcement learning, which operates without the need for expert knowledge, alongside a PD controller as the deterministic controller. These components function in parallel to create a hybrid system. This proposed controller switches the control when observability is not sufficient for the deterministic controller to act, and the RL takes control over and executes the learned policy to robot to take its usual maneuvers. The assessment involved conducting an experiment using a basic two-wheeled line-following robot. During the experiment, the robot's camera view was disabled halfway through to activate a hybrid mechanism transitioning into RL mode. Upon executing this test scenario, the RL mode based on PILCO demonstrated promising outcomes compared to other machine learning methods.

Research further emphasizes the importance of hybrid AI systems to cope with any uncertainty or disastrous situation in future mission critical applications of robotics. Discussion also suggesting future enhancements that can add to this concept as further improvements.

Keywords: Reinforcement Learning, PILCO, Robotic manipulator, Hybrid Controller, ROS, Gazebo, MATLAB.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Description
AI	Artificial Intelligence
GP	Gaussian Processes
RL	Reinforcement Learning
PILCO	Probabilistic Inference for Learning Control
ML	Machine Learning
ROS	Robotic Operating System
URDF	Unified Robot Description Format
RMSD	Root Mean Squared Deviation
BFGS	Broyden–Fletcher–Goldfarb–Shanno algorithm
MDP	Markov decision process
PID	Proportional, Integral and Derivative
SVM	Support Vector Machines

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Prolegomena

There are many actuators used in robotics to get required motion and these actuators should be controlled to get precise and accurate motion. When addressing the control challenges of robot manipulators, factors such as actuator dynamics, various system uncertainties, and joint flexibility are thoroughly taken into account. These system dynamics are time variant and non-liner in nature so cannot be per-defined. Presently, these uncertainties are regarded as time-varying, yet the specific variations in their limits over time remain unknown, adaptive control faces a challenge due to its dynamic nature, making it impractical even with expertise. Because of this nature it is possible to introduce reinforcement learning (RL) to the control of robot manipulators [2], and RL is simply active learning from experience. More importantly, it is not possible to align other ML techniques because they need more data, but with robots it is not always possible collect data since it usually involves excessive wearing of mechanical. Reinforcement learning (RL) has garnered considerable interest in recent years due to its compelling outcomes across various fields. This paper [3] has shown a successful proof of concept of using PILCO (probabilistic inference for learning control) to control robot manipulators using RL. When it is required to quantify these uncertainties of system the best approach that has taken by above approach is to employ probabilistic dynamics model into the reinforcement learning. Even PILCO also needs few trials to learn reasonably good policy, this alone will not solve real industrial applications.

This thesis presents a hybrid controller which contains both classical deterministic controller and the RL based controller in parallel so that robots can act in any unpredictable emergency situations. This solution for robotics is inspired by humans' muscle memory that is one can accomplish the task without consciously thinking about it or act on the environment without directly observing it. In an event of lacking observability of the environment the classical feedback controller is suspended, and RL controller would come forward and act, or if controller dynamics changed then also RL agent will learn new environment and act upon that. Further this thesis describes the complete research project of reinforce learning based hybrid controller for robotic manipulator including background and motivation, technology used, approach, design, implementation, evaluation, and conclusion.

1.2 Objectives.

- Critical review of reinforcement learning in robot control systems
- In depth study of practical applications of modern reinforcement learning
- Design and implement a hybrid PILCO based controller for a robot manipulator in simulated environment.
- Evaluate the system.

1.3 Background and Motivation

While artificial intelligence (AI) has made considerable improvements, particularly in domains like robotics [5], such as image classification and natural language processing, the capabilities of our brains still surpass those of AI in many everyday tasks. For instance, while industrial robots may execute motions faster than human arms, they rely on pre-programmed trajectories and struggle in unfamiliar scenarios. Unlike AI, our brains can adapt and learn from experience, as seen in reinforcement learning, allowing us to excel in unpredictable environments.

RL is mainly learning through experience, and it has more inspirational influence from living things compared to its data driven ML counterpart [4]. Robotics in particular has gained lot of inspirational aspects from human but still not successfully deployed because of practical difficulties of training and data inefficient nature [3].

Although RL has gained a lot of advancement using deep learning techniques it is not so appropriate to robotic controllers since dependency on lot of training and test data. Therefore, when starting from scratch, it necessitates employing a probabilistic dynamics model to articulate the uncertainty inherent in the model. Thus, inherent property data inefficiency in RL [3] can be avoided if we utilize non-parametric probabilistic Gaussian processes (GPs) [6].

1.4 Problem in Brief

From the literature, it is possible to identify there are some deficiencies in connecting practical applications of RL in robotics. But there are many disaster recovery situations where RL can play a crucial role. Here I define the research problem as lack of practical applications of RL in robot manipulation as the rout cause for modern robots still far away from human.

1.5 Proposed solution

Proposal has PILCO based RL system can be deployed in parallel to the standard preprogrammed controller of robotic manipulator to act upon disastrous situation where existing controller could not perceive its environment to act with the lack of observability in completely different environment to that of existed. The environment states and actions were continuously monitored and if it changes the RL based controller immediately activated and classical controller will be suspended.

1.6 Resource requirements

1. Robotic simulator framework.
2. MATLAB software.
3. Computer running Ubuntu 22.04.
4. Software tools to develop ROS networks.

1.7 Structure of the Thesis

The thesis is organized into 8 chapters. The consequent chapters will describe the literature review, technology adopted, approach, design, implementation, evaluation, conclusion, and further work.

In Chapter 1: Introduction; the work of the research is contextualized, which also provides a brief description about the research background and its motives.

In Chapter 2: Literature review; the background information, relevant past works, which has been used for the study of the research problem is described. The past studies and research done on the intended methodologies are also covered in this chapter.

In Chapter 3: Technologies adopted; to develop the solution explained in detailed manner.

In Chapter 4: Reinforcement learning based hybrid controller for robotic manipulator Approach; the implemented solution with their modules is presented.

In Chapter 5: Design; Design and architecture of the technologies utilized to develop the system.

In Chapter 6: Implementation; Detailed instructions are provided for implementing the simulated robotic manipulator.

In Chapter7: Evaluation; A comprehensive overview of the evaluation process employed for the developed solution is described.

In Chapter 8: Work and Conclusion, highlights the research findings, explain various aspects of the work, and makes suggestions for potential future studies are explained.

1.8 Summary

In this chapter, we offer an outline of the study, concentrating on developing a hypothesis for deploying an RL-based controller alongside a deterministic controller to enhance the reliability of industrial robotic manipulators crucial for mission-critical tasks. Furthermore, the chapter described the motives governing the research on applicability of Gaussian process modelling in reinforcement learning for robotic maneuvers.

Secondly, the problem definition and solution to it is highlighted in brief with required resources to develop. Lastly, the flow of the entire thesis is mentioned.

CHAPTER 2

DEVELOPMENT OF REINFORCEMENT LEARNING IN ROBOTICS

2.1 Introduction

In Chapter 1, we have presented the overall picture of this thesis covering the research problem and the essentials of the solution. This chapter gives a review of literature in the area of Reinforcement Learning (RL) in robotics. The literature review has been structured to cover the gestation of RL in robotics, their developments, and future directions. Here we have also summarized the research challenges in RL in robotics and defined our research problem.

2.2 Major developments in reinforcement learning in robotics

Reinforcement Learning (RL) shares strong ties with classical optimal control theory, dynamic and stochastic programming and stochastic search [8]. RL, like optimal control, addresses the challenge of determining a best policy (referred to as the control policy the give problem) that maximizes an objective function (such as cumulative cost or reward). These two approaches depend on the concept of a system defined by a fundamental state, controls, and a plant or model that delineates transitions between states. However, optimal control presumes complete knowledge of the system description through a model (e.g., a function T, representing the subsequent state of the robot based on the current state and action). Despite providing robust assurances for such models, optimal control frequently encounters challenges stemming from model and computational approximations, leading to breakdowns in these guarantees. Conversely, reinforcement learning engages directly with observed data and rewards acquired through interactions with the environment. The research in reinforcement learning has heavily emphasized tackling scenarios that pose analytical challenges through the utilization of approximations and data-centric methodologies. A key strategy in reinforcement learning within robotics revolves around leveraging classical optimal control methods for system models acquired through iterative engagement with the environment [7].

Robot manipulation tasks typically encompass a multitude of actions feasible at any given state. Significantly, proficient human operators often exhibit considerable expertise in selecting the best actions to a subjected state of the robot and can exemplify this appropriate behavior. It's often challenging for such a proficient individual to articulate their approach; nevertheless, their decision-making process typically involves a non-linear amalgamation of various considerations, including stability, energy conservation, actuator constraints, and future objectives. The operator

finds it far simpler to showcase optimal actions than to meticulously list the intricate function being optimized to generate the action [9].

RL is mainly learning through experience, and it has more inspirational influence from living things compared to its data driven ML counterpart [4]. Robotics in particular has gained lot of inspirational aspects from human but still not successfully deployed because of practical difficulties of training and data inefficient nature [3].

Applying RL to robotics presents unique challenges due to the multi-dimensional state and action spaces, as well as safety concerns and hardware limitations. Traditional RL algorithms often struggle with sample efficiency, making it challenging to train robots in the real world where data collection can be expensive or time-consuming. Deep RL algorithms, particularly those based on deep neural networks, have shown promise in learning complex control policies directly from raw sensory inputs. Techniques such as Actor-Critic, Policy Gradient methods, and Deep Q-Networks architectures have been successfully adapted to robotics tasks [15].

2.3 Introducing probabilistic models for reinforcement learning in robotics.

Generally, methods reliant on models, which entail learning the model of dynamics of given environment, offer greater potential for deriving valuable insights from existing data compared to model-free approaches such as Q or TD-learning. A reason for the limited uptake of model-based methods in learning from scratch is their vulnerability to model bias. This bias arises due to inherent assumption which presume accurate reflection of the real environment in the learned dynamic parameters of the model. Model bias becomes especially problematic when there are only limited samples available and lack of comprehensive prior knowledge regarding the target being tackled.

Hence, in commence learning from scratch, it is essential to employ a probabilistic dynamics model that exhibits model uncertainty. For this purpose, researchers used non-parametric Gaussian processes (GPs). Subsequently, considering model uncertainties in policy evaluation becomes crucial. By using deterministic inference methods for policy evaluation, we can pursue policy search using analytic policy gradients, eliminating the need for an explicit value function model. Building on these concepts, researchers suggested a model-based approach known as probabilistic inference for learning control [3] (PILCO). A main uses of PILCO thus includes uses such as physical systems such as robots deserve data efficiency in continuous state-action domains. It is acknowledged as one of the fastest model-based RL methods, offering a data-efficient framework grounded in Bayesian inference using Gaussian processes.

However, PILCO might still necessitate numerous online-learning iterations in practical applications due to the imprecise policy search in the initial phases. Since the

initial policy is restricted to random actions, even minor or inconsequential updates during policy search can potentially lead to control failure in the initial iterations. There is few significant research to bridge this gap to adapt RL into practical applications in robotics through PILCO.

PID control structures serve as the primary control mechanism not only in industrial settings but also in automotive and robotics applications, particularly for low-level control tasks. These controllers need tedious manual tuning or heuristic tuning with rules. It has been shown that integrating PILCO is successful in tuning PID controllers in robotics [12].

In practical application, PILCO alone necessitates numerous online-learning iterations due to the imprecise policy search in the initial phases. Better approach of RL in control system can be seen when PILCO is used with linear combination with PID controller [13].

2.4 Summary of challenges in reinforcement learning in robotics

Although RL has gained lot of advancement using deep learning techniques it is not so appropriate to robotics since taking lot of test data is not feasible with mechanical. Probabilistic dynamics model to infer model uncertainty and employing non-parametric probabilistic GPs are crucial in addressing the data inefficiency of RL.

However, PILCO also has major drawback when it comes to computation complexity of GPs is O(n3) due to the need to invert an $n \times n$ matrix. Therefore, to run PILCO requires high end microprocessors.

2.5 Summary

This chapter presented our literature around RL in robotics by covering 15 research papers with very high citation records. Here we summarized the challenges in the field and defined the research problem as lack of practical application of RL in robotics remains unsolved. This chapter also identified probabilistic dynamics model RL as a potential technology for addressing practical applications into industrial robots. Chapter 3 presents our in-depth study of Gaussian process based probabilistic model-based RL techniques by highlighting their relevance to robotics.

CHAPTER 3

TECHNOLOGY ADAPTED

3.1 Introduction

Chapter 3 presents the employed technologies and software tools used in this thesis. Specifically, software frameworks, software tools, software libraries and programming languages were used to build and evaluate this project. They have been selected based on their effectiveness and timeliness. Chapter tries to briefly but meaningfully outline all the technologies used in this project.

3.2 Robotic operating system (ROS)

The Robot Operating System (ROS) is software platform used in robotics research and development. Originally developed by Willow Garage and Stanford University, ROS has grown into a widely adopted framework for building complex robotic systems.

At its core, ROS delivers tools and libraries which facilitate robotic software development. It offers a distributed computing architecture, enabling seamless communication between different components of a robotic system, such as sensors, actuators, controllers, and high-level algorithms. This communication is achieved through a publish-subscribe messaging system, allowing nodes (individual software modules) to exchange data and commands efficiently.

