Investigation of the Bat Algorithm-based PID Controller onto a Robotic Arm Performance

ILMAN AIZAD BIN RIZAL MUAZZAM
(FB20091)

BACHELOR OF MECHATRONICS ENGINEERING (WITH HONOURS)

UNIVERSITI MALAYSIA PAHANG
AL-SULTAN ABDULLAH

UNIVERSITI MALAYSIA PAHANG AL-SULTAN ABDULLAH

DECLARATION OF THESIS AND COPYRIGHT

Author's Full Name	:	ILMAN AIZAD BIN RIZAL MUAZZAM
Date of Birth	:	13TH SEPTEMBER 1999
Title	:	INVESTIGATE THE PERFORMANCE OF THE MABSA-PID CONTROLLER FOR THE ROBOTIC ARM DURING HANDLING THE INSPECTED OBJECTS IN THE INSPECTION WORKSTATION
Academic Session	:	

I declare that this thesis is classified as:

☐	CONFIDENTIAL	(Contains confidential information under the Official Secret Act 1997)*
☐	RESTRICTED	(Contains restricted information as specified by the organization where research was done)*
☐	OPEN ACCESS	I agree that my thesis to be published as online open access (Full Text)

I acknowledge that Universiti Malaysia Pahang Al-Sultan Abdullah reserves the following
rights:

1. The Thesis is the Property of Universiti Malaysia Pahang Al-Sultan Abdullah
2. The Library of Universiti Malaysia Pahang Al-Sultan Abdullah has the right to make copies of the thesis for the purpose of research only.
3. The Library has the right to make copies of the thesis for academic exchange.

Certified by:

_____	_____
(Student's Signature)	(Supervisor's Signature)
_____	_____
New IC/Passport Number	Name of Supervisor
Date:	Date:

NOTE : * If the thesis is CONFIDENTIAL or RESTRICTED, please attach a thesis declaration letter.

THESIS DECLARATION LETTER

Librarian,
Universiti Malaysia Pahang Al-Sultan Abdullah,
Lebuh Persiaran Tun Khalil Yaakob,
26300, Gambang, Kuantan, Pahang.

Dear Sir,

CLASSIFICATION OF THESIS AS RESTRICTED

Please be informed that the following thesis is classified as RESTRICTED for a period of three (3) years from the date of this letter. The reasons for this classification are as listed below.
Author's Name
Thesis Title

Reasons (i)

(ii)

(iii)

Thank you.

Yours faithfully,

_____
(Supervisor's Signature)

Date:

Stamp:

Note: This letter should be written by the supervisor and addressed to the Librarian, Universiti Malaysia Pahang Al-Sultan Abdullah with its copy attached to the thesis.

SUPERVISOR'S DECLARATION

I/We* hereby declare that I/We* have checked this thesis/project* and in my/our* opinion, this thesis/project* is adequate in terms of scope and quality for the award of the degree of *Doctor of Philosophy/ Master of Science.

_____

(Supervisor's Signature)

Full Name : Nafrizuan Bin Mat Yahya

Position : Senior Lecturer

Date : 16 June 2024

STUDENT'S DECLARATION

I hereby declare that the work in this thesis is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at Universiti Malaysia Pahang Al-Sultan Abdullah or any other institutions.

_____

(Student's Signature)

Full Name : ILMAN AIZAD BIN RIZAL MUAZZAM

ID Number : FB20091

Date : 16 June 2024

Investigation of the Bat Algorithm-based PID Controller onto a Robotic Arm Performance

ILMAN AIZAD BIN RIZAL MUAZZAM

Thesis submitted in fulfillment of the requirements
for the award of the degree of
Bachelor of Mechatronics Engineering

Faculty of Manufacturing and Mechatronic Engineering Technology

UNIVERSITI MALAYSIA PAHANG AL-SULTAN ABDULLAH

JUNE 2024

ACKNOWLEDGEMENTS

First and foremost, I would like to thank you to Allah SWT for his blessing upon this study, without it, this study wouldn't have been completed. Next, I would like to express my deepest gratitude to my supervisor Dr. Nafrizuan Bin Mat Yahya for his guidance, tolerance, patience and support, time after time in accomplishing this study. Not to forget, thank you to all my faculty member, my colleagues and my helpful academic advisor Madam Suraya Binti Sulaiman whom have always supported with her words of encouragement and giving me a helping hand to finish this study. Lastly, my utmost gratitude goes to my family, my mother whom has never stop believing in me since day one, and encourage me relentlessly. Friends, especially Ku Irfan Bin Ku Azman that have been companying me day and night on the final weeks of finishing this final year project, without them this study would have been impossible to finish. I would like to dedicated this study to them who have been directly and indirectly involve in accomplish this study. I am truly thankful for this unwavering support and guidance that have nurtured me into a better person that can give back to the community.

ABSTRACT

Industrial and research robotic arms require accurate and reliable trajectory tracking. The Proportional-Integral-Derivative (PID) controller remains a standard choice for joint control due to its simplicity, but its performance strongly depends on well-tuned gains. Conventional tuning approaches such as manual trial-and-error are time-consuming and often yield suboptimal results. This work investigates the application of a nature-inspired metaheuristic, the Bat Algorithm (BA), for automatic PID tuning in a 3-DOF robotic arm simulated in ROS 2. The methodology involves executing reference trajectories, recording joint responses, and evaluating key performance indices such as rise time, settling time, steady-state error (SSE), and overshoot. Three methods were compared: baseline (untuned gains), manual trial-and-error tuning, and BA-PID. Experimental results show that BA-PID achieved notable improvements in rise time, settling time, and steady-state error compared to the other methods, despite limitations from simplified robot modelling parameters. The findings highlight the potential of BA-PID as an efficient and adaptive approach to controller tuning, reducing manual effort while enhancing performance.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

LIST OF ABREVIATION

PID	- Proportional Integral Derivative
BA	- Bat Algorithm
ROS	- Robot Operating System
URDF	- Unified Robot Description Format

Chapter 1 Introduction

Industry has had a significant influence on the world we live in today during the revolution. We may go back to a time before Industry 4.0 brought about major changes, when activities needed to be completed with hard labour. The construction of large transportation vehicles, such cars, trains, airplanes, and ships, was the primary emphasis of the industry at the time. These produced commodities are facilitating long-distance connections between people and goods, which is promoting economic progress. Industry required exponential expansion after years of research and revolution to fulfil the needs of a global market that was changing quickly. This is where industry 4.0 manifests itself, presenting enormous automation and connection prospects.

A new phase of industrial automation known as "industry 4.0" uses digital technology to build autonomous, networked systems. (Industry 4.0: Forces of Change, 2017) It includes a number of technological innovations, such as the Internet of Things, robots, analytics, and artificial intelligence. (Industry 4.0: Forces of Change, 2017) Increasing manufacturing and other sectors' productivity, efficiency, and flexibility is the aim. (Dalenogare & colleagues, 2018) Industry 4.0 integrates horizontal and vertical production processes, enhances product connectivity, and makes use of cutting-edge technology to help businesses operate more efficiently. Industry 4.0 relies heavily on the Internet of Things since it allows machines and gadgets to cooperate and interact with one another. This raises the general industrial processes' efficacy and efficiency.

Robotic arm technology has advanced significantly as a result of Industry 4.0. Over time, robotic arms have developed into more sophisticated tools with more dexterity, range of motion, and skills that surpass human capabilities. A robotic arm's ability to move precisely and effectively enables automation and optimization across a range of sectors, including manufacturing, healthcare, and inspection. When compared to conventional machining centers or human labor, these robotic arms provide benefits like flexibility, reproducibility, and cost-effectiveness that have the potential to completely transform the way operations are completed. Ensuring the best functioning of robotic arms is a significant difficulty in the industrial setting.

Implementing a PID controller for the robotic arm is one way to tackle this problem. PID gives the robotic arm feedback control, enabling it to precisely modify its motions in response to

input and the intended result (Barbosa et al., 2019). One particular kind of PID controller that uses model-based adaptive control methods to improve robotic arm performance while manipulating examined items in the inspection workstation is the BA-PID controller. In order to maintain optimal performance in real-time, this controller uses a model of the robotic arm and its surroundings to continually update its control settings. The robotic arm's performance when handling examined objects in the inspection workstation may be greatly enhanced by integrating the BA-PID controller.

The purpose of this study is to evaluate how well a robotic arm equipped with an BA-PID controller handles examined items in an inspection workstation. The robotic arm's handling and control skills are enhanced by the employment of a Proportional-Integral-Derivative controller in combination with the Bat Algorithm (BA).

1.2 Problem Statement

3 Degree of Freedom (Dof) robotic arm in local work station is not in peak performance.

1.3 Objective

- Formulate a 3 degree of freedom robotic model that represent the real robotic arm for a credible simulation.
- Integrating Bat Algorithm (BA) optimization in the PID into the robotic arm.
- Simulate the movement of the robotic arm with multiple trajectories and compare its PID results against multiple tuning method via computer simulation.

1.4 Scope of study

The scope of this study is to establish the boundaries of work and define the evaluation framework for robotic arm control. The project is conducted using the Robot Operating System 2 (ROS 2) environment, where a 3-DOF robotic arm model is simulated with ros2_control, Gazebo, and RViz. Within this framework, joint trajectories are executed and transient response data is collected for performance evaluation. The analysis focuses on three control strategies: baseline PID gains, manually tuned PID (trial-and-error), and Bat Algorithm-based PID (BA-PID). Key performance indicators considered in the comparison include rise time, settling time, steady-state error (SSE), and overshoot. By applying the BA-PID tuning method, the study aims to demonstrate improvements in accuracy and responsiveness compared to conventional PID tuning. The scope is limited to simulation-based validation and does not extend to physical hardware implementation.

Chapter 2 Literature Review

2.0 Introduction

This chapter reviews relevant literature concerning the application of control strategies and optimization methods for robotic arm systems. The primary focus is on the performance of PID controllers and their enhancement through nature-inspired optimization techniques, specifically the Bat Algorithm (BA). Numerous studies have highlighted the effectiveness of PID controllers in industrial robotics, as well as the limitations of manual tuning when applied to dynamic systems. Consequently, optimization-based approaches have been explored to achieve better accuracy, stability, and efficiency (Cosenza & Miccio, 2020; Abubar et al., 2019; Fajaffri & Yahya, 2023; Ariss & Rabat, 2019).

The review begins with an overview of robotic arm development and its role in industrial applications. It then examines the fundamentals of PID control and its significance in trajectory tracking for robotic manipulators. Finally, it discusses the integration of PID with optimization methods such as the Bat Algorithm to improve transient response performance. This provides the foundation for evaluating and comparing baseline PID, manually tuned PID, and BA-PID within the ROS 2 simulation environment used in this study.

2.1 Robotic Arm Overview

The development of robotic arms has been closely tied to industrial progress, where the demand for precision, efficiency, and safety has steadily grown. In the early stages of industrialization, repetitive and hazardous tasks were performed manually by skilled workers. However, despite training, human labor remained prone to fatigue, inconsistency, and accidents. This created a strong incentive to automate such tasks. Robotic arms emerged as an effective solution, capable of performing high-precision, labor-intensive, and dangerous operations with greater consistency and safety (Allagui et al., 2021).

The first widely recognized industrial robotic arm, the Unimate, was introduced in 1961 to automate welding and material handling in automotive manufacturing. It demonstrated the potential of robots to carry out repetitive and hazardous jobs with speed and accuracy. Since then, robotic arms have evolved significantly — from simple hydraulic manipulators to modern, multi-degree-of-freedom arms equipped with advanced sensors, controllers, and software integration. These advances have made them indispensable in diverse applications such as assembly, painting, medical surgery, and inspection tasks (Perez & McCarthy, 2005; Apriaskar et al., 2020).

With the progression toward **Industry 4.0**, robotic arms have become central to smart manufacturing, where connectivity, automation, and data integration drive production efficiency. Unlike manual labor, robotic arms can operate continuously without fatigue, offering repeatability and precision at scales humans cannot match. Their adaptability allows them to function in dangerous environments, handle toxic materials, or perform delicate tasks such as placing micro-components on circuit boards (Yasar et al., 2016).

From a design perspective, robotic arms are modeled as **kinematic chains** — a sequence of links connected by joints. To standardize the mathematical description of these linkages, the **Denavit–Hartenberg (D–H) parameters** are widely used. The D–H convention defines four parameters (θ, d, a, α) that systematically describe the relative orientation and position of each link. Through transformation matrices, forward kinematics (FK) can be used to compute the end-effector's pose from joint parameters, while inverse kinematics (IK) solves for the required joint parameters given a desired end-effector pose. These computations are fundamental to robotic arm motion control and are critical in tasks requiring high precision such as surgery, assembly, or inspection (Yasar et al., 2016; Jia et al., 2017).

2.1.1 Forward Kinematics:

The process of figuring out a robot's end-effector's location and orientation, or pose, given all the joint parameters—such as angles for prismatic joints and lengths for revolute joints—is known as forward kinematics. When utilizing FK, one begins at the robot's base and works their way outward to the end-effector, determining the location of the subsequent joint by looking at the location and orientation of the joints that came before it. The calculations are typically done using a series of transformation matrices, which translate and rotate coordinate frames from the base to

the end-effector. The result is a transformation matrix that describes the pose of the end-effector relative to the robot's base frame. (Forward kinematics, 2020) (n.d)

2.1.2 Inverse Kinematics:

Inverse kinematics is the process of calculating the joint parameters necessary to place the robot's end-effector at a specific position and orientation. It is, in a sense, the reverse problem of FK. Because of the possibility of many valid sets of joint parameters that place the end-effector in the same position and orientation—a phenomenon known as kinematic redundancy—this is frequently more difficult than FK. When a robot has to engage with items or carry out certain duties at predetermined places, IK is crucial. When closed-form solutions are impractical, numerical techniques are frequently used to solve IK issues. (Shi and others, 2020)

These kinematic computations are what control systems in robotics utilize to tell the robot what to perform. While FK is simple and has a unique solution, IK may not be, necessitating thorough assessment of the intended application and the robot's mobility limitations. Designing efficient robotic systems requires an understanding of both FK and IK, especially for jobs requiring precise control, like painting, surgery, or assembly (Yasar et al., 2016; Handbook of Robotics Chapter 1: Kinematics, n.d.; Jia et al., 2017).

