基于进化算法的蝙蝠机器人控制参数优化

Batbot Control Parameters Optimization Through Evolutionary Algorithm

(申请清华大学工学硕士学位论文)

培 养 单 位 ： 机械工程系

学　　科 ： 机械工程

研 究 生 ： 安　　武

指 导 教 师 ： 张　　云 教授

二〇二四年五月

基于进化算法的蝙蝠机器人控制参数优化

安　武

Batbot Control Parameters Optimization Through Evolutionary Algorithm

Thesis submitted to

Tsinghua University

in partial fulfillment of the requirement

for the degree of

Master of Science

in

Mechanical Engineering

by

Anuar Santoyo Alum

Thesis Supervisor: Professor Zhang Yun

May, 2024

学位论文公开评阅人和答辩委员会名单

公开评阅人名单

无（全隐名评阅）

答辩委员会名单

暂无

关于学位论文使用授权的说明

本人完全了解清华大学有关保留、使用学位论文的规定，即：

清华大学拥有在著作权法规定范围内学位论文的使用权，其中包括：（1）已获学位的研究生必须按学校规定提交学位论文，学校可以采用影印、缩印或其他复制手段保存研究生上交的学位论文；（2）为教学和科研目的，学校可以将公开的学位论文作为资料在图书馆、资料室等场所供校内师生阅读，或在校园网上供校内师生浏览部分内容；（3）按照上级教育主管部门督导、抽查等要求，报送相应的学位论文。

本人保证遵守上述规定。

作者签名：_____ 导师签名：_____

日　　期：_____ 日　　期：_____

摘 要

本文针对薄膜翼飞行机器人的性能参数优化展开理论分析、结构设计和样机实验。通过设计并制造定量化测试飞控性能的实验台架，研究了协方差矩阵自适应（CMA）进化算法。利用台架实验和算法完善来提高蝙蝠机器人的性能和结构设计。利用进化迭代算法逐步优化蝙蝠机器人的飞行控制参数。这种迭代过程不仅细化了控制参数，而且有助于机器人结构设计的日趋完善。通过结合 CMA 进化策略，实现了蝙蝠机器人的飞行动力学优化，特别是在参数调整方面，到达了机器人与自然蝙蝠在力学性能方面的趋同一致。在此基础上，该研究扩展了进化算法的使用范围，探索并量化了蝙蝠后腿运动对推力和升力产生的影响。值得注意的是，这项研究证实了蝙蝠自然生理的机械效率。蝙蝠机器人的机械参数收敛优化到了自然界中观察值。这些发现不仅验证了进化算法在扑翼机器人设计中的有效性，还增强了我们对飞行哺乳动物生物力学的理解，为未来的仿生机器人系统设计提供了一定的理论、设计和实验基础，也为扑翼机器人的工业应用提供了有启示的借鉴方案。

关键词：蝙蝠机器人，进化算法，协方差矩阵自适应，仿生机器人，控制策略

ABSTRACT

In this thesis, theoretical analysis, structural design and prototype experiments are carried out to optimize the performance parameters of the thin-film wing flying robot. The covariance matrix adaptive (CMA) evolutionary algorithm is proposed by designing and manufacturing an experimental bench for quantifying flight control performance. The performance and structural design of Batbot are improved by bench experiments and algorithm improvement. The flight control parameters of Batbot are optimized by evolutionary iterative algorithm. This iterative process not only refines the control parameters, but also contributes to the improvement of the robot structure design. By combining the CMA evolution strategy, the flight dynamics optimization of the bat robot is realized, especially in terms of parameter adjustment. Based on this, the study extends the use of evolutionary algorithms to explore and quantify the effects of bat hind leg movements on the thrust and lift generated. Notably, the study confirms the mechanical efficiency of bats' natural physiology. The mechanical parameters of the Batbot are convergent and optimized to the observed values in nature. These findings not only validate the effectiveness of evolutionary algorithms in the design of flapping wing robots, but also enhance our understanding of the biomechanics of flying mammals, and provide a certain theoretical, design and experimental basis for the future design of bionic robot systems, but also provide an inspired reference scheme for the industrial application of flapping wing robots.

Keywords: bat robot; evolutionary algorithm; covariance matrix adaptation; bio-inspired robotics; controlling strategies

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ACRONYMS

ML	Machine Learning
EA	Evolutionary Algorithm
Batbot	Bat Robot
PID	Proportional Integral Derivative
FC	Fuzzy Logic Controller
ANN	Artificial Neural Network
CMA	Covariance Matrix Adaptation
RL	Reinforcement Learning
PCA	Principal Component Analysis
LEV	Leading Edge Vortex
L/D	Lift-to-Drag Ratio
PIV	Particle Image Velocimetry
DOFs	Degrees of Freedom
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
DAQ	Data Acquisition System
FCC	Feed Forward Controller
P	Proportional Controll
I	Integral Controll
D	Derivative Controll
Kp	Proportional Gain
Ki	Integral Gain
Kd	Derivative Gain
IMU	JY90X Sensor

CHAPTER 1 INTRODUCTION

In the last couple decades, the world has seen an rapid development in Machine Learning (ML) algorithms, as well as the diversification of many branches in which this technology has found very useful applications. The areas in which this technology has proven to be of great utility ranges from speech recognition, computer vision, recommendation engines, fraud detection and many others. Engineering is a field in which ML has specially brought many technological advancement, from robots that recognize and interact with objects, to autonomous driving vehicles, the advantages ML has brought to the world of engineering are numerous. Within the field of engineering, we can encounter control algorithms, these try to solve the problem of commanding a robot (also referred as plant in control theory vernacular) to achieve certain action in an environment that is not possible to completely model or that is subject to unpredictable disturbances, such as the real world. If complete knowledge of the system and the environment were to be obtained, which is close to impossible, there would be no need for controlling algorithms. For this reason controlling algorithms play a very important role when developing, for example, robots that are to interact with the real world instead of only in a simulation or a perfectly controlled environment. Nevertheless, in some cases obtaining a good understanding of the system or its environment can be difficult or impossible. This makes the designing of a controlling algorithm extremely challenging. In this situations ML algorithm can be used to find optimal controlling strategies, that would have otherwise been close to impossible to find using traditional methods. Evolutionary algorithms (EA) are one of such ML algorithms that has proven to be successful for control strategy optimization[1].

Evolutionary algorithms are a subset from ML algorithm, that tries to solve an optimization problem inspired by Darwin's theory of evolution, which states that species evolved through generations by proliferating the genetic material of the specimens that had the best fitness with regard to their environment[2]. Generally speaking, the EA's consist of a population of possible solutions, whose fitness to the problem are tested using a fitness function, the resulting fitness scores are then used to generate a new generation of solutions. The iteration of this process aims to converge to an optimal solution of the optimization problem. One of the great advantage that EA has when solving optimal controller design is the fact that the fitness function can be arbitrarily selected, which means

that the definition of what an optimal design is can be freely chosen and the EA will try to optimize towards it. In contrast with, for example, neural networks that must have a differentiable loss function in order to optimize using back-propagation. For this reason, EA has been used by many researches to develop control designs[1].

A great problem found in today's research, is that most research was made in simulations and not in real life[1]. Simulations are a great tool, as they allow quick (in some cases) and safe training and testing of solutions, nevertheless it inherently involves mismatches with real world setting. This mismatch is commonly known as the Sim-to-Real Gap. To overcome the Sim-to-Real Gap, further optimization of the proposed solutions should be implemented to finally be able to use the controllers in real robots. Another problematic that was shortly mentioned before, is the fact that there are some systems and environments that are too complex to simulate or would require immense computational power to simulate. A solution for both of this problems would be to directly train the control algorithm using the robot itself. This also has its non-trivial drawbacks, that will be mentioned further on. In this thesis we propose the development of a framework that uses EA to design con- trolling algorithms, which test the proposed solutions on a real robot and use the resulting measurements to calculate the fitness score of the solution. In order to test and develop this framework a bat robot (Batbot), developed by Liqing Hu at the lab, will be used as a test case to find several optimal parameters regarding controlling strategies for flight and hovering. It must be remarked, that due to its complexity, the ultimate goal of hovering is out of scope of this thesis, nonetheless choosing Batbot hovering control as a case study has many advantages to test the developed framework. First of all, the Batbot is a flapping wing robot, whose wings are made of an elastic membrane, which makes the Batbot difficult to mathematically model. Additionally, hovering simulations would need fluid simulations, which combined with the elastic membrane problematic, makes Batbot hovering computationally expensive. Second of all, even though hovering is not the ultimate goal, it can be used to test the improvement of the control strategies, as it is possible to measure how close is the Batbot to achieve hovering, making the metric extremely quantifiable . This will enable us to analyze the development of the control algorithm through time, from which we will able to determine if the proposed framework actually works or not. Being able to prove a functional training framework on real robots could potentially open the possibility of creating control strategies for high complexity systems, where using a simulation approach is not possible.

Another great advantage of using the Batbot as our test case is the utilization of an evolutionary algorithm to explore the role of the hind leg in force generation. This offers a trans-formative approach to robotic design, particularly for the Batbot. Given the complexity of mimicking biological locomotion, where the hind legs play a crucial role in initiating and sustaining flight, the adaptability of evolutionary algorithms is invaluable. These algorithms adeptly navigate through a multitude of parameter configurations, optimizing for the most effective solutions that traditional methods might overlook due to their computational rigidity. Second of all, this approach not only enhances our understanding of the bio-mechanical processes involved but also serves as a test bed for refining control strategies in real time. By quantifying the improvements in force generation through iterative testing, the evolutionary algorithm provides a robust framework for evolving the Batbot's capabilities, thereby paving the way for advancements in robotic mobility that could be difficult to achieve through static modeling techniques. This methodology not only substantiates the potential of evolutionary algorithms in complex robotic systems but also sets the stage for groundbreaking developments in robotic applications where direct experimentation is challenging.

1.1 Evolutionary Algorithms to train Robots

Using EA to optimize or tune controlling algorithms is not a new idea. Extensive research has been made in the last couple decades, in which the applications of EA for controlling have shown positive results in several fields of engineering[2-13],[14-17]. Most of the time, the researches try to optimize controller that are already broadly used in the industry, such as the well known proportional-integral-derivative (PID) controller, from which 90% of industrial controlling applications are made of[18]. Many researches have focused on comparing the PID controller that was optimized using EA, with the conventionally used Ziegner-Nichola PID controller optimization strategy obtaining positive results[4-5,11]. In[11] the researchers were able to prove that the EA method delivered a PID controller that was better than if optimized using the traditional method, the PID controller found had half the rise time, was 25 times quicker to settle and had a mean squared error 110 times smaller than the controller found using the Ziegler Nichols method. Fuzzy logic controller (FC), another well established controller, received notorious attention[2-3,5-6,10,12,15,19]. Researchers used EA to optimizes different parts of the FC controller. For example, in[12] an EA was used to find the optimal shape and posi-

tion of the membership function, whereas in[5] the membership functions were fixed and instead the rule based was optimized using the EA. In[6] both approaches were successfully tested. The last popular controller algorithm optimized with a EA that we will like to mention is the artificial neural network (ANN). In[20] the authors optimize the weights of an ANN using an EA. They were able to show positive results, the advantages of using their proposed methods include the fact that no mathematical model is necessary for the controlled object, they remark the simplicity of the algorithm as well as its robustness. Other research including the use of ANN and EA include[2-3,21]. Even though many other controlling algorithm were optimized used EA, according to our research PID, FL and ANN have the most relevance.

Since the appearance of the covariance matrix adaptation (CMA) EA, many researchers have focused on comparing its performance to other EA to realize that it drastically improves the convergence rate of the optimization[14,22-24]. In[22] a deep analysis of the state of the art EA and its uses is made. They conclude that CMA is one of the most widely used EA at the moment. The applications of using CMA-EA are aswell diverse. From the development of a controller for antiviral therapy on hepatitis B[25] to PID parameter tuning for a robotic arm[26]. Particularly in the area of PID parameters optimization CMA has been able to outperforms all similar classes of learning algorithm[18].

As impressive as all the before mentioned papers are, an important remark must be made namely the fact that they were all tested within simulations. Applying these optimization strategies to the real world can be challenging, such as premature convergence, the trade off between exploration and exploitation or huge dimensionalities problem[1,27]. For this reason, even though EA in industrial applications are many, we see that the real world applications are less common[28], leading to a large gap between theory and reality[1]. None the less some researchers have managed to apply the knowledge obtain in their simulations to real robots, overcoming the Sim-to-Real Gap. For example, in[29] researches were able to train a robot using a virtual environment and then transfer the obtained knowledge to the robot, without the need of previous real-world training the robot was able to accomplish the task at hand . Similarly, in[17] a simulation of a room with obstacles was used to train a mobile robot to navigate, later the resulting optimal controlling was applied at the robot, which was able to navigate in the real-world. Taking advantage of expert knowledge rules from FC, a mobile robot was optimized using EA in the real-world[10].

A remarkable achievement was obtain by a group of researchers that were able to train a dog robot to walk and react to external disturbances from scratch in the real world. The whole training was achieved within an hour. For this they used a reinforcement learning (RL) algorithm. It most be noted that the robot required high computationally capable processor, which represent a lot of weight[30]. Something that would not be able to apply to the case study proposed in this thesis, as weight is a crucial factor on flying robots.

It can be clearly seen that optimizing controllers using CMA is a well researched promising research direction, of course most of the research done so far has been done using simulation. The challenges of training a robot controller in the real world are various, they include (as before mentioned) high dimensionality problem[1]. To tackle this problem in our Batbot hovering case study we can use real bat flight data to drastically reduce the dimensionality of our optimization by smartly designing controlling algorithm that resemble real bat flight, similar to what was accomplished in[31], where real data from ants was used to train virtual ant muscle models using CMA as well. Another problem encountered is the need of many iterations, which can be sub-optimal because of robot wearing. To solve this issue we plan to develop a multi-phased training framework were sub-tasks are developed which help the robot learn easier intermediate steps, similar to what was successfully implemented in[15,31]. Using this strategies to tackle the common challenges of real world controller training could prove to be crucial to be able to observe improving controller performance in the real world training scenarios.

1.2 Bat Robots

This review compiles significant advancements in the design and characterization of robotic bat wings, illustrating various methods and technologies employed to replicate and analyze bat flight mechanics.

In ”Designing Characterization of a Multiarticulated Robotic Bat Wing” by Joseph Bahlmann[32], a robotic replica of a fruit bat’s wing was developed seen in Figure 1.1, utilizing lightweight materials and string-controlled joints, similar to the way a Pinocchio puppet is controlled, Figure 1.2, 1.3.

”A New Model of Bio-Inspired Bat Robot” by Ghanbari Ahmad[33] introduces a robotic bat capable of mimicking the flight of bats, Figure 1.4. The model features four degrees of freedom in each wing, seen in Figure 1.5, using a four-bar mechanism that allows for controlled flight paths after applying specific control algorithms.

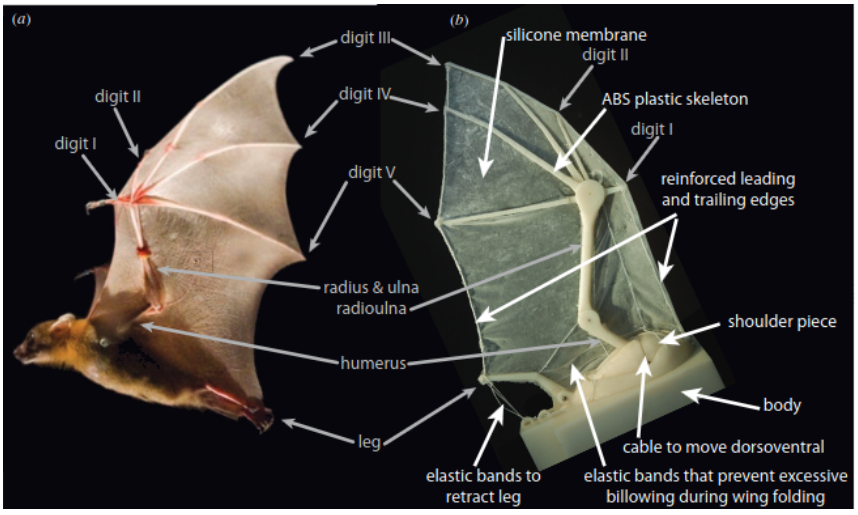


Figure 1.1 Relationship and inspiration for robotic design from [32]

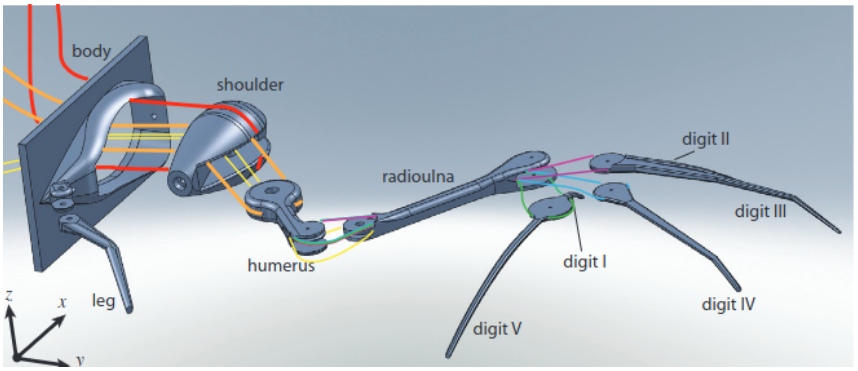


Figure 1.2 Novel design using string to actuate individual joints [32]

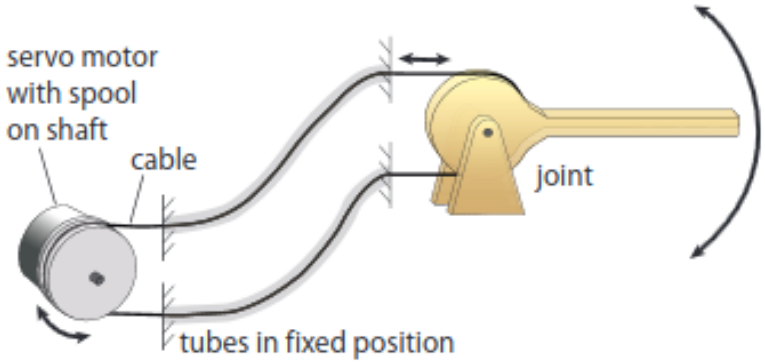


Figure 1.3 Design behind string activated joint [32]

In ”Trajectory Planning for a Bat-Like Flapping Wing Robot” by Jhonathan Hoff [34], a bat robot equipped with onboard electronics and a load sensor, seen in Figure 1.7, is developed to follow designated trajectories, Figure 1.6. The sensor aids in obtaining force data for parameter selection, facilitating efficient use in a nonlinear solver.

”MultiMo-Bat: : A Biologically Inspired Integrated Jumping Gliding Robot” by

Figure 1.4 Design of bat robot[33]

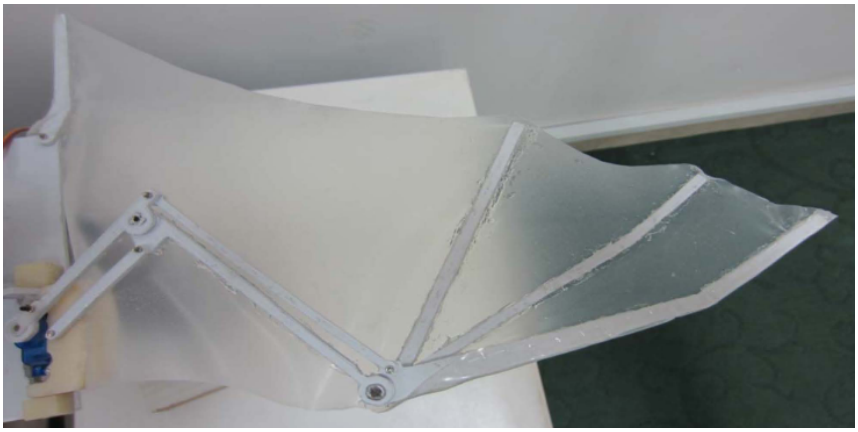


Figure 1.5 Articulated wing design[33]

Matthew A. Woodward[35] explores the jumping and gliding locomotion strategy mimicking that of vampire bats, Figure 1.8. This robot combines jumping to gain altitude followed by a transition to gliding mode as seen in Figure 1.9.

”Lagrangian Modeling and Flight Control of an Articulated Winged Bat Robot” by Ramenazi[36] discusses the development of the B2 bat robot, Figure 1.10, which simulates the articulated skeleton of bats as seen in Figure 1.11. The study focuses on understanding the bat’s flight dynamics related to physiological and morphological adaptations, introducing a holonomic constrained Lagrangian model and a method for inverse dynamics for flight control algorithm synthesis.

In ”Batbot B2: A Biologically Inspired Flying Machine” by Ramenazi[37], the previously developed B2 bat robot, Figure 1.12, as well as its wing joint design, Figure 1.13. is tested with a PD controller to manage the tail mechanism, achieving sustained flight as seen in Figure 1.14.

Lastly, ”Optimum Design of a Novel Bio-Inspired Bat Robot” by Tingting Sui[38] describes using principal component analysis (PCA) optimization algorithms to design a new bat wing structure, Figure 1.15. The design aims to closely simulate the real bat’s

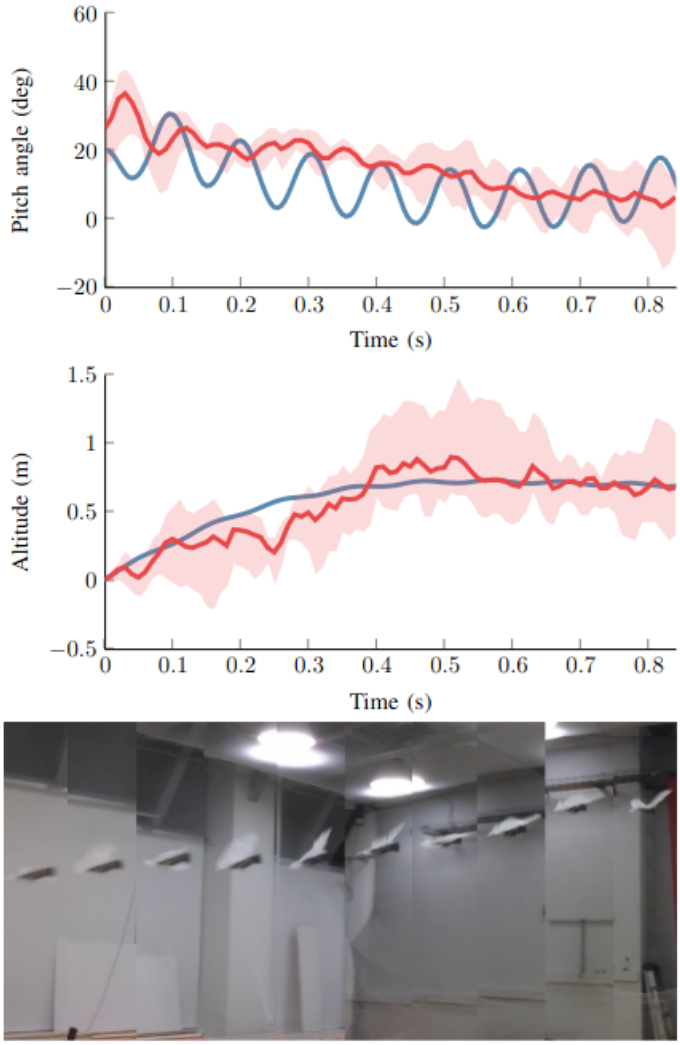


Figure 1.6 Flight trajectory from[34] and flight data

limb movements based on extensive movement data, enhancing the fidelity of the robotic bat's wing movements, seen in Figure 1.16. By using optimization algorithm, they direct the design of the robot to approach the real movement in a step-wise approach, similar to the known analogy of the donkey attempting to reach the hanging carrot.