ROS is modular and flexible which facilitates developers to integrate various software components and hardware devices into their robotic systems seamlessly. It is compatible with a various programming language, such as C++ and Python, so that it is accessible to a diverse community of researchers, engineers, and hobbyists.

Main feature of ROS is intense set of packages, which provides library wise software modules for common robotics tasks such as localization, mapping, navigation, perception, manipulation, and more. These packages, contributed by the ROS community, allow developers to leverage existing solutions and focus on higher-level aspects of their projects.

ROS also includes powerful tools for visualization, debugging, and simulation, such as RViz for 3D visualization, rqt for graphical user interface development, and Gazebo for robot simulation. These tools enable developers to test and validate their algorithms in simulated environments before deploying them on real robots.

Moreover, ROS inculcates networking and dissemination of knowledge among platform within robotics experts through its vibrant ecosystem of users, developers, and researchers. The ROS community actively contributes to the development of new

packages, provides support through forums and mailing lists, and organizes events such as conferences and hackathons.

In summary, ROS is a comprehensive platform for designing robotic systems, offering an array of features, tools plus resources to supporting the improvement and deployment of complex robotics applications. Its open-source structure, modular architecture cum vibrant community supports indispensable tool for robotics research, education, and industry.

ROS2, the next generation of ROS, has been used in this thesis to build the entire robot and its simulation environment.

3.3 Gazebo robotic simulation application

Gazebo is an open-source robotics simulation framework used in research, particularly in robotics and AI. It employs a framework for simulating complex environments, robots, and sensor systems, enabling researchers to test and validate algorithms in a virtual environment before deploying them on real hardware. Moreover, it can be seamlessly coupled with ROS to simulate and test whole robotic systems.

Gazebo offers a 3D simulation environment where users can design and simulate various robotic systems, including manipulators, mobile robots, drones, and even entire robot swarms. It provides realistic physics simulation, allowing researchers to study robot interactions with their environment accurately. It is easily customizable and able to extend, permitting users to create custom robot models, environments, sensors, and controllers. It helps various languages such as C/C++, Python, and XML, allowing developers with various backgrounds.

3.4 MATLAB

MATLAB is a popular end programming language and interactive environment utilized in both academic and industrial settings for numerical computation, data analysis, and algorithm development. It provides a rich set of built-in mathematical functions and libraries for performing numerical computations. It supports matrix and vector operations, linear algebra, optimization, interpolation, integration, and differential equations solving. Researchers can implement complex mathematical algorithms efficiently using MATLAB's intuitive syntax and extensive function library.

At its core, MATLAB is renowned for its powerful matrix manipulation capabilities, allowing users to perform complex mathematical operations and computations efficiently. It offers a vast library of built-in functions and toolboxes for linear algebra, signal processing, image processing, control systems, machine learning, and more, making it a versatile tool for researchers, engineers, scientists, and educators alike.

One of MATLAB's key strengths lies in its ease of use and interactive nature. Its intuitive syntax, coupled with a rich set of documentation and examples, enables users to quickly prototype and implement algorithms, analyze data, and visualize results without requiring extensive programming knowledge. Additionally, MATLAB's integrated development environment (IDE) provides a unified platform for writing, debugging, and testing code, enhancing productivity and efficiency.

MATLAB also offers comprehensive plotting and visualization capabilities, allowing users to create high-quality graphs, charts, and interactive plots to represent data and communicate findings effectively. Its extensive library of plotting functions, combined with customizable graphics options, enables users to create publication-quality visualizations for presentations, reports, and publications.

Furthermore, MATLAB supports interoperability with other programming languages and software tools, facilitating integration with external libraries, data formats, and hardware devices. It provides seamless integration with languages such as C/C++, Java, Python, and Fortran, as well as with external software packages like Microsoft Excel, Simulink, and various CAD tools.

In addition to its desktop version, MATLAB offers specialized editions such as MATLAB Online and MATLAB Mobile, providing users with flexibility in accessing MATLAB's capabilities from different devices and platforms. Moreover, MATLAB's extensive community support, including forums, user groups, and online resources, enables users to seek assistance, share knowledge, and collaborate on projects.

In summary, MATLAB is a comprehensive and versatile platform for numerical computing and technical computing tasks, offering a wide range of tools, functionalities, and resources to support research, development, and innovation in diverse fields such as engineering, science, finance, and beyond. Its ease of use, interactive environment, extensive libraries, and interoperability make it a popular choice for professionals and academics worldwide.

3.5 PILCO software framework

The original implementation of PILCO was done in MATLAB and the project used the latest version that is 0.9 of the same. This has a well-structured implementation and has very impressive documentation to anybody to follow and extend.

This package serves the PILCO framework for RL, focusing on data efficiency. It encompasses the PILCO framework applied to various examples featuring continuous states space, including cart-pole swing-up, double-pendulum, unicycling etc.

3.5 Programming Languages

Python served as the high-level scripting language for developing projects because of its versatility, user-friendly nature, and robust support for third-party libraries. Python gives a huge set of libraries for data processing, analysis, and visualization, which were utilized in developing the system. Among all these libraries numpy, matplotlib, pandas, sklearn, opencv can be highlighted. Additionally, the large developer community is the main plus point of the language.

In addition to the python MATLAB has been used to modify the PILCO algorithm and some evaluation programs. MATLAB is a sophisticated programming language and interactive platform tailored for numerical computation, data analysis, and visualization tasks. It features intuitive syntax, powerful matrix operations, extensive visualization capabilities, and a wide range of built-in functions.

3.5.1 OpenCV Library

OpenCV, short for Open-Source Computer Vision Library, stands as a potent and extensively employed open-source computer vision toolkit. It gives a comprehensive set of tools and functions for image processing, computer vision, and machine learning. In Python, OpenCV is available as a Python package, providing easy-to-use interfaces for developers to leverage its capabilities.

OpenCV in Python include a various task, including reading and writing images and videos, processing images, performing geometric transformations, detecting objects and faces, recognizing patterns, and more. Its rich functionality makes it suitable for various applications, such as robotics, augmented reality, facial recognition, object tracking, and medical image analysis.

3.6 Scikit-learn.

Scikit-learn, often abbreviated as sklearn, is a versatile and user-friendly machine learning library for Python. It gives various algorithms for multiple machine learning tasks which includes classification, regression, clustering, dimensionality reduction, and model selection. Scikit-learn is developed based on other known computing libraries in Python, making it seamlessly integrate with the Python scientific computing ecosystem.

Scikit-learn is mainly utilized in the education sector and industry for multiple machine learning systems, including but not limited to predictive modeling, pattern recognition, natural language processing, computer vision, and recommender systems. Its active development community and regular updates ensure that it stays up to date with AI research and technology.

3.7 Unified Robot Description Format

Unified Robot Description Format (URDF) is an XML-based formatted file used in robotics to describe the structure, kinematics, and dynamics of robots. It provides a standardized and platform-independent way to represent the physical properties and geometry of robotic systems.

In URDF files, robot components such as links, joints, sensors, and other elements are defined using XML syntax. This includes specifying the visual and collision properties of each link, defining the joint types and their parameters, specifying the robot's inertial properties, and incorporating sensor information.

URDF files play a crucial role in robot modeling, simulation, and control. They are commonly used in robot simulation environments such as ROS (Robot Operating System) to simulate robot behavior accurately and efficiently. URDF files are also utilized in robot motion planning, visualization, and virtual prototyping.

Overall, URDF serves as a standardized representation format for describing robotic systems, facilitating interoperability between different robotic platforms and simulation environments, and enabling efficient development, analysis, and deployment of robotic applications.

3.8 Summary

This chapter provides a discussion of multiple technologies that were adapted for this research project. We started by introducing ROS which is used to build the robot and its environment of this project, then the gazebo yet another open-source framework for robotics were discussed. Secondly, we brief about MATLAB, one of the best scientific programming languages. Later we explained about PILCO software framework and its capabilities. Finally, we briefly mentioned the programing languages and URDF which were used in the project with libraries which were used in this project. In essence, the choice of suitable technologies was pivotal in crafting the reinforcement learning-based hybrid controller.

CHAPTER 4

AN APPROACH TO REINFORCEMENT LEARNING BASED HYBRID CONTROLLER FOR ROBOTIC MANIPULATOR

4.1 Introduction

Chapter 3 justified the suitability of probabilistic model-based RL techniques for enhancing industrial robot manipulators. This chapter presents our RL based approach to building hybrid robotic controller by covering our hypothesis, input, output, process, features, and users. Among others, the process specifically identifies the task within hybrid controller which provides insight into design of the solution.

4.2 Hypothesis

Proposal has hypothesized that PILCO based RL system to be deployed in parallel to the standard preprogrammed controller through a ROS network. PID controller-based ROS nodes are mainly doing the job, while the environment is continuously monitored with cameras or any other localization technique. If the robot deviates from the desired path the RL controller is taking over the control and the best learned policy will be applied to the desired goal. This application also refrains from the fatal paths that could happen in pre-learning of RL. Instead, the devised approach allows the deterministic controller and the RL policy to synergize, continually enhancing each other through ongoing learning.

4.3 Input

Input to the hybrid controller,

1. State of the robot through a camera or by localization technique

2. Feedback from different sensors, tachometers, position sensors, limit switches and accelerometers/gyros

4.4 Output

Outputs

1. Optimized policy for the robot manipulator.

4.5 Process

Overall process includes PILCO based learned optimized policy to be run in a separate ROS node parallel with the deterministic controller and the environment is continuously monitored. Another node monitors the accuracy of the robot and if significant change of environment is identified, then RL based controller takes over the control and pre-programmed controller will be suspended.

The PILCO controller algorithm is shown in figure 4.1 and the whole algorithm is modular wise shown in figure 4.2.

Algorithm 1 PILCO

1: **init:** Set controller parameters ψ to random.
2: Apply random control signals and record data.
3: **repeat**
4: Learn probabilistic GP dynamics model using all data
5: **repeat** ▷ Model-based policy search
6: Approx. inference for policy evaluation: get $J^\pi(\psi)$
7: Gradients $\mathrm{d}J^\pi(\psi)/\mathrm{d}\psi$ for policy improvement
8: Update parameters ψ (e.g., CG or L-BFGS).
9: **until** convergence; **return** ψ^*
10: Set $\pi^* \leftarrow \pi(\psi^*)$.
11: Apply π^* to robot (single trial/episode); record data.
12: **until** task learned

Figure 4.1: PILCO algorithm

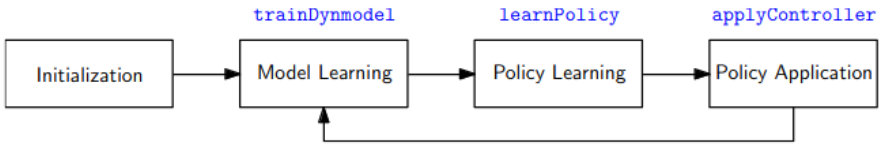


Figure 4.2: Policy Search

The overall system containing both classical controller and the RL controller is shown in figure 4.3. Hybrid controller takes input from camera or indoor localization technique to detect any significant changes in the environment.

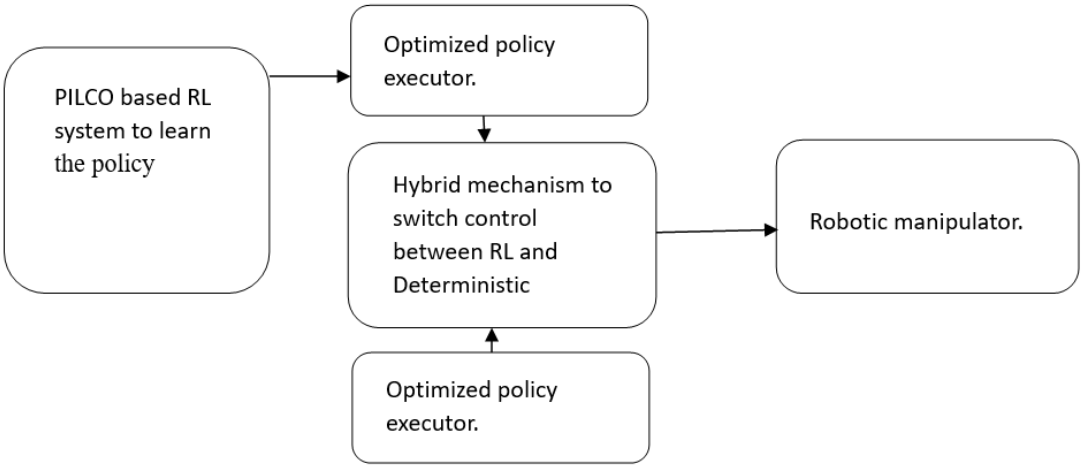


Figure 4.3: High level picture of Hybrid control system

Actions of RL controller is then fed to the motor control unit to drive manipulator and rewards are taken based on camera inputs. Classical controller on the other hand takes the feedback to the controller again from depth camera.

4.6 Features

Major features of the system are,

1. Any disastrous lack of observability can be mitigated by the RL base controller.
2. Robotic manipulators will be more autonomous and fail proof.

4.7 Users

Users of the new hybrid controller are industrial robot manipulators which are vulnerable to huge losses due to disastrous situations.