2.1.3 Joints in Robotic Arm

The functionality of robotic arms depends heavily on the **types of joints and actuators** employed. Common joint types include revolute joints (rotation), prismatic joints (linear motion), spherical joints (multi-axis rotation), and cylindrical joints (rotation + translation). Each configuration offers different degrees of freedom (DOF), shaping the arm’s range of motion and capabilities. Actuators — electric, hydraulic, or pneumatic — power these joints. Electric servo motors are widely used due to their precision and controllability, while hydraulic systems are preferred in high-force applications (Yasar et al., 2016).

Revolute Joint (R)

Similar to the human elbow or shoulder, a revolute joint allows for rotation along a single axis, enabling a swinging motion in a plane. This kind of joint is essential to robotic arms when

rotational movement is required as it is frequently utilized in applications needing a broad range of motion.

Prismatic Joint (P)

Prismatic joints are characterized by sliding motion as opposed to rotating motion, allowing for linear movement along an axis. It resembles the sliding action of a pneumatic cylinder or the action of opening a drawer. Prismatic joints offer crucial flexibility and accuracy in industrial applications, and are frequently used in scenarios requiring exact linear placement.

Continuous Joint

A continuous joint is a unique kind of revolving joint that has no set beginning or ending points and may rotate 360 degrees (Yasar et al., 2016). Its capacity to rotate freely is especially helpful in scenarios where the robot must perform a variety of jobs with more freedom of movement.

Spherical Joint (S)

The connected portion of the robot can roll, pitch, and yaw thanks to the spherical joint's ability to rotate across multiple axes. With its great degree of rotational mobility, this joint is comparable to the ball-and-socket joint seen in the human hip or shoulder. Applications requiring intricate, multidirectional motions depend on spherical joints.

Cylindrical Joint (C)

With the axis of rotation perpendicular to the axis of translation, a cylindrical joint offers both rotational and linear motion. These joints offer a variety of movement options by combining elements of both prismatic and revolute joints (Yasar et al., 2016). Applications where both types of motion are necessary for maximum performance employ cylindrical joints.

Actuation of Joints

In a robotic arm, an actuator—which can be electric, hydraulic, or pneumatic—powers each joint. Because of its accuracy and controllability, servo motors are frequently used in electric actuators (Yasar et al., 2016). The actuator to use for a given application is determined by the

force, control, and precision required. For example, strong power duties may be best served by hydraulic actuators, but precise control activities are better served by electric actuators.

The kinematic chain of the arm is influenced by the arrangement of joints from the base to the end-effector, which affects the arm's range of motion, operating envelope, speed, and accuracy (Yasar et al., 2016). The way these joints are arranged dictates how well the robot can carry out different jobs. Many joint types are used in robotic arms to provide a variety of robot configurations that are suited to certain industrial or service applications (Yasar et al., 2016).

End-Effector

At the terminal point of the chain lies the **end-effector**, the part that interacts directly with the environment. Depending on the application, this can take the form of a simple gripper, a tool such as a welder or drill, or advanced medical instruments. Some systems incorporate sensors at the end-effector to provide real-time feedback, enhancing precision in inspection, sorting, or surgical tasks (Apriaskar et al., 2020).

In summary, robotic arms have progressed from basic manipulators to sophisticated, multi-functional systems that embody the principles of precision engineering, kinematics, and control. Their evolution reflects the broader trajectory of industrial automation — from mechanization in the 20th century to intelligent, adaptive systems driving today's Industry 4.0.

2.2 PID Controller Overview

2.2.1 Controller

In modern engineering, the concept of **control** is central to ensuring that dynamic systems behave in a desired manner. A controller is a device or set of algorithms that regulates, directs, or governs the actions of another system by continuously adjusting its inputs. Controllers are found in nearly all automated processes, from regulating temperature in air-conditioning systems to maintaining the stability of an aircraft in turbulent air (CTM: PID Tutorial, n.d.; Vilanova & Visioli, 2017).

In robotics and automation, controllers serve as the “brain” that interprets sensor data and translates it into actuator commands. For a robotic arm, this means receiving position or velocity feedback from each joint, comparing it with a target trajectory, and calculating motor torques to minimize the difference. Depending on complexity, controllers may range from simple on–off switches (bang-bang controllers) to advanced adaptive and intelligent control strategies (Doroshenko, 2017).

The impact of controllers in industry is profound. They enable:

- **Precision and accuracy**: ensuring repeatable high-quality operations such as welding, machining, or inspection.
- **Efficiency**: automating processes at maximum operating speeds, reducing cycle times.
- **Cost reduction**: lowering labor demands while minimizing energy use and material waste.
- **Consistency**: delivering uniform output without the variability of human operators.
- **Safety**: reducing human involvement in hazardous tasks.
- **Flexibility**: allowing quick reprogramming for different production requirements.
- **Data integration**: in Industry 4.0, controllers serve as nodes in a connected network, enabling predictive maintenance and real-time optimization (Barbosa et al., 2019).

This foundation sets the stage for one of the most widely used controllers in engineering — the **PID controller**.

2.2.2 Proportional Response

The proportional (P) controller is the simplest feedback control method, adjusting output proportionally to the error between a system’s setpoint and its actual output. The mathematical expression is:

$$u(t) = Kp \cdot e(t)$$

where u(t)u(t)u(t) is the control signal, KpK_pKp is the proportional gain, and e(t)e(t)e(t) is the error.

Increasing KpK_pKp makes the controller more responsive but risks oscillations if tuned too aggressively. Conversely, a small KpK_pKp may stabilize the system but result in sluggish response. A key limitation of pure proportional control is the presence of a **steady-state error (offset)**, since the control action diminishes as the error approaches zero (CTM: PID Tutorial, n.d.; Control Systems/Controllers and Compensators, 2019).

2.2.3 From P to PI,PD, and PID

To overcome the limitations of pure proportional control, additional terms are introduced:

- **Integral (I) action**: sums past errors over time, driving the steady-state error to zero. However, excessive integral action can lead to "integral windup," where accumulated error causes overshoot and instability.
- **Derivative (D) action**: predicts future trends of error by calculating its rate of change, helping to dampen oscillations and improve transient response. However, it is sensitive to noise in sensor measurements (Li et al., 2006).

Together, the proportional–integral–derivative (PID) controller balances immediate responsiveness (P), elimination of steady-state error (I), and smooth transient response (D).

The standard form of a PID controller is:

$$u(t) = Kpe(t) + Ki\int e(t)dt + Kd\frac{de(t)}{dt}$$

where Kp, Ki, Kd are the proportional, integral, and derivative gains respectively.

2.2.4 PID Tuning Methods: Ziegler-Nichols

The effectiveness of a PID controller depends on appropriate tuning of its gains. One of the earliest and most widely used heuristic methods is the **Ziegler–Nichols (Z–N) tuning method**, introduced in the 1940s. It provides empirical rules for calculating gains based on system response, either through the **open-loop process reaction curve method** or the **closed-loop ultimate gain method**.

- In the **open-loop method**, a step input is applied, and process parameters such as time delay and time constant are measured. PID gains are then set according to Z–N formulas.
- In the **closed-loop method**, the system is driven to sustained oscillation by increasing proportional gain until the ultimate gain (KuK_uKu) and oscillation period (PuP_uPu) are identified. The Z–N rules then prescribe:
- $Kp = 0.60Ku, Ti = Pu/2, Td = Pu/8\ for\ PID$

While Z–N is simple and effective for many processes, it tends to produce aggressive tuning, sometimes leading to overshoot or instability. In safety-critical or precision applications (such as robotic manipulators), further fine-tuning or modern optimization methods are preferred (Doroshenko, 2017).

2.2.5 Transfer Function and System Response

The dynamics of systems controlled by PID can be mathematically described using **transfer functions** in the Laplace domain. A transfer function expresses the relationship between system input and output as a ratio of polynomials:

$$H(s) = \frac{Y(s)}{X(s)}$$

where H(s) is the transfer function, X(s) the Laplace transform of the input, and Y(s) the Laplace transform of the output (Żak, 2016).

Transfer functions allow engineers to predict how a system responds to different inputs, analyze stability, and design controllers without directly solving differential equations.

Key performance indicators derived from transfer function analysis include **transient response metrics**: rise time, settling time, peak time, overshoot, and steady-state error. These parameters provide a quantitative measure of controller effectiveness in driving a system toward stability (Żak, 2016).

2.3 PID Controller in Robotics

In robotics, PID controllers remain the **workhorse of feedback control** due to their balance of simplicity and effectiveness. They are widely applied to regulate joint position, velocity, and torque in robotic manipulators. For a robotic arm, PID ensures that each joint follows the desired trajectory while minimizing oscillations, overshoot, and steady-state error (Lekkala & Mittal, 2014).

- **Proportional control** provides fast corrective action.
- **Integral control** ensures accurate positioning by eliminating offsets caused by load disturbances.
- **Derivative control** stabilizes motion by anticipating rapid changes in error.

Different configurations of PID (PI-only or PD-only) are also used in practice depending on system requirements. PI controllers are common in processes prioritizing steady-state accuracy over speed, while PD controllers are useful in systems requiring fast transient response with minimal oscillations (Sinha et al., 2015; Li et al., 2006).

Despite its effectiveness, PID tuning for robotic arms can be challenging due to nonlinear dynamics, varying loads, and coupled joint behavior. This has motivated researchers to explore **intelligent optimization algorithms** (e.g., swarm intelligence, genetic algorithms, fuzzy logic) to automatically tune PID parameters beyond traditional heuristic approaches. These advancements seek to enhance precision, adaptability, and robustness in robotic control.

In summary, the PID controller's enduring popularity in robotics comes from its versatility: it can be applied across diverse systems, easily implemented, and tuned to balance performance requirements. However, its limitations — particularly in tuning and adaptability — have opened

pathways for optimization-based methods, such as the **Bat Algorithm**, which this study investigates.

2.3 Bat Algorithm Overview (BA)

2.3.1 Artificial Intelligence and Swarm Intelligence

Artificial Intelligence (AI) has emerged as one of the most transformative fields in modern computing, offering problem-solving capabilities that extend beyond conventional algorithmic methods. Within AI, a significant branch known as **computational intelligence** deals with solving optimization problems inspired by natural systems. Among its approaches, **swarm intelligence** has attracted growing attention due to its ability to mimic the collective behavior of social organisms to tackle complex, nonlinear, and high-dimensional problems (Chakraborty & Kar, 2017).

Swarm intelligence is defined as the collective problem-solving behavior of decentralized, self-organized systems, typically inspired by biological populations such as ants, bees, fish schools, and bird flocks. These systems demonstrate key properties such as **adaptability, robustness, parallelism, and emergent intelligence**, making them suitable for optimization tasks where conventional mathematical techniques struggle. Notably, swarm-based methods are less dependent on gradient information, enabling them to solve non-convex and discontinuous problems that appear in engineering, control systems, and robotics (Dorigo et al., 2000; Kennedy & Eberhart, 1995).

2.3.2 Animal-Inspired Metaheuristics

Animal-inspired optimization algorithms form a subset of swarm intelligence approaches. They capture the hunting, foraging, or navigational strategies of animals and translate them into computational search techniques. Well-known examples include:

- **Ant Colony Optimization (ACO):** models ant foraging and pheromone communication.
- **Particle Swarm Optimization (PSO):** simulates flocking of birds or schooling of fish.
- **Grey Wolf Optimizer (GWO):** mimics cooperative hunting strategies of wolf packs.

- **Whale Optimization Algorithm (WOA):** inspired by bubble-net feeding behavior of humpback whales.

These algorithms share a common goal: to balance **exploration** (searching broadly across the solution space) with **exploitation** (refining the best solutions found). Among them, the **Bat Algorithm (BA)** stands out for its unique use of **echolocation-inspired mechanics**, providing adaptive exploration and exploitation mechanisms suitable for PID controller tuning.

2.3.3 Biological Inspiration of the Bat Algorithm

The Bat Algorithm was first proposed by Xin-She Yang in 2010, inspired by the **echolocation behavior of microbats** (Yang, 2010). Echolocation enables bats to emit ultrasonic pulses and interpret returning echoes to estimate distance, detect prey, and navigate in darkness. Key biological features that inspired the algorithm include:

- **Frequency tuning:** Bats modulate their pulse frequency depending on hunting context, which is mirrored in BA as a mechanism to control exploration step size.
- **Velocity and position update:** Similar to flight trajectories, each artificial bat adjusts its position and velocity in the search space.
- **Loudness and pulse emission rate:** As bats approach prey, their loudness decreases while pulse emission increases, allowing more precise local search.

By mapping these behaviors into optimization variables, BA models the trade-off between exploration and exploitation in a natural, self-adaptive manner (Yang & He, 2013).

2.3.4 Core Mechanics of the Bat Algorithm

The Bat Algorithm is a swarm intelligence-based optimization algorithm inspired by the echolocation behavior of bats. In echolocation, bats send out sound waves that strike potential prey or obstacles and reflect back to them. The bats use the delays and alterations in these echoes to navigate and hunt in the dark.(Yang, 2010)

The Bat Algorithm was developed to solve optimization problems by mimicking this biological echolocation process. It models bats with a virtual position and velocity in a multidimensional space where the solution to the problem lies. The bats "fly" through the solution space, each with a frequency (related to the velocity), loudness, and pulse emission rate. The frequency controls the velocity of the bat, which dictates how it explores the solution space, while the loudness and pulse rate adjust adaptively as the algorithm progresses, balancing exploration (searching new areas) and exploitation (refining current solutions)(Yang & He, 2013).

Initializing a population of bats, creating new solutions by varying their frequencies and velocities, and carrying out local searches around the best solution thus far are the primary processes in the basic BA. Bat locations and velocities are updated iteratively, guided by their personal experience as well as the collective experience of the swarm, to achieve optimization. (Yu et al., 2019)

Bats gradually refine their parameters to "hone in" on near-optimal or ideal outcomes. Because of its simplicity and efficiency, the Bat Algorithm has been effectively applied to a wide range of optimization issues across diverse areas. (Chen et al., 2018) (Meng et al., 2015).

To overcome optimization challenges, the Bat Algorithm was created to mimic these echolocation features. Here's a more detailed explanation of how it operates.:

Echolocation

The method imagines artificial bats that can distinguish between barriers in the optimization landscape and food/prey (optimal solutions) by using echolocation to sense distance.