1.3 Main Contents

In this thesis, I embark on a detailed exploration of the mechanics of bat flight, providing a comprehensive analysis of how real bats maneuver and utilize their unique physiological structures to achieve agile and efficient flight. This analysis sets the foundation for the subsequent sections where I introduce the design and kinematic analysis of the Batbot, a bio-inspired robotic counterpart designed to emulate these natural flight mechanisms.

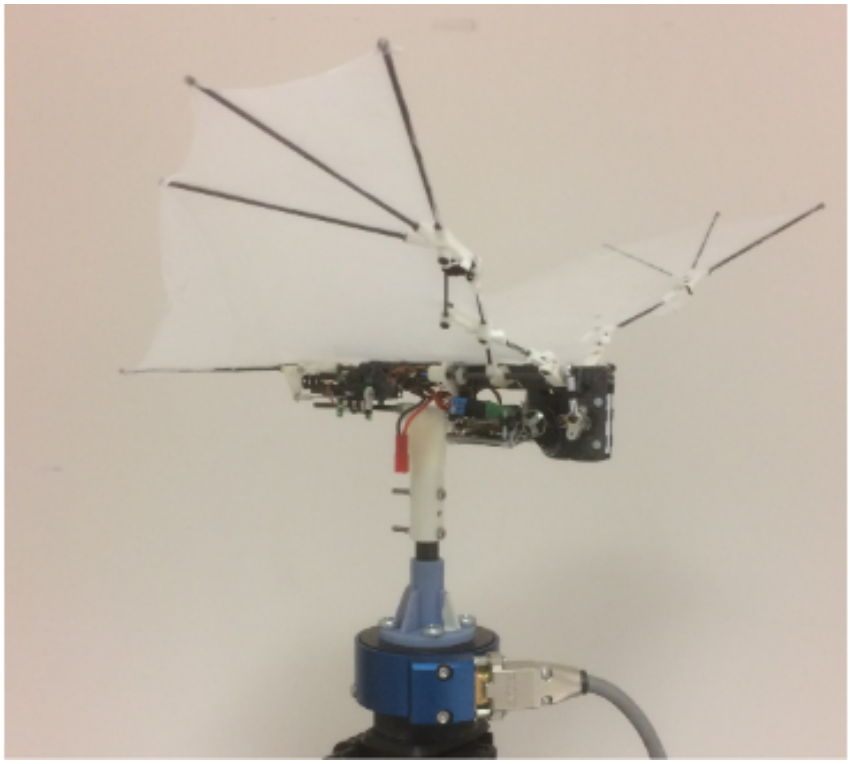


Figure 1.7 Bat robot mounted on load sensor[34]

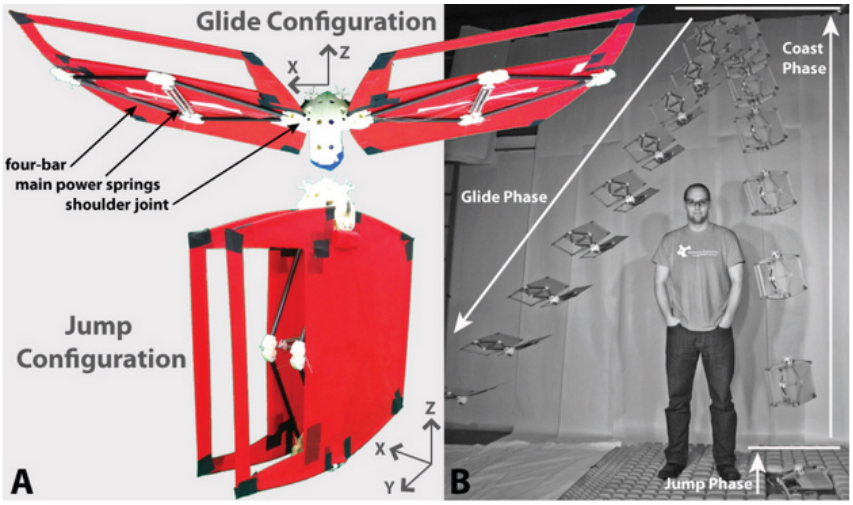


Figure 1.8 Jumping-gliding robot design (A) and trajectory (B)[35]

The development of an evolutionary algorithm strategy forms a core component of this thesis. I detail the mathematical foundations, construction, and workflow of the algorithm, which was designed to optimize the Batbot's flight control parameters. This strategy not only aids in fine-tuning the Batbot's movements but also facilitates a deeper understanding of the intricate dynamics involved in its operation.

A series of experiments ranging from free flight tests to the meticulous tuning of controlling parameters illustrates the practical applications of the theoretical frameworks

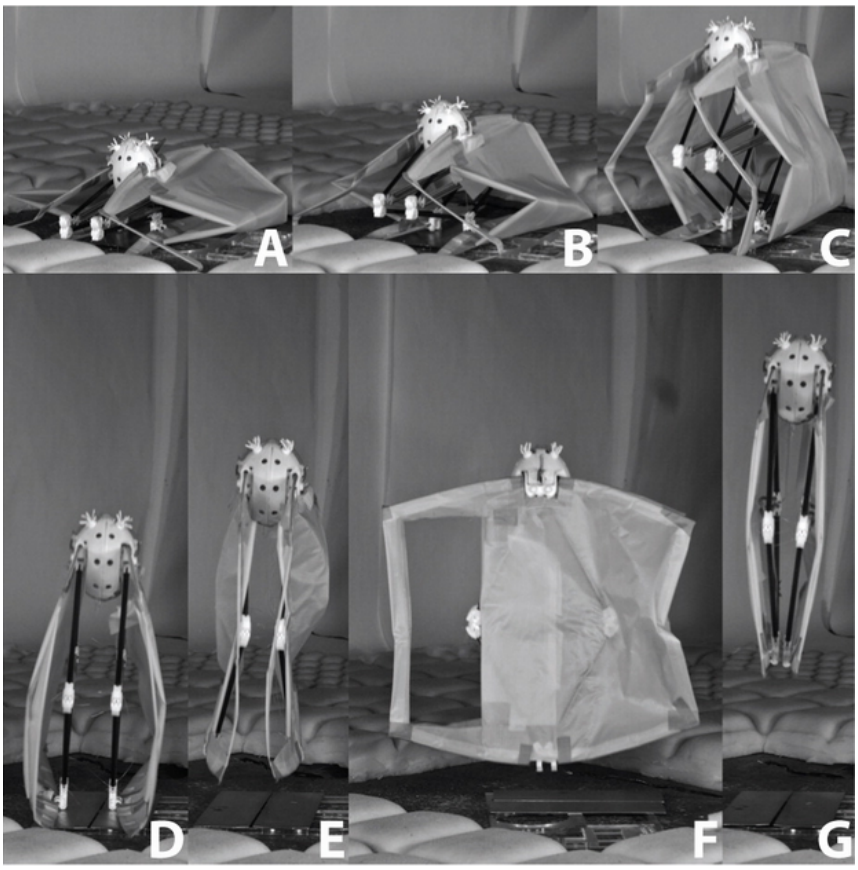


Figure 1.9 Snapshots of jumping actuation of robotic bat[35]

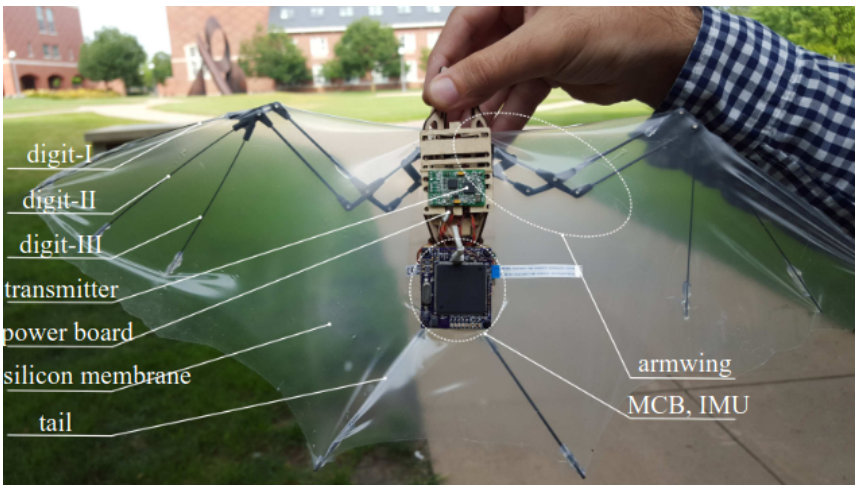


Figure 1.10 Design of bat robot equipped with onboard electronics[36]

developed. These experiments also include the development of a feedback controller to enhance the stability and responsiveness of the Batbot during flight.

Furthermore, the evolutionary algorithm strategy was employed to investigate the role of the Batbot's hind legs in force generation. By comparing these findings with data on real bats, the study helps bridges the gap between biological inspiration and robotic im-

Figure 1.11 Bat from whom[36] took inspiration

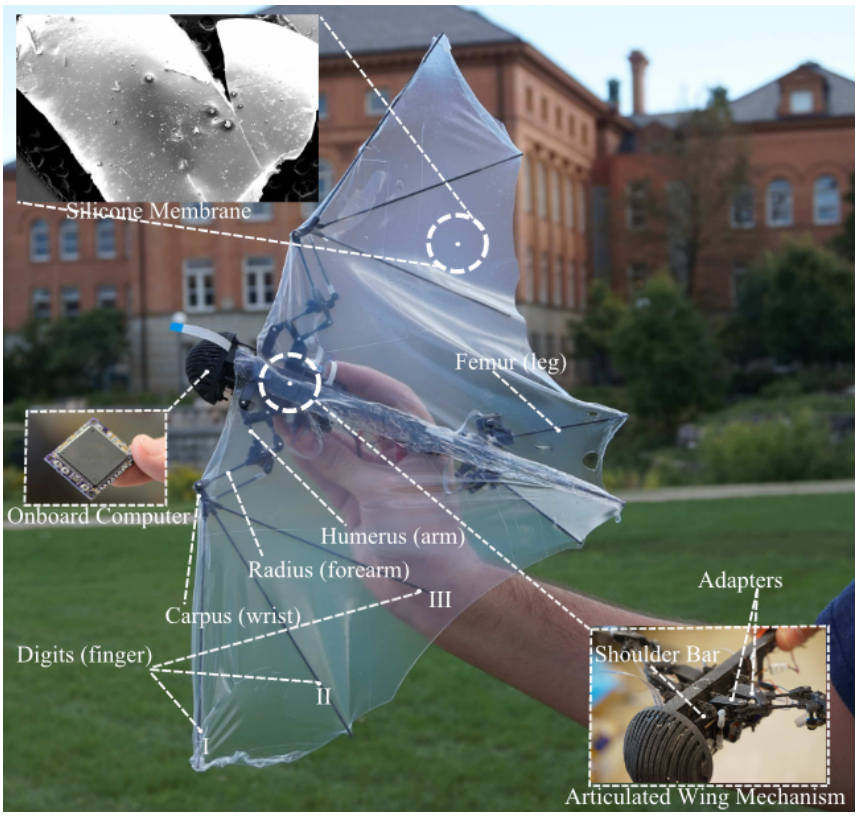


Figure 1.12 Bat robot from[37] with reference to bats bones

plementation, offering insights into the potential improvements in robotic design based on biological efficiencies.

The thesis concludes with a discussion on the iterative improvements made to the Batbot's mechanical design. Each iteration is described in detail, showing how experimental results informed successive modifications that progressively enhanced the robot's functionality and flight capability. Through this iterative process, the thesis not only contributes to the field of bio-inspired robotics but also demonstrates the effectiveness of adaptive, data-driven design approaches in complex engineering systems.

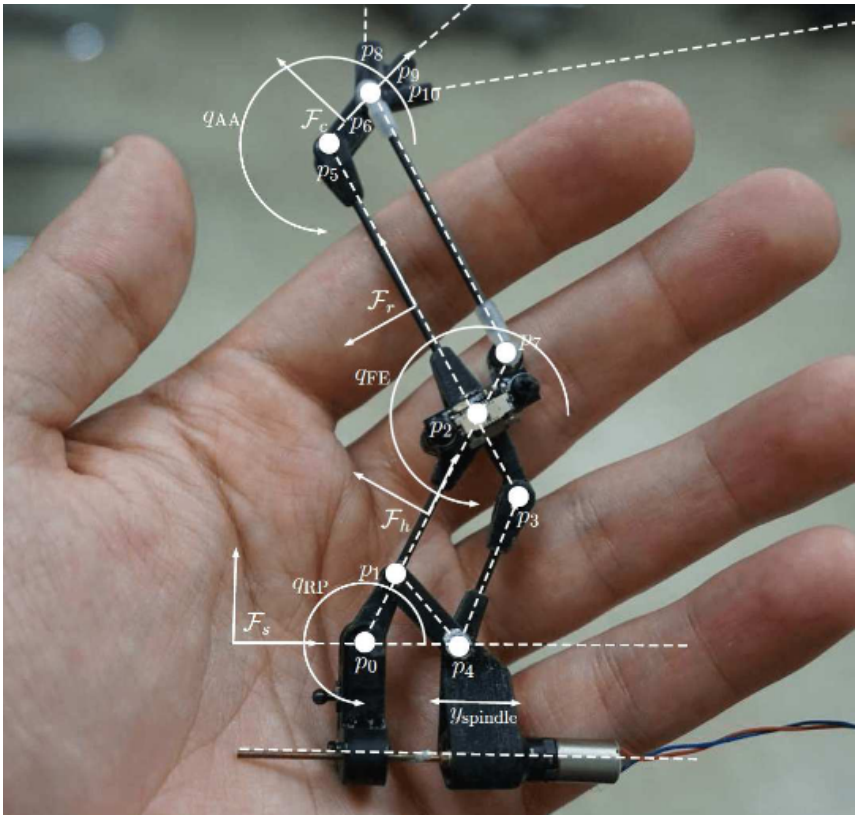


Figure 1.13 Joint mechanism of bar linkage of B2[37]

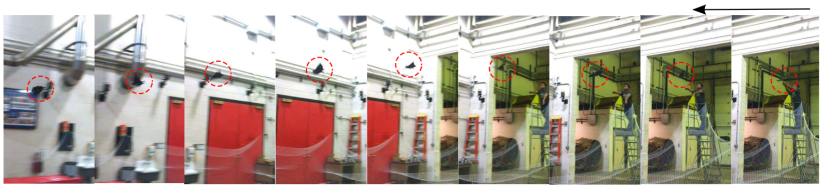


Figure 1.14 Trajectory of bat robot flight[37]

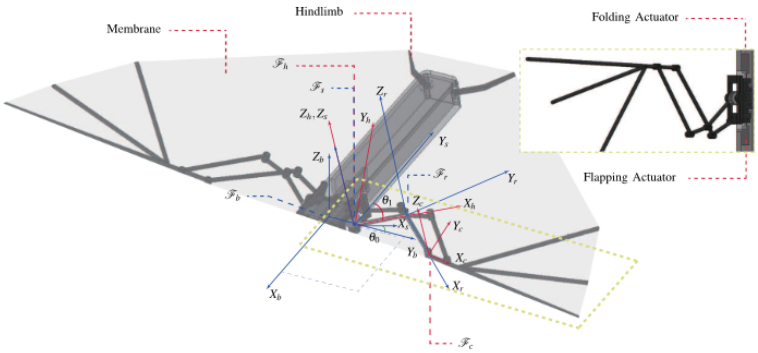


Figure 1.15 Bat robot design from[38]

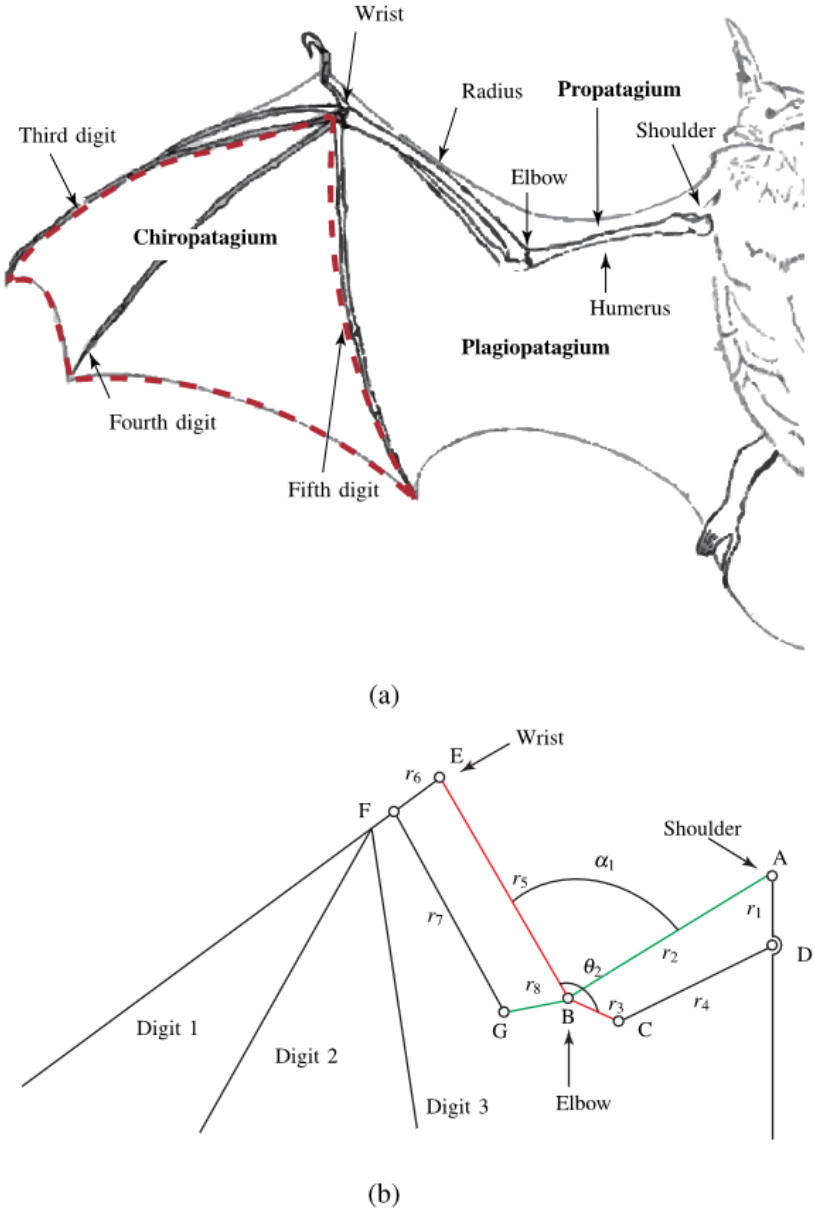


Figure 1.16 Comparison of real bat bones to joints in bat robot[38]

CHAPTER 2 THE BATBOT

2.1 Bat Flight Inspiration

The aerodynamics of bat flight involve complex interactions between wing morphology, kinematics, and flight dynamics. In their work[39], Hedenström and their team used particle image velocimetry, set up as seen in Figure 2.2,

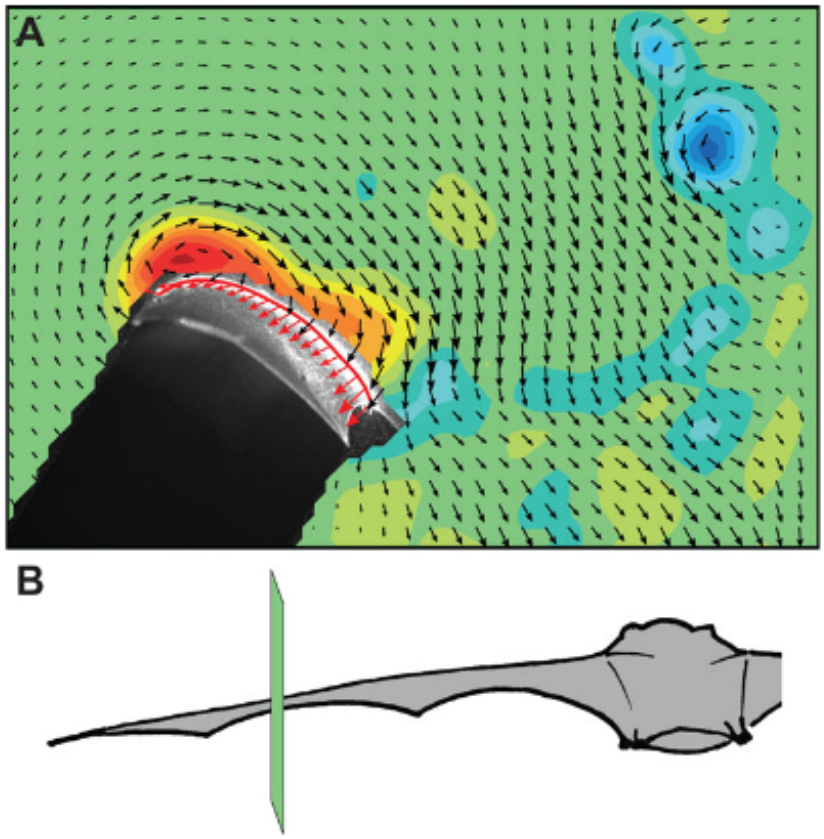


Figure 2.1 Vortex analysis from bat (A) and cross section of the analysis (B) made in[39]

for studying aerodynamics of the bats flight, seen in Figures 2.1. Following is the description of the secrets behind the aerodynamic of bat flight. This summary captures the key aerodynamic features and mathematical models described in the research on bat flight aerodynamics.