4.8 Summary

This chapter explains our approach to hybrid controller based on probabilistic models with RL. We have defined the signals to be input into the system. More importantly, we have identified main modules that need to be coordinated themselves to accommodate RL to act in disastrous situations. The next chapter describes the design of hybrid controller with reference to the process identified here.

CHAPTER 5

DESIGN

5.1 Introduction

Implementation of hypothesis is designed using a simulated robot using gazebo and to make it simple the robot was given line following task to simulate the hypothetical scenario to test the reinforcement learning based hybrid controller.

Initially the robot was driven in a deterministic way and once the PILCO was learned the camera view was switched off to get the PILCO node to take over control of robot.

Then the whole paths are taken by the original control and the PILCO control to get evaluation data.

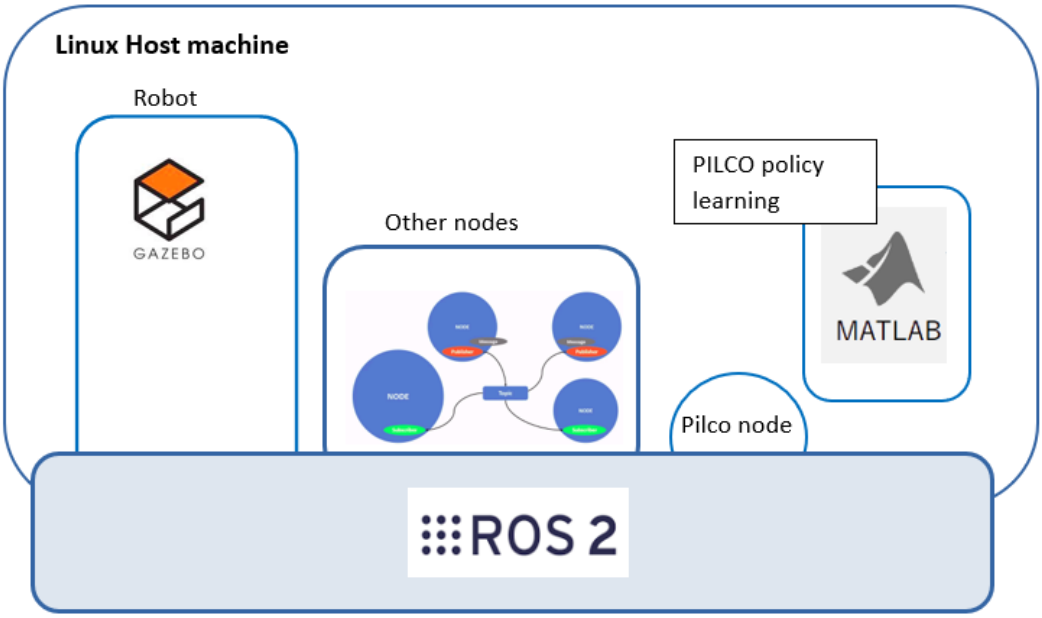


Fig 5.1: High level design

As very high-level figure 5.1 shows the main building blocks of the design. The proposed robot is built on gazebo simulation environment with the control is governed by ROS nodes and PILCO node which is responsible of running optimized policy learned by PILCO algorithm which is implemented using MATLAB. Each module in the figure will be described in rest of this chapter.

5.2 Robot and its environment.

The two-wheel robot with caster wheel was designed with URDF with integrated a camera to view the front and a differential drive to drive through the environment as shown in figure 5.2. The collision and physical properties were described according to the carefully calculated 3x3 rotational inertia and geometrical values.

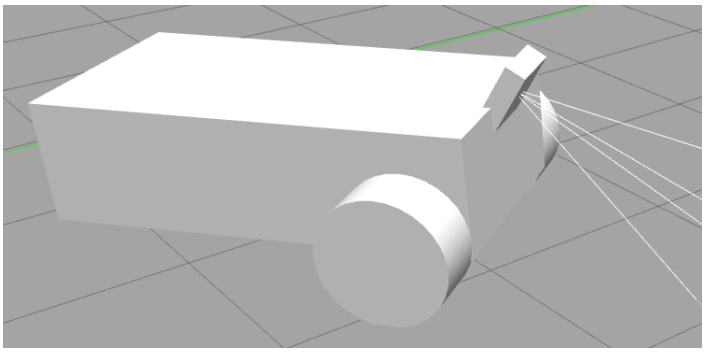


Figure 5.2: Two Wheel robot in gazebo

Environment for the robot to travel along the line was also implemented with URDF and shown in following Figure 5.3.

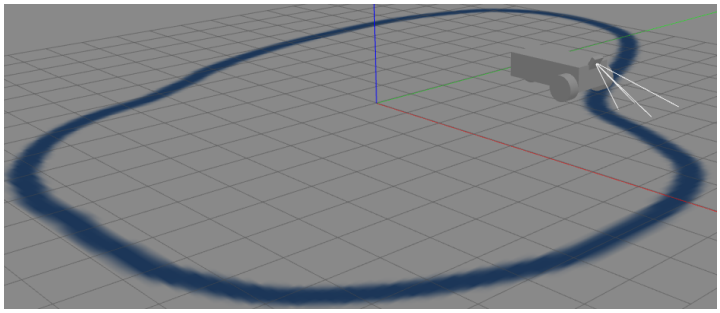


Figure 5.3: Pre-Designed Path for the robot to travel.

5.3 ROS nodes for controlling the robot.

ROS network for controlling the robot along the pre-defined line was designed as following figure 5.4. Mainly the follower node is responsible for planning the trajectory along the line using raw camera images from the front camera. On the contrary, the observer node monitors the robot's state and sensor inputs to ascertain if their values adhere to the same sample distribution. If any of the input deviate from the desired range the observer node decreases the confidence level for the deterministic controller and once it hit the threshold level the follower node hand over the control to PILCO node and vise-versa. So, this way hybrid mechanism will work depending on the confidence level for each control path.

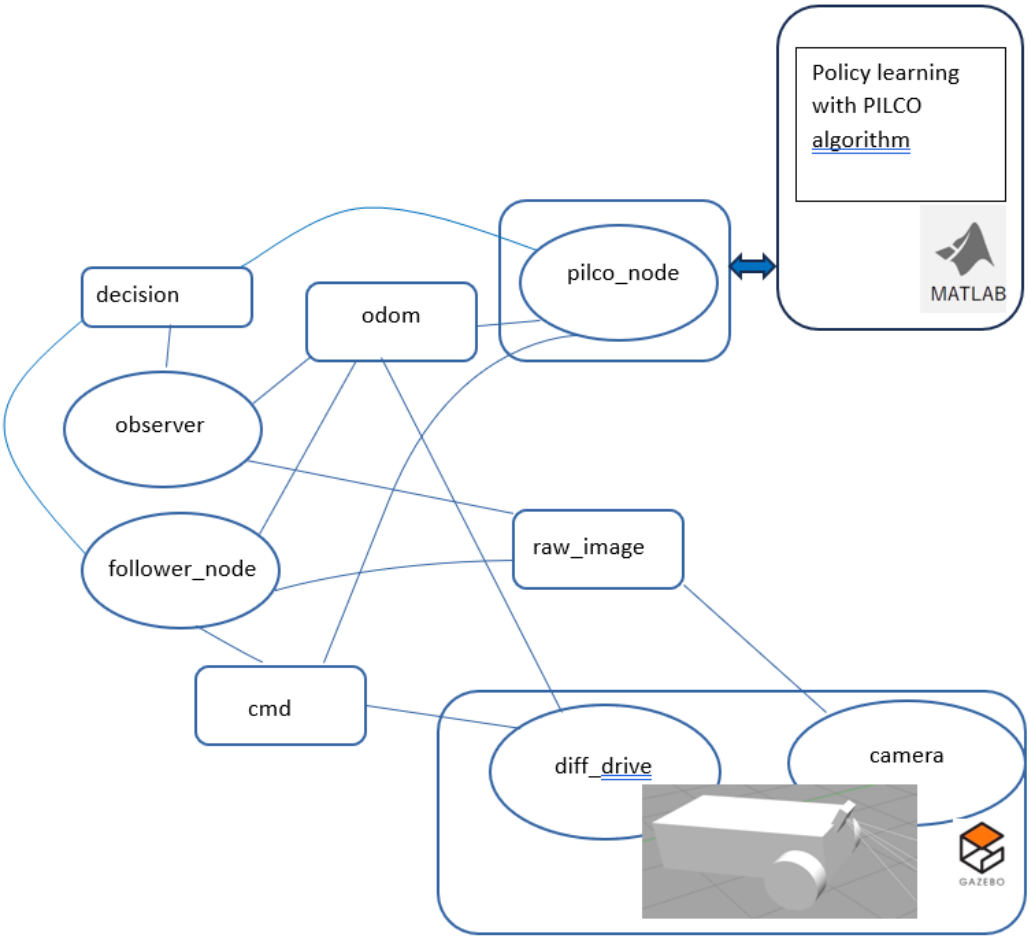


Figure 5.4: High level ROS design

5.4 PILCO node

Though the original PILCO was built to learn the policy from scratch, in this thesis the PILCO was trained with the correct line following trajectories and only the controlling with the learned policy was built into the PILCO node.

As shown in figure 5.5 the training part is done offline to limit the amount of development time of the project and a new data logging ROS node was introduced to get data as csv files for training.

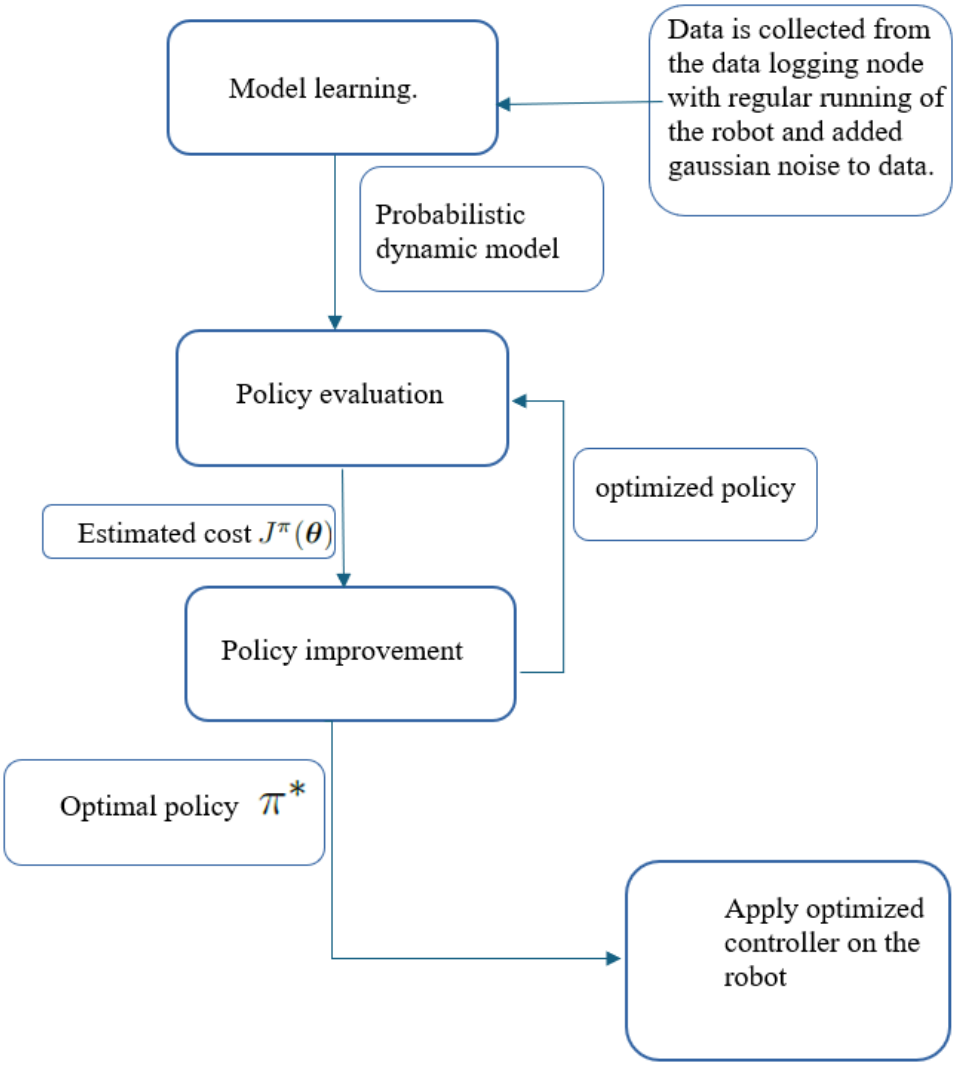


Figure 5.5: Modified PILCO flow diagram

We primarily address discrete control issues involving continuous states $\mathbf{x} \in \mathbb{R}^D$ and external control signals referred to as actions, $\mathbf{u} \in \mathbb{R}^F$. The dynamics of the system are outlined through a Markov decision process (MDP), which serves as a

computational model for decision-making amidst uncertain conditions. This framework comprises four components: the state space, the control space (or action space), the one-step transition function denoted as *f*, and an immediate cost function, *c(x)*, assessing the desirability of a given state *x*. The transition dynamics are elaborated by equation 5.1.