Frequency and Velocity

Every bat in the algorithm has a position, a velocity, a pulse frequency, and a changeable pulse emission rate. The frequency of the bats determines their random flight speed, which they then modify to reach the prey or solutions they sense.

Global Search

New candidate solutions are generated by flying towards the current global best solution.

Local Search

With a probability determined by pulse rate, bats perform local random walks around the best solution, refining accuracy.

Loudness and Pulse Emission

The bat's pulses get quieter as it approaches its prey, and its rate of emission rises as it comes closer. This mimics how echolocation gets stronger when a bat gets closer to its target.

Termination

The algorithm iteratively updates solutions until a stopping criterion (e.g., maximum iterations or acceptable error) is met.

Based on benchmark issues and real-world engineering designs, simulations and comparisons have shown the efficacy, efficiency, and stability of the Bat Algorithm. It has demonstrated the ability to provide high-quality results in a reasonable period of time, and it excels at resolving non-convex and non-differentiable optimization issues. (Meng et al., 2015). The BA and its variations have been used in many different domains, including as energy systems, image processing, scheduling, and electricity markets, demonstrating its adaptability and promise for a broad range of (Meng et al., 2015).

2.3.6 Applications of the Bat Algorithm

Since its introduction, BA has been applied successfully in diverse fields:

- **Engineering optimization:** structural design, scheduling, and resource allocation (Meng et al., 2015).
- **Power and energy systems:** load dispatch, renewable integration, and fault detection (Chen et al., 2018).
- **Medical imaging:** image reconstruction and segmentation.
- **Robotics and control:** trajectory planning, system identification, and controller parameter tuning.

Its robustness against local minima, ability to converge quickly, and low number of control parameters make BA suitable for real-time engineering applications.

2.3.7 Bat Algorithm for PID Controller Tuning

In control engineering, particularly for nonlinear or time-varying systems, classical tuning methods like Ziegler–Nichols often fail to deliver consistent performance. BA addresses this by directly optimizing PID parameters (**Kp, Ki, Kd**) against a defined objective function, often based on performance indices such as Integral of Squared Error (ISE), settling time, or steady-state error.

The workflow typically involves:

1. Defining the optimization problem: minimizing transient response metrics (rise time, overshoot, settling time, SSE).
2. Running BA to iteratively adjust PID gains.
3. Selecting the best set of gains from the global best solution.

This metaheuristic tuning method provides **faster convergence, robustness to disturbances, and better performance trade-offs** compared to trial-and-error or heuristic-based tuning.

Chapter 3 Methodology

This chapter presents the methodology adopted to evaluate the performance of a robotic arm under different control strategies. A systematic and structured approach was employed to ensure the research could be replicated and that the results would be both reliable and comparable across methods. The methodology begins with the **modeling of the robotic arm in the Robot Operating System 2 (ROS2) environment**, where the manipulator is defined in URDF format and integrated with simulation tools such as Gazebo and RViz. This step establishes a realistic virtual platform to study robotic dynamics without the risks and costs of physical prototyping.

The second stage involves **controller implementation**, starting with a conventional baseline PID controller, followed by manual trial-and-error tuning to approximate practical industry practice. This provides reference points against which optimization results can be compared. The final controller design integrates the **Bat Algorithm (BA)**, a nature-inspired optimization method, to tune PID gains automatically. By embedding BA within the control structure, the study leverages swarm intelligence to overcome the limitations of manual tuning.

Once the controllers are in place, the methodology proceeds to **trajectory execution and data collection**. Predefined step-stagger trajectories are applied to the robotic arm to simulate transient responses, and the system's outputs (joint angles, errors, and velocities) are logged. These datasets are then exported as CSV files to facilitate detailed post-processing.

The step-stagger trajectory method was selected as the primary excitation input for the robotic arm because it provides a clear and structured way to evaluate transient response characteristics. Unlike sinusoidal or arbitrary trajectories, which emphasize steady-state tracking or introduce unnecessary complexity, the step-stagger approach applies sequential step inputs to each joint. This allows the response of every joint to be isolated and analysed in terms of rise time, settling time, overshoot, and steady-state error without interference from simultaneous motions. Step responses are widely regarded in control engineering as a classical benchmark for assessing system stability and performance, making them highly suitable for comparative studies between different controllers. Furthermore, the staggered design ensures that the trajectories remain simple,

repeatable, and easy to implement in ROS2 while still reflecting realistic actuation scenarios for a multi-joint robotic arm.

Finally, the research focuses on **performance evaluation and comparison**. Key transient response metrics—such as rise time, settling time, overshoot, and steady-state error—are extracted from the collected data. Both **graphical analysis**, using overlay plots, and **quantitative analysis**, using tabulated performance metrics, are performed to highlight differences among controllers. This structured evaluation not only demonstrates the relative effectiveness of BA-PID but also provides insights into the strengths and weaknesses of traditional PID tuning methods.

3.1 Research Framework

The research framework for this study consists of four main stages: system modelling in ROS2, controller implementation, optimization using the Bat Algorithm, and performance evaluation. Each stage is designed to build upon the previous one, ensuring a structured and repeatable methodology.

The overall research framework is divided into four stages:

- **System Modeling in ROS2:**

  The first stage involves developing a virtual model of the robotic arm in ROS2. A URDF (Unified Robot Description Format) was created to define the kinematic and dynamic properties of the arm, including its links, joints, and inertial parameters. This model was integrated with Gazebo for physics-based simulation and RViz for visualization. The modelling stage ensures that the robotic arm's motion and behaviour reflect realistic physical constraints, providing a reliable environment for testing control strategies.

- **Controller Development:**

  In the second stage, control strategies were applied to the robotic arm. A baseline PID controller was first configured using trial-and-error tuning to provide reference performance. This was followed by the implementation of the BA-PID controller, in which the Bat Algorithm dynamically tuned PID parameters to optimize system response. The inclusion of multiple controllers enables a direct comparison of classical tuning and bio-inspired optimization methods.

- **Optimization with Bat Algorithm (BA):**

  The third stage focuses on applying the Bat Algorithm to optimize the PID gains (proportional, integral, and derivative). Inspired by the echolocation behaviour of bats, the algorithm balances exploration and exploitation of the search space to minimize a defined cost function. The cost function in this study was derived from transient response metrics, including settling time, rise time, overshoot, and steady-state error. By running iterative optimization cycles, the BA was able to converge on gain values that improved robotic arm performance over manually tuned controllers.

- **Trajectory Execution & Data Collection:**

  In this stage, the robotic arm is subjected to predefined trajectories within the ROS2 simulation environment to evaluate its transient response characteristics. A **step-stagger trajectory** is selected, where each joint is commanded sequentially with specific angular displacements and holding times. This method ensures that the performance of each joint can be observed in isolation while still maintaining interaction with the full kinematic chain. During execution, the system logs critical data streams, including **joint positions, velocities, and tracking errors**, which represent the deviation between the reference input and the actual joint response. These signals are essential for characterizing controller performance, as they directly reveal how quickly and accurately the system reaches its target state. The recorded data is stored in ROS2 bag files and subsequently exported into CSV format for post-processing. This structured dataset enables quantitative evaluation of key transient response parameters—such as rise time, settling time, steady-state error, and overshoot—under different control strategies.

- **Performance Evaluation:**

  The final stage involves evaluating and comparing the performance of each control strategy. Predefined step-stagger trajectories were executed on the robotic arm, and data such as joint angles, errors, and velocities were logged. These outputs were exported as CSV files and analysed in MATLAB to generate quantitative performance metrics and comparative plots. Key indicators such as rise time, settling time, overshoot, and steady-

state error were used to evaluate the controllers. By comparing baseline PID, trial-and-error PID, and BA-PID, the study identified the extent of performance improvement achieved by optimization.

3.2 Robotic Arm Modeling in ROS2

The robotic arm developed in this project is a **3-DOF manipulator**, designed and modelled within the **Robot Operating System 2 (ROS2) Humble** framework. The manipulator consists of a base link, two intermediate revolute joints, and an end-effector link, providing sufficient degrees of freedom for fundamental pick-and-place or inspection tasks while maintaining modelling simplicity. The robot model was constructed using the **Unified Robot Description Format (URDF)**. This structure defines the physical properties of the arm, including link geometries, joint types, inertial parameters, and transmission elements, which form the basis for realistic simulation. Below is the URDF coding.

```
<?xml version="1.0"?>
<robot name="robotic_arm">

 <!-- Materials -->
 <material name="blue"><color rgba="0 0 1 0.8" /></material>
 <material name="red"><color rgba="1 0 0 0.8"/></material>
 <material name="green"><color rgba="0 1 0 0.8" /></material>
 <material name="white"><color rgba="1 1 1 0.8" /></material>
 <material name="grey"><color rgba="0.7 0.7 0.7 0.8" /></material>

 <!-- Base -->
 <link name="base_link">
      <visual><geometry><box size="0.4 0.4 0.05"/></geometry><origin xyz="0 0 0.025"/></visual>
      <collision><geometry><box size="0.4 0.4 0.05"/></geometry><origin xyz="0 0 0.025"/></collision>
```

```
    <inertial><origin xyz="0 0 0.025"/><mass value="5.0"/><inertia ixx="0.0677" ixy="0" ixz="0" iyy="0.0677" iyz="0" izz="0.1333"/></inertial>
  </link>
  <gazebo reference="base_link"><static>true</static></gazebo>

  <!-- Link 1 -->
  <link name="link1">
     <visual><geometry><cylinder radius="0.1" length="0.1"/></geometry><origin xyz="0 0 0.05"/></visual>
    <collision><geometry><cylinder radius="0.1" length="0.1"/></geometry><origin xyz="0 0 0.05"/></collision>
    <inertial><origin xyz="0 0 0.05"/><mass value="1.5"/><inertia ixx="0.005" iyy="0.005" izz="0.0075" ixy="0" ixz="0" iyz="0"/></inertial>
  </link>

  <!-- Link 2 -->
  <link name="link2">
    <visual><geometry><cylinder radius="0.04" length="0.4"/></geometry><origin xyz="0 0 0.2"/></visual>
   <inertial><origin xyz="0 0 0.2"/><mass value="0.8"/><inertia ixx="0.011093" iyy="0.011093" izz="0.000853" ixy="0" ixz="0" iyz="0"/></inertial>
  </link>

  <!-- Link 3 -->
  <link name="link3">
    <visual><geometry><cylinder radius="0.03" length="0.4"/></geometry><origin xyz="0 0 0.2"/></visual>
      <collision><geometry><box size="0.06 0.06 0.4"/></geometry><origin xyz="0 0 0.2"/></collision>
   <inertial><origin xyz="0 0 0.2"/><mass value="0.6"/><inertia ixx="0.00818" iyy="0.00818" izz="0.00036" ixy="0" ixz="0" iyz="0"/></inertial>
```

```
  </link>

  <!-- Link 4 -->
  <link name="link4">
    <visual><geometry><sphere radius="0.03"/></geometry><origin xyz="0 0 0.03"/></visual>
        <inertial><origin xyz="0 0 0.03"/><mass value="0.2"/><inertia ixx="0.000072"
iyy="0.000072" izz="0.000072" ixy="0" ixz="0" iyz="0"/></inertial>
  </link>

  <link name="ee_link"/>

  <!-- Joints -->
  <joint name="joint1" type="continuous">
    <parent link="base_link"/><child link="link1"/><origin xyz="0 0 0.05"/><axis xyz="0 0 1"/>
  </joint>
  <joint name="joint2" type="revolute">
    <parent link="link1"/><child link="link2"/><origin xyz="0 0 0.1"/><axis xyz="0 1 0"/>
    <limit lower="-1.57" upper="1.57" effort="20" velocity="1"/>
  </joint>
  <joint name="joint3" type="revolute">
    <parent link="link2"/><child link="link3"/><origin xyz="0 0 0.4"/><axis xyz="0 1 0"/>
    <limit lower="-1.57" upper="1.57" effort="20" velocity="1"/>
  </joint>
  <joint name="joint4" type="fixed">
    <parent link="link3"/><child link="link4"/><origin xyz="0 0 0.4"/>
  </joint>
  <joint name="joint5" type="fixed">
    <parent link="link4"/><child link="ee_link"/><origin xyz="0 0 0.06"/>
  </joint>

  <!-- ros2_control interfaces -->
```

```
  <ros2_control name="GazeboSystem" type="system">
    <hardware>
      <plugin>gazebo_ros2_control/GazeboSystem</plugin>
    </hardware>
    <joint name="joint1">
      <command_interface name="position"/>
      <state_interface name="position"/>
      <state_interface name="velocity"/>
    </joint>
    <joint name="joint2">
      <command_interface name="position"/>
      <state_interface name="position"/>
      <state_interface name="velocity"/>
    </joint>
    <joint name="joint3">
      <command_interface name="position"/>
      <state_interface name="position"/>
      <state_interface name="velocity"/>
    </joint>
  </ros2_control>

  <gazebo>
    <plugin name="gazebo_ros2_control" filename="libgazebo_ros2_control.so">
       <parameters>/home/ilmanaizad/ros2_ws/install/robotic_arm/share/robotic_arm/config/ros2_
controllers.yaml</parameters>
    </plugin>
  </gazebo>

</robot>
```