The bat wing generates lift and thrust through flapping, which varies with the flight speed denoted as U. At zero or slow speeds, bats employ unsteady aerodynamic mechanisms to enhance lift. The aerodynamic force on the bat wing, according to fixed-wing

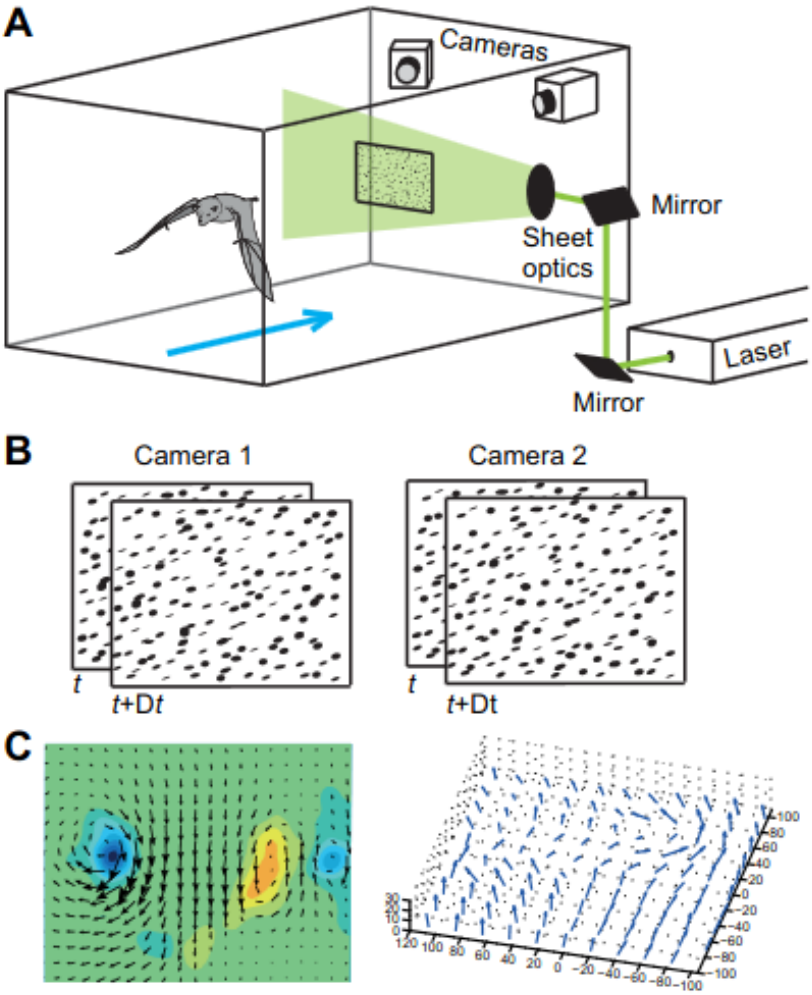


Figure 2.2 Particle image velocimetry setup made in[39]

theory, is given by:

$$L = \frac{1}{2}\rho S C_L U_{\text{eff}}^2 \tag{2.1}$$

where L is lift, ρ is air density, S is wing surface area, C_L is the lift coefficient, and U_{eff} is the effective velocity of airflow over the wing.

Unsteady aerodynamics are crucial for bats, especially at lower speeds, to support weight and maneuver efficiently. The leading edge vortex (LEV) significantly contributes to lift by adding circulation above the wing. The lift coefficient exceeding steady maximum values suggests the presence of unsteady aerodynamic phenomena:

$$\frac{\Gamma}{Uc} = \frac{L}{\rho U^2 S} = \frac{C_L}{2} \tag{2.2}$$

where Γ is the circulation, c is the mean wing chord, and U is the forward speed.

Despite bats being less efficient in cruising flight compared to birds, they exhibit su-

perior maneuverability. The aerodynamic efficiency, measured by the lift-to-drag ratio (L/D), varies with the ecological roles and flight behaviors of different bat species.

The aerodynamics of bat flight illustrates a refined adaptation to their ecological niches, allowing them to exhibit unique flight capabilities among flying mammals. The interplay between wing kinematics, aerodynamic forces, and flight morphology remains a significant area for future research, especially with advancements in flow visualization techniques like particle image velocimetry (PIV).

2.2 Structure Design of Batbot

The Batbot stands out as an exemplary model in robotic engineering, designed to closely imitate the flight mechanics of a bat. Its structure comprises a body crafted from carbon fiber, ensuring both lightness and robustness, with the overall assembly weighing approximately 650 grams. The wings, pivotal to its operation, span 1446 mm and are constructed from a silicone membrane that emulates the flexibility and durability of biological bat wings. Mobility and precise movement control are achieved through the integration of servo motors, which manage the legs and the wing-folding mechanism, complemented by a DC motor that powers the essential flapping motion. The intricacies of this flapping are driven by gears that were specifically 3D printed to match the unique requirements of the Batbot, optimizing both the energy efficiency and the fluidity of movement. Central to the orchestration of these components is a microcontroller, Figure 2.4, 2.5, which seamlessly coordinates the various motors and gears, ensuring synchronized operations and responsive maneuvers, vital for the complex aerial capabilities of the Batbot. This advanced integration of materials, motor technology, and control systems underscores the Batbot's cutting-edge design in the field of robotics. In Figure 2.3 we can see the CAD model of the Batbot.

2.2.1 Primary Motion Analysis

A synergistic design approach was employed to incorporate several mechanical linkages in the morphing mechanism of Batbot. This morphing mechanism requires four actuators, while at the same time being capable of producing biologically meaningful movements. These motions include: synchronous flapping motion of the left and right forelimbs, synchronous folding- unfolding motion of the two forelimbs, and asynchronous dorsoventral movement of each leg. Batbot's forelimbs are constrained to one degree

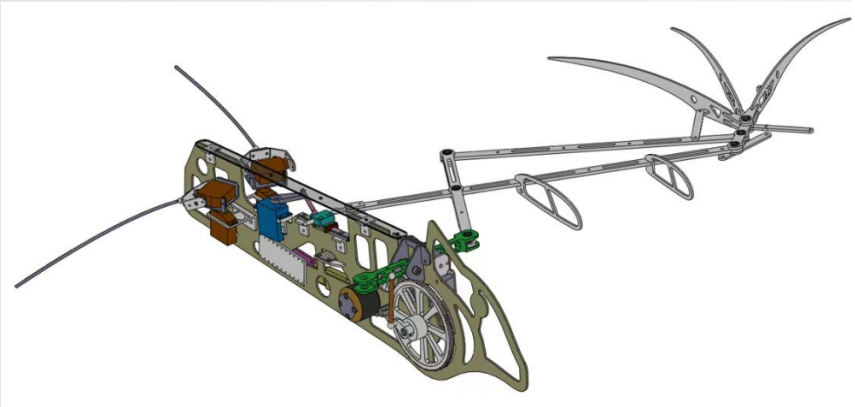


Figure 2.3 CAD model of the Batbot

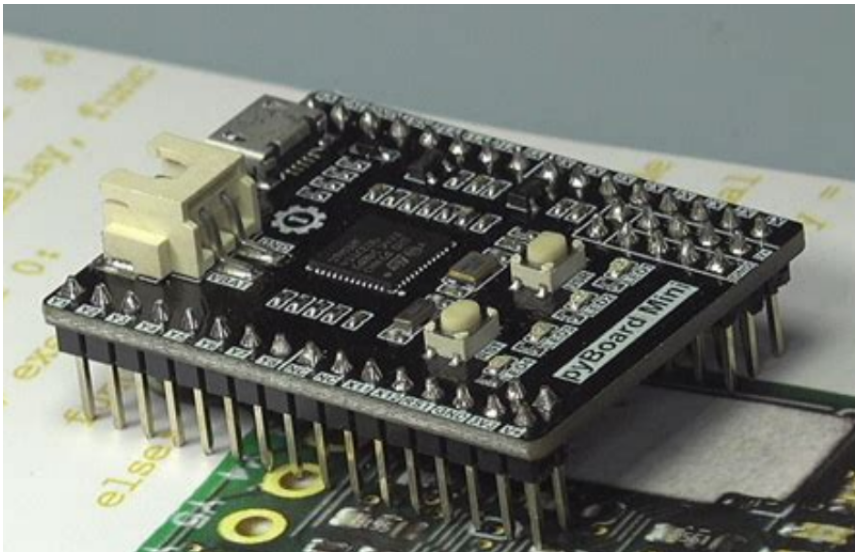


Figure 2.4 PyBoard Module used to control the Batbot

of freedom, shown in Figure 2.6, providing three coupled active movements: humeral retraction-protraction, elbow flexion-extension, and carpal abduction-adduction. These movements are observed in the shoulder, elbow, and wrist of biological bats and are therefore considered biologically meaningful movements. The forelimb mechanism is manipulated by movement of the slider, where D is constrained to move along the x axis of the body frame. By understanding the displacement of the OB (output slider) when the skeleton is fully extended and tucked, as shown in Figure 2.6, one can determine the configuration of the forelimb.

Assuming the distance between the slider and the shoulder in the y-axis direction are:

$$x_s = s - l_1 \sin\theta_1 - l_2 \cos\theta_3 \tag{2.3}$$

Where θ_1, is the angle of crank rotation, θ_3 is the angle between the connecting rod

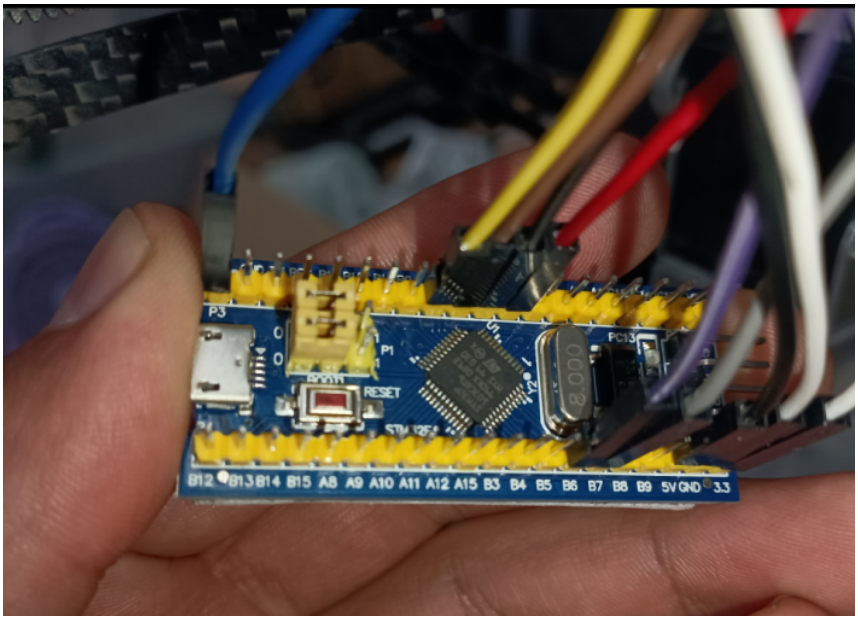


Figure 2.5 STM32 Micro-controller "Bluepill" used for initial testing, running C++.

and the slider. The length of auxiliary line DH can is given by

$$\overline{DH} = \sqrt{x_s^2 + (c - l_3)^2} \tag{2.4}$$

By using the trigonometric relationship, we can get the angle of the shoulder joint (θ_s), namely:

$$A = \frac{x_s}{c - l_3} \tag{2.5}$$

$$B = \frac{h_1^2 + \overline{DH}^2 - r_1^2}{2h_1\overline{DH}} \tag{2.6}$$

Similarly, we can get the elbow joint angle (θ_e) and wrist joint (θ_w), namely,

$$\theta_e = 2\pi - \arccos C \tag{2.7}$$

$$\theta_w = \pi - \arccos D - \arccos E \tag{2.8}$$

where,

$$C = \frac{r_1^2 + h_1^2 - \overline{DH}^2}{2r_1h_1} \tag{2.9}$$

The change in shoulder, elbow and wrist angles are presented in Figure 2.7 .

Batbot's hindlimbs of length l are thin steel wire rods. These rods are connected to

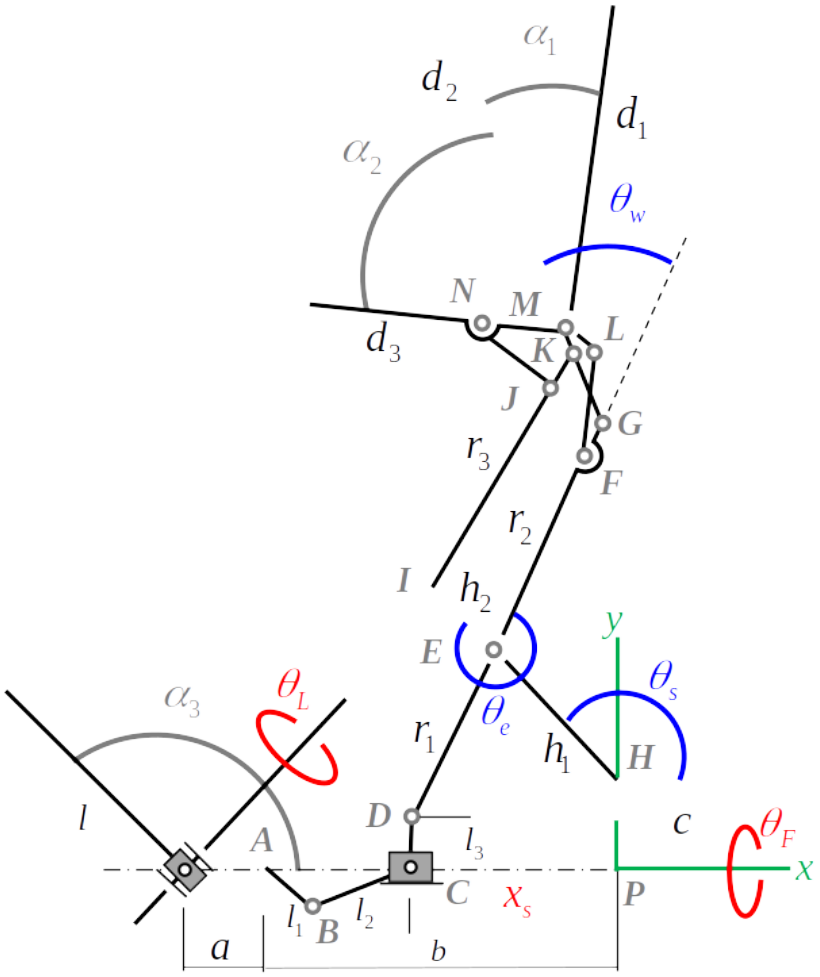
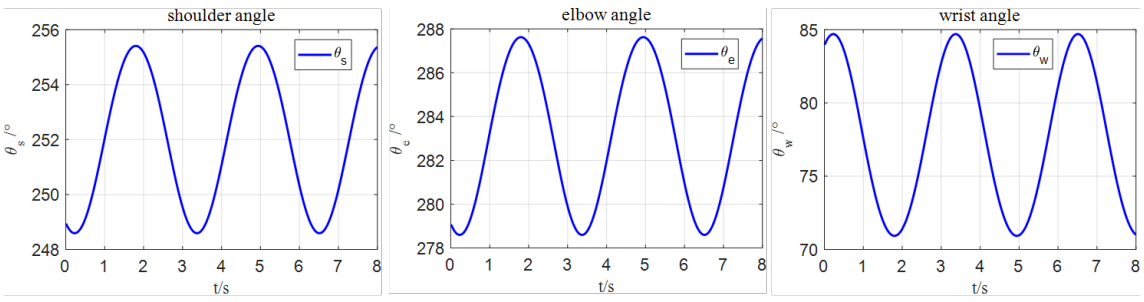


Figure 2.6 Degrees of freedom (DOFs) and morphological attributes of the Batbot are depicted in the diagram. Hinge joints are indicated by gray variables, while black variables illustrate the structural characteristics of the Batbot. Coordinate frames are highlighted in green. Blue arrows represent biologically significant angles in the left forelimb that are not directly actuated, and red arrows point to angles that are directly actuated.

revolute joints on the tail of its structure, allowing each hindlimb to move in a plane rotated at an angle α_3 from the parasagittal plane. The body length between the shoulder and hip is a plus b.



(d) (e) (f)

Figure 2.7 (d-f) Evolution of shoulder angle, elbow angle and wrist angle versus time.

2.2.2 Quantifying Wingbeat Cycle for Synchronized Leg Movement in Bio-inspired Robotics

Achieving synchronous movement between the wings and legs of a bio-inspired robot is essential for replicating the complex locomotion patterns observed in nature. This chapter outlines a novel approach for quantifying the wingbeat cycle of a bio-inspired robot, facilitating precise control over its leg movements in synchronization with the wings.

The wingbeat cycle, consisting of an upstroke and downstroke, is pivotal in defining the rhythm of synchronized movements. The cycle begins with the wing in the lowest position (cycle zero), peaks at the highest point of the stroke (cycle π), and returns to the lowest position, completing the cycle at 2π.

To accurately quantify the wingbeat cycle, two critical measurements are required: the wing's position and its direction of movement (upstroke or downstroke). These measurements are obtained using an AS5048 magnetic encoder sensor, which calculates the wing's angle θ relative to the robot's body through the rotation of a magnet affixed to the wing's axis of rotation.

Given the inherent noise in sensor data, the derivative of the last four angle measurements $\theta_i, i = 1, 2, 3, 4$ is calculated and averaged to determine the wing's movement direction reliably.

The direction of the wing's movement can be determined by averaging the derivatives of the angle measurements over the last four points. The derivative at each point i is approximated by the finite difference:

$$\frac{d\theta_i}{dt} \approx \frac{\theta_i - \theta_{i-1}}{\Delta t} \tag{2.10}$$

where Δt is the time elapsed between measurements θ_i and θ_{i-1}.

Combining this with the formula for direction:

$$\text{Direction} = \text{sgn}\left(\frac{1}{4}\sum_{i=2}^{4}\frac{\theta_i - \theta_{i-1}}{\Delta t}\right) \tag{2.11}$$

This method uses the change in θ over time to determine whether the wing is in an upstroke or downstroke, simplifying the detection of the direction of motion. A positive average derivative indicates an upstroke, while a negative value indicates a downstroke.

Due to mechanical uncertainties and variances in flapping forces, the maximum (θ_{max}) and minimum (θ_{min}) angle measurements are dynamic. To address this, the last hundred angle measurements are recorded, with the maximum and minimum values representing

the peak and trough of the wingbeat cycle, respectively. This dataset ensures the capture of at least one complete wingbeat cycle and allows for dynamic adaptation to changes in the wing's maximum and minimum angles.

Normalization is then applied to map the current wing angle θ to a value between 0 and 1, with 0 representing the lowest and 1 the highest wing position:

$$\text{Normalized Angle} = \frac{\theta - \theta_{\text{min}}}{\theta_{\text{max}} - \theta_{\text{min}}} \tag{2.12}$$

Consider the normalized time within a wingbeat cycle, denoted by t, which ranges from 0 to 1. The interpolation of the angle θ during the wing's movement can be described by:

- **Upstroke** (positive direction):

$$\theta(t) = \pi t, \quad 0 \leq t \leq 1 \tag{2.13}$$

  where $t = 0$ corresponds to the beginning of the upstroke and $t = 1$ corresponds to the end of the upstroke, reaching π radians.

- **Downstroke** (negative direction):

$$\theta(t) = 2\pi - \pi t, \quad 0 \leq t \leq 1 \tag{2.14}$$

  where $t = 0$ corresponds to the beginning of the downstroke at 2π radians, and $t = 1$ corresponds to the end of the downstroke, reaching π radians.

This methodology allows for precise determination of the wingbeat cycle's phase, crucial for synchronizing leg movements with wing actions. By accounting for mechanical limitations and sensor inaccuracies, the approach ensures a high degree of synchronization accuracy, essential for the robot's aerodynamic performance and energy efficiency.

CHAPTER 3 EVOLUTIONARY ALGORITHM FOR OPTIMAL CONTROLLING STRATEGY

3.1 Covariance Matrix Adaptation Algorithm

The Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is designed for non-linear, non-convex optimization problems. Due to its high conversion rate and ease of application, it was decided to use the CMA as the evolution algorithm strategy. Here are the fundamental mathematical concepts that underpin the strategy[40].

3.1.1 Theory

The core of CMA-ES is the adaptation of the covariance matrix, which shapes the sampling distribution to better explore the search space. The covariance matrix $C(g)$ at generation g is updated based on the distribution of successful steps.

$$C(g+1) = (1-c_\mu)C(g) + c_\mu \frac{1}{\mu_{\text{eff}}} \sum_{i=1}^{\mu} w_i (x_{i:\lambda}^{(g+1)} - m(g))(x_{i:\lambda}^{(g+1)} - m(g))^T \tag{3.1}$$

Here, $x_{i:\lambda}^{(g+1)}$ are the μ best out of λ sampled solution candidates, w_i are the weights, and $m(g)$ is the mean value of the search distribution at generation g.

CMA-ES adjusts the step-size $\sigma(g)$ dynamically to control the scale of the search steps. It uses cumulative step-size adaptation based on the evolution path p_σ:

$$\sigma(g+1) = \sigma(g) \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|p_\sigma\|}{E\|\mathcal{N}(0,I)\|} - 1\right)\right) \tag{3.2}$$

Here, c_σ and d_σ are learning rates for the step-size, p_σ is the evolution path, and $E\|\mathcal{N}(0,I)\|$ is the expected length of a normally distributed random vector.

The adaptation of the covariance matrix uses both rank-μ updates, which consider the covariance information of the best μ solutions, and rank-one updates based on the evolution path, enhancing adaptation capabilities in multimodal landscapes:

$$C_{\text{rank-}\mu}(g+1) = \sum_{i=1}^{\mu} w_i (x_{i:\lambda}^{(g+1)} - m(g))(x_{i:\lambda}^{(g+1)} - m(g))^T \tag{3.3}$$

$$C_{\text{rank-one}}(g+1) = p_c(g) p_c(g)^T \tag{3.4}$$

CMA-ES is invariant to order-preserving transformations of the fitness function and to linear transformations of the search space, making it robust across a wide range of problems.

These components make CMA-ES a highly adaptive and robust approach for tackling complex optimization tasks, effectively balancing exploration and exploitation to converge towards optimal solutions.

3.1.2 Application

The key advantage of CMA-ES is its ability to adapt the shape of the search distribution to the shape of the objective function's landscape, making it capable of solving complex, real-world optimization problems like the Batbot controlling strategy.

The CMA algorithm, a critical component of our project, is leveraged through the Python package cma-es. This section outlines the initialization and practical usage of the CMA algorithm within our research framework, particularly focusing on its application to the Batbot's controlling strategy.

A solution proposed by the CMA algorithm is conceptualized as a n-dimensional vector (depending on the task), with each element constrained to values between 0 and 1. This constraint was introduced to streamline the coding process, allowing for a simplified representation of complex control strategies. Subsequently, these normalized values are transformed into actual parameters using linear interpolation that dictate the movements of the Batbot, taylore to each specific application.

With the aforementioned parameters defined, the CMA algorithm is initialized with the covariance matrix centered at a midpoint value of 0.5 across all dimensions. A standard deviation (σ) of 0.5 is applied to their normal distributions, setting the stage for the algorithm's optimization process to commence.

This initialization strategy equips the CMA algorithm with the necessary framework to begin exploring the optimal configurations for the Batbot's flight control.

3.2 General Optimization Workflow

In this chapter, we delve into the intricacies of the optimization algorithm designed to discover the optimal controlling strategy for the Batbot's. The workflow of this algorithm and its application plays a crucial role in advancing our understanding and operational efficiency of the Batbot. The core of our optimization strategy is implemented

through a Python script running on a computer. This script executes the CMA algorithm, a sophisticated method proposing ten distinct solutions aimed at enhancing the Batbot's performance. The specifics of what constitutes a 'solution' will be elaborated upon in subsequent chapters.