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) \tag{5.1}$$

We assume that the immediate cost function acts as the major role in the design. In reinforcement learning, the aim is to discover a policy that minimizes the anticipated long-term cost.

$$V^{\pi}(\mathbf{x}_0) = \mathbb{E}_{\boldsymbol{\tau}}\left[\sum_{t=0}^{T} c(\mathbf{x}_t)\right] = \sum_{t=0}^{T} \mathbb{E}_{\mathbf{x}_t}\left[c(\mathbf{x}_t)\right] \tag{5.2}$$

The function V is referred to as the value function, and V (x_0) denoting the value of state x_0 under policy π.

In this context, Quasi-Newton techniques are employed for optimization, employing gradient-based methods such as BFGS to enhance the policy. The gradients of the expected long-term cost, J, in relation to the policy parameters are calculated using analytical methods. The choice of the cost function aligns with Equation 5.3.

$$c(x) = 1 - \exp\left(-\frac{1}{2a^2}\left[d(x, x_{target})^2 + (w \cdot error)^2\right]\right) \tag{5.3}$$

d represents the Mahalanobis distance between present state to target state, error is measured in center of image to the centroid of largest contour in the camera image. Other parameters like a and *w* are weights for the tuning purposes.

$$J^{\pi} = \sum_{t=1}^{T} \mathbb{E}[c(\boldsymbol{x}_t)|\pi], \qquad p(\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) \tag{5.4}$$

For the state x of the robot, linear position, angular orientation, linear velocity and angular velocity are selected as shown below.

$$[x_1, x_2, \dot{x_1}, \dot{x_2}, \dot{\Theta}, \Theta]$$

and cost function is selected as follows from Euclidean distance to the target and the error.

5.4.1 Modified PILCO learning

The original PILCO algorithm written on MATLAB is modified to learn offline and then the learned optimized policy has been used by *pilco_node.*

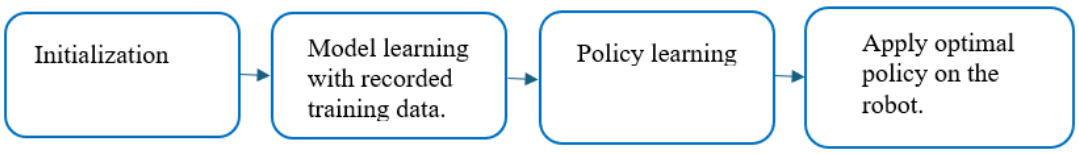


Figure 5.6: Module wise flow of modified algorithm

Following initialization, depicted in figure 5.6, the subsequent module is tasked with training a Gaussian Process (GP) using recorded accurate trajectory data. The second module focuses on policy learning, encompassing both policy evaluation and enhancement. The third module operates within a ROS node, managing the implementation of the learned policy on the robot. Data gathered from the robot at regular intervals is used for model updates, thus restarting the cycle.

5.5 Summary

The design of the RL based hybrid controller is discussed in this chapter. Since PILCO is data efficient learning only few iterations of training were expected to learn an optimized policy, the system endeavors to enhance the accuracy and overall performance of robot trajectories when the observations from the regular sensors were uncertain. This approach takes advantage of the Gaussian process model learning. The subsequent sections will describe in detail the implementation details, evaluation, and results of the system to provide extra safety for critical applications of robotics.

CHAPTER 6

IMPLEMENTATION

6.1 Introduction

In this section, we provide an in-depth explanation of the RL implementation-based hybrid controller for two-wheel line follower is provided. Including setting up the Ubuntu host machine, building the two-wheel robot with gazebo, implementation of ros2 nodes network and PILCO algorithm development is clearly explained. Furthermore, some utilities implemented using Python and MATLAB for data collection and evaluation are described. For implementing the whole scenario, a simple two-wheel robot was built in gazebo to simulate and test the robot trajectory.

6.2 Setting up the Linux host machine.

Since almost all the software tools are open source, quite a lot of work was needed to set up the infrastructure to build the project. All the installation stuff and setting up can be found in the appendix.

6.3 Building the robot in gazebo.

Main building blocks of the two-wheel robot contains chassis, left wheel, right wheel, caster wheel, camera and differential drive. All these components are listed in URDF in world file *gazebo_ros_2wbot.world*

6.3.1 Chassis

The following figure 6.1 *shows the chassis implemented in URDF under the link tag as a simple rectangular box with inertia calculated in each axis.*

```
<link name='chassis'>
  <pose>-0.151427 -0 0.175 0 -0 0</pose>
  <inertial>
    <mass>1.14395</mass>
    <inertia>
      <ixx>0.126164</ixx>
      <ixy>0</ixy>
      <ixz>0</ixz>
      <iyy>0.416519</iyy>
      <iyz>0</iyz>
      <izz>0.481014</izz>
    </inertia>
  </inertial>
  <visual name='visual'>
    <geometry>
      <box>
        <size>2.01142 1 0.568726</size>
      </box>
    </geometry>
  </visual>
  <collision name='collision'>
    <geometry>
      <box>
        <size>2.01142 1 0.568726</size>
      </box>
    </geometry>
  </collision>
</link>
```

Figure 6.1: Implementation of Chassis

6.3.2 Left and Right Wheels

```
<link name='left_wheel'>
  <pose>0.554283 0.625029 -0.025 -1.5707 0 0</pose>
  <inertial>
    <mass>2</mass>
    <inertia>
      <ixx>0.145833</ixx>
      <ixy>0</ixy>
      <ixz>0</ixz>
      <iyy>0.145833</iyy>
      <iyz>0</iyz>
      <izz>0.125</izz>
    </inertia>
  </inertial>
  <visual name='visual'>
    <geometry>
      <cylinder>
        <radius>0.3</radius>
        <length>0.284363</length>
      </cylinder>
    </geometry>
  </visual>
  <collision name='collision'>
    <geometry>
      <cylinder>
        <radius>0.3</radius>
        <length>0.284363</length>
      </cylinder>
    </geometry>
    <surface>
      <friction>
        <ode>
          <mu>1</mu>
          <mu2>1</mu2>
          <slip1>0</slip1>
          <slip2>0</slip2>
        </ode>
      </friction>
      <contact>
        <ode>
          <soft_cfm>0</soft_cfm>
          <soft_erp>0.2</soft_erp>
          <kp>1e+13</kp>
          <kd>1</kd>
          <max_vel>0.01</max_vel>
          <min_depth>0.01</min_depth>
        </ode>
      </contact>
    </surface>
  </collision>
</link>
```

Figure 6.2: Implementation of two wheels

Figure 6.2 shows the left wheel link with correct inertia calculated on each axis and other parameters like visual, collision respectively. The right wheel is also identical with only the position value changes.

6.3.2 Castor wheel

To make it simple, a castor wheel is selected as a ball as shown in figure 6.3 and its only purpose is to balance the cart.

```
151     <link name='caster'>
152       <pose>-0.957138 -0 -0.125 0 -0 0</pose>
153       <inertial>
154         <mass>1</mass>
155         <inertia>
156           <ixx>0.1</ixx>
157           <ixy>0</ixy>
158           <ixz>0</ixz>
159           <iyy>0.1</iyy>
160           <iyz>0</iyz>
161           <izz>0.1</izz>
162         </inertia>
163       </inertial>
164       <visual name='visual'>
165         <geometry>
166           <sphere>
167             <radius>0.2</radius>
168           </sphere>
169         </geometry>
170       </visual>
171       <collision name='collision'>
172         <geometry>
173           <sphere>
174             <radius>0.2</radius>
175           </sphere>
176         </geometry>
177       </collision>
178     </link>
```

Figure 6.3: Castor wheel

6.3.3 Joints connecting all the above

Both the left wheel and right wheel are connected to chassis with these two joints as shown in 6.4.

```
<joint name='left_wheel_joint' type='revolute'>
  <parent>chassis</parent>
  <child>left_wheel</child>
  <axis>
    <xyz>0 0 1</xyz>
    <limit>
      <lower>-1.79769e+308</lower>
      <upper>1.79769e+308</upper>
    </limit>
  </axis>
</joint>

<joint name='right_wheel_joint' type='revolute'>
  <parent>chassis</parent>
  <child>right_wheel</child>
  <axis>
    <xyz>0 0 1</xyz>
    <limit>
      <lower>-1.79769e+308</lower>
      <upper>1.79769e+308</upper>
    </limit>
  </axis>
</joint>

<joint name='caster_wheel' type='ball'>
  <parent>chassis</parent>
  <child>caster</child>
</joint>
```

Figure 6.4: Implementation of two-wheel joints

6.3.4 Camera

Main camera is integrated into the robot using following source as shown in figure 6.5.

```
243     <joint name="camera_optical_joint" type="fixed">
244       <parent> camera_link</parent>
245       <child> camera_link_optical</child>
246     </joint>
247
248     <link name="camera_link_optical">
249     <pose>0.9 0 0.50 0 0.55 0</pose>
250       <sensor type="camera" name="camera1">
251         <update_rate>60</update_rate>
252         <visualize>true</visualize>
253         <camera name="head">
254           <horizontal_fov>1.3962634</horizontal_fov>
255           <image>
256             <width>640</width>
257             <height>480</height>
258             <format>R8G8B8</format>
259           </image>
260           <clip>
261             <near>0.02</near>
262             <far>300</far>
263           </clip>
264           <noise>
265             <type>gaussian</type>
266             <!-- Noise is sampled independently per pixel on each frame.
267                  That pixel's noise value is added to each of its color
268                  channels, which at that point lie in the range [0,1]. -->
269             <mean>0.0</mean>
270             <stddev>0.007</stddev>
271           </noise>
272         </camera>
273         <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
274           <ros>
275             <namespace>demo_cam</namespace>
276             <remapping>image_raw:=image_demo</remapping>
277             <remapping>camera_info:=camera_info_demo</remapping>
278           </ros>
279           <!-- camera_name>omit so it defaults to sensor name</camera_name-->
280           <!-- frame_name>omit so it defaults to link name</frame_name-->
281         </plugin>
282       </sensor>
283     </link>
```

Figure 6.5: Front Camera Implementation

For gazebo simulation camera plugin *libgazebo_ros_camera.so* has been used in line 273 and connected to the link in robot to see path. This plugin has a ROS node, and it publishes its raw images the *image_raw* topic in the ROS2 environment.

6.3.5 Differential Drive

The differential drive system serves as the primary propulsion mechanism for the robot and has been widely adopted in numerous research projects. The left and right wheels of the two wheeled robot are linked to the differential drive, allowing this to alter its direction by adjusting the relative rotation rates of its wheels. Consequently, no additional steering motion is necessary.

Differential drive also implemented as a gazebo plugin, and it uses *libgazebo_ros_diff_drive.so* as shown in figure 6.6 line 297.

```
297    <plugin name='diff_drive' filename='libgazebo_ros_diff_drive.so'>
298
299      <ros>
300        <namespace>/demo</namespace>
301        <remapping>cmd_vel:=cmd_demo</remapping>
302        <remapping>odom:=odom_demo</remapping>
303      </ros>
304
305      <!-- wheels -->
306      <left_joint>left_wheel_joint</left_joint>
307      <right_joint>right_wheel_joint</right_joint>
308
309      <!-- kinematics -->
310      <wheel_separation>1.25</wheel_separation>
311      <wheel_diameter>0.6</wheel_diameter>
312
313      <!-- limits -->
314      <max_wheel_torque>20</max_wheel_torque>
315      <max_wheel_acceleration>1.0</max_wheel_acceleration>
316
317      <!-- output -->
318      <publish_odom>true</publish_odom>
319      <publish_odom_tf>true</publish_odom_tf>
320      <publish_wheel_tf>true</publish_wheel_tf>
321
322      <odometry_frame>odom_demo</odometry_frame>
323      <robot_base_frame>chassis</robot_base_frame>
324
325    </plugin>
```

Figure 6.6: Differential Drive Implementation

6.4 Ground plane as the path for the robot to follow.

The robot operates within an environment consisting of a ground plane with a non-linear path designated for the robot to navigate, resembling a line. This path was also integrated into URDF for Gazebo simulation, depicted in Figure 6.7.

```
<?xml version="1.0"?>
<sdf version="1.4">
<model name="my_ground_plane">
  <static>true</static>
    <link name="link">
      <collision name="collision">
        <geometry>
          <plane>
            <normal>0 0 1</normal>
            <size>50 50</size>
          </plane>
        </geometry>
        <surface>
          <friction>
            <ode>
              <mu>100</mu>
              <mu2>50</mu2>
            </ode>
          </friction>
        </surface>
      </collision>
      <visual name="visual">
        <cast_shadows>false</cast_shadows>
        <geometry>
          <plane>
            <normal>0 0 1</normal>
            <size>50 50</size>
          </plane>
        </geometry>
        <material>
          <script>
            <!-- <uri>model://my_ground_plane/materials/scripts/my_ground_plane.material</uri>
            <uri>model://my_ground_plane/materials/scripts</uri>
            <uri>model://my_ground_plane/materials/textures</uri>
            <name>MyGroundPlane/Image</name>
          </script>
        </material>
      </visual>
    </link>
  </model>
</sdf>
```

Figure 6.7: Ground Plane with Line

6.5 ROS network

ROS network consists of 5 main nodes namely */follower*, */pilco_node*, */diff_drive*, */camera controller and /odom subscriber* shown in figure 6.8.