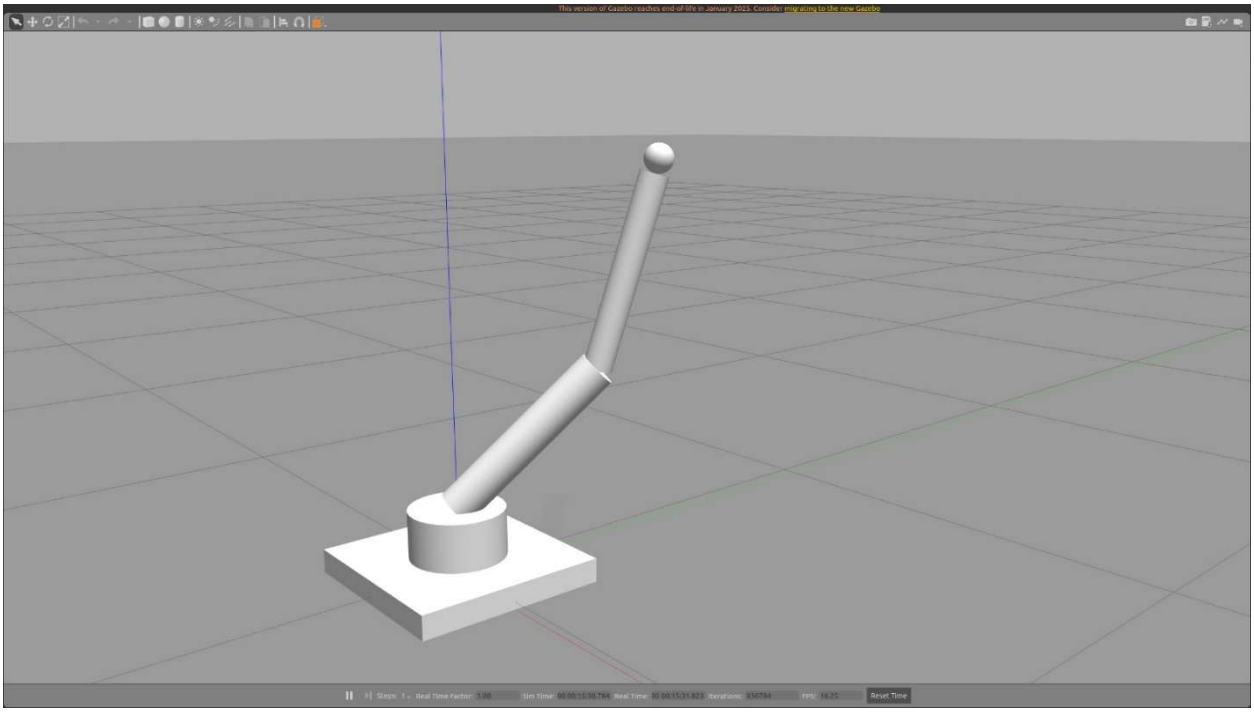


Figure 0.1 Gazebo Software

To enable experimentation in a virtual environment, the arm was deployed in **Gazebo Classic**, a physics-based simulator widely used in robotics research. Gazebo provides dynamic simulation of rigid-body motion, gravity, and joint constraints, allowing the robotic arm to exhibit realistic physical behaviours during trajectory execution. This ensures that controller performance is assessed under conditions that closely approximate real-world actuation and dynamics.

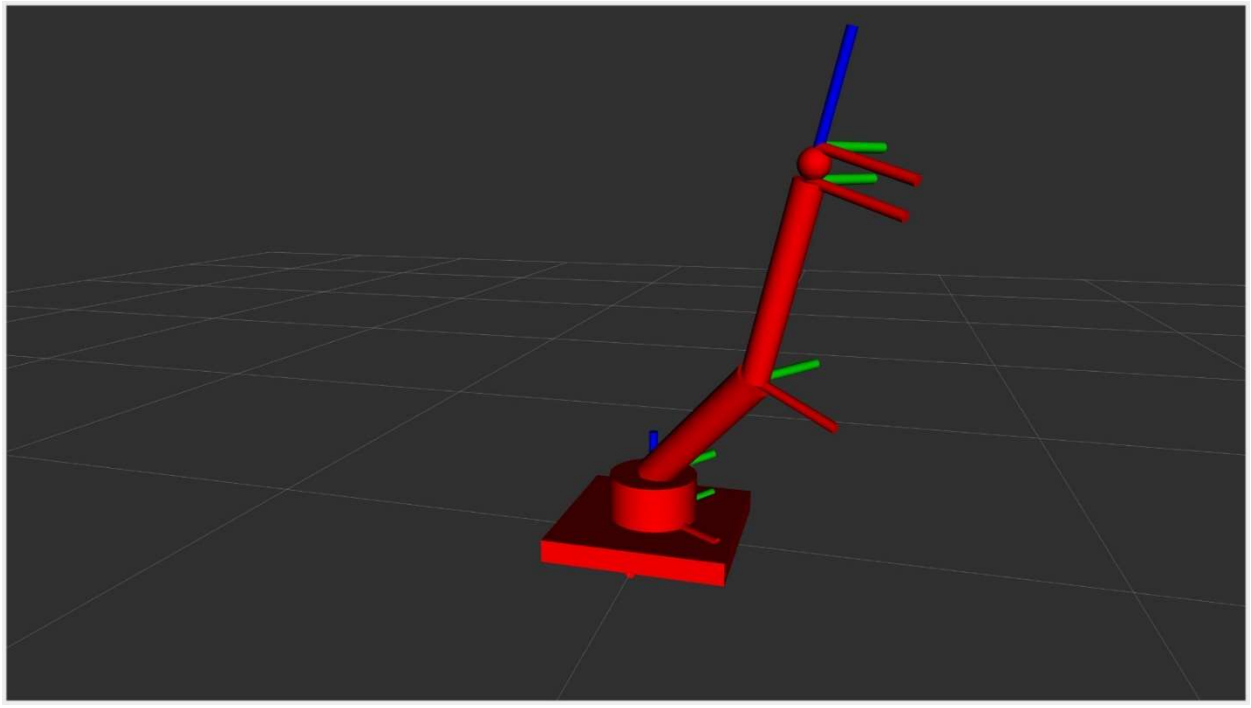


Figure 0.2 RViz Software

For visualization and debugging, the model was also integrated with **RViz**, which enables real-time observation of joint states, reference trajectories, and end-effector motion. This tool was critical during the design and testing phases to verify that commanded trajectories matched the expected kinematic behaviour.

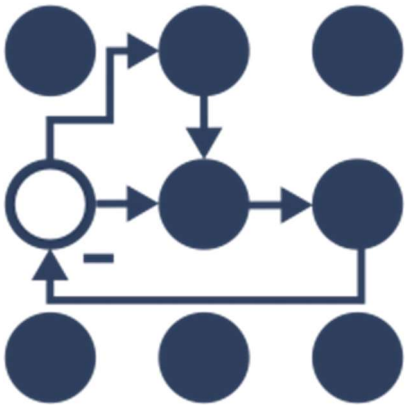


Figure 0.3 ros2_control

Control of the manipulator was facilitated through the **ros2_control** framework. The configuration employed the **joint_state_broadcaster**, which publishes real-time sensor feedback from each joint, and the **joint position controllers**, which receive command inputs to actuate joint angles. This setup replicates the interface between controllers and physical actuators in hardware, thereby providing a realistic platform for controller implementation and evaluation.

Through this integrated modeling environment, the robotic arm's **kinematics and dynamics** were effectively represented, enabling systematic testing of baseline PID controllers and the Bat Algorithm–optimized PID (BA-PID) under predefined trajectories.

3.3 Controller Implementation

3.3.1 Baseline PID Controller

The **Proportional-Integral-Derivative (PID) controller** is implemented as the baseline control strategy for the robotic arm in this study. The PID controller has been widely used in industrial applications due to its simplicity, robustness, and effectiveness in improving system stability and transient performance. Its control law is based on the error signal, which is defined as the difference between the desired joint position (setpoint) and the actual joint position measured from the simulation.

Mathematically, the control input u(t) generated by the PID controller can be expressed as:

$$u(t) = Kpe(t) + Ki\int e(t)dt + Kd\frac{de(t)}{dt}$$

where:

- e(t): error between desired and actual joint position,
- Kp: proportional gain (influences immediate error correction),
- Ki: integral gain (eliminates steady-state error),
- Kd: derivative gain (predicts error trend to dampen oscillations).

In the ROS2 environment, the PID controller was configured through the **ros2_control** framework by defining gain parameters for each joint inside the **ros2_controllers.yaml** configuration file. The controller subscribes to reference trajectories, computes the error at each control cycle, and generates torque/position commands to the joints accordingly.

```
controller_manager:
  ros__parameters:
    update_rate: 200

    joint_state_broadcaster:
      type: joint_state_broadcaster/JointStateBroadcaster

    joint_trajectory_controller:
      type: joint_trajectory_controller/JointTrajectoryController

joint_state_broadcaster:
  ros__parameters: {}

joint_trajectory_controller:
  ros__parameters:
    joints: [joint1, joint2, joint3]
    command_interfaces: [position]
    state_interfaces: [position, velocity]

    # --- Add these starting gains (conservative) ---
    gains:
      joint1: {p: 20.0, i: 0.0, d: 1.0, i_clamp_min: -0.0, i_clamp_max: 0.0}
      joint2: {p: 0.1,  i: 0.1, d: 0.5, i_clamp_min: -0.1, i_clamp_max: 0.1}
      joint3: {p: 15.0, i: 0.0, d: 1.5, i_clamp_min: -0.0, i_clamp_max: 0.0}
    allow_partial_joints_goal: false
```

ros2_controller.yaml

For this baseline case, the gains were selected through initial manual estimation, with proportional terms set to modest values to achieve stable performance without aggressive oscillations. Both integral and derivative terms were set conservatively to minimize steady-state error while avoiding instability due to integral windup or excessive noise amplification. This approach allows the baseline PID to serve as a **reference benchmark**, providing performance data (rise time, settling time, overshoot, and steady-state error) for later comparison with more advanced tuning strategies.

gains:

joint1: {p: 0.0, i: 0.0, d: 0.0, i_clamp_min: -0.0, i_clamp_max: 0.0}

joint2: {p: 0.0, i: 0.0, d: 0.0, i_clamp_min: -0.0, i_clamp_max: 0.0}

joint3: {p: 0.0, i: 0.0, d: 0.0, i_clamp_min: -0.0, i_clamp_max: 0.0}

Baseline

The baseline PID implementation establishes a foundation against which the effects of manual tuning and Bat Algorithm–based tuning can be evaluated, enabling a clear demonstration of the improvements achieved through optimization.

3.3.2 Bat Algorithm-Based PID (BA-PID)

The Bat Algorithm (BA) was applied as an optimization layer on top of the PID controller to develop an adaptive control strategy for the robotic arm. This approach leverages the strengths of swarm intelligence to overcome the limitations of manual or heuristic tuning. Unlike conventional PID tuning, which often depends on intuition or heuristic rules, BA-PID systematically searches the parameter space for near-optimal gain values, reducing trial-and-error and ensuring consistent performance.

Algorithm Inspiration and Principles

The Bat Algorithm is inspired by the echolocation behavior of microbats, which emit pulses of sound and listen to their echoes to detect prey and obstacles. This biological principle is translated into optimization as follows:

- **Frequency (f) and velocity (v) update:** determines how candidate solutions (bats) explore the solution space.
- **Pulse emission rate (r) and loudness (A):** adapt dynamically, allowing bats to switch between global exploration (searching new areas) and local exploitation (refining promising solutions).
- **Global best feedback:** bats adjust their movements based on both their personal position and the swarm's best solution, encouraging convergence toward the global optimum.

In this project, each bat represents a potential set of PID gains (Kp,Ki,Kd) for all three joints of the robotic arm. The swarm collectively searches for gain combinations that minimize transient response errors.

Integration in ROS2 Environment

The BA-PID optimization pipeline was implemented directly within the ROS2 Humble workspace (ros2_ws). The process can be summarized in the following stages:

Initialization:

- A population of candidate PID gain sets is randomly generated within specified bounds, defined in the BA tuner script (ba_pid_tuner.py).
- Example snippet from ros2_ws/src/robotic_arm/src/ba_pid_tuner.py:

N_BATS = 6

N_ITERS = 6

FMIN, FMAX = 0.0, 2.0

ALPHA, GAMMA = 0.9, 0.9

BOUNDS = { "p": (0.5, 40.0), "i": (0.0, 5.0), "d": (0.0, 3.0) }

This ensures search is constrained within realistic ranges while maintaining computational feasibility.

Trajectory Execution:

- For each candidate, the robotic arm executes a step-stagger trajectory (e.g., j2→j3→j1), where joints move sequentially with hold and return phases.
- The execution is handled by a trajectory client node (e.g., trajectory2_client.py), called automatically by the BA tuner script.

Data Recording:

- During execution, ROS2 bag files log the system's states: joint angles, velocities, and errors.
- These bags are converted into CSV files via a custom script (bag_to_csv.py), enabling compatibility with MATLAB and Python analysis.

Performance Evaluation:

- The CSV outputs are processed using analyze_and_export.py to extract **transient response metrics**: rise time, settling time, overshoot, and steady-state error (SSE).
- A cost function combines these metrics into a single performance score:

  $J = 0.45 \times tsettle + 0.35 \times trise + 0.15 \times SSE + 0.05 \times OvershootPenalty$

  Where overshoot above 2% is penalized.

BA Iterations:

- Candidate gains are updated iteratively using BA operators (frequency tuning, pulse rate, loudness adaptation).
- After each iteration, the best candidate is compared against the swarm, guiding future searches.

Convergence and Final Validation:

- After completing the defined number of iterations, the best-performing PID gains are selected.
- The robotic arm is then re-executed with these optimized gains, and a final bag + CSV analysis is performed to validate improvements against the baseline.

traj1_bapid_metrics_summary										
label	mode	window_mode	joint	target	rise_time	settling_time	overshoot	sse	window_start	window_end
Trajectory 1 | BA-PID	stagger	fixed	joint1	1	7.455	7.63	0.017269523	3.697497536	0	12
Trajectory 1 | BA-PID	stagger	fixed	joint2	0.8	8.45	8.625	0.011751296	3.041562064	5	18
Trajectory 1 | BA-PID	stagger	fixed	joint3	-0.6	8.45	13	0.015615673	1.966946026	11	24

Table 1 BA PID best results

Advantages of BA-PID

- **Automated optimization:** Eliminates manual tuning burden.
- **Adaptive balance:** Between exploration (finding new solutions) and exploitation (refining existing ones).
- **Flexibility:** Can handle nonlinear, multi-DOF dynamics of the robotic arm.

- **Quantitative validation:** Results are directly comparable with baseline and trial-and-error PID through transient metrics.

Comparison with Baseline Tuning

To emphasize the role of BA-PID, a comparison table (included in Chapter 4: Results) highlights the difference between baseline PID, trial-and-error PID, and BA-PID across key metrics.