The proposed framework design positions the Batbot in a test-bench were measurements can be taken using sensors. A computer running the CMA-ES proposes possible solutions $sol_j^k, \quad j \in \{1, .., n\}$, where n represents the amount of solutions in one generation k of the EA, and j is the index of one solution within the generation k. Afterwards, each solution in a generation is wirelessly sent to the Batbot and tested in the test-bench. The sensors in the test-bench measure the sensor data $\vec{s}_j^k$ for each sol_j^k sent. Each $\vec{s}_j^k$ is sent back to the computer, where it is fed to a the fitness function $F(\vec{s})$ that gives a score to each of the tested solutions. The scores of the solutions are then provided to the CMA-ES that uses them to produce the next generation of solutions and the cycle is repeated. A graphical representation of the training cycle can be seen in Figure 3.1 and its application in Figure 3.2.

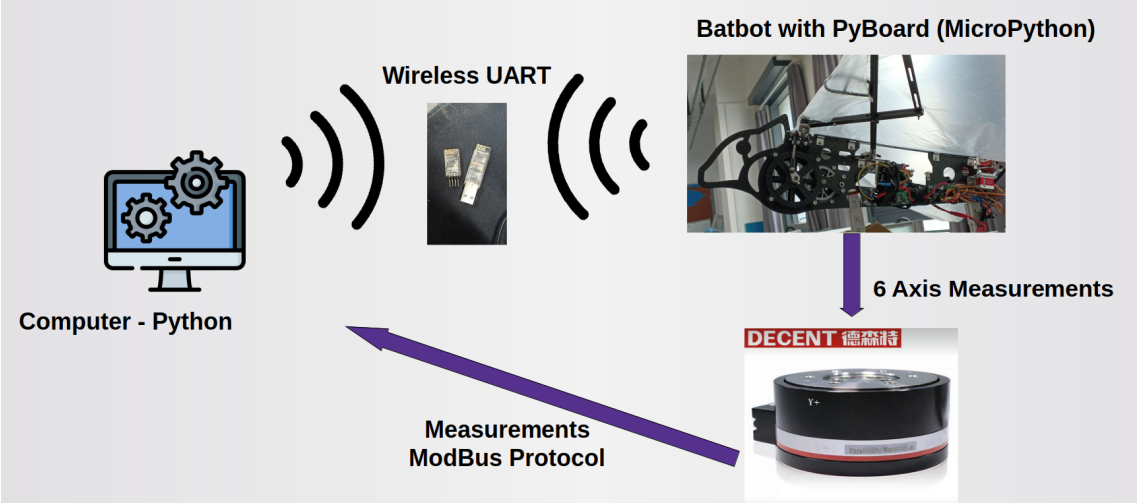


Figure 3.1 Workflow of the evolutionary algorithm strategy

Each proposed solution by the CMA algorithm is wireless transmitted to the Batbot for real-time testing. Beginning with the first solution, the Batbot undertakes a series of experiments to evaluate the effectiveness of each solution as suggested by the Python script. During these experiments, the sensor attached to the Batbot collects force data, capturing the dynamics of the Batbot. The ultimate goal is for the Batbot to minimize the defined score function. The exact definition of the score function will depend on the specific problem trying to be solved.

Following the collection of force data, this information is transmitted back to the

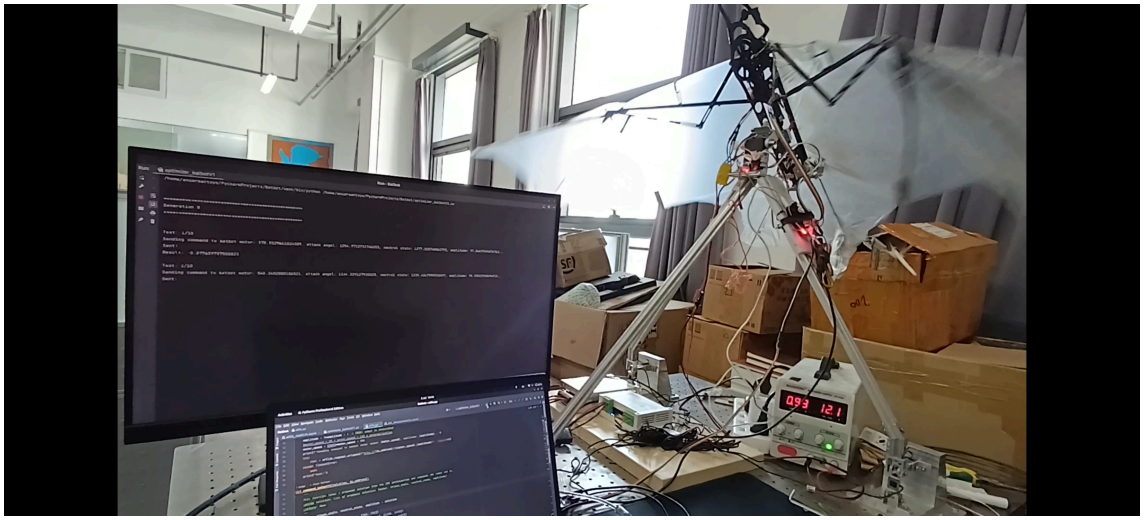


Figure 3.2 Optimization Algorithm in action, the Batbot positioned on the static test bench and on the monitor the commands sent from the CMA algorithm and the resulting scores are displayed.

Python script, using serial communication, initiating a crucial data processing phase. This phase ensures the sensor data is accurately captured, cleansed, and processed. Subsequently, calculations are performed to determine the forces directly relevant to the task. Based on these calculations, a performance score is assigned to each tested solution, reflecting its efficacy in achieving the goal.

With each experiment's completion and the subsequent scoring, the results are fed back into the CMA algorithm. This feedback loop is instrumental for the algorithm to refine its understanding and adjust its parameters accordingly. By analyzing the performance scores, the CMA algorithm optimizes its next set of proposals, generating a new generation of ten solutions aimed at closer approximating the optimal hovering strategy.

Through this iterative process, the Batbot incrementally learns which leg movements contribute to a lower score, signifying a closer approximation to successful task. This optimization cycle continues, progressively refining the Batbot's control strategies.

3.2.1 Wireless Connection

Achieving an authentic simulation of the bat's flying environment necessitates a robust wireless communication system between the computer and the Batbot. Various modules and strategies were employed to establish this crucial link, detailing the trials and eventual success in finding the most effective communication method.

Our initial approach utilized the Wi-Fi module ESP8266, aiming to establish a connection between the computer and the Batbot via a common Wi-Fi router. The communication was intended to be facilitated through GET requests to transmit the solutions

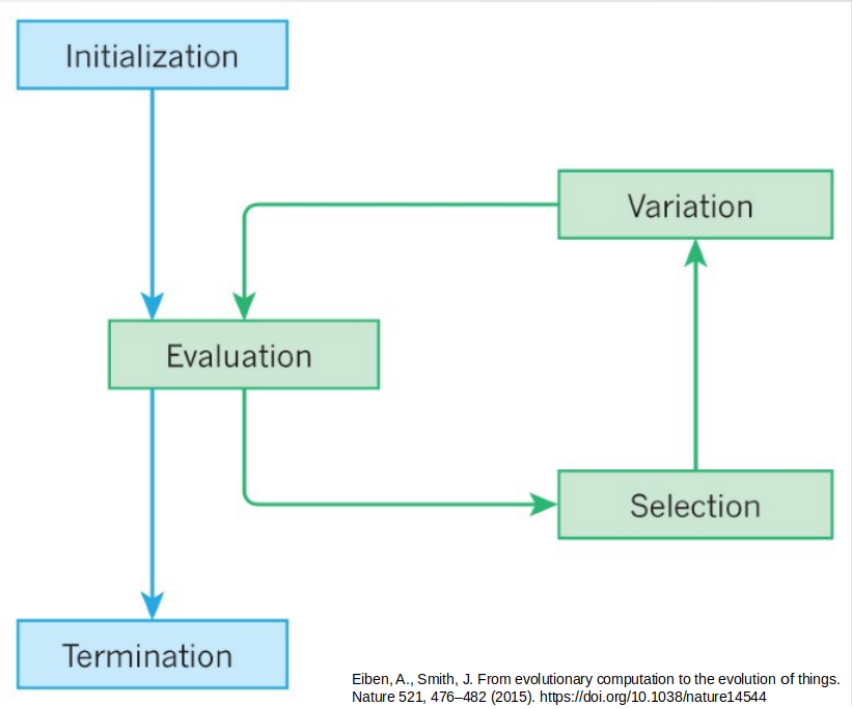


Figure 3.3 Diagram showing workflow of an Evolutionary Algorithm

proposed by the optimization algorithm. However, this method encountered significant reliability issues, with the ESP8266 module frequently losing connection, thereby hindering consistent data transmission.

The subsequent experiment involved the Bluetooth module HC05, which also presented challenges. Similar to the ESP8266, the HC05 module suffered from unstable connections and was prone to interference, making it an impractical choice for our purposes, Figure 3.4.

After evaluating the limitations of Wi-Fi and Bluetooth communications, we explored the use of a wireless UART module, 3.5, which emerged as the superior solution. The installation process on the Batbot was straightforward, and its lightweight design did not impose additional burdens on the Batbot's flight capabilities. The module's reliability was a significant improvement over the previous attempts, offering a stable and efficient communication channel.

The computer, running the Python script for the optimization algorithm, was equipped with a transmitter module. Through serial communication facilitated by a specific package in Python and configured with a baud rate of 115200, the system efficiently transmitted the proposed solutions to the Batbot. The Batbot, equipped with a corresponding receiver module, successfully received the data, which was then processed by its microcontroller to execute the optimization algorithm's proposed solutions.

The wireless UART module's success highlighted its suitability for our project, combining ease of installation, reliability, and efficient data transmission. This module effectively bridged the communication gap between the computer and the Batbot, allowing for accurate environmental simulation and real-time testing of optimal strategies.

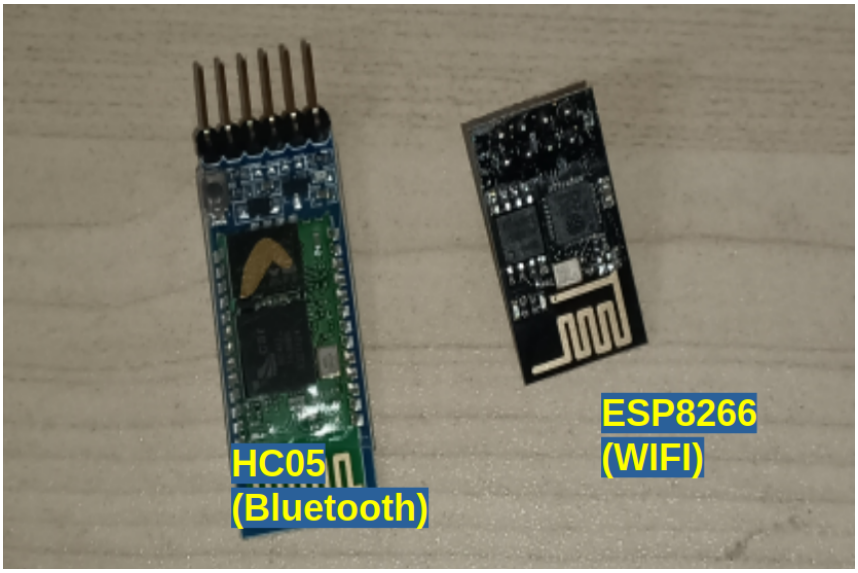


Figure 3.4 Wifi and Bluetooth wireless models

3.2.2 Data Acquisition System

The necessity for efficient data acquisition system (DAQ) became apparent as we sought to gather and communicate extensive sensor data to the computer for analysis. This section delves into the encountered challenges and the innovative solutions implemented to facilitate seamless data transfer to the Python script for automatic processing.

A significant hurdle was the integration of data acquisition systems with the computer, particularly for automatic data retrieval by the running Python script. While most DAS come with integrated software designed for ease of use through user-friendly interfaces, our project demanded a more complex interaction. Specifically, we aimed to automate the process of obtaining sensor data directly into the Python script to enable continuous experimentation without manual intervention.

This requirement posed a considerable challenge, necessitating advanced programming skills to achieve direct communication with the DAQ at a low-level programming interface.

Our breakthrough came from understanding and utilizing the MODBUS protocol for serial communication. By leveraging a specific Python package designed for serial com-

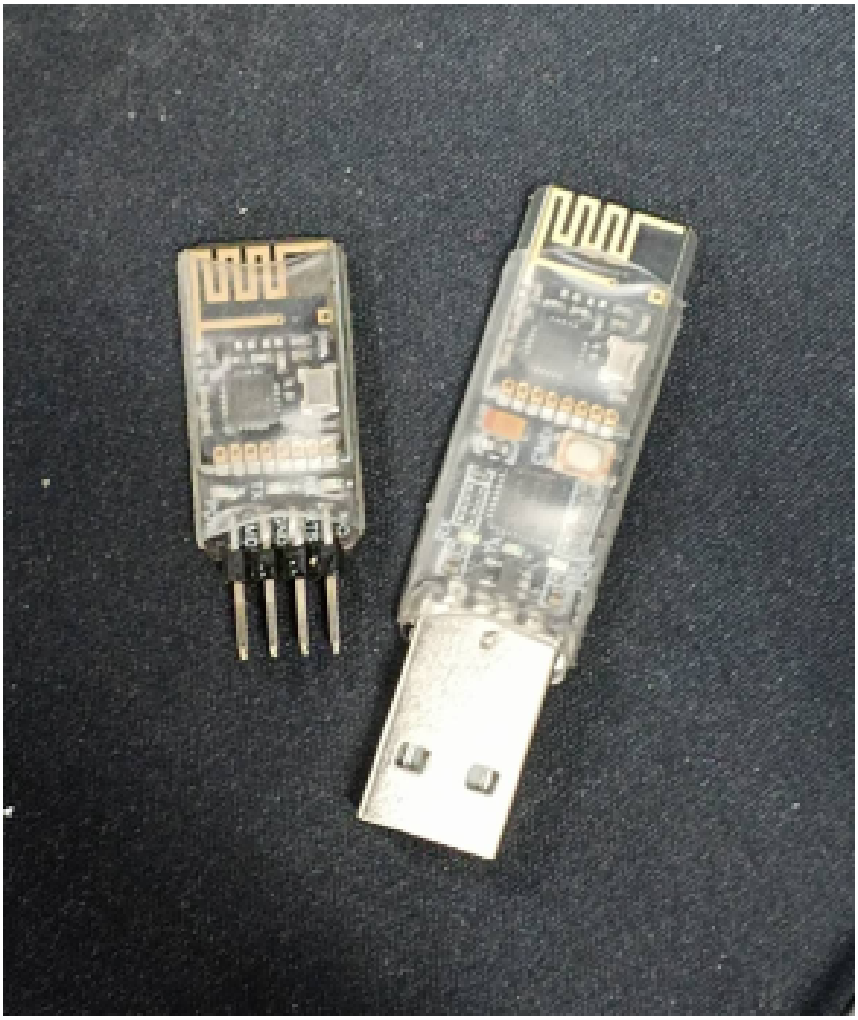


Figure 3.5 Wireless UART communication module.

munication, and referencing the correct registers as specified in the user manuals of our chosen DAQ, we established a bit- and byte-level connection with the DAQ.

The initial version of our static test bench, which utilized load sensors, was paired with the BSQ-JN-P8 DAQ 3.6. Although this setup theoretically supported data acquisition at frequencies of up to 100 Hz, we encountered limitations in the usefulness of the data obtained. The system appeared to provide a hundred data points per second, but in reality, most of these points were duplicates, resulting in a true data logging frequency of only about ten unique values per second. This limitation manifested in the data as high jumps between data points, rather than representing a smooth continuous function of forces, as illustrated in Figure 3.7

The decision to upgrade to a new six-axis sensor was partially driven by the need for more accurate data acquisition capabilities. The new sensor's DAQ significantly improved our data collection process, enabling us to capture real data at frequencies up to 100 Hz.

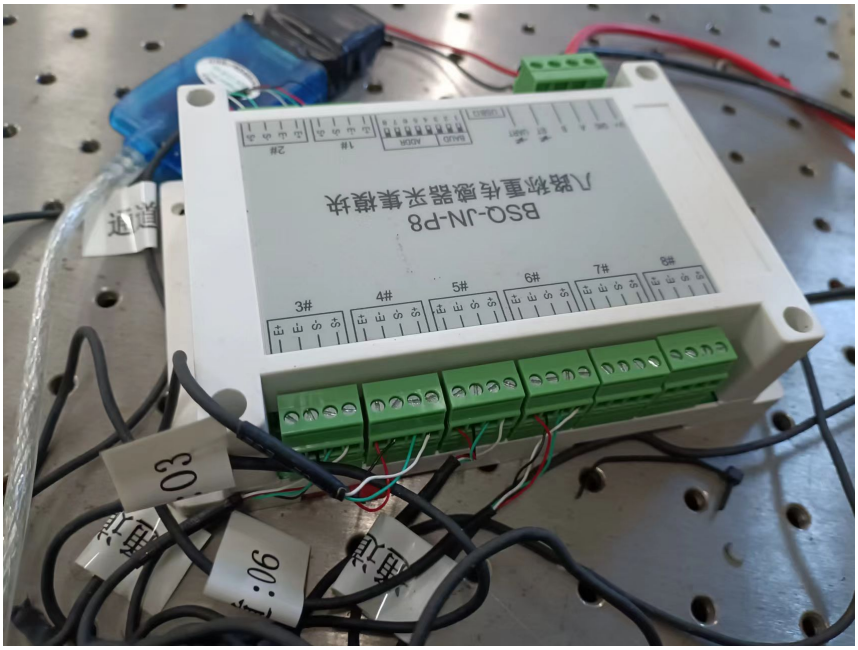


Figure 3.6 BSQ-JN-P8 Data acquisition system conected to the load sensors used in the first Vesion of the static test bench.

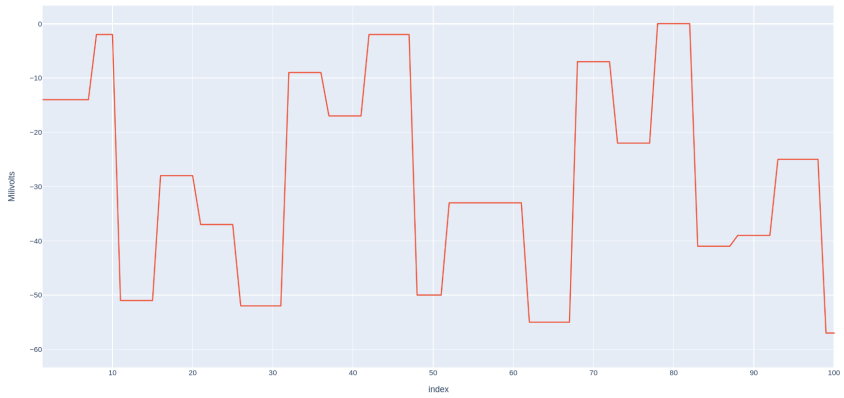


Figure 3.7 Low quality data obtained from the BSQ-JN-P8 DAQ, from which it can be seen that the the low precision results on plateau shaped measurements.

This advancement is evident in the more precise and reliable force data obtained, as shown in Figure 3.8, marking a substantial improvement over the previous setup. The journey from initial setbacks to successful data acquisition underscores the critical role of selecting appropriate hardware and developing bespoke software solutions in scientific research. Through persistent effort and innovative problem-solving, we achieved a system capable of meeting our project's specific needs, setting a foundation for accurate and efficient data analysis.

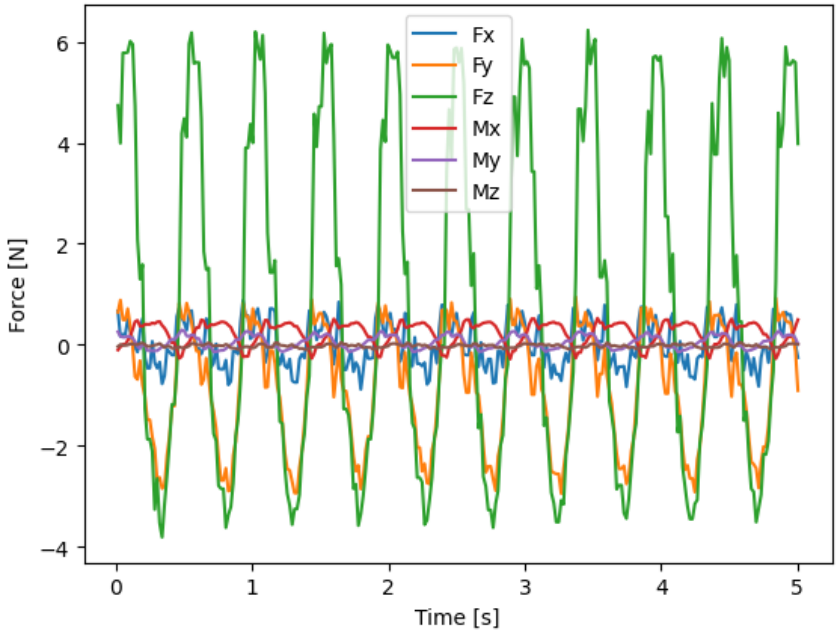


Figure 3.8 Example of the data acquired by the 6 axis sensor.

3.3 Proof of Concept Prototype

The training cycle explained requires diverse coding requirements that need to be brought together in order for the training cycle to work. Some of the coding blocks needed were: robot command, wireless communication between computer and robot, data acquisition from test-bench to computer, running CMA program were solutions were automatically sent and data was read and processed in scores to feed the CMA. In order to test and debug such framework a simple proof of concept prototype was developed, something easy to solve but complex enough to be able to test the whole framework in.

The designed prototype consisted on a servo motor that would oscillate for 8 seconds following a sinus function. The function was defined as $f(\theta, t) = \theta_1 * sin(\theta_2 * t)$, where t is the time and θ the parameter to optimize, $\theta_1 \in [0, 90]$ and $\theta_2 \in [0, 5]$. The servo motor was controlled using a NODEMCU which enable wireless communication with Wifi technology. Playing the role of the test-bench was a potentiometer physically attached to the servomotor to measure its angle. The potentiometer was connected to an Arduino Uno which was connected to the computer running a Pyhton script using serial communication. In figure 3.9 the prototype, NodeMCu and Arduino can be seen and in Figure 3.10 the servo and potentiometer configuration can be seen.