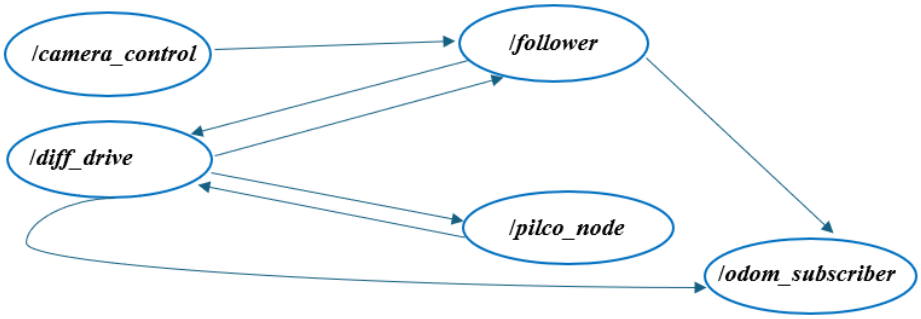


Figure 6.8: ROS node graph

The following figure. 6.9 shows the topics which are used to communicate between nodes. Camera node publishes *image_raw* topic which subscribed by the follower node to calculate the twist command to the *cmd_demo* topic.

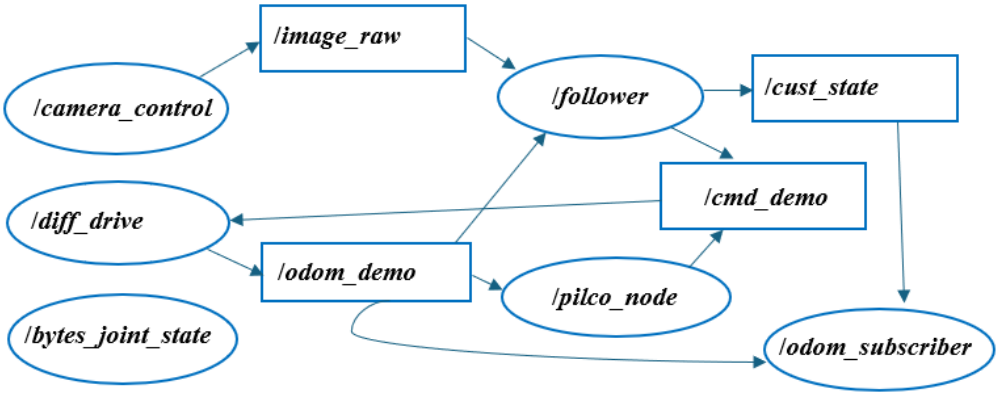


Figure 6.9: ROS Node and Topics

diff_drive node responsible of driving the wheels according to the twist commands from the cmd_demo topic.

6.5.1 Line follow mechanism of the *follower* node using *image_raw* topic

image_raw topic gives an image as shown in figure 6.10. This image then passed to get_contour_data function as following listing and OpenCV library is used to find the centroid of largest blue color mass using contour separation technique shown in figure 6.11.

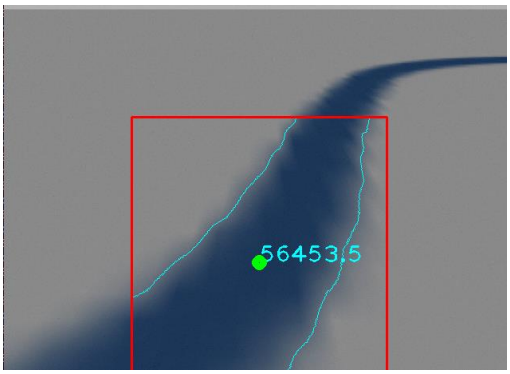


Figure 6.10: Camera view

```python
def get_contour_data(mask, out):
    """
    Return the centroid of the largest contour in
    the binary image 'mask' (the line)
    and return the side in which the smaller contour is (the track mark)
    (If there are any of these contours),
    and draw all contours on 'out' image
    """
    # get a list of contours
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

    mark = {}
    line = {}

    for contour in contours:

        M = cv2.moments(contour)
        # Search more about Image Moments on Wikipedia :)

        if M['m00'] > MIN_AREA:
        # if countor.area > MIN_AREA:

            if (M['m00'] > MIN_AREA_TRACK):
                # Contour is part of the track
                line['x'] = crop_w_start + int(M["m10"]/M["m00"])
                line['y'] = int(M["m01"]/M["m00"])

                # plot the area in light blue
                cv2.drawContours(out, contour, -1, (255,255,0), 1)
                cv2.putText(out, str(M['m00']), (int(M["m10"]/M["m00"]), int(M["m01"]/M["m00"])),
                    cv2.FONT_HERSHEY_PLAIN, 2, (255,255,0), 2)
                #break

            else:
                # Contour is a track mark
                if (not mark) or (mark['y'] > int(M["m01"]/M["m00"])):
                    # if there are more than one mark, consider only
                    # the one closest to the robot
                    mark['y'] = int(M["m01"]/M["m00"])
                    mark['x'] = crop_w_start + int(M["m10"]/M["m00"])

                    # plot the area in pink
                    cv2.drawContours(out, contour, -1, (255,0,255), 1)
                    cv2.putText(out, str(M['m00']), (int(M["m10"]/M["m00"]), int(M["m01"]/M["m00"])),
                        cv2.FONT_HERSHEY_PLAIN, 2, (255,0,255), 2)


    if mark and line:
    # if both contours exist
        if mark['x'] > line['x']:
            mark_side = "right"
        else:
            mark_side = "left"
    else:
        mark_side = None

```

Figure 6.11: Line Following Algorithm

The follower node has two modes to accommodate hybrid controller and whenever the camera view is not available, the mode changes to the PILCO mode and twist message is taken by requesting it from the PILCO node as in figure 6.12.

```
341     else:
342         print("pilco mode")
343         state = state_stack.pop()
344         print(state)
345         np_state = np.array(state)
346         np_state_reshaped = np_state.reshape(1, -1)
347         message.linear.x = 1.47
348         revz = gpr_rosbot.predict(np_state_reshaped)
349         message.angular.z = float(revz)
```

Figure 6.12: PILCO mode implementation

gpr_rosbot. predict calls the service *pilco_service* running in *pilco_node* as in figure 6.14.

```
34  def pilco_service_callback(self,request, response):
35      """
36      pilco service
37      """
38      state = self.state_stack.pop()
39      np_state = np.array(state)
40      print("stop: move")
41      response.linx = 1.5
42      response.revz = gpr_rosbot.predict(np_state)
43      self.get_logger().info('Incoming request\na: %d b: %d' % (response.linx, response.revz))
44      return response
```

Figure 6.13: Service callback function

Full source for the *pilco_node* is shown in the following figure 6.14.

```python
import rclpy
import numpy as np
from rclpy.node import Node
from nav_msgs.msg import Odometry
from geometry_msgs.msg import Pose, Twist
from more_interfaces.srv import PilcoTwist
from py_2wbot_ctrl.gpr_rosbot import GPR_rosbot
from std_srvs.srv import Empty


class PilcoNode(Node):
    def __init__(self):
        super().__init__('pilco_node')
        self.req = Empty.Request()
        self.state_stack = []
        self.publisher = self.create_publisher(Twist, '/demo/cmd_demo', rclpy.qos.qos_profile_system_default)
        self.subscription = self.create_subscription(
        Odometry,
        '/demo/odom_demo',  # Topic name
        self.odom_callback,  # Callback function
        1  # QoS profile depth
        )
        self.gpr_rosbot = GPR_rosbot()
        self.gpr_rosbot.loadModel()

        self.pilco_service = self.create_service(PilcoTwist, 'pilco_service', self.pilco_service_callback)
        self.start_pilco_client = self.create_client(Empty, 'start_pilco')
        while not self.start_pilco_client.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('service not available, waiting again...')
        self.send_pilco_request()


    def pilco_service_callback(self,request, response):
        """
        pilco service
        """
        state = self.state_stack.pop()
        np_state = np.array(state)
        print("stop: move")
        response.linx = 1.5
        response.revz = gpr_rosbot.predict(np_state)
        self.get_logger().info('Incoming request\na: %d b: %d' % (response.linx, response.revz))
        return response

    def odom_callback(self, msg):
        # This function will be called each time an Odometry message is received
        # Access the pose
        state = []
        #timestr = time.strftime("%H%M%S%f")
        pose = msg.pose.pose
        position = pose.position
        state.append(position.x)
        state.append(position.y)

        orientation = pose.orientation
        state.append(orientation.z)

        # Access the twist (linear and angular velocities)
        twist = msg.twist.twist
        linear_velocity = twist.linear
        state.append(linear_velocity.x)
        state.append(linear_velocity.y)

        angular_velocity = twist.angular
        state.append(angular_velocity.z)

        self.state_stack.append(state)


    def send_pilco_request(self):
        self.future = self.start_pilco_client.call_async(self.req)
        rclpy.spin_until_future_complete(self,self.future)
        return self.future.result()

def main(args=None):
    rclpy.init(args=args)
    pico_node = PilcoNode()
    rclpy.spin(pico_node)
    pico_node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Figure 6.14: PILCO Node

6.5.2 Logging the state action data to csv file.

This task is done by the *odom_subscriber* node which is subscribed for */odom_demo* and */cust_state* topics that are publishing joint states and action commands respectively. The source for this task is highlighted in 6.15.

```
1  import rclpy
2  from rclpy.node import Node
3  from nav_msgs.msg import Odometry
4  from std_srvs.srv import Empty
5  import time
6  from datetime import datetime
7  import csv
8  from more_interfaces.msg import CustomState
9
10 RUN_TIME = 10
11 LOG_FILE_PATH = "/home/lasantha/"
12
13 class OdomSubscriber(Node):
14     def __init__(self):
15         super().__init__('odom_subscriber')
16         self.subscription = self.create_subscription(
17             Odometry,
18             '/demo/odom_demo',  # Topic name
19             self.odom_callback,  # Callback function
20             1  # QoS profile depth
21         )
22
23         self.state_subscription = self.create_subscription(
24             CustomState,
25             '/demo/cust_state',  # Topic name
26             self.joint_state_callback2,  # Callback function
27             1  # QoS profile depth
28         )
29
30         self.start_client = self.create_client(Empty, 'start_follower')
31         while not self.start_client.wait_for_service(timeout_sec=1.0):
32             self.get_logger().info('service not available, waiting again...')
33
34         self.stop_client = self.create_client(Empty, 'stop_follower')
35         while not self.stop_client.wait_for_service(timeout_sec=1.0):
36             self.get_logger().info('service not available, waiting again...')
37
38         timestr = time.strftime("%Y%m%d-%H%M%S")
39         self.filename = LOG_FILE_PATH + timestr + ".csv"
40         self.logdata = []
41         self.req = Empty.Request()
42         self.start_time = time.time()
43         self.subscription  # Prevent unused variable warning
44
```

Figure 6.15: Subscribe into the odom topic.

On line 16 and 23 in figure 6.15, the node subscribed to two topics mentioned above. *join_state_callback2* as shown in figure 6.16 is responsible for logging error and the action.

```python
    def joint_state_callback2(self, msg):
        # This function will be called each time a JointState message is received
        # Access joint names
        #robot_name = msg.name
        row = []
        timestr = datetime.now().strftime("%H:%M:%S.%f")#time.strftime("%H%M%S%f")
        row.append(timestr)