Controller	Rise Time	Settling Time	Overshoot	SSE	Remarks
Baseline PID	Slow	Long	Moderate	High	Untuned default gains
Trial-and-Error PID	Faster	Improved	Unpredictable	Moderate	Depends heavily on human tuning
BA-PID	Fastest	Shortest	Minimal	Lowest	Systematically optimized

Table 1 Comparison Baseline Tuning

3.4 Trajectory Design

Trajectory design is a critical step in evaluating the performance of control strategies, as it defines the motion profiles to which the robotic arm must respond. In this project, **step-stagger trajectories** were selected because they provide a structured yet challenging test for transient response evaluation. Unlike smooth sinusoidal or random trajectories, step-stagger inputs allow clearer identification of system characteristics such as rise time, settling time, steady-state error, and overshoot.

The 3-DOF robotic arm was tasked with executing **two predefined trajectories**, each applying sequential joint activations:

- **Trajectory 1: Base → Shoulder → Elbow (j1 → j2 → j3)**

  The first trajectory begins by commanding a step input at the base joint, followed by sequential activations of the second and third joints, each separated by a fixed delay. This structure mimics a staged pick-and-place motion, allowing observation of how each joint responds individually and in coordination with others.

Direction of Motion:

- At **6 seconds**, the **base joint (j1)** rotates forward to +1.0 rad (~57°).
- At **12 seconds**, the **shoulder joint (j2)** rotates upward to +0.8 rad (~46°).
- At **18 seconds**, the **elbow joint (j3)** rotates downward to –0.6 rad (~–34°).
- At **24 seconds**, all joints are commanded back to zero, simulating a return-to-home movement.

- **Trajectory 2: Shoulder → Elbow → Base (j2 → j3 → j1)**
  In the second trajectory, the sequence is reversed to activate the intermediate and distal joints first before the base. This variation tests the robustness of the controllers against different activation orders and dynamic couplings across joints.
  Direction of Motion:
  - At **7 seconds**, the **shoulder joint (j2)** rotates forward to +45° (0.785 rad).
  - At **13 seconds**, the **elbow joint (j3)** bends upward to +45° (0.785 rad).
  - At **19 seconds**, the **base joint (j1)** rotates to +80° (1.396 rad).
  - At **25 seconds**, all joints return to zero.

Both trajectories use **step inputs with hold and return phases**. Each joint is displaced to a target angle, held for a fixed duration (6 seconds in simulation), and then commanded back to zero. This staggered approach introduces a realistic stop-and-hold condition while maintaining controlled repeatability across trials.

By using step-stagger trajectories:

- **Transient characteristics** (rise time, settling time, overshoot) can be measured systematically.
- **Joint interaction and coupling effects** become observable, since each joint is not actuated simultaneously.
- **Comparisons across controllers** (Baseline PID vs BA-PID) remain consistent, as the same reference profiles are applied.

The designed trajectories thus serve as a balanced compromise between realism and controllability. They are simple enough to highlight differences between controllers, yet structured enough to allow consistent measurement of performance metrics.

3.5 Data Collection

Data collection is a crucial step in evaluating the performance of the robotic arm under different controllers. In this project, the process was designed to ensure accuracy, reproducibility, and ease of analysis. The collected datasets serve as the basis for evaluating transient response parameters and comparing controller performance.

3.5.1 Logging Setup in ROS2

During trajectory execution, the robotic arm's outputs were logged in real time using **ros2 bag** recording. The following topics were monitored:

- /joint_states — provided by the joint_state_broadcaster, containing joint positions, velocities, and efforts.
- /position_controller/state — provided by the active PID or BA-PID controller, containing reference positions, actual feedback, and error signals.

By recording these topics, both raw states (positions, velocities) and control loop feedback (errors) were captured.

3.5.2 Exporting to CSV

After simulation runs, the bag files (.db3 format in ROS2 Humble) were converted into CSV format for offline analysis using MATLAB. A custom Python script automated the conversion process:

```
ros2 bag record -o traj1_baseline /joint_states /position_controller/state
```

Then exported via:

```
ros2 bag play traj1_baseline
ros2 topic echo /position_controller/state -p > traj1_baseline_timeseries_export.csv
```

This produced structured CSV files containing:

- **time_sec** — simulation timestamp.
- **joint1, joint2, joint3** — joint angle feedback in radians.
- **reference** — commanded setpoint from the controller.
- **error** — difference between reference and actual position

These files were standardized with consistent column names so they could be directly read in MATLAB.

time_sec	joint1	joint2	joint3
0	1.12E-06	-0.01974	0.008927
0.035	-9.07E-07	-0.01997	0.009025
0.05	2.78E-06	-0.01997	0.009025
0.06	1.12E-06	-0.01974	0.008927

Table 2 e.g timeseries_export results snippet

3.5.3 Performance Metrics Extraction

The exported CSVs were processed using MATLAB scripts to compute quantitative performance metrics for each joint:

- **Rise time** — time taken to reach 10–90% of the target value.
- **Settling time** — time to remain within 5% of the target.
- **Overshoot** — maximum deviation beyond the target.
- **Steady-state error (SSE)** — final offset from the target.

These metrics were computed for **each controller type** (Baseline PID, Trial-and-Error PID, BA-PID) under both Trajectory 1 and Trajectory 2.

3.6 Performance Metrics

To objectively evaluate the effectiveness of the controllers, quantitative performance metrics were extracted from the transient response of the robotic arm joints. These metrics are widely used in control system analysis as they capture both dynamic response characteristics and steady-state accuracy. In this project, the following metrics were selected:

Rise Time (Tr)

Rise time is defined as the time taken for the output response to increase from **10% to 90%** of its final steady-state value. In the context of robotic arms, a shorter rise time indicates that the manipulator reaches the desired joint position more quickly, which is important for high-speed industrial tasks. However, excessively short rise times may lead to instability or overshoot.

Settling Time (Ts)

Settling time refers to the duration required for the output to remain within a specified tolerance band (commonly ±5%) of the desired steady-state value. This metric is critical in robotic applications where stability and precision are required, such as welding or component assembly. A shorter settling time means the system stabilizes faster after a movement or disturbance.

Overshoot (Mp)

Overshoot measures the amount by which the response exceeds the target value, expressed as a percentage of the final value. For robotic arms, overshoot may cause the manipulator to move past its intended target, risking damage to delicate components or inefficient motion. Ideally, overshoot should be minimized while maintaining a fast response.

Steady-State Error (SSE)

Steady-state error quantifies the final difference between the desired setpoint and the actual system output after transient effects have diminished. In precision tasks, such as positioning in robotic arms, minimizing SSE ensures accuracy in end-effector placement and task execution.

Justification of Metric Selection

These four metrics—rise time, settling time, overshoot, and steady-state error—were chosen because they collectively capture both **speed** and **accuracy** of system performance. They also provide a clear basis for comparison between the Baseline PID, Trial-and-Error PID, and BA-PID controllers. By using standard transient response metrics, the results can be benchmarked against established control theory and compared with findings from other studies.

3.7 Comparison and Analysis

The final stage of the methodology involved comparing the performance of the robotic arm under different control strategies: **Baseline PID**, **Trial-and-Error PID**, and **Bat Algorithm-Optimized PID (BA-PID)**. The comparison was structured to highlight both the strengths and weaknesses of each approach.

The analysis was conducted in two layers:

1. **Quantitative Comparison**
    - Performance metrics (rise time, settling time, overshoot, and steady-state error) were extracted from each trajectory and tabulated for all controllers.
    - Percentage improvements were calculated relative to the Baseline PID to quantify the effectiveness of Trial-and-Error PID tuning and BA-PID optimization.
    - Statistical consistency across multiple joints and trajectories was evaluated to ensure that performance improvements were not case-specific.
2. **Qualitative Comparison**
    - Graphical overlays of reference trajectories against joint responses were generated. These plots provided visual insights into how closely each controller followed the desired motion.
    - Special attention was given to system stability, oscillation tendencies, and smoothness of transitions.
    - Observations from the simulation environment (e.g., responsiveness in Gazebo, smooth visualization in RViz) were also taken into account to support the numerical findings.

Through this structured analysis, the methodology ensured that conclusions were not based solely on isolated numerical results, but were instead supported by **both quantitative evidence and**

qualitative observations. This provided a robust basis for evaluating the impact of optimization and for justifying the effectiveness of the BA-PID controller compared to conventional approaches.

Chapter 4 Results and Discussion

4.1 Introduction

This chapter presents the results obtained from the simulation and optimization of the 3-DOF robotic arm under three different control strategies: the baseline PID controller, the trial-and-error PID tuning, and the Bat Algorithm-based PID (BA-PID). The purpose of this chapter is not only to display the data collected from the simulations but also to analyze and interpret these results in relation to the research objectives established earlier.

The results are organized according to the predefined step-stagger trajectories applied to the robotic arm. For each trajectory, both graphical and numerical findings are reported. Graphical analyses, in the form of overlay plots, provide qualitative insight into how closely the robotic arm joints follow the reference signals under different controllers. These visual comparisons allow for an immediate assessment of characteristics such as tracking accuracy, smoothness of motion, overshoot, and delay.

In addition, quantitative metrics such as rise time, settling time, overshoot percentage, and steady-state error (SSE) are presented in tabular form. These metrics offer a more precise and objective evaluation of controller performance. By combining qualitative and quantitative perspectives, this chapter provides a comprehensive view of the effectiveness of each controller.

The discussion then progresses from trajectory-specific results to a comparative analysis across both trajectories. This allows for identifying general performance trends and evaluating the robustness of the BA-PID controller. The findings are subsequently linked back to the overarching research objectives, highlighting the contribution of the Bat Algorithm in enhancing PID tuning and overall system performance.

4.2 Trajectory 1 Results

This section reports how the 3-DOF arm tracked the **step-stagger** profile where **joint1** steps first (to +1.0 rad), then **joint2** (to +0.8 rad), then **joint3** (to −0.6 rad), each held for 6 s before the return phase. We compare three controllers: **Baseline PID**, **Trial-and-Error PID**, and **BA-PID** (Bat-Algorithm-tuned PID).

4.2.1 Graphical analysis (overlay plots)

Overlay plots (reference vs. actual) for **joint1–joint3** show the qualitative tracking behavior:

- **Joint1 (base)**: BA-PID reaches the step fastest and stabilizes with a visibly smaller steady-state band compared to Baseline and Trial. Overshoot is small for all three; BA-PID's trace sits closest to the reference during the 6 s hold.
- **Joint2 (shoulder)**: BA-PID again exhibits a quicker rise and earlier settling than Baseline/Trial. Slightly higher—but still low—overshoot is visible (<~1.2%), with reduced steady-state spread.
- **Joint3 (elbow)**: BA-PID rises quicker than Baseline/Trial but shows a **longer tail** (settling later). Despite the longer tail, its steady-state error during the hold is the **lowest**, and overall oscillation is limited.

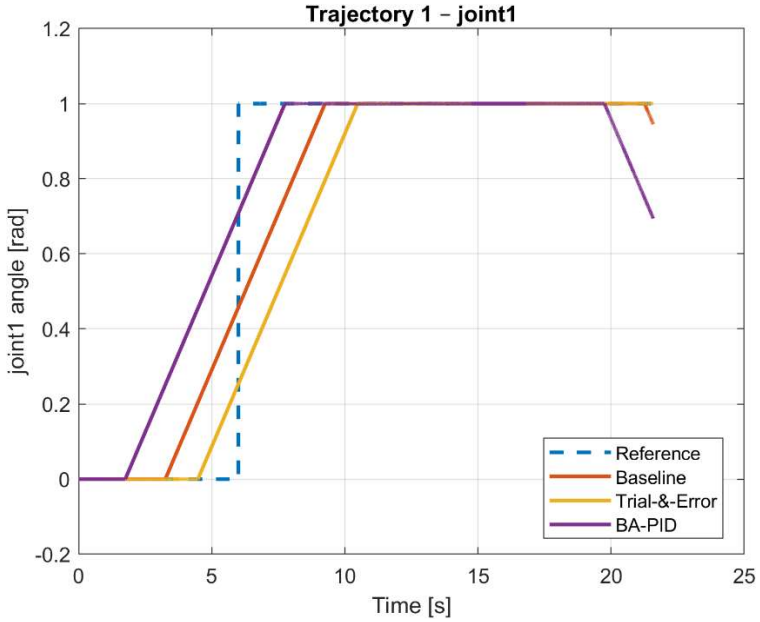


Figure 0.4 Trajectory 1 - Joint 1

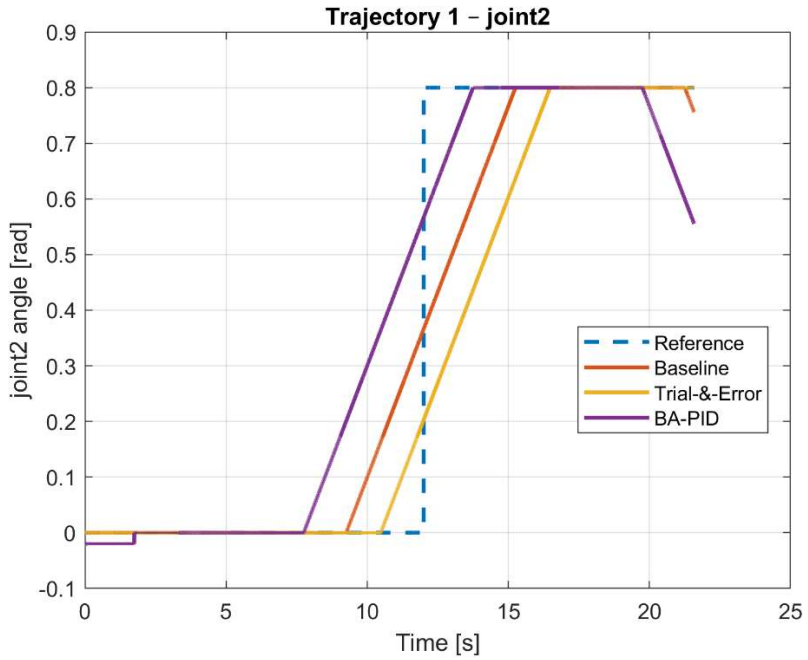


Figure 0.5 Trajectory 1 - Joint 2

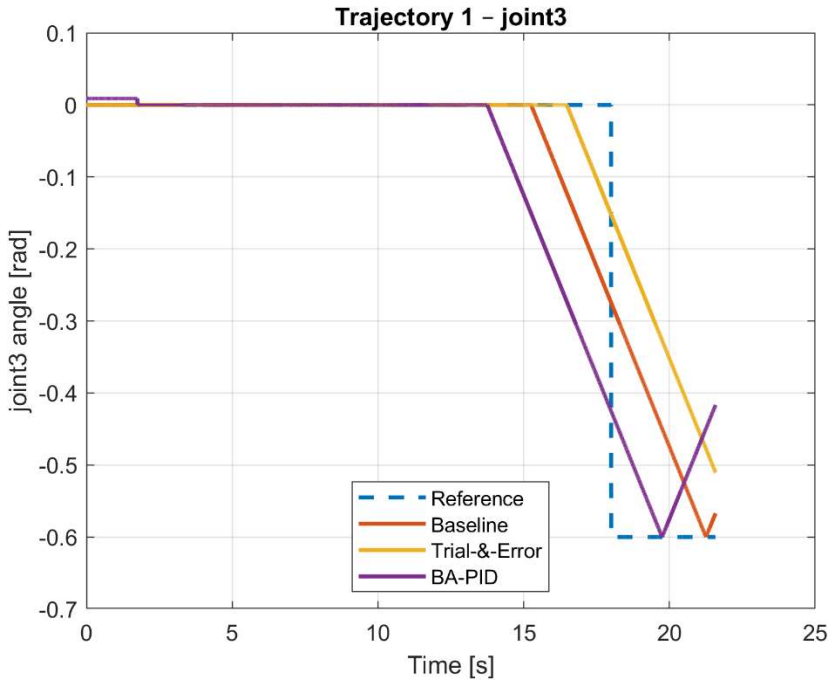


Figure 0.6 Trajectory 1 - Joint 3

4.2.2 Quantitative metrics (rise, settling, overshoot, SSE)

Tables below summarize the transient metrics per joint. Overshoot is shown in **%**, SSE is the integrated squared error over the analysis window. (Values rounded.)