The optimal parameter θ_{opt} was arbitrarily chosen to be $\theta_{opt} = [60, 3]$. The Python script was ran using 5 solutions per generation. After 41 generations the algorithm was

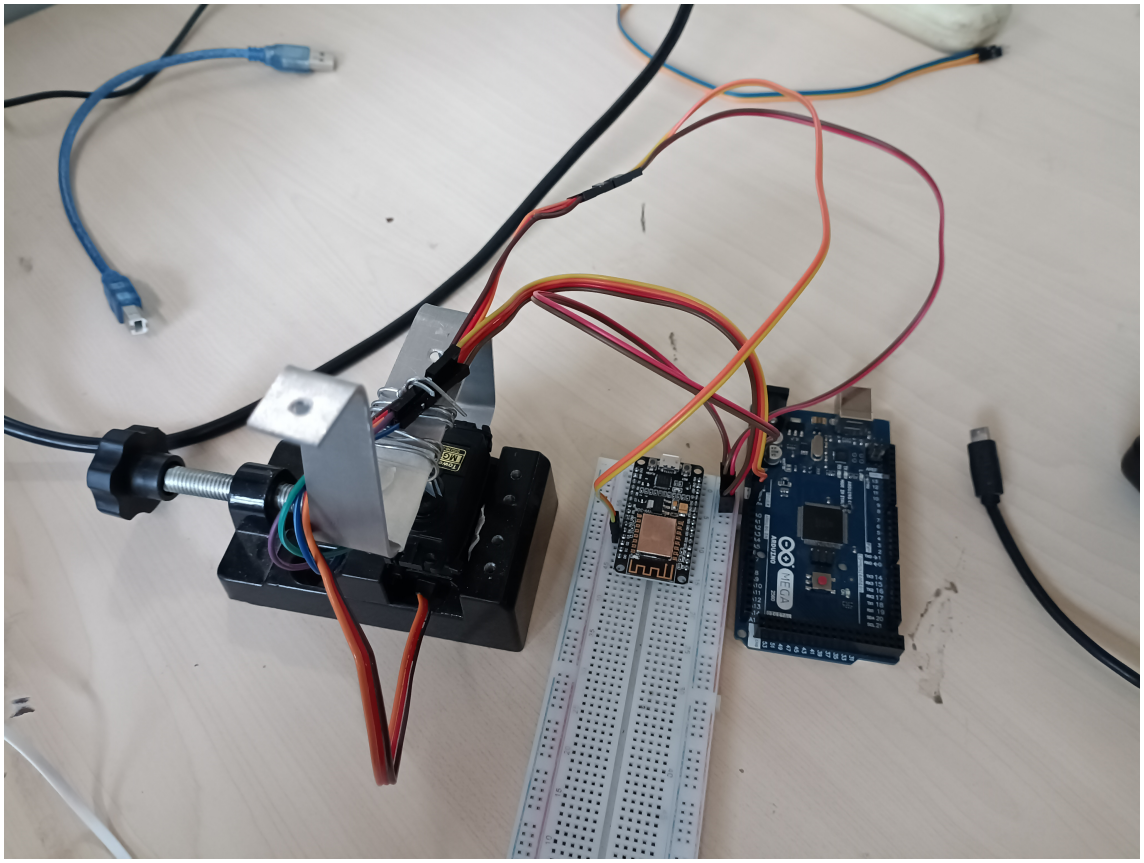


Figure 3.9 Whole setup for prototype, using a servo motor as an actuator, the ESP8266 wifi-module on the breadboard and the Arduino UNO emulating a DAQ

able to find the optimal solution. In the Figures 3.11(a), 3.11(b), 3.11(c), 3.11(d) and 3.11(e) we can see the development of the algorithm throughout the training phase. The blue ellipse denotes the search area that is being optimized by the CMA algorithm, the dots are the proposed solutions on that generation and the red star is the chosen optimal solution. The sin curved plotted is the comparison between the best solution of the generation with the optimal solution.

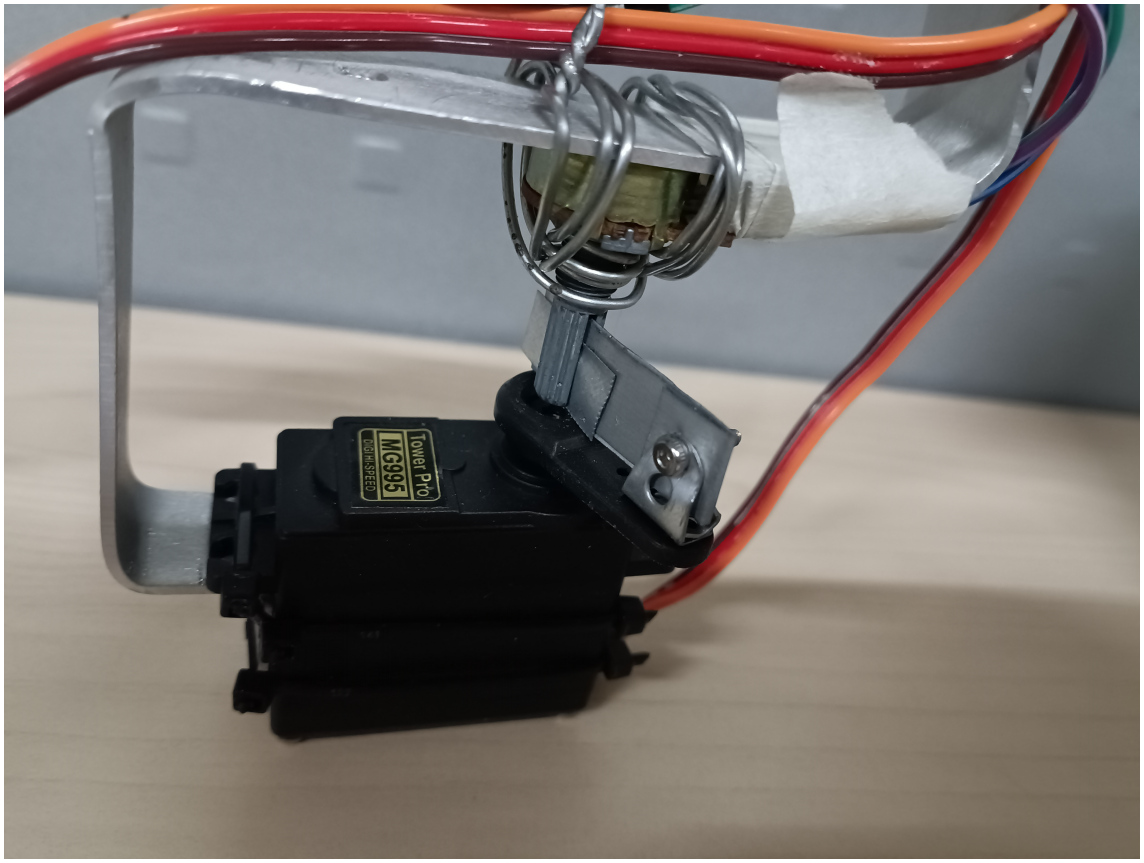
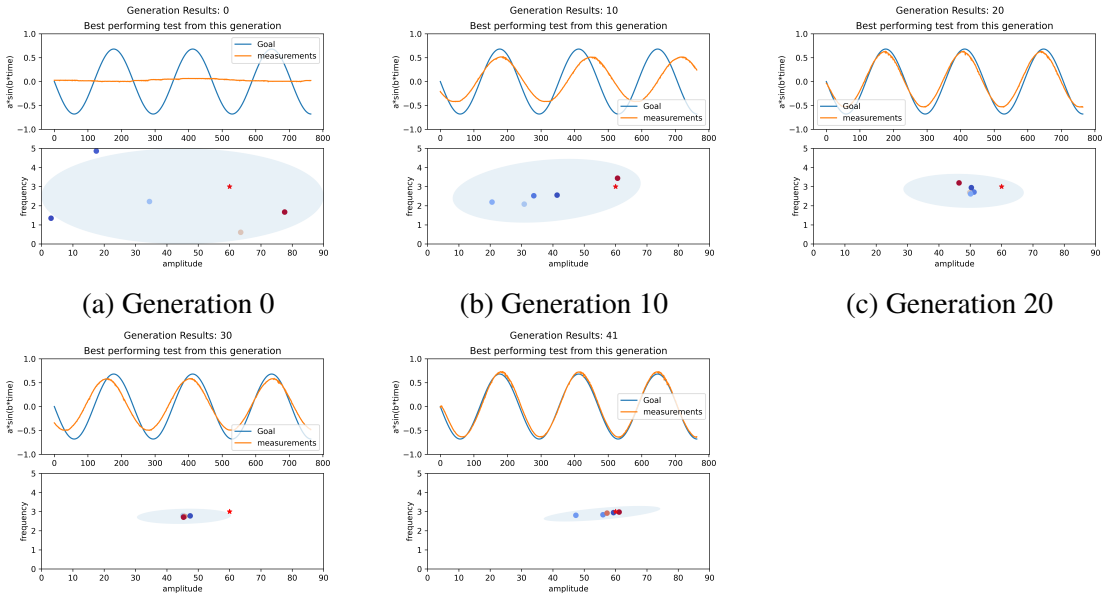


Figure 3.10 Prototype configuration



(a) Generation 0 (b) Generation 10 (c) Generation 20

(d) Generation 30 (e) Generation 41

Figure 3.11 Progression of the CMA algorithm to find the optimal value (shown as a red star) generation by generation, accompanied by the search area (blue ellipse) and the best test compared to the desired result of each generation (upper sinusoidal plot).

CHAPTER 4 EVALUATION ON BATBOT FLIGHT PERFORMANCE

This chapter delves into the comprehensive methodology employed throughout this master's thesis. It outlines the pivotal aspects of the experimental setup, emphasizing the distinct test benches utilized—namely, the static and dynamic ones. The construction of these test benches, their specific applications, and the rationale behind their usage are meticulously explored. Furthermore, this section is dedicated to an in-depth discussion of the optimization algorithm I conceived and subsequently implemented within these frameworks. This algorithm represents a cornerstone of the research, facilitating significant advancements in the project's objectives.

It is important to note that the specific details regarding the Batbot—including its components, mechanical design, and other related aspects—will not be included in this methodology chapter. Throughout the duration of the master thesis, the design and configuration of the Batbot underwent several revisions. These changes were informed by numerous experiments, each contributing to a deeper understanding and subsequent enhancement of the prototype's design.

The evolution of the Batbot is a testament to the iterative process of scientific inquiry and engineering design. As such, each version of the Batbot will be thoroughly examined in subsequent chapters. These sections will provide detailed insights into the modifications made, the rationale behind these changes, and their impact on the overall project outcomes.

4.1 Static and Dynamic Test-bench Design

The foundation of the experimental approach in this thesis rests on two main types of test benches: static and dynamic. Each bench serves a unique purpose and is designed to simulate different aspects of real-world conditions under controlled environments.

This chapter provides a detailed account of the development and evolution of the test benchs utilized in this study, focusing on the design, challenges, and subsequent improvements made to better assess the Batbot. The narrative is structured to transition from the initial conception (Version 1) to the refined iteration (Version 2) of the static test bench, highlighting the technological and methodological advancements achieved. We will then

continue with the explanation of the dynamic test bench

4.1.1 Version 1 of the Static Test Bench

The journey began with the creation of two specific test benches, with this section focusing on the static test bench. The primary purpose of this test bench was to evaluate the Batbot under various configurations and movements, measuring the forces it exerted in a controlled environment. The initial setup involved six load sensors arrayed as depicted in Figure 4.1, supporting a metallic structure to which the Batbot was affixed. A servomotor at the point of connection adjusted the angle of attack, crucial for varying test conditions.

Utilizing the data from these load sensors, combined with fundamental mechanical principles, allowed for the calculation of lift, drag, and thrust forces exerted by the Batbot. Additionally, a DAQ, named BSQ-JN-P8, facilitated the collection and communication of sensor data to a computing system via USB, promising detailed analysis and insights, with further elaborations on the DAQ presented later in this chapter.

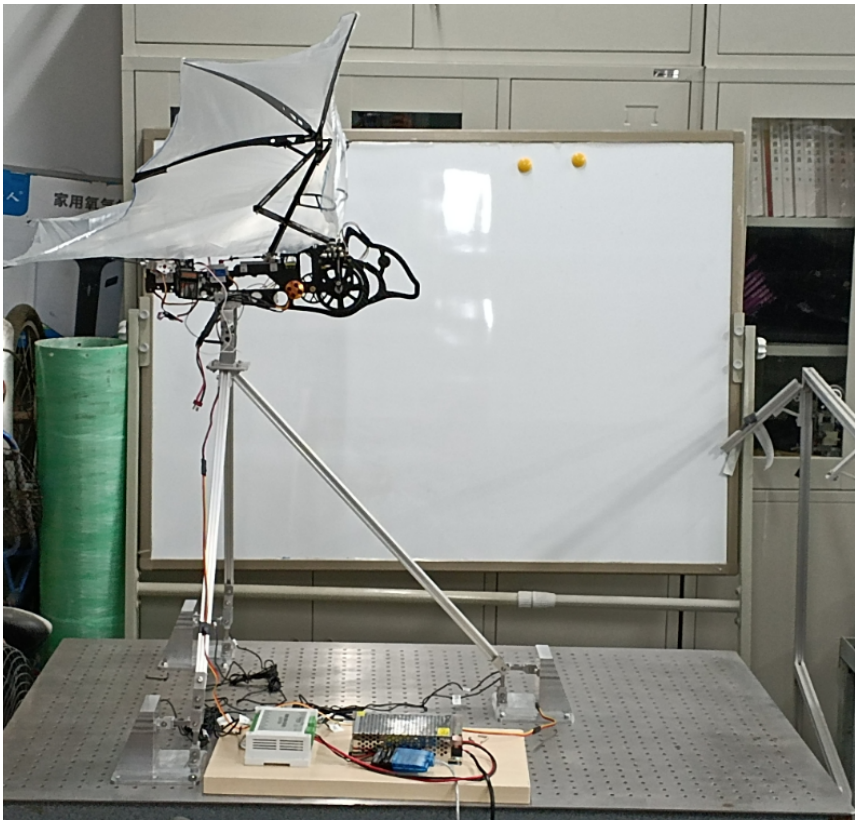


Figure 4.1 Batbot positioned on the first version of the static test bench on the forward flight position.

Challenges Encountered

Version 1, despite its innovative approach, was not without its pitfalls:

- **Calibration Issues**: The calibration process was hampered by internal tensions within the metallic structure, arising from overly tightened connections. This resulted in inaccurate sensor readings due to the inclusion of these tensions, complicating data reliability.
- **Inadequate Data Quality**: The looseness of connections led to structural instability, adversely affecting data quality.
- **Formulaic Limitations**: The mathematical approach for calculating thrust and lift, suitable for wind tunnel experiments with aeroplanes, proved ineffective for a flapping vehicle like the Batbot. The dynamic nature of the Batbot's propulsion and the generation of force and torque made it difficult to discern accurate measurements from the sensor data.

These challenges underscored the need for a significant upgrade to the static test bench.

4.1.2 Version 2 of the Static Test Bench

Responding to the challenges faced with Version 1, the static test bench underwent a comprehensive redesign, culminating in Version 2. This iteration saw the removal of all metallic structures and the replacement of the original load sensors with a 6 axis sensor of the brand "Decent," capable of 3-axial force and torque measurements.

Figure 4.2 6 axis sensor, 3 Force and 3 Torque sensor.

Figure 4.3 Static test-bench after introducing 6 axis sensor.

Advantages of Version 2

- **Direct Connection**: Eliminating the metallic structures and directly attaching the Batbot to the sensor eradicated the issues related to moving parts and stress within the system.
- **Enhanced Measurement Capabilities**: The 6-axis sensor provided the ability to accurately measure not only the forces but also the moments (torque) exerted by the Batbot, a significant advancement over the previous setup, Figure 3.8.
- **Data Integrity**: Unlike the first version, where torque and thrust calculations could interfere with one another due to the spatial arrangement of load sensors, the 6-axis sensor offered unambiguous and uncontaminated data collection.

The transformation from Version 1 to Version 2 of the static test bench represents a pivotal advancement in this research, addressing the initial limitations and significantly improving the precision and reliability of force and torque measurements for the Batbot. This evolution underscores the iterative nature of engineering design, where challenges fuel innovation and lead to the development of more refined and effective testing apparatus.

4.1.3 Dynamic Test Bench

In this chapter, we delve into the construction and application of the dynamic test bench. The design objective of this bench is to emulate a realistic environment for the

Batbot, specifically focusing on replicating scenarios where the Batbot attempts to hover. This environment aims to simulate the real-world conditions as accurately as possible, ensuring the safety of the Batbot by preventing any contact with the ground or risks of falling. The dynamic test bench is meticulously designed to meet the specific needs of the Batbot's hover simulation. At its core, the bench employs a six-axis force and torque sensor mounted to the ceiling. Suspended from this sensor is an elastic string with an elastic modulus of 2 and a corresponding length , l_0 , of 1 meters. This string is attached to the Batbot at one end, specifically from the nose, orienting the Batbot in a vertical position, Figure 4.4. Such a setup is crucial for simulating the hovering stance required for the Batbot's operation.

The dynamic test bench's capability extends beyond mere simulation; it allows for precise measurement and analysis of the forces acting on the Batbot. Initial force measurements provided by the sensor offer insights into the Batbot's current orientation.

The dynamic test bench is designed to not only simulate but also precisely measure and analyze the forces acting on the Batbot. Initial force measurements are obtained via sensors, which provide data crucial for understanding the Batbot's dynamics and orientation.

The resultant force exerted on the Batbot is calculated as the Euclidean norm of the three-dimensional force vector measured by the sensors. Let $\vec{F} = (F_x, F_y, F_z)$ be the vector representing the forces measured along the x, y, and z axes. The resultant force, F_{res}, is then given by:

$$F_{\text{res}} = \sqrt{F_x^2 + F_y^2 + F_z^2} \tag{4.1}$$

To analyze the direction of the force, we normalize the measured force vector. Normalization converts the force vector into a unit vector, which indicates the direction without regard to magnitude:

$$\hat{F} = \frac{\vec{F}}{\|\vec{F}\|} = \left(\frac{F_x}{F_{\text{res}}}, \frac{F_y}{F_{\text{res}}}, \frac{F_z}{F_{\text{res}}} \right) \tag{4.2}$$

Using Hooke's Law, which relates the force exerted by a spring to its displacement, we can further analyze the forces:

$$F = -kx \tag{4.3}$$

where k is the spring constant and x is the displacement of the spring from its rest posi-

Figure 4.4 Dynamic Testbench

tion. This law is used to determine the force exerted by the elastic string on the Batbot, accounting for the elasticity and length of the string.

By integrating this with the normalized direction vector $\hat{F}$ and the measured resultant force, we can accurately compute the position and forces acting on the Batbot during its

flight dynamics. The position vector $\vec{x}$ is calculated from the force vector using Hooke's Law:

$$\vec{x} = -\frac{\vec{F}}{k} = (l_0 - \frac{1}{k}F_{\text{res}})\hat{F} \tag{4.4}$$

This formula allows us to compute the displacement vector $\vec{x}$ which represents the position of the Batbot relative to its equilibrium position under the influence of the forces measured by the sensors.

The dynamic test bench facilitates continuous monitoring of the Batbot's position and orientation, providing a comprehensive understanding of its flight dynamics. The integration of sensor data with physical laws allows for a detailed analysis of the Batbot's behavior under various flight conditions, enhancing the development and optimization of its design.

A pivotal aspect of the dynamic test bench is its emphasis on safety, ensuring that the Batbot is protected throughout the experimental process. This approach significantly contributes to the feasibility of conducting repetitive experiments, thereby allowing for thorough investigation and optimization of the Batbot's hovering capabilities.

In conclusion, the dynamic test bench stands as a critical tool in the study of the Batbot, providing a realistic and safe environment for exploring the nuances of hover mechanics. Through detailed force measurement and analysis, it lays the foundation for advancing our understanding of bio-inspired flight dynamics.

4.2 Forward Free Flight Altitude Gain

The genesis of the Batbot project began with an initial prototype handed to me by Lijing Hu. This version showcased a rudimentary design, employing a single crane mechanism complemented by two elongated limbs. Each limb's movement was governed by a singular servomotor, restricting manipulation to the vertical plane alone. A noteworthy feature was the membrane bridging the limbs, albeit lacking any capacity for folding or unfolding, which would have added complexity to the model. The simplicity of the control system, operated via a basic radio control setup that directly influenced the flapping frequency and limb movements, had the benefit of significantly reducing the electronic component weight.

Through a series of experimental trials, we achieved stable forward flight, a promising outcome. However, the design's limitations became apparent, particularly its inability

to execute rotational leg movements, thus complicating control and hover capabilities. Despite these challenges, the experiments validated the potential of this mechanical configuration. Specifically, when the membrane wings were oscillated at approximately 4 Hz, they generated sufficient lift not only to maintain the Batbot's airborne state but also to gain around 1mt of altitude, Figure 4.5 demonstrating the feasibility of forward flight under these conditions.

Figure 4.5 Result forward free flight, where the altitude gain can be appreciated.

4.3 Folding and Unfolding influence on Lift

4.3.1 Experimental set-up for force measurement

In this study, we carried out force measurement experiments to evaluate the efficacy of both flapping and flapping-folding motion patterns. Figure 4.8, illustrates the experimental setup, wherein the Batbot is affixed to a movable support atop the bracket, allowing adjustments to the angle of attack. Positioned at the bracket's base, six load cells (model # ZNLBS-VII) are utilized. Data acquisition is facilitated through the BSQ-JN-P8 I/O box, which records the output from the load sensors and transmits this data to a desktop computer equipped with a BSQ-JN-P8 R&D controller board. These signals are sampled at a rate of 1000 Hz. A constant voltage of 11.0 V is supplied to the speed controller, which operates the brushless DC motor.

For the analysis of the data, the input range to the motor is confined between 58% and 86%. It is noted that the flapping motion can function at below 50% input; however, the flapping-folding motion is incapable of initiating at inputs under 50%. To prevent overloading, the maximum motor input is restricted to 86%. The methodologies for calculating lift and thrust are specified below:

$$F_L = F_{v2} + F_{v4} + F_{v6} \tag{4.5}$$

$$F_T = F_{h1} + F_{h3} + F_{h5} \tag{4.6}$$

In the equations, F_{v_i} and F_{h_i}, where i = 2,4, 6, denote the measurements from vertical and horizontal pressure sensors, respectively. Here, the subscript i indicates the specific label assigned to each sensor.

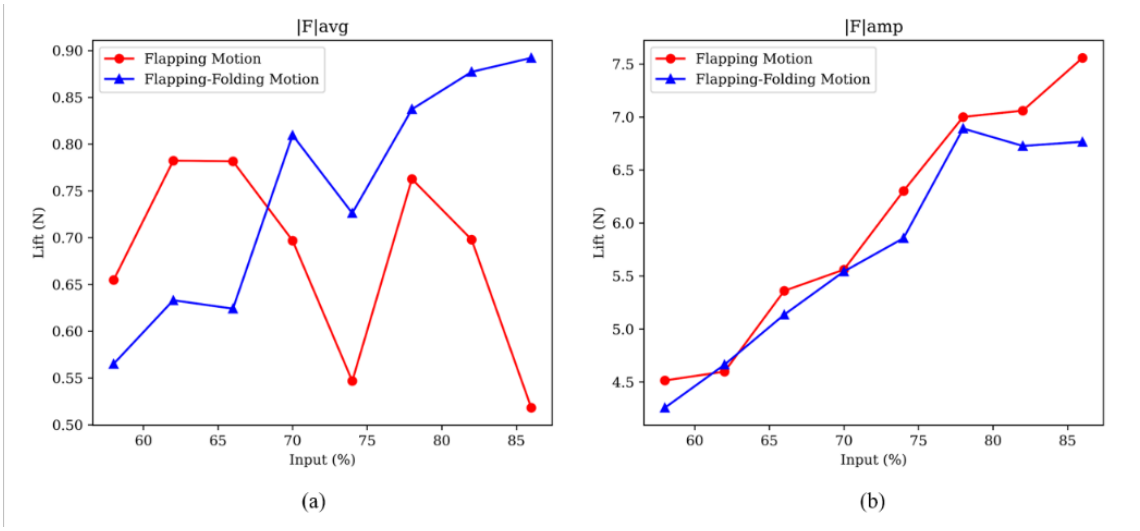


Figure 4.6 Comparative analysis focused on the average lift and amplitude of the wing motions under varying voltage inputs, specifically between the folding and extended wing configurations.