        robot_error = msg.err
        row.append(msg.err)
        # Access joint positions
        robot_linear_x = msg.linear_x
        row.append(msg.linear_x)
        row.append(msg.linear_y)
        row.append(msg.linear_z)
        # Access joint velocities
        robot_angular_z = msg.angular_z
        row.append(msg.angular_x)
        row.append(msg.angular_y)
        row.append(msg.angular_z)
        # Access joint efforts
        print("robot_error : {}".format(robot_error))
        #print("robot_linear_x : {}".format(robot_linear_x))
        #print("robot_angular_z : {}".format(robot_angular_z))
        self.logdata.append(row)
```

Figure 6.16: Joint state callback

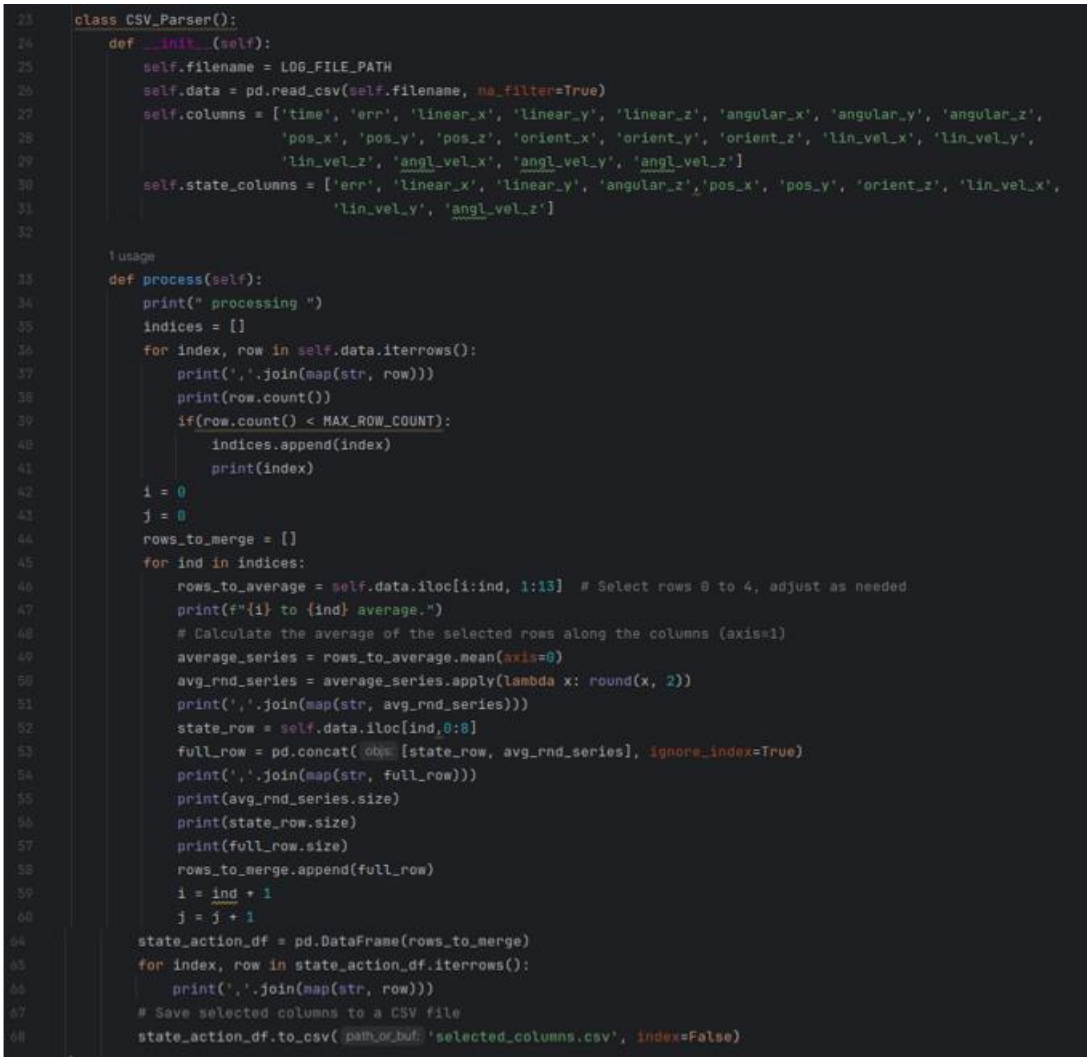
odom_callback function in figure 6.17 is responsible for logging the robot state of linear position, linear velocity, angular orientation and angular velocity.

```python
    def odom_callback(self, msg):
        # This function will be called each time an Odometry message is received
        # Access the pose
        row = []
        #timestr = time.strftime("%H%M%S%f")
        timestr = datetime.now().strftime("%H:%M:%S.%f")
        row.append(timestr)

        pose = msg.pose.pose
        position = pose.position
        row.append(position.x)
        row.append(position.y)
        row.append(position.z)

        orientation = pose.orientation
        row.append(orientation.x)
        row.append(orientation.y)
        row.append(orientation.z)
        # Access the twist (linear and angular velocities)
        twist = msg.twist.twist
        linear_velocity = twist.linear
        row.append(linear_velocity.x)
        row.append(linear_velocity.y)
        row.append(linear_velocity.z)

        angular_velocity = twist.angular
        row.append(angular_velocity.x)
        row.append(angular_velocity.y)
        row.append(angular_velocity.z)
        end_time = time.time()
        elapsed_time = end_time - self.start_time
        #print("elapsed_time: {}".format(elapsed_time))
        self.logdata.append(row)
        if(elapsed_time > RUN_TIME):
            self.log_data_to_csv()
            self.send_stop_request()
            print("TIME has finished")
```

Figure 6.17: Odom callback

6.5.3 Parsing the csv file and rewriting them to be compatible with MATLAB.

joint states and actions are not recorded synchronically so CSV_Parser is doing some aggregation to get synchronized data in lines 49 to 53 in figure 6.18.



```
class CSV_Parser():
    def __init__(self):
        self.filename = LOG_FILE_PATH
        self.data = pd.read_csv(self.filename, na_filter=True)
        self.columns = ['time', 'err', 'linear_x', 'linear_y', 'linear_z', 'angular_x', 'angular_y', 'angular_z',
                        'pos_x', 'pos_y', 'pos_z', 'orient_x', 'orient_y', 'orient_z', 'lin_vel_x', 'lin_vel_y',
                        'lin_vel_z', 'angl_vel_x', 'angl_vel_y', 'angl_vel_z']
        self.state_columns = ['err', 'linear_x', 'linear_y', 'angular_z','pos_x', 'pos_y', 'orient_z', 'lin_vel_x',
                              'lin_vel_y', 'angl_vel_z']

    1 usage
    def process(self):
        print(" processing ")
        indices = []
        for index, row in self.data.iterrows():
            print(','.join(map(str, row)))
            print(row.count())
            if(row.count() < MAX_ROW_COUNT):
                indices.append(index)
                print(index)
        i = 0
        j = 0
        rows_to_merge = []
        for ind in indices:
            rows_to_average = self.data.iloc[i:ind, 1:13]  # Select rows 0 to 4, adjust as needed
            print(f"{i} to {ind} average.")
            # Calculate the average of the selected rows along the columns (axis=1)
            average_series = rows_to_average.mean(axis=0)
            avg_rnd_series = average_series.apply(lambda x: round(x, 2))
            print(','.join(map(str, avg_rnd_series)))
            state_row = self.data.iloc[ind,0:8]
            full_row = pd.concat( objs: [state_row, avg_rnd_series], ignore_index=True)
            print(','.join(map(str, full_row)))
            print(avg_rnd_series.size)
            print(state_row.size)
            print(full_row.size)
            rows_to_merge.append(full_row)
            i = ind + 1
            j = j + 1
        state_action_df = pd.DataFrame(rows_to_merge)
        for index, row in state_action_df.iterrows():
            print(','.join(map(str, row)))
        # Save selected columns to a CSV file
        state_action_df.to_csv( path_or_buf: 'selected_columns.csv', index=False)
```

Figure 6.18: CSV parser Implementation

6.6 PILCO base RL algorithm for learning the policy.

6.6.1 Introduction

This MATLAB implementation is an extended version of PILCO Software Package V0.9 (2013-07-04) [14]. The initial iteration is a MATLAB library designed for use PILCO algorithm for problems.

Aside from given scenarios of this, a new one called *rosbot* is created to cater for this reinforcement learning based hybrid controller project as in figure 6.19.

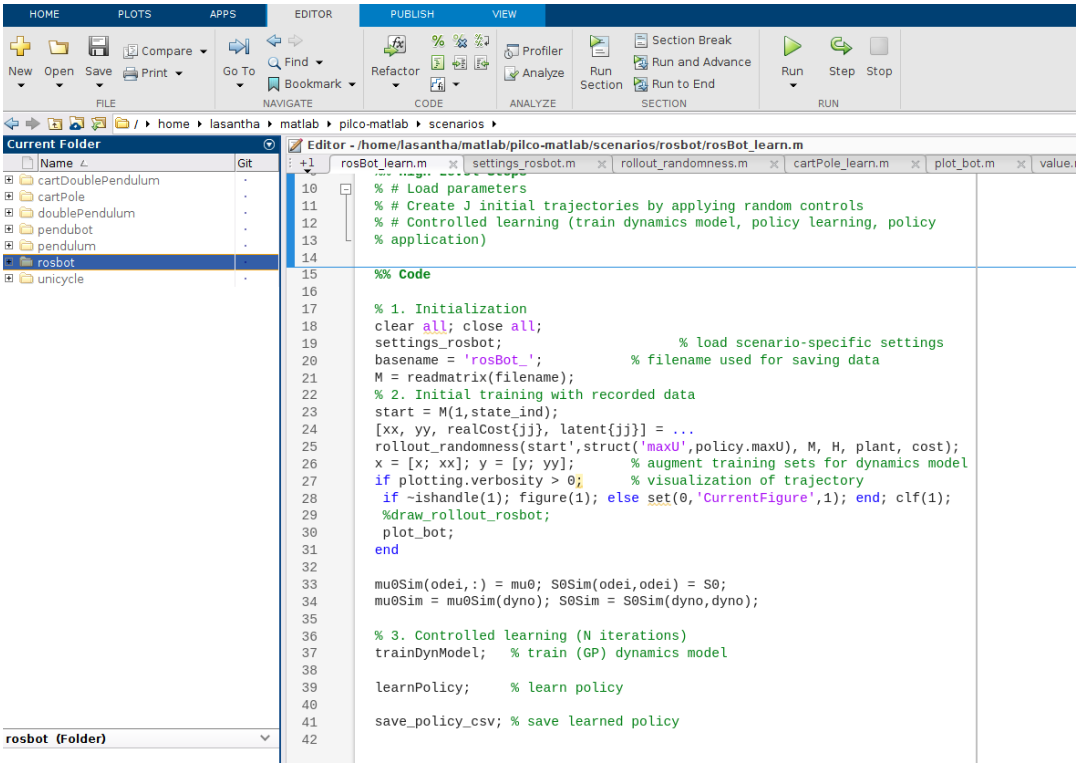


Figure 6.19: PILCO framework

Then new files of *draw_rollout_rosbot.m*, *settings_rosbot.m*, *rosBot_learn.m* and *rosbot_results_plot.m* are added to implement the learning process of line follower robot.

PILCO implementation contains three high level modules as described in the design and they are Model learning, Policy learning, Policy application respectively. Top level MATLAB implementation is given in figure 6.18.

```
17  % 1. Initialization
18  clear all; close all;
19  settings_rosbot;                        % load scenario-specific settings
20  basename = 'rosBot_';                % filename used for saving data
21  M = readmatrix(filename);
22  % 2. Initial training with recorded data
23  start = M(1,state_ind);
24  [xx, yy, realCost{jj}, latent{jj}] = ...
25  rollout_randomness(start',struct('maxU',policy.maxU), M, H, plant, cost);
26  x = [x; xx]; y = [y; yy];       % augment training sets for dynamics model
27  if plotting.verbosity > 0;      % visualization of trajectory
28   if ~ishandle(1); figure(1); else set(0,'CurrentFigure',1); end; clf(1);
29   %draw_rollout_rosbot;
30   plot_bot;
31  end
32
33  mu0Sim(odei,:) = mu0; S0Sim(odei,odei) = S0;
34  mu0Sim = mu0Sim(dyno); S0Sim = S0Sim(dyno,dyno);
35
36  % 3. Controlled learning (N iterations)
37  trainDynModel;   % train (GP) dynamics model
38
39  learnPolicy;     % learn policy
40
41  save_policy_csv; % save learned policy
```

Figure 6.18: Top Level MATLAB interface

6.6.2 Initialization of setup

line 19 calls the *setting_rosbot* which initializes the parameters as shown below.

```
43  filename = 'sim_data.csv';
44  state_ind = [8 9 14 15 19 13];
45  action_ind = [2 7];
46  cost_ind = 1;
47  plot_ind = [8 9];
48  dt = 0.0625;
```

Figure 6.19: Initial Settings I

In line 43 of figure 6.19, filename is the logged data from logging ROS node which saved the training data to csv files.

```
60  T = 10.0;                          % [s] initial prediction horizon time
61  H = ceil(T/dt);                   % prediction steps (optimization horizon)
62  mu0 = [0 0 0 0 0 0]';                % initial state mean
63  S0 = diag([0.1 0.1 0.1 0.1 0.1 0.1].^2);   % initial state covariance
64  N = 15;                           % number controller optimizations
65  J = 1;                            % initial J trajectories of length H
66  K = 1;                            % no. of initial states for which we optimize
```

Figure 6.20: Initial Settings II

Here in figure 6.20 listing T is horizon time where the training data is collected and mu0, S0 are initial mean state and covariance.

6.6.2 Collecting samples from CSV file.

In the file *rollout_randomness.m* code is listed below.