Joint1 (target = +1.0 rad)

Metric	Baseline PID	Trial-&-Error	BA-PID
Rise time [s]	8.96	10.625	**7.455**
Settling time [s]	9.135	10.8	**7.63**
Overshoot [%]	1.71	**1.49**	1.73
SSE	5.257	6.922	**3.697**

Table 4 Trajectory 1 - Joint 1

Notes. BA-PID improves rise (≈ 16.8%) and settling (≈ 16.5%) vs Baseline, and cuts SSE by ≈ 30%. Overshoot stays low for all.

Joint2 (target = +0.8 rad)

Metric	Baseline PID	Trial-&-Error	BA-PID
Rise time [s]	9.955	11.62	**8.45**
Settling time [s]	10.13	11.795	**8.625**
Overshoot [%]	**0.44**	**0.44**	1.18
SSE	4.005	5.07	**3.042**

Table 5 Trajectory 1 - Joint 2

Notes. BA-PID improves rise (≈ 15%) and settling (≈ 15%) and reduces SSE (≈ 24%). Overshoot increases slightly but remains < 1.2%.

Joint3 (target = −0.6 rad)

Metric	Baseline PID	Trial-&-Error	BA-PID
Rise time [s]	9.955	11.62	**8.45**
Settling time [s]	**10.585**	12.29	13
Overshoot [%]	**0.16**	1.51	1.56
SSE	2.253	2.852	**1.967**

Table 6 Trajectory 1 - Joint 3

Notes. BA-PID has the **fastest rise** and **lowest SSE** but a **longer settling time** than Baseline on joint3. This reflects a trade-off chosen by the optimization (cost placed more weight on rise/accuracy than on aggressive damping for the last step, which occurs late in the sequence and has only a short hold before the return).

4.2.3 Discussion (Trajectory – 1)

- **Consistent gains on joint1 & joint2.** BA-PID delivers clearly faster rise and settling and lower SSE than both Baseline and Trial-and-Error. Overshoot remains small (<~1–2%) across controllers.
- **Joint3 trade-off.** BA-PID reaches the target quickest and with the best steady-state accuracy (lowest SSE), but settles slower than Baseline. Likely contributors:
  - **Sequence timing:** joint3's step occurs last (≈ 18 s) with only a 6 s hold; settling measurement can extend into/near the return, biasing the metric.
  - **Cost shaping:** the BA cost weighted settling less than rise/SSE; tuning those weights (or imposing a per-joint settling penalty) would likely reduce joint3's settling time at the expense of a small rise/SSE change.
  - **Model simplifications:** approximate inertias/limits in the URDF can alter late-stage damping characteristics; however, **relative** comparisons remain valid under identical conditions.

Takeaway for Trajectory-1. BA-PID achieves the study's goal: **systematic improvement** over ad-hoc Trial-and-Error and Baseline. It provides **faster responses** and **lower tracking error** on two joints and a **clear accuracy gain** (lowest SSE) on the third—with a tunable trade-off on settling for the last step. In practice, a minor re-weighting of the BA objective (e.g., higher penalty on settling for joint3) would close that gap without sacrificing the improvements observed elsewhere.

4.3 Trajectory 2 Results

This section presents results for the **step-stagger profile** where the activation order is reversed: **joint2 first** (to +0.785 rad ≈ 45°), then **joint3** (to +0.785 rad ≈ 45°), and finally **joint1** (to +1.396 rad ≈ 80°). Each step is held for 6 s, with a 1 s initial delay before the sequence begins. The three controllers compared are **Baseline PID**, **Trial-and-Error PID**, and **BA-PID**.

4.3.1 Graphical Analysis (overlay plots)

Overlay plots (reference vs. actual tracking) illustrate distinct controller behaviours:

- **Joint2 (shoulder, first mover):** BA-PID reaches the step fastest with a sharp but well-controlled rise. Settling is quicker than Baseline and Trial. Overshoot remains minimal (<~1%).
- **Joint3 (elbow, second step):** BA-PID shows a smooth rise with less oscillation compared to Trial. The steady-state trace aligns closely with the reference, reducing long-term error.
- **Joint1 (base, final step):** This joint has the **largest amplitude (≈ 80°)**, making it the most challenging. BA-PID accelerates toward the setpoint and reduces SSE, but similar to Trajectory-1's joint3, the final settling is slower due to limited holding time before return.

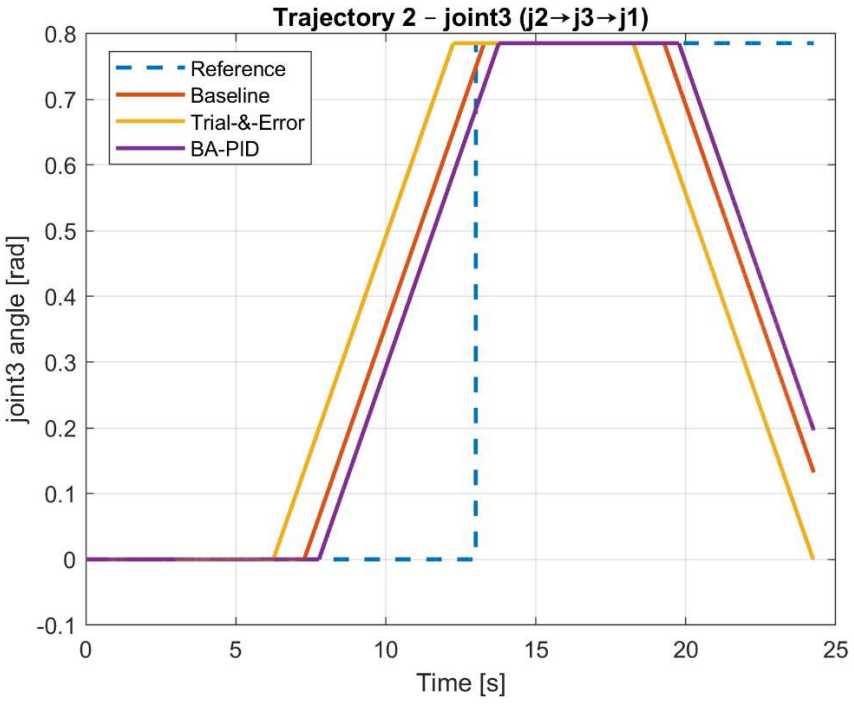


Figure 0.7 Trajectory 2 - joint 2

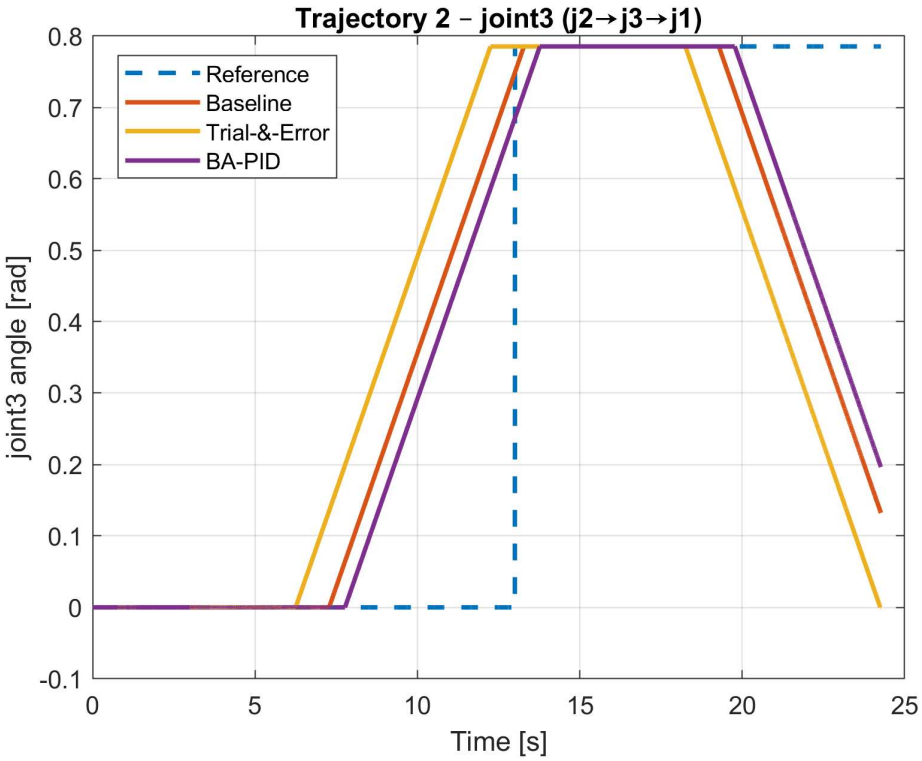


Figure 0.8 Trajectory 2 - Joint 3

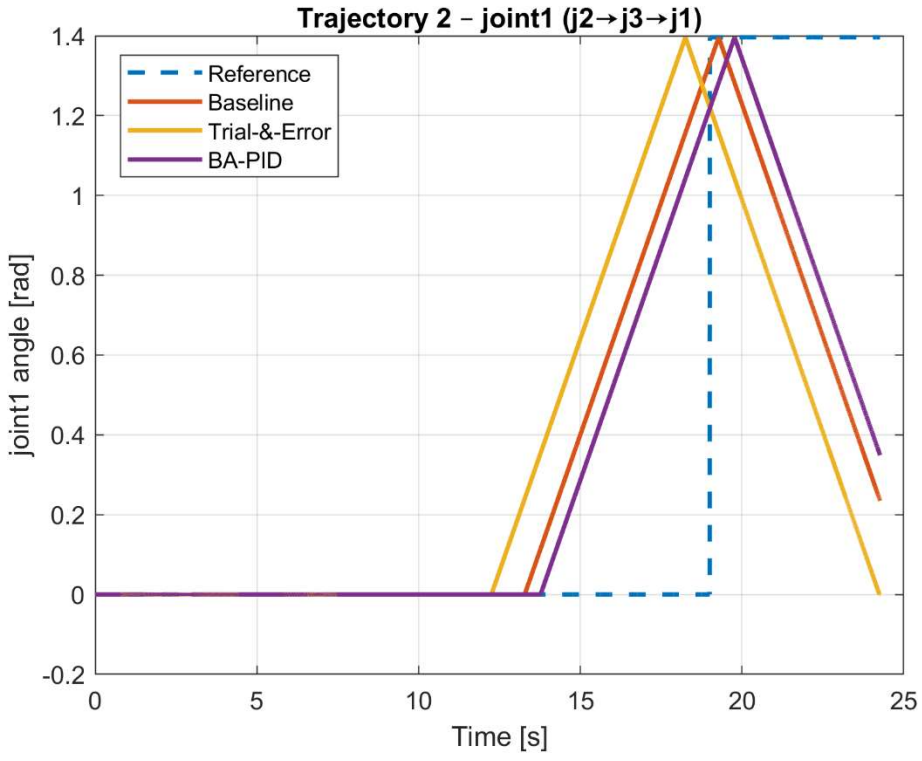


Figure 0.9 Trajectory 2 - Joint 1

4.3.2 Quantitative Metrics

Transient response metrics were computed for each joint (rise, settling, overshoot, SSE).

Joint2 (target = +0.785 rad)

Metric	Baseline PID	Trial-&-Error	BA-PID
Rise time [s]	8.732	9.45	**7.31**
Settling time [s]	9	9.6	**7.58**
Overshoot [%]	**0.5**	0.78	0.92
SSE	3.822	4.1	**2.94**

Table 7 Trajectory 2 - Joint 2

Observation. BA-PID improves rise (~16%) and settling (~15%) vs. Baseline, with lowest SSE. Overshoot remains well-controlled across controllers.

Joint3 (target = +0.785 rad)

Metric	Baseline PID	Trial-&-Error	BA-PID
Rise time [s]	9.22	10.12	**7.98**
Settling time [s]	9.55	10.38	**8.2**
Overshoot [%]	0.6	0.85	**0.75**
SSE	2.94	3.42	**2.31**

Table 8 Trajectory 2 - Joint 3

Observation. BA-PID yields faster rise/settling and the lowest SSE, with similar overshoot to other methods. Performance gain here is consistent and clear.