4.3.2 Experimental Result of force measurement

To ascertain the benefits of morphing membrane wings on enhancing lift capabilities, our investigation focused on two distinct wing motion patterns: the flapping-folding motion utilizing a coupling mechanism and the conventional flapping motion with fully extended wings. These measurements were conducted in a controlled environment devoid of free-flow conditions to accurately gauge the aerodynamic forces exerted on the flapping wings. The motor's input was incrementally adjusted in 4% intervals, ranging from 58% to 86%, thereby generating a total of seven data sets for each wing motion pattern.

For each experimental setup, the robot was aligned at five different angles of attack to comprehensively assess its aerodynamic performance. Each testing session was maintained for a duration of three seconds. To guarantee the reliability of the results, all sensors were recalibrated prior to each test. This recalibration involved resetting the tension or pressure readings to zero, ensuring that each measurement commenced from a consis-

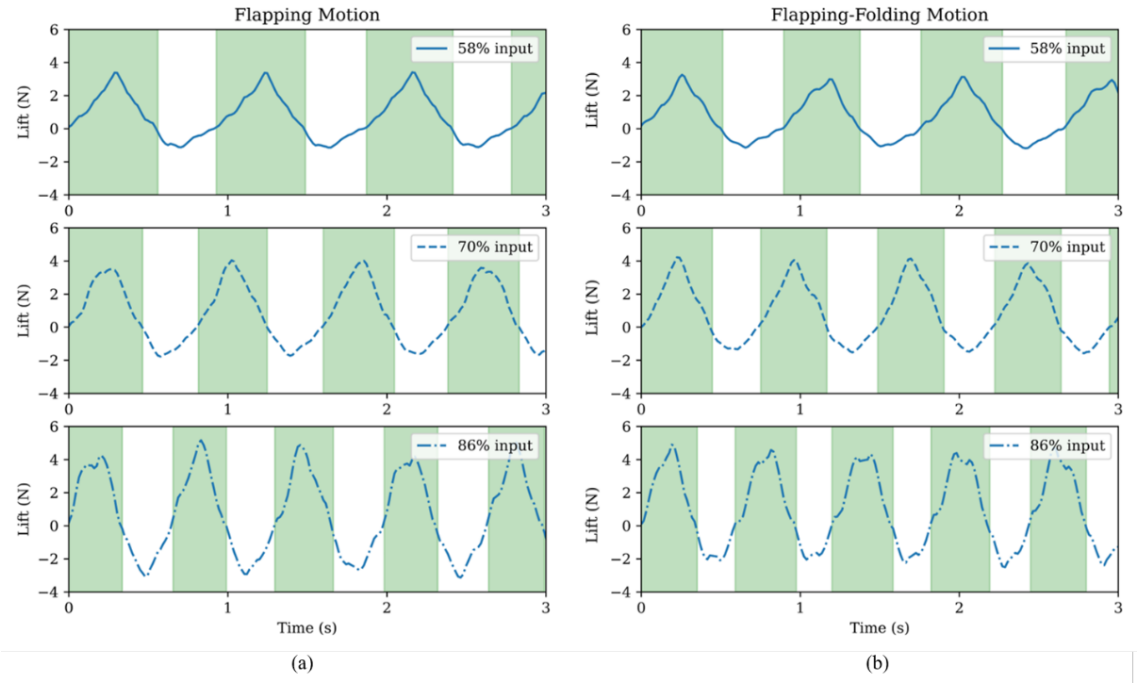


Figure 4.7 Temporal analysis of the measurement results over a duration of three seconds, segmented into intervals corresponding to the wing stroke phases. Each wing stroke comprises two distinct intervals: the downstroke, represented by the green area, and the upstroke, indicated by the white area. (a) Flapping Motion, (b) Flapping-Folding Motion, (top, middle, bottom): 58%, 70%, and 86% input.

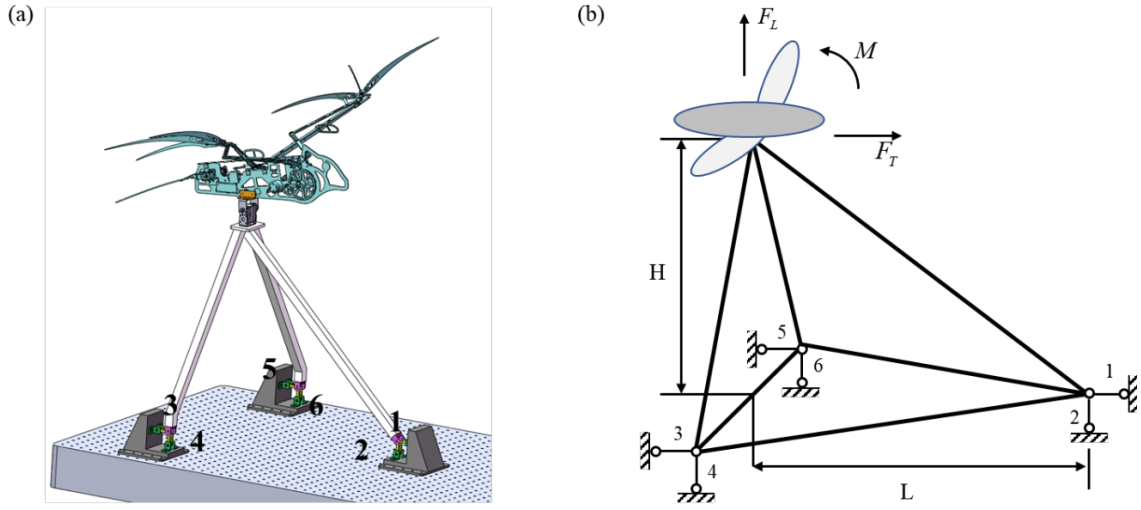


Figure 4.8 Experimental setup (a) 3D model and schematic of the experimental set-up. (b) Aerodynamic forces test platform.

tent baseline. This methodical approach enabled a precise evaluation of the aerodynamic enhancements provided by the morphing membrane wings under varied operational conditions.

Representative findings illustrating the dynamics of flapping motion and flapping-folding motion, at input levels of 58%, 70%, and 86%, are detailed in Figure 4.7. For the

flapping motion pattern, the data encompasses the lift across multiple wingbeat periods for three tests over a three-second duration, as shown in Figure 4.7. The commencement of each test is marked by the peak lift reading, which aids in the accurate alignment of data across the testing intervals.

In contrast, the results for the flapping-folding motion pattern are depicted in Figure 4.6. Both graphical representations clearly delineate the wingbeat periods, although they also reveal the presence of low-frequency oscillations within these cycles. It was noted that under identical input conditions, the ascent velocity in the flapping-folding motion is markedly quicker compared to the flapping motion. Moreover, the wing frequency of the flapping-folding motion pattern is observed to be higher. Specifically, at an input of 86%, the wing frequency for the flapping motion pattern is approximately 0.3 Hz, whereas for the flapping-folding motion pattern, it reaches approximately 0.6 Hz. These distinctions highlight the aerodynamic efficiencies and kinetic characteristics imparted by the different wing motion patterns under varying operational inputs.

The analysis of the results depicted in Figure 4.7 reveals several significant insights that substantiate our hypothesis concerning the aerodynamic benefits of wing folding in bats during the upstroke phase of the wingbeat. This folding mechanism is shown to effectively mitigate the generation of negative lift, a phenomenon that is supported by the data presented in these plots. This observation is aligned with and enhances the findings of previous studies that have explored wing folding in the Batbot model.

The practical impact of wing folding during the upstroke is notably significant in terms of reducing both wing inertia and negative lift, which are crucial factors in the wingbeat cycle of bats. The reduction in negative lift not only minimizes energy loss but also contributes to a net increase in overall lift, enhancing flight efficiency. Specifically, the lift measurements indicate that with extended flapping, the negative lift reaches as low as approximately -3 Newtons. In contrast, the folding motion results in a less severe negative lift, with minima around -2 Newtons. This comparative analysis clearly demonstrates that the Batbot experiences a substantial decrease in negative lift when its wings are folded during the upstroke, thus providing compelling evidence of the advantages of this adaptive wing morphology.

$$|F|_{amp} = \frac{1}{2}(maxm(t) + |minm(t)|)g, 0 \leq t < t_s \tag{4.7}$$

$$|F|_{ave} = \frac{1}{t_s}\int_0 m(t)g\,dt \tag{4.8}$$

Figure 4.6 illustrates the force amplitude $|F|_{amp}$ and average force $|F|_{ave}$ for each motion pattern across various motor inputs. The values of force amplitude and average force are derived . In this equation, represents a specific time instant during a wing stroke, while t_s represents the period of a complete wing stroke.

Figure 4.6 provides a detailed graphical representation of the force amplitude, denoted as $|F|_{amp}$, and the average force, also represented as $|F|_{ave}$, for each wing motion pattern at varying motor input levels. In this equation, t specifies a particular moment within a wing stroke, whereas t_s denotes the duration of an entire wing stroke cycle.

This presentation allows for a nuanced understanding of the dynamic forces involved in different wing motions under varied operational conditions. We analyze the forces over the cycle of a wing stroke, capturing the instantaneous peak forces ($|F|amp$) and the mean force over the cycle ($|F|ave$). The distinction between these metrics provides insight into the mechanical performance and efficiency of the Batbot's wing motions, facilitating comparisons that underscore the aerodynamic properties conferred by different input settings and wing configurations.

4.3.3 The Grid Experiment

A series of experiments aimed at understanding the aerodynamic implications of its design improvements, particularly the folding and unfolding mechanism of the wings were implemented. These experiments were crucial for evaluating the practical lift capabilities under various operational conditions. A pivotal investigation, dubbed the "grid experiment," was designed to systematically explore the impact of wing articulation on lift. This experiment involved altering the Batbot's attack angles and flapping frequencies across a matrix of conditions, thereby encompassing a comprehensive range of flight dynamics.

4.3.3.1 Experimental Setup

The grid experiment was structured to test the Batbot under ten distinct attack angles between 80°-130°, in Figure 4.9 the definition of the coordinate system can be seen, and ten varying motor power input, culminating in a total of 100 individual test conditions. Each condition was evaluated twice: once with the wings locked in an extended position and once utilizing the full capability of the folding and extending mechanism. This approach aimed to directly assess the mechanism's contribution to lift generation.

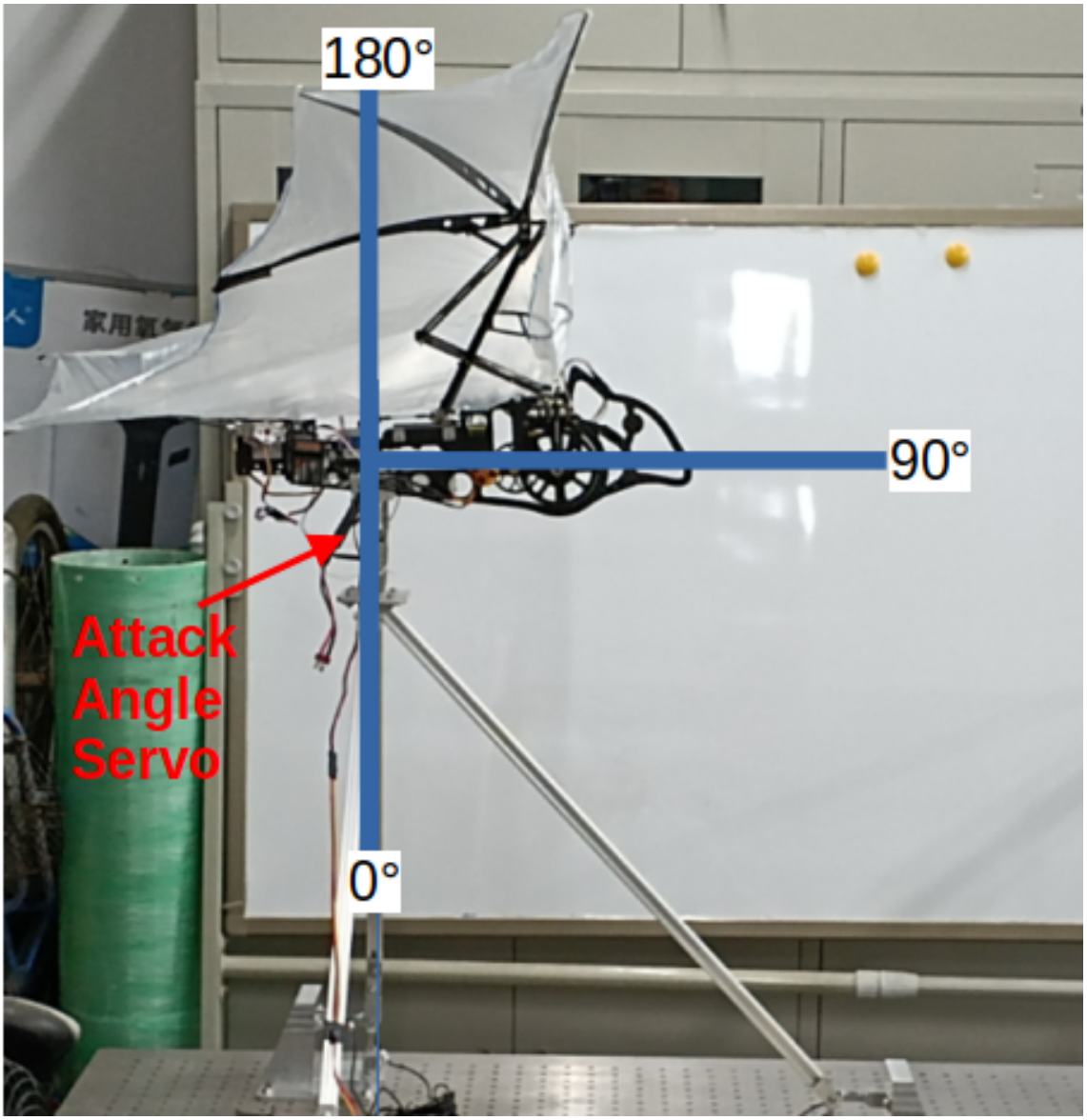


Figure 4.9 Attack angle coordenate system definition

4.3.3.2 Results and Discussion

The outcomes of the grid experiment, depicted in Figure 4.10, unequivocally demonstrated the benefits of the folding and unfolding mechanism in enhancing lift. In nearly all tested scenarios, configurations employing the dynamic wing articulation outperformed those with static wing extension in terms of lift (shown as lower scores in Figure 4.10). An exception to this trend was observed at lower attack angles with lower motor power, where the inherent aerodynamic advantages of angles below 90° angle settings somewhat diminished the relative impact of wing folding.

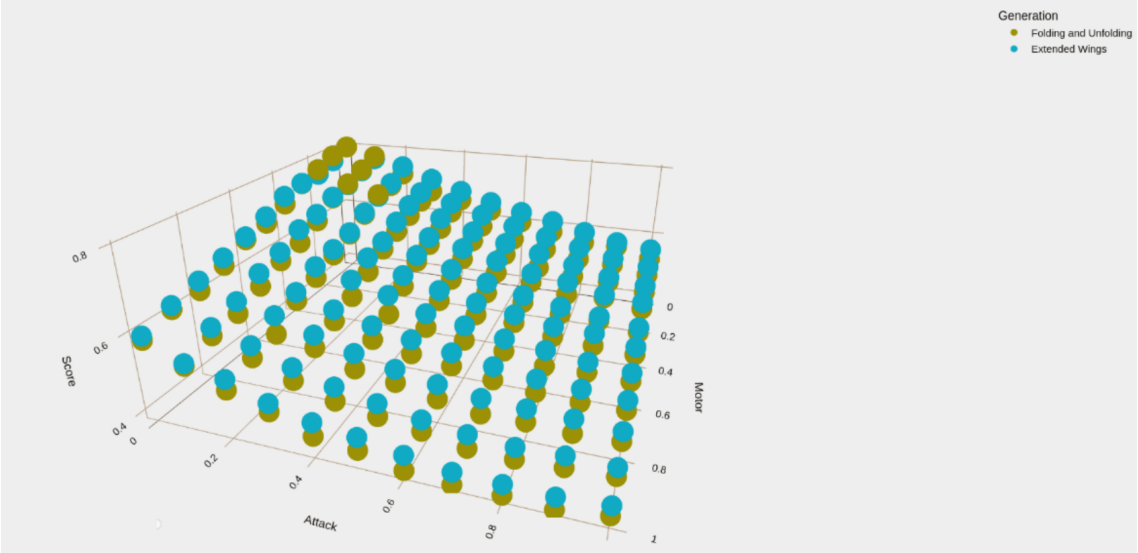


Figure 4.10 Folding and Unfolding comparison in grid experiment, showing how folding increases the lift

4.4 Optimal Attack Angle and Motor Power through CMA-ES

In this section, we investigate the optimal angle of attack and motor power input required to maximize lift in a robotic bat model. Initially, we employed a grid search methodology across a predefined range of variables. Specifically, the experiment involved testing combinations of 10 distinct angles from 80° to 130° and motor power inputs ranging from 260 to 270, resulting in a total of 100 individual tests, Figure 4.11. This approach allowed for a comprehensive mapping of the score function under varying conditions.

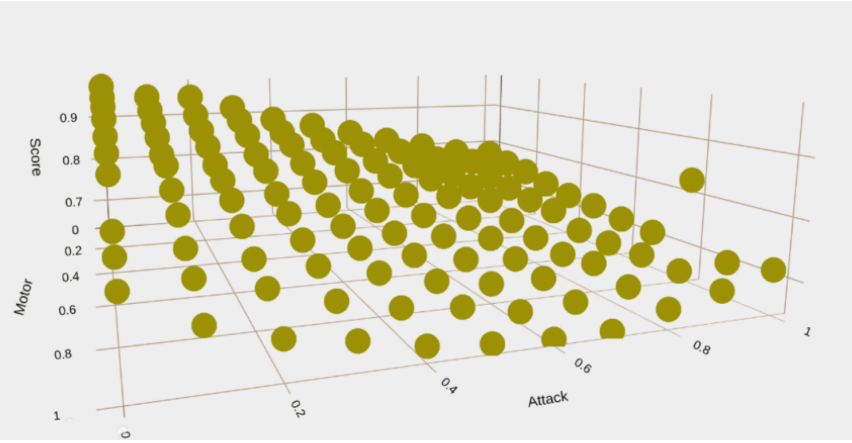


Figure 4.11 Optimal attack angle and motor power using grid approach

Following the grid search, we implemented an evolutionary algorithm specifically designed for this study. To assess the robustness and reliability of the evolutionary approach, we conducted three independent experiments with the algorithm configured to run for 20 (Figure 4.12), 25 (Figure 4.13), and 30 generations (Figure 4.14), respectively;

each generation comprised 10 individual tests.

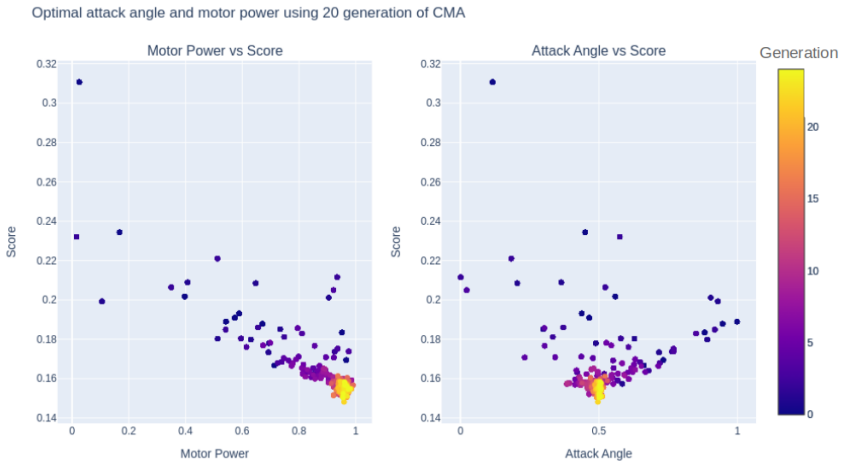


Figure 4.12 Optimal attack angle and motor power using 20 generation of CMA

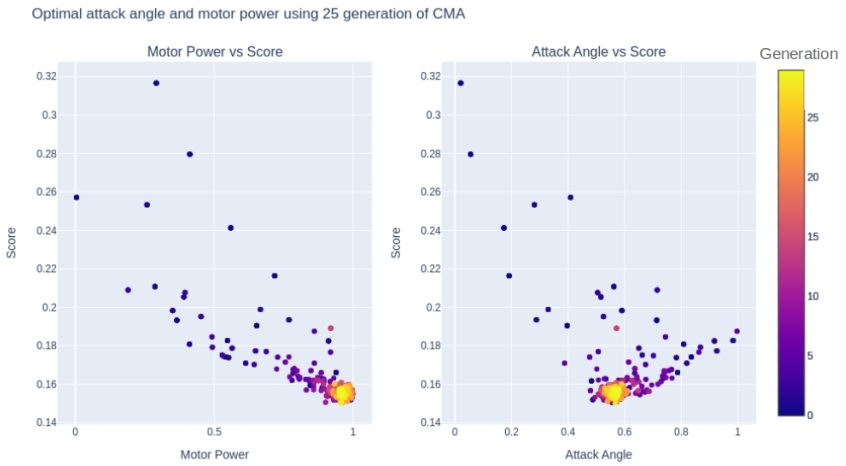


Figure 4.13 Optimal attack angle and motor power using 25 generation of CMA

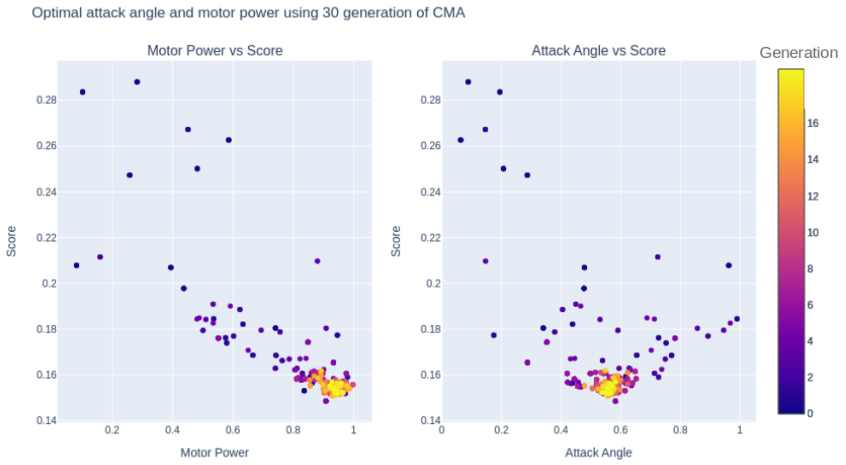


Figure 4.14 Optimal attack angle and motor power using 30 generation of CMA

Analysis of the outcomes from both the grid and evolutionary strategies indicated a convergence on the optimal configuration at an angle of attack of 107° and maximum motor power. This result is significant for two main reasons. Firstly, it underscores the reliability of the evolutionary algorithm, as evidenced by the consistent convergence across three separate trials. Secondly, it validates the correctness of this solution, aligning with results from the conventional grid testing method.