```
50
51  function [x y L latent] = rollout_randomness(start,policy, M, H, plant, cost)
52  %% Code
53
54  %simi = plant.state_ind;
55  simi = [1, 2, 3, 4, 5, 6];%sort([plant.state_ind]);
56  %%
57  nX = length(simi); nU = length(policy.maxU); nA = 1;% nX=6 nA=1 nU=2
58
59  state(simi) = start; %state(augi) = plant.augment(state);      % initializations
60  x = zeros(H+1, nX+2*nA);% x is (H + 1) x 6 --> 41x6
61  x(1,simi) = start' + randn(size(simi))*chol(plant.noise);
62  %x(1,augi) = plant.augment(x(1,:));
63  u = zeros(H, nU); latent = zeros(H+1, size(state,2)+nU);
64  y = zeros(H, nX); L = zeros(1, H); next = zeros(1,length(simi));
65
66  u(1:H,:) = M(1:H,plant.action_ind);
67
68  for i = 1:H % ------------------------------------------- generate trajectory
69
70    latent(i,:) = [state u(i,:)];                              % latent state
71
72    % 2. Simulate dynamics ------------------------------------------------------
73    next(simi) = M(i+1,plant.state_ind);
74    %next(subi) = plant.subplant(state, u(i,:));
75
76
77    % 4. Augment state and randomize --------------------------------------------
78    state(simi) = next(simi); %state(augi) = plant.augment(state);
79    x(i+1,simi) = state(simi) + randn(size(simi))*chol(plant.noise);
80    x(i+1,[7 8]) = [cos(x(i+1,6)),sin(x(i+1,6))];
81    %x(i+1,augi) = plant.augment(x(i+1,:));
82
83    % 5. Compute Cost -----------------------------------------------------------
84    if nargout > 2
85      L(i) = M(i,1);%cost.fcn(cost,state(dyno)',zeros(length(dyno)));
86    end
87  end
```

Figure 6.21*:* Addition of Gaussian noise

Here the random Gaussian noise is added to the data for GP to have tolerance in the measured points.

6.6.3 Plot graphs for evaluation