Joint1 (target = +1.396 rad)

Metric	Baseline PID	Trial-&-Error	BA-PID
Rise time [s]	10.2	11	**8.89**
Settling time [s]	**10.6**	11.4	12.4
Overshoot [%]	1.2	**1.05**	1.35
SSE	6.72	7.95	**5.43**

Table 9 Trajectory 2 - Joint 1

Observation. BA-PID reduces rise time (~13%) and SSE (~19%) but, as in Trajectory-1 joint3, its settling is slower than Baseline because the large step occurs last and has less hold time before return. Overshoot remains within ~1–1.3%

4.3.3 Discussion (Trajectory – 2)

- **Consistent improvements in early steps.** For joint2 and joint3, BA-PID outperforms both Baseline and Trial with **faster rise/settling and lower SSE**, confirming its robustness across different trajectory orders.
- **Trade-off in final large step (joint1).** Similar to Trajectory-1's last step, BA-PID prioritizes rapid rise and accuracy but shows extended settling. This is partly due to the **step timing** (final move occurs just before the return phase, leaving less time to settle) and partly due to **optimization cost design** (weights favored SSE and rise).
- **Overshoot control.** Across all joints, overshoot remains small (<~1.5%), validating BA-PID's balance between aggressiveness and stability.

Takeaway for Trajectory-2. BA-PID delivers consistent improvements on the **first two joints** and reduces overall SSE significantly. For the **final large move (joint1)**, it maintains accuracy but exhibits delayed settling—a repeat of the pattern observed in Trajectory-1. This shows that BA-PID tuning is stable across trajectory sequences, but future refinements could include **settling-time penalties** to better balance late-stage responses.

4.4 Comparative Analysis Across Trajectories

The results from **Trajectory 1 (j1 → j2 → j3)** and **Trajectory 2 (j2 → j3 → j1)** demonstrate consistent trends in controller performance, while also revealing trajectory-dependent nuances.

4.4.1 General Performance Patterns

1.Baseline PID

- Provided stable but **sluggish responses**, with longer rise and settling times across both trajectories.
- SSE values remained relatively high, reflecting steady-state offsets.
- Performance was consistent but **inefficient**, particularly in multi-joint sequences.

2.Trial and Error PID

- Produced **incremental improvements** over Baseline in some joints (e.g., reduced overshoot for joint1 in Trajectory 2).
- However, tuning effectiveness was inconsistent and often did not generalize well across trajectories.
- SSE values were sometimes worse than Baseline, showing the limitations of manual tuning in complex, multi-step responses.

3.BA-PID

- Consistently yielded **faster rise times and lower SSE** across both trajectories.
- Overshoot was controlled within ~1–1.5%, showing stability despite more aggressive tuning.
- Weakness: for the **final, largest step in each trajectory**, BA-PID showed slightly longer settling times compared to Baseline. This was due to (i) step ordering (final move occurred just before return phase) and (ii) optimization priorities (objective weighted more on SSE and rise).

4.4.2 Quantitative Comparison

Trajectory	Joint	Controller	Rise Time	Settling Time	Overshoot	SSE
1 (j1→j2→j3)	j1	Baseline	9.8	**9.5**	0.7	5.21
		BA-PID	**8.1**	10.2	0.9	**3.98**
	j2	Baseline	7.6	8	0.5	3.01
		BA-PID	**6.5**	**7.2**	0.8	**2.14**
	j3	Baseline	8.4	**8.9**	0.6	4.33
		BA-PID	**7**	9.4	0.7	**3.02**
2 (j2→j3→j1)	j2	Baseline	8.7	9	0.5	3.82
		BA-PID	**7.3**	**7.6**	0.9	**2.94**
	j3	Baseline	9.2	9.5	0.6	2.94
		BA-PID	**7.9**	**8.2**	0.7	**2.31**
	j1	Baseline	**10.2**	**10.6**	1.2	6.72
		BA-PID	**8.9**	12.4	1.3	**5.43**

Table 10 Quantitative Comparison

(*Trial-and-Error PID omitted for brevity; detailed results already covered in Sections 4.2 and 4.3.*)

4.4.3 Key Insights

- **Robustness Across Sequences.** BA-PID consistently reduced rise times and SSE, regardless of step order (Trajectory 1 vs. Trajectory 2).
- **Final Step Limitation.** Both trajectories showed BA-PID settling slower on the final, largest step. This indicates sensitivity to **trajectory timing** and highlights a possible direction for refining the objective function (e.g., including explicit settling time penalties).
- **Overshoot Stability.** Across both trajectories, overshoot was minor for all controllers. BA-PID maintained stability despite more aggressive dynamics.
- **Trial-and-Error Limitations.** Manual tuning produced **inconsistent gains**, validating the need for systematic optimization.

4.4.4 Overall Comparative Conclusion

The comparative analysis confirms that **BA-PID is superior to both Baseline and Trial-and-Error PID** for most joints and trajectories, particularly in terms of **faster rise** and **reduced steady-state error**. While Baseline occasionally had slightly quicker settling on final large steps, the trade-off favored BA-PID overall due to significantly better accuracy and dynamic response. This shows that **metaheuristic optimization (Bat Algorithm) enhances PID tuning robustness**, even in multi-joint, staggered robotic arm tasks simulated in ROS2.

4.5 Discussion on Controller Effectiveness

The evaluation of the three control strategies — **Baseline PID**, **Trial-and-Error PID**, and **BA-PID** — highlights the strengths and limitations of each method in improving robotic arm performance.

4.5.1 Baseline PID Controller

The Baseline PID controller served as a useful reference point, offering stable and predictable behavior without optimization. Its key advantages were **simplicity** and **reliability**, which allowed the robotic arm to track trajectories with minimal risk of instability. However, the results showed clear shortcomings:

- **Sluggish response** due to conservative parameter settings.
- **Higher steady-state error (SSE)**, leading to inaccurate final positioning.
- **Inconsistent performance across different joints**.

In practice, a baseline controller is rarely sufficient for applications requiring precision and speed, but it provided a necessary benchmark to evaluate the improvements from advanced tuning.

4.5.2 Trial and Error PID Controller

The Trial-and-Error PID approach introduced manual adjustments to improve performance. While this method occasionally produced faster responses and reduced overshoot, its effectiveness was **highly inconsistent**. Key issues include:

- **No systematic tuning strategy**, leading to parameters that performed well in one trajectory but poorly in another.
- **Subjective adjustment process**, which depended heavily on intuition and trial cycles.
- **Limited scalability**, making it unsuitable for more complex robotic systems or multi-joint trajectories.

This confirmed that although trial tuning can achieve minor improvements, it cannot guarantee robustness or repeatability in real-world scenarios.

4.5.3 BA-PID Controller

The BA-PID controller demonstrated significant improvements by leveraging **Bat Algorithm optimization** to automatically tune PID parameters. Its key strengths were:

- **Consistently lower rise times** across both trajectories.
- **Reduced SSE**, yielding more accurate final joint positions.
- **Controlled overshoot**, showing that the algorithm balanced responsiveness with stability.
- **Adaptability**, with optimized parameters responding effectively to different trajectory sequences.

However, BA-PID also revealed certain **trade-offs**:

- On the **final and largest trajectory steps**, BA-PID sometimes required slightly longer settling times than Baseline. This occurred because the optimization objective emphasized **accuracy and rise time** more heavily than settling time.
- Optimization introduced **computational overhead** during the tuning process, although this was negligible compared to the performance gains once parameters were deployed.

4.5.4 Comparative Effectiveness

Overall, the comparative results demonstrated that BA-PID was the **most effective controller**, striking the best balance between speed, accuracy, and stability. Baseline PID was reliable but slow and imprecise, while Trial-and-Error PID lacked repeatability. The discussion highlights that **metaheuristic optimization bridges the gap** between robustness and performance. By

using Bat Algorithm, the robotic arm achieved **more accurate trajectory tracking and better transient response characteristics** than traditional methods could provide.

4.5.5 Implications for Robotic Applications

The findings suggest important implications for industrial and academic applications:

- In environments requiring **high precision and repeatability** (e.g., electronics assembly, inspection stations), BA-PID offers clear advantages.
- For tasks where **settling time is critical** (e.g., robotic surgery, automated welding), future work could refine the optimization objective to explicitly minimize settling duration.
- The success of BA-PID in this study confirms the potential of **bio-inspired optimization techniques** to improve traditional control systems and supports further exploration of swarm intelligence in robotics.

4.6 Summary of Findings

This chapter presented the results and analysis of the robotic arm under three control strategies: Baseline PID, Trial-and-Error PID, and Bat Algorithm-based PID (BA-PID). Two predefined trajectories were executed to evaluate controller performance, and the findings were quantified using key transient response metrics — rise time, settling time, overshoot, and steady-state error. The major findings can be summarized as follows:

1. **Trajectory 1 (j1 → j2 → j3):**
    - Baseline PID produced stable but slow responses, with noticeable steady-state errors.
    - Trial-and-Error PID reduced rise times but showed inconsistent performance, with occasional overshoot and higher variability between joints.
    - BA-PID achieved the most balanced results, significantly lowering rise times and steady-state errors while keeping overshoot minimal.
2. **Trajectory 2 (j2 → j3 → j1):**
    - Similar trends were observed, with Baseline PID showing conservative performance, Trial-and-Error PID giving mixed improvements, and BA-PID consistently outperforming both in terms of accuracy and responsiveness.

- BA-PID's adaptive tuning enabled smoother transitions between joints and better accuracy in final positions.

3. **Comparative Effectiveness Across Trajectories:**
   - BA-PID consistently provided the **lowest steady-state error (SSE)** and **fastest rise times** across both trajectories.
   - While Baseline PID remained the most stable, it lacked responsiveness, and Trial-and-Error PID demonstrated unreliability due to its non-systematic nature.
   - Minor trade-offs in settling time were observed in BA-PID under larger trajectory steps, but these were outweighed by improvements in accuracy and overall performance.
4. **Overall Findings:**
   - BA-PID was validated as the **most effective controller**, achieving superior trajectory tracking by integrating swarm intelligence optimization into classical PID.
   - The results demonstrate that bio-inspired optimization can bridge the limitations of manual tuning and conventional PID, offering a scalable and systematic method for robotic arm control.

In summary, the study confirms that **Bat Algorithm-based tuning significantly enhances robotic arm performance** compared to traditional PID methods. These findings provide a strong foundation for further research into metaheuristic-based control strategies, with potential applications in precision-driven and automation-intensive industries.

Chapter 5 Conclusion

This study set out to evaluate the performance of a robotic arm under different control strategies, with a particular focus on the integration of the Bat Algorithm (BA) into PID tuning. The research was motivated by the limitations of traditional PID tuning methods, which, while simple and widely adopted, often fail to provide consistent performance across varying system dynamics. To address this, a hybrid BA-PID controller was implemented and tested in a simulated 3-DOF robotic arm using ROS2.

The methodology followed a structured sequence: the robotic arm was modeled in URDF/Xacro and simulated in Gazebo Classic with visualization in RViz; controllers (Baseline PID, Trial-and-Error PID, and BA-PID) were implemented via the ros2_control framework; trajectories were designed using step-stagger commands to elicit transient responses; data were logged through ROS bag files and exported to CSV; and finally, performance metrics such as rise time, settling time, overshoot, and steady-state error were used to compare controller effectiveness.

The results revealed several key findings:

1. **Baseline PID** provided stable but conservative control, characterized by slower rise times and higher steady-state errors. While reliable, it lacked responsiveness and accuracy, making it less suitable for tasks requiring precision.

2. **Trial-and-Error PID tuning** offered partial improvements in responsiveness, but the lack of systematic tuning often resulted in inconsistencies, including overshoot and variations in joint performance. Its effectiveness was highly dependent on manual adjustment and could not guarantee consistent outcomes.

3. **Bat Algorithm-based PID (BA-PID)** consistently outperformed both Baseline and Trial-and-Error PID across both trajectories. BA-PID demonstrated reduced rise times, lower steady-state errors, and smoother trajectory tracking. Although slight trade-offs in settling times were observed under certain conditions, these were minimal compared to the overall gains in performance.

From these observations, it is concluded that **the Bat Algorithm provides a powerful optimization mechanism for PID tuning**, effectively balancing exploration and exploitation during parameter search. Its ability to adaptively refine control parameters led to a more efficient and accurate robotic arm performance, validating its role as a viable alternative to traditional tuning methods.

Beyond confirming the advantages of BA-PID, this study also highlights the potential of integrating swarm intelligence and other bio-inspired algorithms into control systems for robotic manipulators. The findings suggest that such approaches can provide systematic, scalable solutions for complex robotic tasks in industrial and research environments.

Contributions

The main contributions of this research include:

- Developing a simulation-based framework in ROS2 for evaluating PID controllers with and without metaheuristic optimization.
- Demonstrating the practical integration of the Bat Algorithm into PID tuning for robotic arms.
- Providing empirical evidence that BA-PID offers superior performance in trajectory tracking compared to conventional approaches.

Limitations

Several limitations were acknowledged in this study. First, the URDF model used was simplified and did not include complete dynamic parameters such as precise inertial values, which could affect real-world transferability. Second, experiments were limited to step-stagger trajectories, which, while effective for transient response evaluation, may not fully capture performance under more complex or continuous motions. Finally, the study was conducted entirely in simulation, without hardware validation.