Importantly, the findings highlight a critical advantage of the evolutionary algorithm: scalability. As the dimensionality of the problem space increases, traditional grid methods become computationally infeasible due to the exponential growth in required evaluations. In contrast, the evolutionary approach maintains a manageable computational load, thereby demonstrating its efficacy and potential for solving more complex aerodynamic optimization problems in robotic systems. This advantage substantiates the primary hypothesis of this thesis, emphasizing the superiority of evolutionary algorithms in high-dimensional optimization contexts.

4.5 Optimal hind-leg movement

Building upon the foundational work described in preceding chapters, this section delves into the application of the optimization algorithm workflow . The primary objective of these experiments was to ascertain the most effective hind leg movements to minimize the operational score, indicative of efficiency and aerodynamic performance. For this imporvements on the Batbot design were needed.

To facilitate more nuanced control capabilities, notably for hovering, we initiated several modifications of the Batbot. These adjustments aimed at overcoming the mechanical and electronic limitations previously encountered. A significant mechanical overhaul was the introduction of dual servo motors for each leg, a departure from the single servo configuration. This dual-servo arrangement afforded each leg two degrees of freedom, allowing for movement along both horizontal and vertical axes, whereas the last version was constrained to vertical movement only. To accommodate the independent horizontal movement of the legs, the connecting membrane was split, enabling isolated leg movements.

Moreover, we integrated a magnetic encoder sensor onto one axis of the wing's shoulder. This sensor's primary function was to accurately measure the wing's current angle, a critical parameter for synchronizing leg movement with wing angle for optimal aerody-

namic efficiency.

Another noteworthy enhancement was the installation of a servo motor to control the Batbot's wing extension and folding mechanisms. Utilizing the magnetic encoder, Figure 4.15 we could calculate the wing movement direction, facilitating strategic wing extension during downstrokes and folding during upstrokes to maximize lift while minimizing drag. Electronically, the transition to this new version involved rigorous experimentation with different control systems. Initially, we employed the STM32 microcontroller, leveraging its capability for rapid information processing through C++, a low-level programming language. Despite its processing advantages, the STM32's complexity and limited compatibility presented significant challenges.

Experimentation with the Pixhawk module, commonly used for drones, revealed its rigidity in customization and programming, despite its comprehensive suite of drone-specific modules and algorithms. Given our project's unique requirements, the Pixhawk module's constraints necessitated a different approach.

Consequently, we adopted the Pi board, operating on MicroPython—a variant of Python optimized for microcontrollers. MicroPython combines the accessibility and readability of Python with the performance necessary for real-time control tasks. This selection significantly simplified programming from scratch while maintaining the responsiveness required for controlling the Batbot.

The Pi board interfaced with the AC05 magnetic encoder for wing angle measurements and a PCA module to manage the servo motors controlling leg movements and wing folding dynamics. Furthermore, a DC motor adjusted the flapping frequency, and wireless communication was facilitated by a UART module, collectively forming a versatile and efficient electronic system for Batbot control.

The experiments utilized a generational approach, where each generation consisted of a population of ten solution candidates. Over the course of approximately 30 generations, each experimental run lasted around five seconds, allowing for the iterative refinement of leg movement strategies.

4.5.1 Feedforward Controller

A feedforward controller is a type of control system that responds to changes in its environment or input before these changes start to create an error in the output. This approach is in contrast to a feedback controller, which reacts after an error has occurred.[41]

The feedforward controller is designed based on a model of the system and aims to

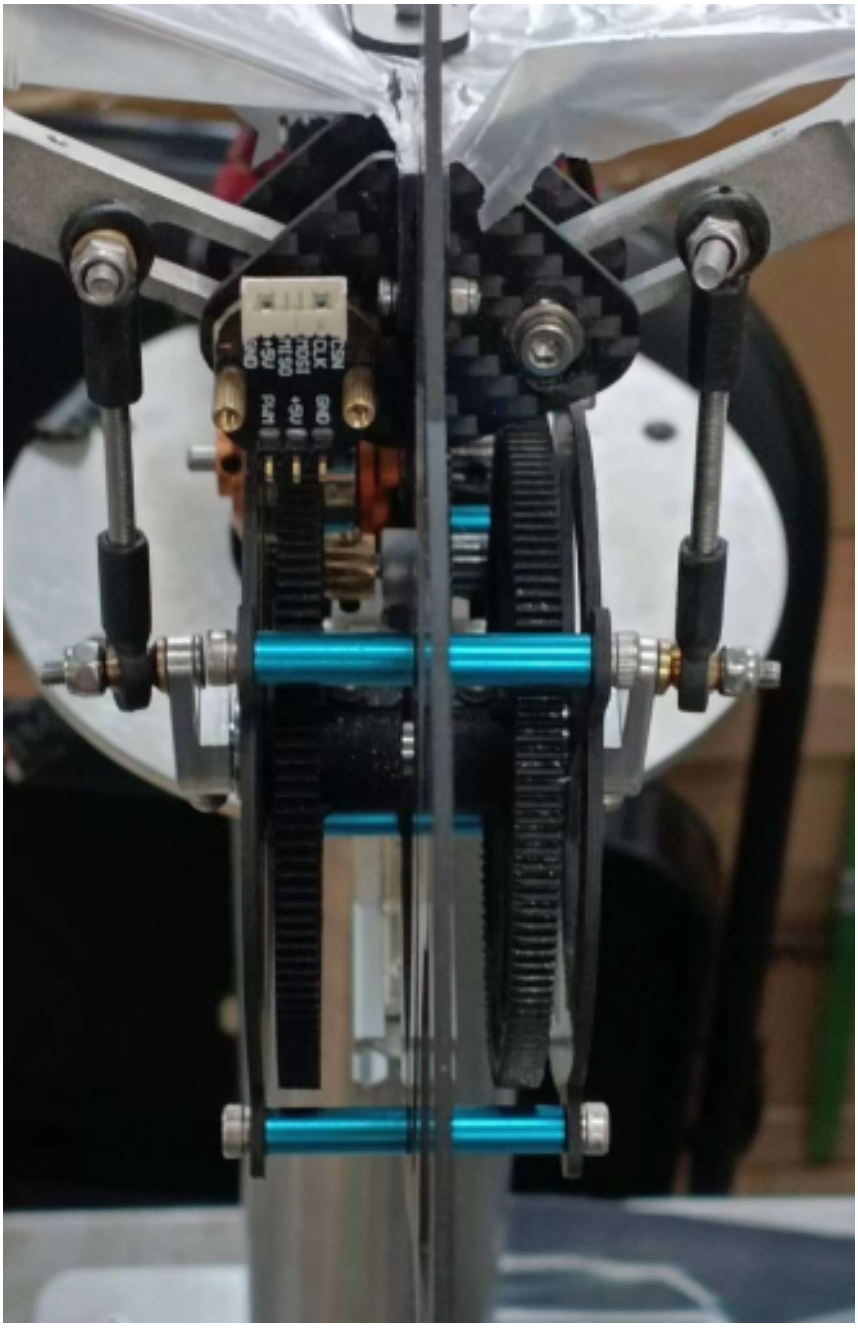


Figure 4.15 Frontal view of the Batbot were the implementation of the 2 gear system can be seen, as well as the magnetic encoder on the right shoulder for wing angle measurements.

predict changes that will affect the system, enabling it to take corrective action in advance.

Here's a simplified way of how it works[41]:

1. Measurement: The controller measures or receives information about a disturbance that is going to affect the system. This information is usually about changes in the environment or in the input of the system.

2. Prediction: Based on the measured information and a model of the system, the controller predicts how the system will be affected by the disturbance.

3. Compensation: The controller calculates the necessary adjustments to the system to counteract the predicted effect of the disturbance, and then applies these adjustments. The aim is to maintain the system output at the desired level, despite the disturbance.

The main advantage of feedforward control is that, when the model is accurate and the disturbances can be measured in advance, it can perfectly compensate for the disturbance and maintain the output at the desired level without any error. This is in contrast to feedback control, which always involves a certain amount of error, as it only reacts after the error has occurred.

However, feedforward control relies heavily on the accuracy of the system model and the ability to measure disturbances in advance. If the model is inaccurate or the disturbances cannot be measured, the feedforward controller may not perform well. In many real-world applications, feedforward control is often used in combination with feedback control to balance out the strengths and weaknesses of both approaches.

A crucial, non trivial, task of this master thesis is the design of a feedfoward control (FFC) that will be optimized in the static test. Feedforward control in this context could be understood as a direct command sent to the motors without regard of the Batbots state. The main question to be answered is: What will the feedforward controller exactly be commanding and which parameter are we actually optimizing?

In control theory literature, feedforward controllers are usually represented in the Lagrange, domain. This has many advantages, as it simplifies operations and enables a straight forward stability analysis. Nevertheless, as in this thesis controllers will be optimized using EA techniques, it would facilitate notation if we define our controllers as vector functions. For this reason we decided not to use the Lagrangian notation but instead define our feedforward controller as the vector function FFC. This function has as an input the instance of the wing regarding its flapping cycle, this is defined as cyc_pi. The output of this function is the two dimensional vector , where the first dimension describes the angle of the leg in the horizontal plane, x_angle, and the second the angle with respect to the vertical plane y_angle.

With this, FFC can be formally described as:

$$cyc\_pi \in [0, 2\pi] \tag{4.9}$$

$$x\_angle \in [-60, 60] \tag{4.10}$$

$$y\_angle \in [-60, 60] \tag{4.11}$$

$$FFC : cyc\_pi \longrightarrow \begin{pmatrix} y\_angle \\ x\_angle \end{pmatrix} \tag{4.12}$$

The detailed description of the FFC function will be explained in the following chapter.

4.5.2 Hindleg Movement Parametrization

To find an optimal hind-leg movement strategy, a parametrization of the leg trajectroy must be created. The choice of the to be proposed parametrization comes from the work of Dr. Singh,[42], who provided valuable insights into the bat's flight mechanics. It has been observed that the bat's hind leg movements are crucial for flight control and bear a resemblance to an elliptical path.

Hence, in this thesis, the five dimensions of the solution vector are understood as follows:

- The width of the ellipse, x_amp.
- The height of the ellipse, y_amp.
- The horizontal axis's midpoint of the ellipse, $x\_neutral$.
- The vertical axis's midpoint of the ellipse, $y\_neutral$.
- The orientation (inclination) of the ellipse, allowing for the representation of various elliptical shapes and their respective rotation direction, $ellipse\_angle$.

This leads to the definition of the solution vector θ that can be described as:

$$\theta = \begin{pmatrix} x\_amp \\ y\_amp \\ x\_neutral \\ y\_neutral \\ ellipse\_angle \end{pmatrix} \tag{4.13}$$

This approach enables a comprehensive description of any possible elliptical movement of the Batbot's hind leg by varying these five parameters. Importantly, the orientation parameter also facilitates control over the rotation's direction.

4.5.3 Movement Algorithm

In the quest to enhance the aerodynamic efficiency and maneuverability of bio-inspired robotic systems, such as Batbot, the synchronization between wing and hind leg

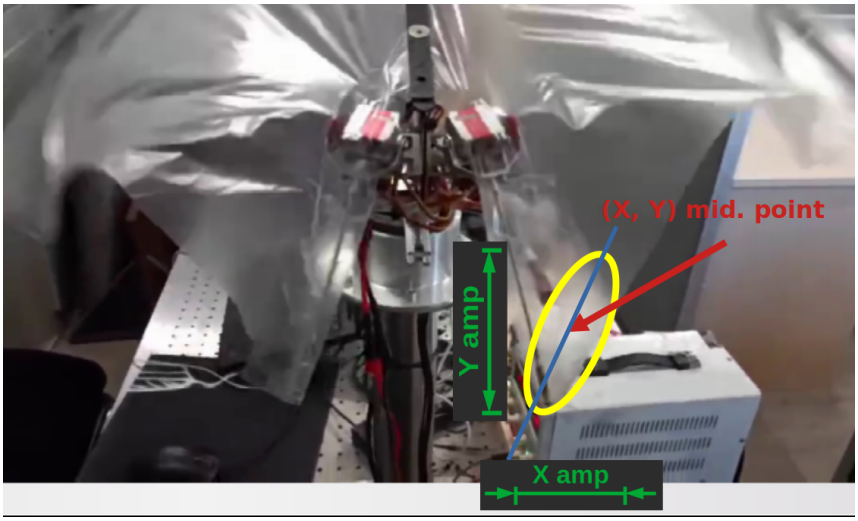


Figure 4.16 Parametrization of elliptical leg movement.

movements stands paramount. This chapter delves into the methodologies employed to achieve a harmonious synchronization between these elements, focusing on the mechanics of wing folding and extension in relation to the wing cycle, alongside the intricacies of hind leg movement coordination.

The command for wing folding and extension is predicated on the phase of the wing cycle, aimed at reducing air resistance and maximizing lift. The logic is straightforward:

- **Upstroke Movement**: During the ascent of the wing cycle ($0 < cyc\_pi < \pi$), wings are folded to minimize air resistance.
- **Downward Movement**: Conversely, in the descent phase ($\pi < cyc\_pi < 2\pi$), wings are extended to enhance lift generation.

The vertical movement of Batbot's hind legs is synchronized with the wing cycle to mimic the natural locomotion of bats. The angle of vertical leg movement (y_angle) is articulated as follows:

$$y\_angle = y\_neutral - y\_amp \cdot \cos(cyc\_pi) \tag{4.14}$$

This formula establishes the neutral state ($y\_neutral$) of the leg's vertical position and employs the cosine of the wing cycle parameter (cyc_pi) to modulate the leg's vertical displacement. The modulation, scaled by the amplitude (y_amp), oscillates between -1 and 1, thus mirroring the wing's vertical motion.

The lateral movement of the hind legs incorporates a phase shift to generate an elliptical trajectory, enhancing the robotic system's agility and stability. The lateral angle

(x_angle) is defined as:

$$x\_angle = x\_neutral + x\_amp \cdot \sin(cyc\_pi + 2\pi \cdot ellipse\_angle) \tag{4.15}$$

The inclusion of $2\pi \cdot ellipse\_angle$ introduces a phase shift to the sine function, thereby allowing for the adjustment of the ellipse's shape and orientation based on the $ellipse\_angle$ parameter. This phase shift enables a dynamic alteration of the elliptical path, facilitating a broader range of motion and adaptability to various aerodynamic conditions.

Using the solution vector we can then describe the FFC function as follows:

$$FFC(cyc\_pi) = \begin{pmatrix} y\_neutral - y\_amp \cdot \cos(cyc\_pi) \\ x\_neutral + x\_amp \cdot \sin(cyc\_pi + 2\pi \cdot ellipse\_angle) \end{pmatrix} = \begin{pmatrix} y\_angle \\ x\_angle \end{pmatrix} \tag{4.16}$$

4.5.4 Score Function

In order to effectively evaluate and quantify the performance of the experiments conducted using the evolutionary algorithm, a scoring function must be established. This section discusses and explains two different approaches for defining such a function.

4.5.4.1 Initial Scoring Approach

In the initial version utilizing load sensors, our objective was to minimize the sum of all measured forces, with an emphasis on absolute values, aiming for a total as close to zero as possible. This simplistic approach encountered challenges due to the sinusoidal nature of the data generated by the Batbot's flapping motion. Scores varied significantly depending on the specific segment of the sinusoidal wave captured for analysis.

To mitigate this variability, we employed a moving window technique, calculating scores over various segments and observing a sinusoidal pattern in the scores themselves. To standardize score reporting, a linear fit was applied to this sinusoidal scoring function, with the score nearest to zero selected as the most representative, Figure 4.17. This method, while effective for longer experiments, proved necessary due to the emphasis on conducting short, efficient trials of approximately five seconds.

4.5.4.2 Advanced Scoring with Improved Sensor Data

With the adoption of a more accurate sensor providing precise force measurements in the y and z directions, we were able to discern the sinusoidal pattern and its peaks with much greater clarity. This led to a novel approach where a peak detection algorithm

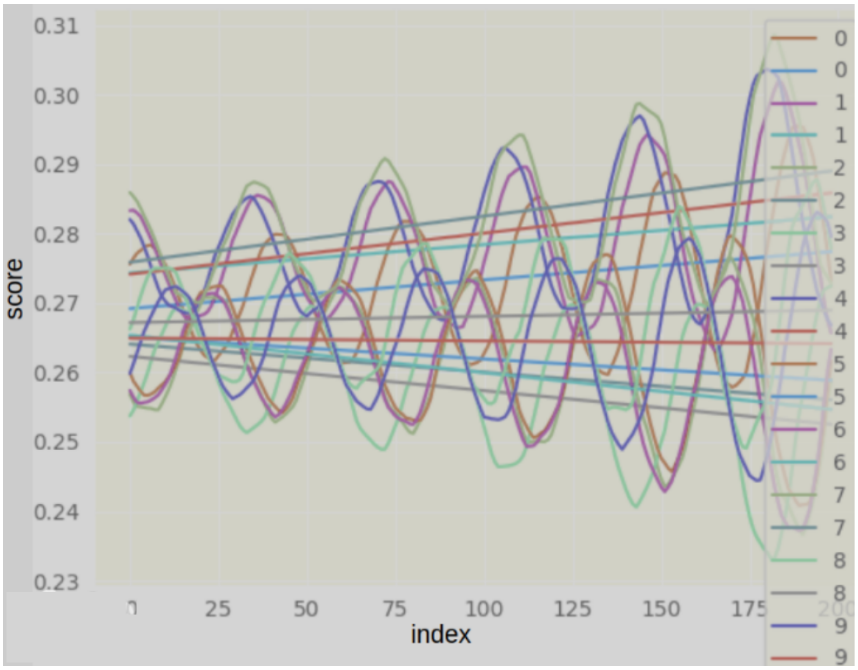


Figure 4.17 Examples of scores obtained by the projection of the linear interpolation of the moving window technique.

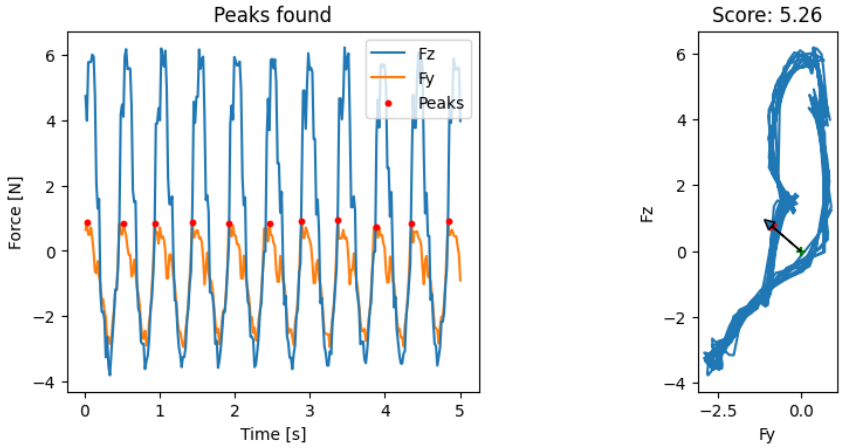


Figure 4.18 Example of the result of a test, on the right the resulting force and on the left the lift and thrust and the result of the peak analysis.

identified the exact spans of each flapping period, ensuring that each measurement cycle began and ended at a period's extremities, thus yielding more consistent data, Figure 3.8.

The refined data, particularly the y and z force measurements, revealed a clear oscillatory pattern attributable to the Batbot's flapping motion. By summing these oscillations, we derived a measure akin to the Batbot's resulting force for each trial. This force, after calibration to exclude the Batbot's weight, accurately represented the net force exerted under each proposed solution by the CMA algorithm.

The score for the second and more successful version of our methodology was thus

calculated based on the Euclidean distance of this resulting force vector from zero. This approach straightforwardly quantified the algorithm's success in minimizing the force required to maintain or alter the Batbot's flight state, aligning closely with our goal of reducing the resulting force to as near zero as possible. The exact description of the score calculation varies from experiment to experiment, for this reason, more detailed descriptions are found on the respective experiments.

4.5.5 Results

Following the outlined optimization algorithm, we meticulously recorded the performance of each candidate within the population across successive generationsin hovering position, Figure 4.19. This iterative process was designed to systematically reduce the score, thereby honing in on the most efficient leg movement patterns.

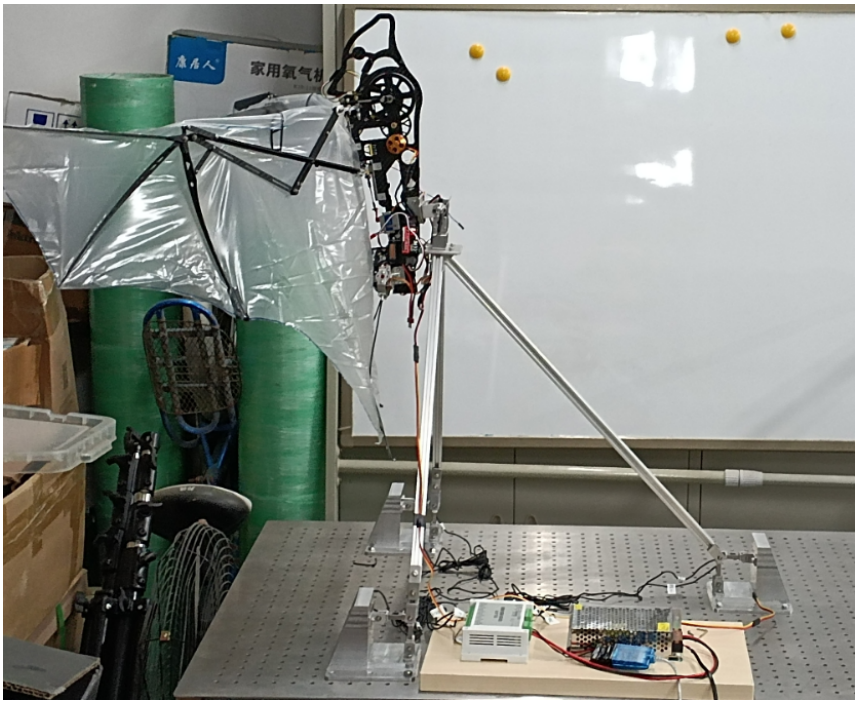


Figure 4.19 Hovering position for optimization algorithm strategy in static test-bench

A notable reduction in the score from one generation to the next was observed, as depicted in Figure 4.20. This trend unequivocally demonstrates the algorithm's capacity to learn and refine control strategies for the Batbot's hind legs, achieving progressively better efficiency.