```
M = {};
result = zeros(5,1);
% Define an array of 12 different colors
colors = {'r', 'g', 'b', 'c', 'm', [0, 0, 0], 'k', ...  % Basic colors
          [0.5, 0, 0], [0, 0.5, 0], [0, 0, 0.5], ... % RGB values
          [1, 0.5, 0], [0, 0, 0], [1, 1, 1]};    % Custom colors
lineStyles = {'-', '-','-','-','-','--'...
    ,'-','-','-','-','-','--'};
for i = 1 : length(filenames)
    disp(['filename is: ', filenames(i)]);
    M{1,i} = readmatrix(filenames(i));
end
Ref = readmatrix(ref_file);
Ry = Ref(1:160,plot_ind);
for j = 1 : length(M)
    S = cell2mat(M(1,j));
    xy = S(1:160,plot_ind);
    squaredDistances = sqrt((sum((sqrt(Ry(:,1).^2 + Ry(:,2).^2) - sqrt(xy(:,1).^2 + xy(:,2).^2)).^2))
    result(j) = squaredDistances;
    %E = rmse(Ry,xy);
    d = sprintf('%f rmse is %f.',squaredDistances,E);
    disp(d);
    %plot(xy(1:160,1),xy(1:160,2), ...
     %   'linestyle', lineStyles{j},...
     %   'Color', colors{j});
    %hold on;
end
```

Figure 6.21: Plotting the results.

6.7 Summary

In this chapter we presented the implementation of reinforcement learning based hybrid controller for simulated robot in gazebo. The implementation consists of building the robot and its environment using URDF in gazebo framework, creating ROS network for controlling the robot and learning with optimization of policy with PILCO framework in MATLAB for the robot trajectories. Finally, MATLAB again implemented the plotting of results to generate graphs. The next chapter will focus on evaluating these results to find the accuracy and efficiency of the PILCO based hybrid controller.

CHAPTER 7

EVALUATION

7.1 Introduction

This chapter presents the outcomes of RL based hybrid controller by simulating the robot in the gazebo environment with graphs and values. It also tries to show evaluation criteria that has been taken for measuring the performance of the hybrid controller. These values are recorded when the line follower robot goes through trajectories calculated by the RL based controller.

The chapter briefly evaluates this approach over other methods available to carry out the same task and tries to justify the significance by carefully examining the results.

7.2 Experimental design

Firstly, a path segment is chosen from the circular type of irregular shape line and data were recorded for ten second duration for robot to travel with camera-based line following method. This data was recorded 12 times to build the test data for the MATLAB based RL controller to learn the policy. Secondly, the camera view was shut down to create a hypothetical emergency, so the PILCO based RL controller took over the control. Again, the same set of data was recorded for the evaluation and figure 7.1 shows the same position data plotted on to a graph.

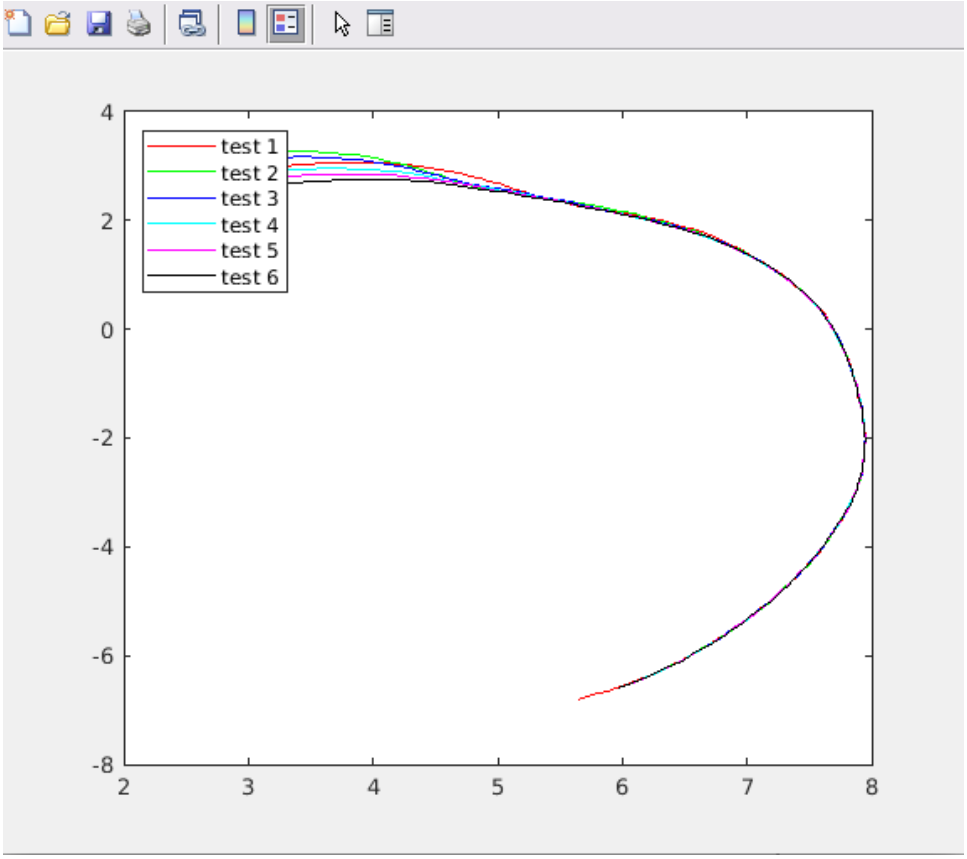


Figure 7.1: Trajectories taken by robot during initial test runs.

7.3. PILCO policy learning results.

The above test data was fed to the PILCO algorithm to get the optimized policy to perform the line following task. Immediate cost has been recorded over time and plotted onto the graphs as shown in figure 7.2 and we can clearly see the convergence happens after few iterations of learning.

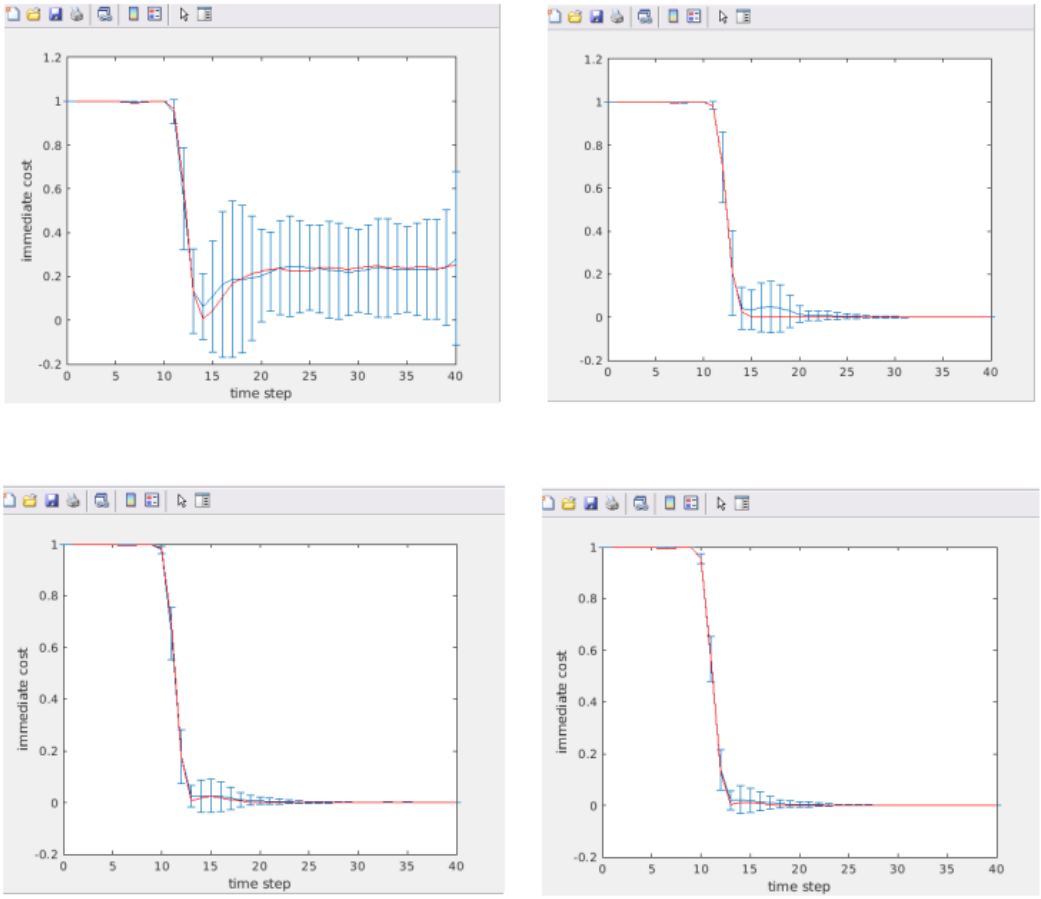


Figure 7.2*: Decay of immediate cost over iterations*

7.4 Evaluation Metrics

Root mean squared deviation has been selected as the evaluation metric to the project, so the error is taken as the distance from the predicted point to the reference point on the respective trajectories. In figure 7.3 black path shows the reference line and others show the trail trajectories taken by the PILCO learned policies. RMSD serves as a common error metric in academic works. Lower values signify superior performance, with RMSD being minimal when the inferred state distribution's mean closely aligns with the true state.

$$\mathrm{RMSD} = \sqrt{\frac{\sum_{i=1}^{N} (x_i - \hat{x}_i)^2}{N}} \tag{7.1}$$

7.5 Performance Evaluation

It was observed that the RMSD has been decayed over few iterations and shows the data efficiency in PILCO learning.

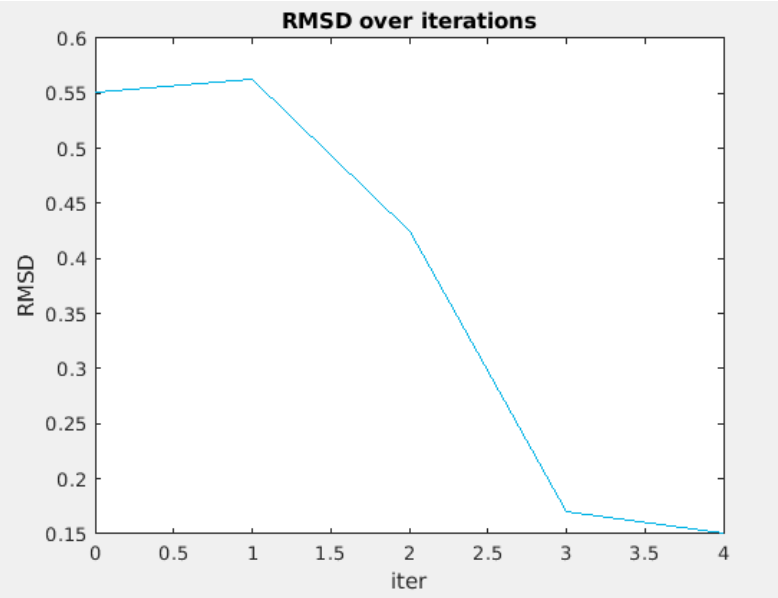


Figure 7.3*: RMSD over iterations*

Table 7.1 also shows how values have been decreased.

Iteration	RMSD
1	0.551629
2	0.562189
3	0.424239
4	0.169850
5	0.151137

7.6 Trajectories traced on each iteration to see the improvement on the plot.

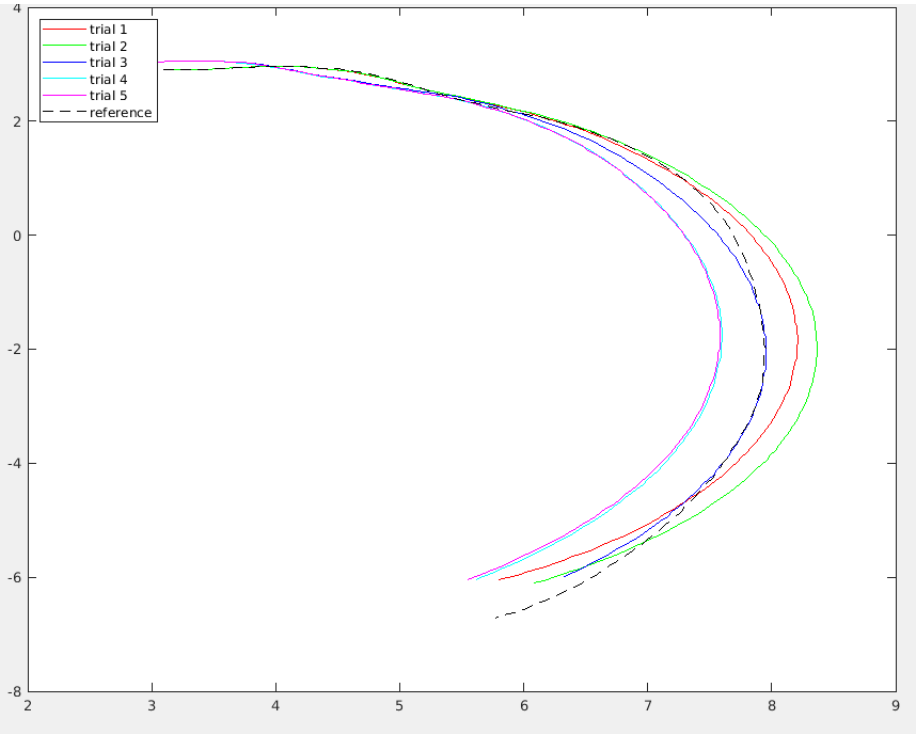


Figure 7.4: Trajectories of robot

It was clearly visible how the PILCO learn quite fast and get to the correct trajectory in few iterations in figure 7.4.

7.6.1 Trajectories traced by other algorithms

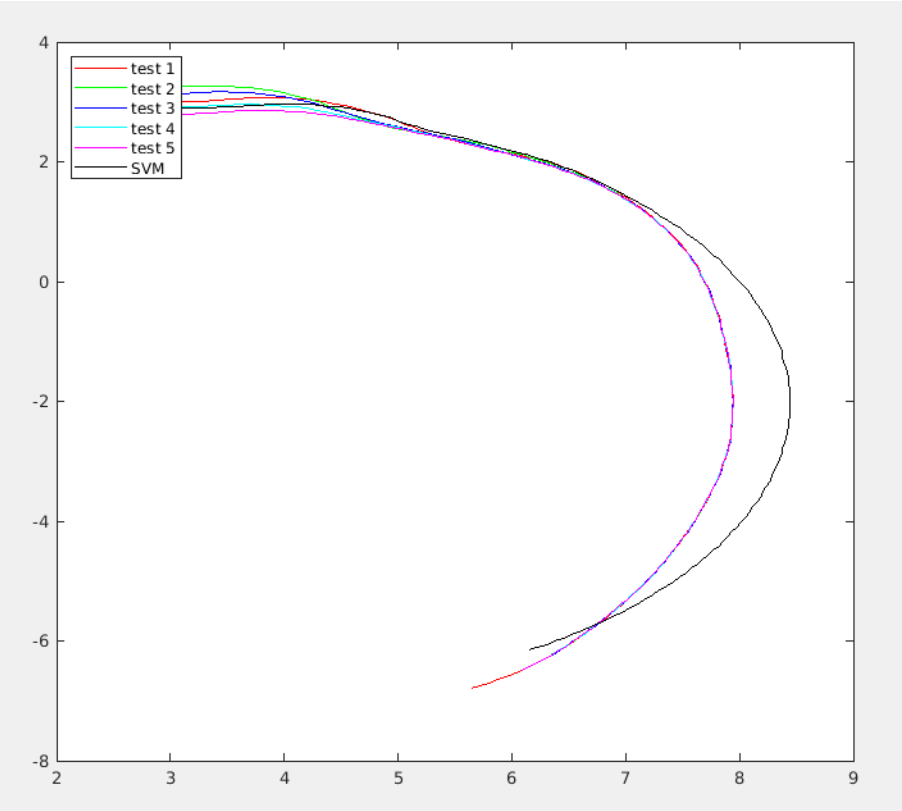


Figure 7.5: Trajectories comparison with other algorithms

In this comparison, we juxtapose Support Vector Machines (SVM) with PILCO due to their similarity in utilizing the Radial Basis Function (RBF) kernel. However, it becomes evident that SVM requires significantly more iterations to converge compared to PILCO, shown in figure 7.5. This implies that SVM needs more computational steps or iterations to reach a stable and optimal solution compared to PILCO when applied to similar tasks or datasets.

7.7 Significance of using PILCO in a hybrid controller.

The original PILCO algorithm has demonstrated superior data efficiency compared to other continuous state reinforcement learning methods. PILCO is recognized as the most effective RL approach in with regard to learning speed, typically by a significant margin. This is largely attributed to its systematic approach to mitigating model bias in RL. Mainly PILCO does not depend on expert knowledge, aligning with the essence of AI. Moreover, PILCO enables unparalleled data-efficient learning from scratch in continuous state and control domains.

7.8 Discussion and Findings

Thesis shows that the RL algorithms are very useful when used in parallel to the deterministic controllers so that first few iterations that can be harmful for the practical applications can be avoided. We also observed that PILCO, like GP based algorithms more suited to the robotic manipulators since their movements are continuous in nature.

Furthermore, this approach would be beneficial for many robotic applications to build more robust systems in future.

7.9 Summary

In this chapter we presented the evaluation of the project by comparing the predicted trajectories of the robot with deterministically found ones using the RMSD values. It showed that the PILCO based RL technique quickly optimized the policy to have the desired trajectories.

Furthermore, it also highlighted the significance of PILCO over other reinforcement learning techniques in this robot control field.

This chapter provided a comprehensive assessment of the reinforcement learning based hybrid controller, setting the stage for the subsequent chapter that will describe the system's strengths, limitations, and further research.

CHAPTER 8

CONCLUSION AND FURTHER WORK

In this thesis we were able to develop reinforcement learning based hybrid controller for robotic manipulator in simulated world using data efficient RL technique. The system demonstrated promising results by accurately predicting the desired trajectories for the line follower robot, showing the potential to address critical applications of robotics that need robust fail-safe system.

The objective of the thesis is to fill the gap of practical applications using reinforcement learning in the industry, because RL has the human instinct of learning by doing. Another aspect we want to cover in the thesis is to apply the inspiration took from muscle memory in humans. With this hybrid mechanism we have shown that robots can take actions even without acting on direct observations but with pre-learned policies like humans do.

Today, the robotics field has witnessed remarkable improvements driven by cutting-edge technologies and innovative approaches. Modern developments in robotics have led to the creation of robots that are more capable, versatile, and intelligent than ever before. A prominent trend in contemporary robotics involves merging artificial intelligence (AI) and machine learning methodologies. Robots enhanced with AI can adjust to dynamic surroundings, acquire knowledge through experience, and autonomously make decisions, enhancing their capability to execute intricate tasks more efficiently and precisely. These advancements have facilitated the emergence of autonomous vehicles, humanoid robots, and robotic systems capable of providing support across diverse industries, spanning from manufacturing and healthcare to agriculture and logistics. All these developments lead to prefect humanoid that should have almost all the human instincts built on to it, so this research will definitely add value to going research and industry in particular to develop.

Although we have developed the hybrid mechanism to automatically switch between deterministic controller and RL based controller, this hybrid mechanism is very primitive in nature and it does not contain intelligence on it. Therefore, we feel there should be more research on making the system have true hybrid mechanism to smoothly switch between several algorithms to have optimized manure that suit to the situation. Our research also suggests that the best way to achieve such mechanism is to train a ML algorithm to build confidence level of each algorithm and chose that having the highest level.

Furthermore, in our research the PILCO algorithm was not run in dynamic ROS environment, but the policy was trained offline using CSV files to apply through ROS. This is mainly because the PILCO algorithm was to be only available with MATLAB and there were some python-based implementations which are cumbersome to build. So, it is good to have readily available sources in many languages to use for the research community to further be digging up into it. Future of edge AI definitely need

these types of algorithms to adopt and run even in smaller micro controllers, so that every robot can have RL capabilities on hand. Therefore, we have also identified that more research should be done in PILCO implementation to write in many programming languages to make it more efficient so that to reduce the latency in policy learning.

In conclusion, this research project successfully developed a reinforcement learning based hybrid controller for robotic manipulator. The system demonstrated promising results in accurately generating trajectories even when robot faces lack of observations on the run. The findings from this research lay a foundation for further advancements in practical application of RL in robotics to gain muscle memory concept into reality.

REFERENCES

[1] C. M.-C. Huang A-C, Adaptive Control of Robot Manipulators: A Unified regressor-free approach. World Scientific Publishing Company 2010.

[2] M. Deisenroth, C. Rasmussen, and D. Fox, Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning. 2011.

[3] M. P. Deisenroth and C. E. Rasmussen, "PILCO: a model-based and data-efficient approach to policy search," presented at the Proceedings of the 28th International Conference on International Conference on Machine Learning, Bellevue, Washington, USA, 2011.

[4] Y. Matsuo et al., "Deep learning, reinforcement learning, and world models," (in eng), Neural Netw, vol. 152, pp. 267-275, Aug 2022, doi: 10.1016/j.neunet.2022.03.037.

[5] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," The International Journal of Robotics Research, vol. 32, no. 11, pp. 1238-1274, 2013.

[6] M. Kuss and C. Rasmussen, "Gaussian processes in reinforcement learning," Advances in neural information processing systems, vol. 16, 2003.

[7] R. Sutton, A. Barto, and R. Williams, Reinforcement Learning is Direct Adaptive Optimal Control. 1991, pp. 2143-2146.

[8] W. B. Powell, "AI, OR and Control Theory: A Rosetta Stone for Stochastic Optimization," 2012.

[9] V. N. Vapnik, The Nature of Statistical Learning Theory. New York: Springer, 1995.

[10] J. G. Schneider, "Exploiting Model Uncertainty Estimates for Safe Dynamic Control Learning," in Neural Information Processing Systems, 1996.

[11] M. P. Deisenroth, “Efficient Reinforcement Learning using Gaussian Processes.” Accessed: Apr. 28, 2024. [Online]. Available: https://publikationen.bibliothek.kit.edu/1000019799

[12] J. Yoo, D. Jang, H. J. Kim, and K. H. Johansson, "Hybrid Reinforcement Learning Control for a Micro Quadrotor Flight," IEEE Control Systems Letters, vol. 5, no. 2, pp. 505-510, 2021, doi: 10.1109/LCSYS.2020.3001663.

[13] M. Deisenroth, A. McHutchon, J. H. Carl, and E. Rasmussen. PILCO Software Package V0.9 [Online] Available: https://github.com/UCL-SML/pilco-matlab/blob/master/README.md

[14] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: lessons we have learned," The International Journal of Robotics Research, vol. 40, no. 4-5, pp. 698-721, 2021, doi: 10.1177/0278364920987859.

APPENDIX A

ROS node graphs

ROS has an inbuild infrastructure to see running graph to be shown with rqt_graph tool. The following figures of A.1 and A.2 shows the ROS nodes with topics and nodes alone in subjected hybrid controller.

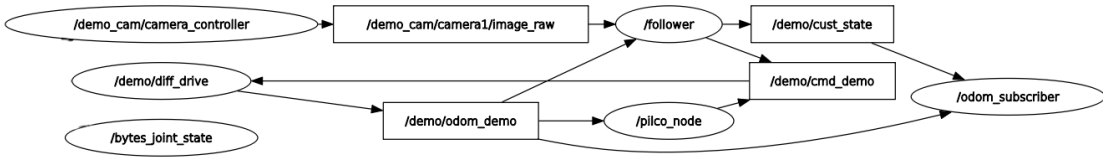


Figure A.1: Nodes and Topics with their interaction from *rqt_graph*

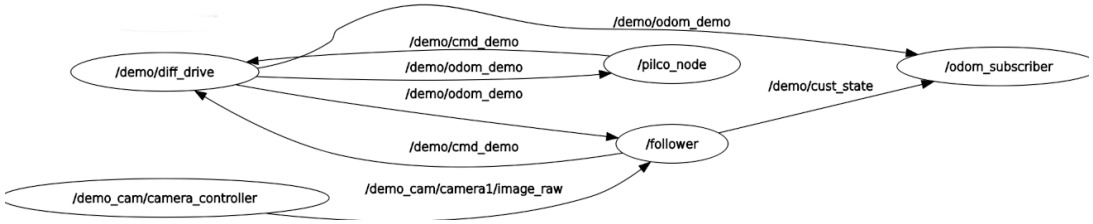


Figure A.2: Nodes with their interaction from *rqt_graph*

APPENDIX B

Install ROS2, Gazebo on Ubuntu

1. Install ROS2 on ubuntu 22.04.

```
sudo locale-gen en_US en_US.UTF-8

sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8


sudo apt update && sudo apt install curl gnupg2 lsb-release

curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc

sudo apt-key add - sudo sh -c 'echo "deb [arch=$(dpkg --print-architecture)]

http://packages.ros.org/ros2/ubuntu $(lsb_release -cs) main" >

/etc/apt/sources.list.d/ros2.list'

sudo apt install ros-iron-desktop
```

2. Install Gazebo

Ensure that your Ubuntu 22.04 system is up to date by running

```
sudo apt update && sudo apt upgrade
```

Gazebo is often installed as part of the ROS distribution, but you might need to install it separately if it's not included. You can install Gazebo on Ubuntu 22.04 by following the instructions on the Gazebo website or by using the package manager. Here are the general steps:

```
sudo apt install gazebo11
```