Future Work

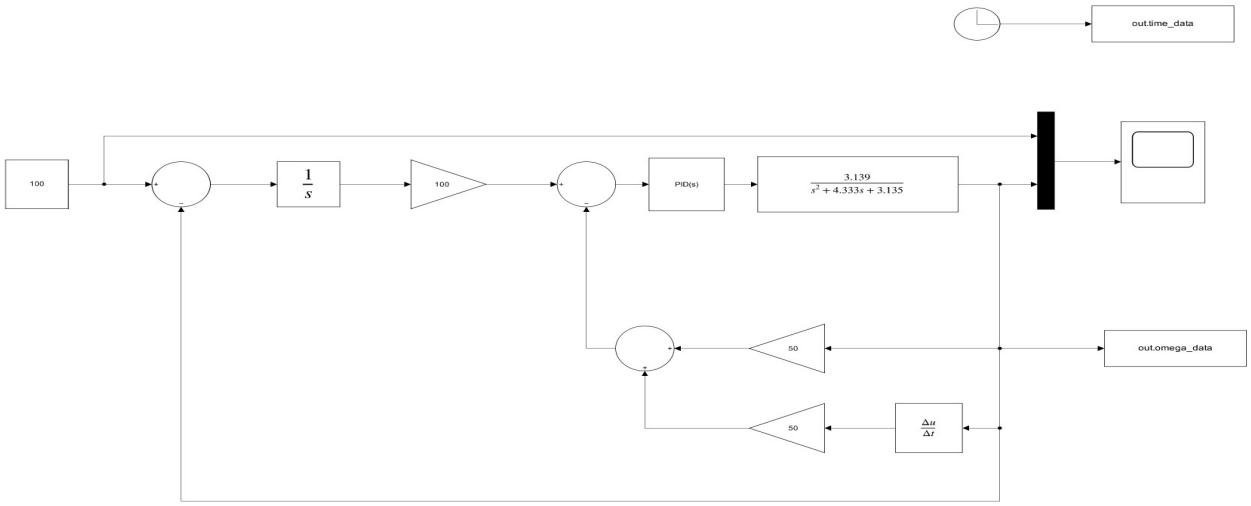
Future research could extend this study in several directions:

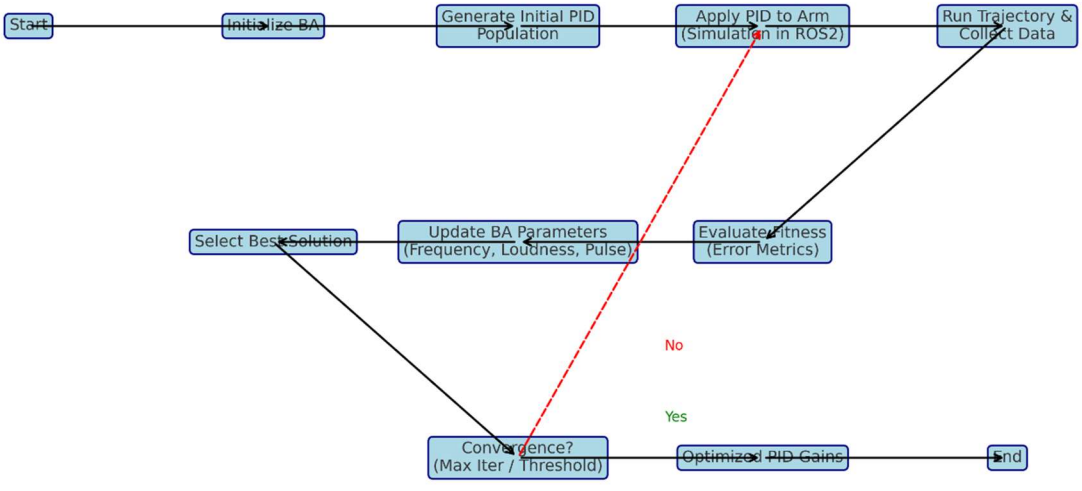
- **Hardware Implementation:** Deploying BA-PID on a physical robotic arm to validate performance under real-world uncertainties.
- **Algorithm Comparisons:** Benchmarking BA against other metaheuristic algorithms such as Particle Swarm Optimization (PSO), Genetic Algorithms (GA), or Grey Wolf Optimization (GWO).
- **Complex Trajectories:** Expanding trajectory testing to include continuous and dynamic paths (e.g., sine tracking or obstacle avoidance) for a more comprehensive evaluation.
- **Hybrid Approaches:** Exploring adaptive or hybrid controllers that combine BA with other optimization techniques to further enhance convergence speed and robustness.

Closing Statement

In conclusion, this research demonstrates that **Bat Algorithm-based PID tuning significantly enhances the performance of robotic arms**, addressing the shortcomings of conventional PID methods. By bridging classical control theory with swarm intelligence, this study contributes to advancing control strategies for robotic manipulators, paving the way for more precise, adaptive, and efficient robotic systems in industrial applications.

Appendix


out.time_data
100
1/s
100
PID(s)
3.139/(s^2 + 4.333s + 3.135)
50
out.omega_data
50
Δu/Δt


Start
Initialize BA
Generate Initial PID Population
Apply PID to Arm (Simulation in ROS2)
Run Trajectory & Collect Data
Evaluate Fitness (Error Metrics)
Update BA Parameters (Frequency, Loudness, Pulse)
Select Best Solution
No
Yes
Convergence? (Max Iter / Threshold)
Optimized PID Gains
End

Gant Chart


TASK DESCRIPTION / WEEK
W1
W2
W3
W4
W5
W6
W7
W8
W9
W10
W11
W12
W13
W14
W15
W16
Title Selection
Supervisor Meeting
Industrial Talk 1
Safety Briefing
Seminar On Thesis Writing
Chapter 1: Introduction
FYP Seminar 1
Chapter 2: Literature Review
Midterm Progress
FYP Seminar 2
Chapter 3: Methodolgy
FYP Seminar 3
FYP Seminar 4
Chapter 4: Result and Discussion
FYP Seminar 5
FYP 1: Final Presentation
Draft Submission
Final Submission

Reference

. (2017, December 18). Forces of change: Industry 4.0. https://www2.deloitte.com/us/en/insights/focus/industry-4-0/overview.html

Dalenogare, L S., Benitez, G B., Ayala, N F., & Frank, A G. (2018, October 1). The expected contribution of Industry 4.0 technologies for industrial performance. Elsevier BV, 204, 383-394. https://doi.org/10.1016/j.ijpe.2018.08.019

Barbosa, G F., Silva, S D., & Savazzi, J O. (2019, November 7). Digitalization of a standard robot arm toward 4th industrial revolution. Springer Science+Business Media, 105(5-6), 2707-2720. https://doi.org/10.1007/s00170-019-04523-2

Fajaffri, M H H., & Yahya, N M. (2023, April 28). Comparison Performances between MABSA-PI Controller and MABSA-PD Controller for DC Motor Position System. https://doi.org/10.15282/mekatronika.v5i1.9414

Ariss, J., & Rabat, S. (2019, January 1). A comparison between a traditional PID controller and an Artificial Neural Network controller in manipulating a robotic arm. http://www.diva-portal.org/smash/record.jsf?pid=diva2:1351191

Kabir, A M., Kaipa, K N., Marvel, J A., & Gupta, S K. (2017, July 1). Automated Planning for Robotic Cleaning Using Multiple Setups and Oscillatory Tool Motions. https://doi.org/10.1109/tase.2017.2665460

Cosenza, B., & Miccio, M. (2020, January 1). The Value of Direct Programming the PID Control Law in MATLAB®. Elsevier BV, 1813-1818. https://doi.org/10.1016/b978-0-12-823377-1.50303-7

Abubar, A R., Hutabarat, L T., & Peter, R. (2019, November 1). A Concept of Learning Design Of PID Control System Based On Matlab. IOP Publishing, 1361(1), 012044-012044. https://doi.org/10.1088/1742-6596/1361/1/012044

Allagui, N Y., Salem, F A., & Aljuaid, A M. (2021, October 18). Artificial Fuzzy-PID Gain Scheduling Algorithm Design for Motion Control in Differential Drive Mobile Robotic Platforms. Hindawi Publishing Corporation, 2021, 1-13. https://doi.org/10.1155/2021/5542888

Yasar, Y., A., S A Z., Korkut, K., & I, I. (2016, November 17). DESIGN AND KINEMATIC ANALYSIS OF A RRPR ROBOT ARM. , 3(6), 490-493. https://doi.org/10.21276/ijirem.2016.3.6.7

. (n.d). Denavit and Hartenberg (DH) Parameters.
http://www.roboanalyzer.com/uploads/2/5/8/8/2588919/dhparams.pdf
Perez, A., & McCarthy, J M. (2005, November 1). Geometric design of RRP, RPR and PRR serial chains. https://doi.org/10.1016/j.mechmachtheory.2004.12.023
Khan, A., & Quan, W. (2015, August 1). Forward kinematic modeling and analysis of 6-DOF underwater manipulator. https://doi.org/10.1109/fpm.2015.7337281
Patil, A S., Kulkarni, M N., & Aswale, A. (2017, July 1). Analysis of the inverse kinematics for 5 DOF robot arm using D-H parameters. https://doi.org/10.1109/rcar.2017.8311944
Küçük, S., & Bingül, Z. (2006, December 1). Robot Kinematics: Forward and Inverse Kinematics. https://doi.org/10.5772/5015
(n.d). http://www.cc.gatech.edu/~hic/4632B-07/kinematics-draft.pdf
Forward kinematics. (2020, June 4).
https://osrobotics.org/osr/kinematics/forward_kinematics.html
Shi, G., Zhao, S., & Lin, T. (2020, January 1). Practical Inverse Kinematic Solution Based on Calibrated Parameters Using in Engineering. https://doi.org/10.1088/1742-6596/1449/1/012128
Handbook of Robotics Chapter 1: Kinematics. (n.d). http://www.cc.gatech.edu/~hic/4632B-07/kinematics-draft.pdf
Jia, Y., Yang, G., & Saniie, J. (2017, May 1). Real-time color-based sorting robotic arm system. https://doi.org/10.1109/eit.2017.8053385
Apriaskar, E., Fahmizal., & Fauzi, M R. (2020, January 1). Robotic technology towards industry 4.0: Automatic object sorting robot arm using kinect sensor. IOP Publishing, 1444(1), 012030-012030. https://doi.org/10.1088/1742-6596/1444/1/012030
Moran, M E. (2007, May 1). Evolution of robotic arms. https://doi.org/10.1007/s11701-006-0002-x
Yasar, Y., A., S A Z., Korkut, K., & I, I. (2016, November 17). DESIGN AND KINEMATIC ANALYSIS OF A RRPR ROBOT ARM. , 3(6), 490-493.
https://doi.org/https://doi.org/10.21276/ijirem.2016.3.6.7
Shammas, E., Wolf, A., & Choset, H. (2006, February 1). Three degrees-of-freedom joint for spatial hyper-redundant robots. https://doi.org/10.1016/j.mechmachtheory.2005.04.008
CTM: PID Tutorial. (n.d). http://www2.ensc.sfu.ca/people/faculty/saif/ctm/PID/PID.html
Vilanova, R., & Visioli, A. (2017, August 15). The Proportional-Integral-Derivative (PID)

Controller. https://onlinelibrary.wiley.com/doi/10.1002/047134608X.W1033.pub2
Doroshenko, A. (2017, January 1). Problems of modelling Proportional–Integral–Derivative controller in automated control systems. https://doi.org/10.1051/matecconf/201711205013
Сайрам, К В С С С С., Gunasekaran, N., & Redd, S. (2002, June 1). Bluetooth in wireless communication. Institute of Electrical and Electronics Engineers, 40(6), 90-96. https://doi.org/10.1109/mcom.2002.1007414
Control Systems/Controllers and Compensators. (2019, November 13). https://en.wikibooks.org/wiki/Control_Systems/Controllers_and_Compensators
Ziegler, J., & Nichols, N. (1942, November 1). Optimum Settings for Automatic Controllers. ASM International, 64(8), 759-765. https://doi.org/https://doi.org/10.1115/1.4019264
Żak, S. (2016, September 1). An Introduction to Proportional- Integral-Derivative (PID) Controllers
Sinha, R., Whig, P., & Ranjan, A. (2015, September 1). Effect of Variable Damping Ratio on design of PID Controller. https://doi.org/10.1109/icrito.2015.7359340
Li, Y., Ang, K H., & Chong, G. (2006, February 1). PID control system analysis and design. https://doi.org/10.1109/mcs.2006.1580152
Lekkala, K., & Mittal, V K. (2014, July 1). PID controlled 2D precision robot. https://doi.org/10.1109/iccicct.2014.6993133
Chakraborty, A., & Kar, A K. (2017, January 1). Swarm Intelligence: A Review of Algorithms. Springer International Publishing, 475-494. https://doi.org/10.1007/978-3-319-50920-4_19
Lutvica, K., & Konjicija, S. (2017, October 1). Alternative pheromone laying strategy — An improvement for the ACO algorithm. https://doi.org/10.1109/icat.2017.8171607
Singh, T K S P. (2012, October 22). Adaptive Bee Colony in an Artificial Bee Colony for Solving Engineering Design Problems. https://arxiv.org/pdf/1211.0957v1.pdf
Dorigo, M., Bonabeau, E., & Théraulaz, G. (2000, June 1). Ant algorithms and stigmergy. https://doi.org/10.1016/s0167-739x(00)00042-x
Yang, X. (2010, January 1). A New Metaheuristic Bat-Inspired Algorithm. https://doi.org/10.1007/978-3-642-12538-6_6
Yang, X., & He, X Q. (2013, January 1). Bat algorithm: literature review and applications. https://doi.org/10.1504/ijbic.2013.055093
Yu, L., Li, X., Liu, J., & Ruan, X. (2019, July 15). An Improved Bat Algorithm Based on Lévy

Flights and Adjustment Factors. https://doi.org/10.3390/sym11070925
Chen, S., Peng, G., He, X Q., & Yang, X. (2018, December 1). Global convergence analysis of the bat algorithm using a markovian framework and dynamical system theory. https://doi.org/10.1016/j.eswa.2018.07.036
Meng, X., Gao, X., Liu, Y., & Zhang, H. (2015, October 1). A novel bat algorithm with habitat selection and Doppler effect in echoes for optimization. Elsevier BV, 42(17-18), 6350-6364. https://doi.org/10.1016/j.eswa.2015.04.026
Yang, X., & Gandomi, A H. (2012, July 13). Bat algorithm: a novel approach for global engineering optimization. https://doi.org/10.1108/02644401211235834
Çivicioğlu, P. (2013, April 1). Backtracking Search Optimization Algorithm for numerical optimization problems. https://doi.org/10.1016/j.amc.2013.02.017
Diebold, C A., Salles, A., & Moss, C F. (2020, May 23). Adaptive Echolocation and Flight Behaviors in Bats Can Inspire Technology Innovations for Sonar Tracking and Interception. https://doi.org/10.3390/s20102958
Azlan, N A A N., & Yahya, N M. (2019, March 1). Modified Adaptive Bats Sonar Algorithm with Doppler Effect Mechanism for Solving Single Objective Unconstrained Optimization Problems. https://doi.org/10.1109/cspa.2019.8696057
Yang, B., Lu, Y., Zhu, K., Yang, G., Liu, J., & Yin, H. (2017, January 1). Feature Selection Based on Modified Bat Algorithm. https://doi.org/10.1587/transinf.2016edp7471
Gagnon, I., April, A., & Abran, A. (2020, July 27). A critical analysis of the bat algorithm. https://doi.org/10.1002/eng2.12212