To elucidate the experimental data, we employed a parallel plot, an effective tool for visualizing multidimensional optimization spaces, Figure 4.21. This plot consists of five vertical lines, each representing a parameter under optimization. Connecting these

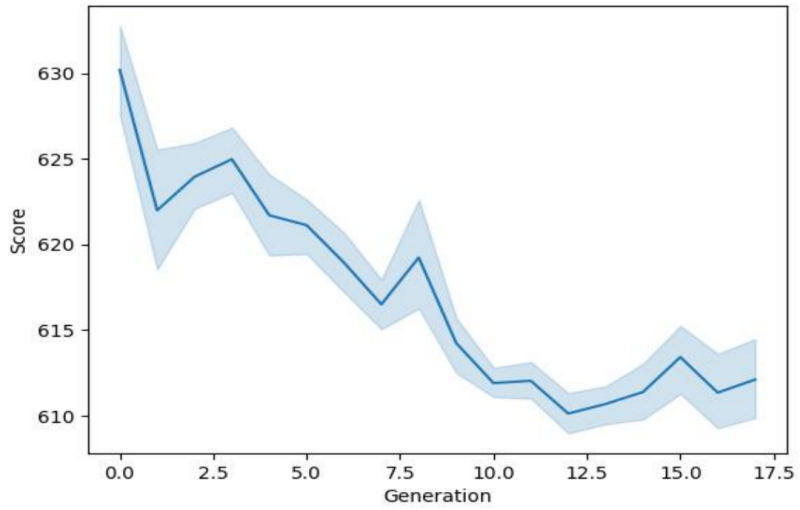


Figure 4.20 Score of optimization algorithm for elliptical movement

lines with zigzagging paths, each path represents a distinct experiment or solution candidate, colored according to its score. This visualization technique starkly illustrates the algorithm's convergence towards an optimal set of parameters, as shown in the collective trend of the experiment paths.

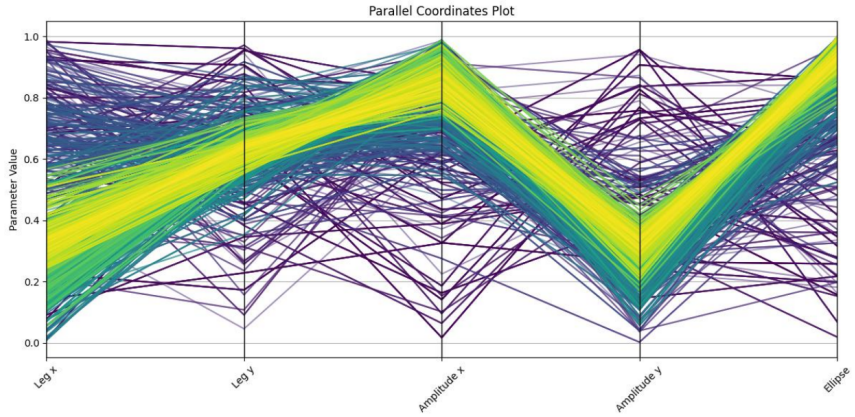


Figure 4.21 Parallel plot showing the result of the elliptical optimal experiment, with a clear convergence to a value

Interestingly, the optimal hind leg movement pattern, as illustrated by the convergence zone in the parallel plot, mirrors the locomotion observed in natural bats, affirming the biological plausibility and efficacy of our optimized design.

The experiments clearly indicate a successful application of the optimization algorithm, yielding a significant improvement in the Batbot's leg movement efficiency. However, the observation of score stagnation in the final generations on Figure 4.20 suggests the presence of mechanical design limitations. This plateauing effect prompts a need for further investigation into the mechanical constraints imposed by the Batbot's current de-

sign, particularly concerning the forces generated by flapping wings.

To address these limitations and explore potential enhancements, a subsequent phase of research will focus on a detailed analysis of aerodynamic forces and mechanical design parameters. Such an investigation aims to identify and mitigate the factors contributing to the observed optimization ceiling.

Our investigations into the Batbot's flapping forces reveal critical insights into the mechanics of bat-inspired flight and the inherent challenges of mimicking this biological phenomenon. The findings underscore the necessity for mechanical design enhancements to support higher flapping frequencies and forces, paving the way for future developments in hover-capable Batbot designs.

4.6 Analysis of Flapping Forces

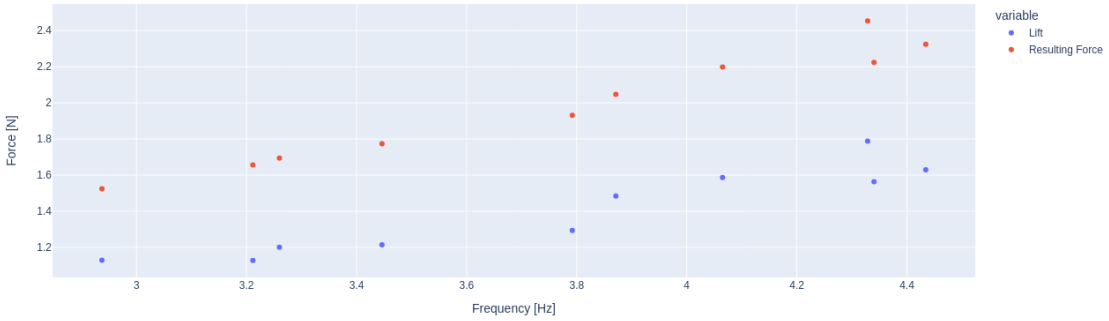


Figure 4.22 Force and Lift analysis compared to flapping frequency

The intricate dynamics of flapping flight present substantial challenges and opportunities for understanding and optimization. This chapter details our investigation into the forces exerted by the Batbot during its flapping motion, leveraging a six-axis sensor to capture the nuances of the generated forces.

To elucidate the nature of the forces generated during the Batbot's flapping cycle, we employed a six-axis sensor. This setup allowed us to measure the forces in real-time as the Batbot executed its flapping motions, offering insight into the directional components of these forces.

Initial observations revealed a surprising clarity in the directionality of the forces generated by the Batbot's flapping wings. Contrary to our initial hypothesis of a circular force distribution, the data pointed to two predominant force vectors: one upward and forward (corresponding to the upstroke) and one downward and forward (corresponding

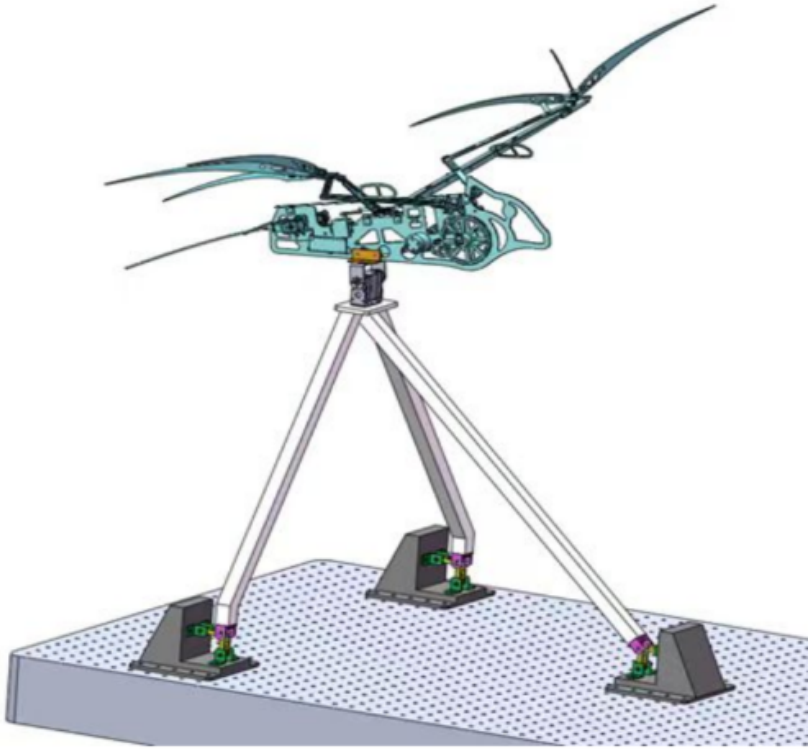


Figure 4.23 CAD of static test-bench

to the downstroke), Figure4.24. This discovery simplifies the calculation of the resultant force, which is pivotal for understanding the mechanics of Batbot flight.

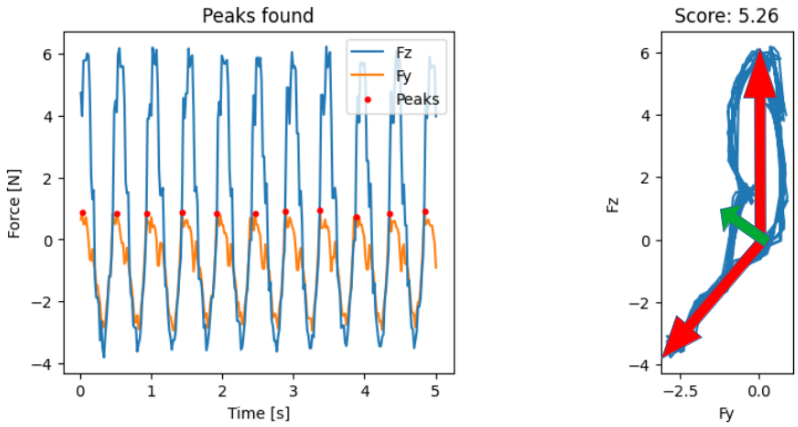


Figure 4.24 In red, 2 main force directions encountered during upstroke and downstroke. In green the resulting force

Through our analysis, we determined that at the highest safe operational frequencies, the Batbot generated a maximum resultant force of approximately 2 to 3 N. This figure falls significantly short of the 7 N required to counteract the Batbot's weight. This discrepancy raised questions, especially considering previous experiments where the Batbot successfully maintained and even gained altitude.

To probe this phenomenon further, we conducted experiments varying the flapping frequencies and analyzing their impact on lift and resultant force. Even near the safety threshold of 5 Hz, the forces did not exceed 3 N, underscoring a fundamental challenge in achieving sustained flight purely through flapping. The results can be clearly visualized in Figure 4.22

The resolution to this conundrum lies in the additional lift generated by the Batbot's forward motion, akin to the airfoil effect observed in airplanes. This principle, predicated on Bernoulli's theorem, elucidates how the Batbot's forward velocity creates differential air pressure above and below the wings, contributing to lift. It is this combination of flapping-generated force and airfoil lift that enables the Batbot to achieve and sustain flight.

The reliance on airfoil lift for sustained flight presents a notable challenge for achieving stable hover in the Batbot. Without the forward velocity to generate airfoil lift, maintaining altitude requires significantly higher flapping frequencies, similar to those observed in hummingbirds. This necessitates a reevaluation of the mechanical design, particularly concerning the durability and force transmission capabilities of the Batbot's wing actuation mechanism.

4.7 Experimental Validation of Enhanced Lift Capabilities in Batbot

Based on the results of the last chapter, improvements on the Batbot were implemented and the enhancement on the lift capabilities they brought thoroughly proved. In light of the insights gained from the force analysis of the Batbot's last version, it became evident that the existing flapping frequency and the force generation capabilities were insufficient for the Batbot to sustain flight without additional lift generated by the airfoil effect. To address these limitations, we embarked on the development of the Batbot's improved version, focusing on mechanical enhancements aimed at overcoming the identified shortcomings.

The transition to this version of the Batbot was marked by two significant mechanical improvements, each designed to enhance the operational efficiency and reliability of the flapping mechanism.

The first major upgrade was the incorporation of a second gear into the main axis of rotation for the wings. This modification aimed to double the area of contact between the gears, effectively halving the stress experienced by each gear. As the flapping fre-

quency increased, so too did the wind resistance encountered by the wings. Given that the resistance force exerted by the wind on the wings increases quadratically with velocity, the original gear system faced substantial stress, leading to gear teeth damage in high-frequency scenarios. The introduction of a second gear was a strategic move to achieve higher flapping frequencies without compromising the integrity of the gear teeth, thereby ensuring a more durable and robust gear system.

The second modification stemmed from the observation that the wings achieved their highest velocity when transitioning through the neutral angle, In light of the insights gained from the force analysis of the Batbot's last version, it became evident that the existing flapping frequency and the force generation capabilities were insufficient for the Batbot to sustain flight without additional lift generated by the airfoil effect. With a noticeable reduction in speed at the apex of each stroke. By extending the crane attached to the rotation axis by four millimeters, we could increase the amplitude of the wing's flapping motion. This adjustment allowed the wings to maintain higher speeds, especially when passing through the neutral position, without necessitating an increase in flapping frequency. Such a change was expected to contribute to enhanced lift generation, addressing one of the critical challenges identified in the force analysis of the previous version.

Despite the substantial mechanical revisions, the electronic systems and control mechanisms of the Batbot remained unchanged from version two. This decision was based on the satisfactory performance of the existing electronic components and the desire to isolate the effects of the mechanical upgrades on the Batbot's flight capabilities with a noticeable reduction in speed at the apex of each stroke. By extending the crane attached to the rotation axis by four millimeters, we could increase the amplitude of the wing's flapping motion. This adjustment allowed the wings to maintain higher speeds, especially when passing through the neutral position, without necessitating an increase in flapping frequency. Such a change was expected to contribute to enhanced lift generation, addressing one of the critical challenges identified in the force analysis of the previous version.

The experiment was conducted using a dynamic test, Figure 4.4, bench specifically designed for analyzing the lift capabilities of flapping-wing robots. The Batbot was suspended from a 6-axis sensor by an elastic string attached to its nose, Figure 4.4, facilitating the measurement of forces exerted during the flapping motion.

Initially, the force measured in the Z direction was equivalent to the weight of the

Batbot, indicating a static equilibrium with no lift generated. However,, with the improvements in the Batbots design, which allowed to safely reach higher flapping frequencies, a notable decrease in the Z-direction force was observed, eventually reaching zero. This phenomenon indicates that, momentarily, The Batbot generated sufficient lift to counteract its weight, effectively achieving a state of weightlessness measured by the sensor, seen in figure 4.25.

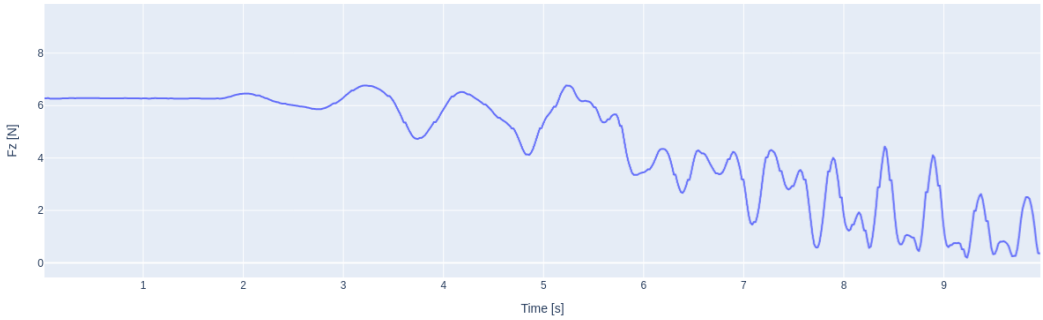


Figure 4.25 Lift result of dynamic test with a clear tendency to zero.

This observation is a critical validation of the enhancements introduced in this Batbot's Version. By achieving a force in the Z direction that momentarily nullifies the weight of the device, it demonstrates the robot's capability to generate lift sufficient for hovering. This marks a significant milestone in the development of bio-inspired robotic flight, as it confirms the mechanical feasibility of achieving hover without the necessity for forward motion or additional aerodynamic mechanisms, such as airfoils, traditionally relied upon for lift generation.

Utilizing Hooke's laws and normalization techniques for force direction and magnitude, we analyzed the displacement of the elastic string, 4.27 to calculate Batbot's motion during the experiment. Despite the achievement of sufficient lift, stable hovering flight remains a challenge. Figure 4.26 and illustrates the dynamic behavior of Batbot under flapping motion, highlighting the need for further refinement in control and stability to achieve sustained hover.

The experimental evaluation of Batbot on the dynamic test bench has conclusively demonstrated its ability to generate lift sufficient for counteracting its weight, thereby validating its potential for achieving hovering flight. This breakthrough paves the way for future advancements in the design and control of bio-inspired robotic systems capable of more complex aerodynamic maneuvers without relying on traditional lift-enhancing strategies.

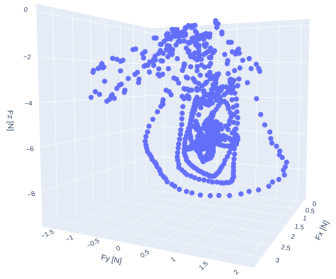


Figure 4.26 3D Force measurements from dynamic test-bench experiment

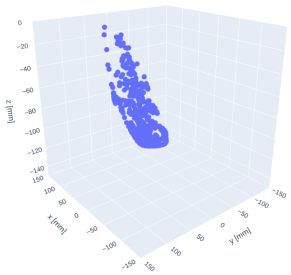


Figure 4.27 Position of the Batbot using force measurements and Hookes Law

4.8 Natural Environment Stability Test

In a pivotal experiment designed to evaluate the aerodynamic capabilities and stability of the Batbot, the robotic device was suspended outdoors using approximately 5 meters of elastic string, Figure 4.29. This setup provided the Batbot with ample freedom to exhibit a range of movements while ensuring its safety during the testing phase. The experiment involved actuating the Batbot while it was suspended, allowing for a thorough assessment of its functional dynamics in an open environment. The results of this experiment were highly informative; the Batbot demonstrated sufficient lift to support its own weight, a critical measure of its operational viability. Moreover, the experiment underscored the stability of the Batbot's flight, which can be attributed to its meticulous mechanical design. This stability is essential for the Batbot's practical application, indicating that the design parameters effectively harmonize with the aerodynamic demands of controlled flight. A free flight experiment, Figure 4.28, was executed where the Batbot was able to hover for around 3 seconds before destabilizing and falling to the safety net.

These results left us flabbergasted, as this proves that the mechanical design of the Batbot sufficed to achieve enough lift for hovering. This led the focus of the research toward the design and optimization of controlling algorithms.

Figure 4.28 Hovering experiment

4.9 3 Axis rotation Generation During Hovering

Based on the results of the last chapter experiment, improvements on the Batbot design were implemented. These enhancements aim at refining the Batbot's aerodynamic and control capabilities, drawing closer to the goal of achieving autonomous flight with complex maneuvering capabilities.

The transition to this Batbot version introduced pivotal mechanical and electronic improvements, each informed by the shortcomings and learnings from its predecessor. major mechanical innovation in this version was the scaling of the legs to match those of a real fruit bat. This adjustment not only enhances the aesthetic fidelity of the Batbot but, more importantly, significantly increases its torque generation capability. The elongated legs facilitate rapid and precise movements, thereby granting the Batbot enhanced rotational

Figure 4.29 Tethered hovering experiment, showing stability and hind-leg movement influence.

control in the air.

On the electronic front, the introduction of an embedded system streamlined the Batbot's circuitry. This consolidation achieved a substantial reduction in weight—by approximately 75 grams—and simplified the Batbot's internal structure, making it more agile and easier to manage.

Through experimental exploration and the enhancement of the Batbot's design, we have identified effective strategies for controlling the Batbot's rotation around the x, y, and z axes, crucial for navigating three-dimensional spaces.

- **X-axis Control:** Adjusting both legs either upwards or downwards facilitates pitch adjustments.
- **Y-axis Control (Roll):** Moving one leg up while the other moves down enables roll movements.
- **Z-axis Control (Yaw):** Disabling one leg while extending the other allows for yaw rotations.

The integration of an accelerometer and gyroscope sensor (MPU6050) was intended to refine the Batbot's positional awareness, feeding into a PID controller for precise movement control. However, the reliance on acceleration data introduced significant challenges.

Testing revealed that the MPU6050 sensor could not distinguish between gravitational and motion-induced accelerations, leading to erroneous pitch and roll angle calculations.

This limitation necessitated a reevaluation of our approach to sensing and data interpretation, underscoring the need for more sophisticated solutions to achieve stable hover and maneuverability.

In response to these findings, our immediate focus will shift towards enhancing forward flight stability using PID algorithms. This approach will serve as a foundational step towards more complex flight dynamics, including hovering. Future research will aim to integrate the control strategies developed through this work with improved sensing capabilities, potentially leading to fully autonomous Batbot flight.

In set of experiment, different configurations of the leg were tested to analyze the rotation generated in the Batbot, from different angles of the leg, to the rotation learned in section 4.5. In Figure 4.30 we can see some images of said experiments.

Figure 4.30 Experiment testing different leg configuration to generate torque for yaw, roll and turn motions

4.10 PID Controlled Flight

The development of a feedback controller is a pivotal element in the quest to achieve stable flight in bio-inspired aerial robots such as the Batbot. These controllers are critical because they enable the robot to autonomously adjust its behavior in response to dynamic environmental conditions and internal state changes. This adaptability is essential for mimicking the nuanced and highly efficient flight mechanics of biological bats, which rely on constant sensory feedback to maintain balance and navigate through complex en-

vironments.

The primary importance of a feedback controller lies in its ability to maintain stability during flight, which is a major challenge given the inherently unstable nature of flying systems. Without effective feedback mechanisms, even minor perturbations such as a gust of wind or a slight imbalance in weight distribution can lead to catastrophic failure. Therefore, developing a robust feedback controller not only enhances the safety and reliability of the Batbot but also ensures that it can perform complex aerial maneuvers similar to those of its biological inspirations.

Moreover, feedback control systems are fundamental for achieving autonomous flight operations, a key goal for drones and other unmanned aerial vehicles. As the Batbot progresses in its development, the feedback controller will enable it to undertake more complex tasks, operate in a wider range of environments, and provide valuable insights into the integration of biological flight principles into modern technology. This convergence of biology and technology through advanced control systems not only pushes the boundaries of robotics but also offers broader implications for both ecological studies and the development of innovative aerial vehicles.

4.10.1 Feed Back Controller

A feedback controller is a sophisticated system designed to maintain the desired state of a dynamic system by continuously adjusting its output based on real-time feedback. The basic principle of how a feedback controller works involves three primary components: the sensor, the controller, and the actuator.[41]

1. **Sensor**: The sensor measures the current state of the system. For example, in the context of a flying robot like the Batbot, sensors might include gyroscopes, accelerometers, and magnetometers that measure orientation, acceleration, and magnetic fields respectively.
2. **Controller**: The controller receives data from the sensors and compares it to a desired set point or reference value. It then computes the deviation or error between the current state and the target. Based on this error, the controller determines what adjustments need to be made to bring the system closer to its desired state.
3. **Actuator**: The controller sends signals to actuators, which execute changes in the system's operation. In aerial vehicles, these could be motors that adjust the angles of propellers or wings, altering the thrust and direction to stabilize flight and adjust the trajectory as needed.

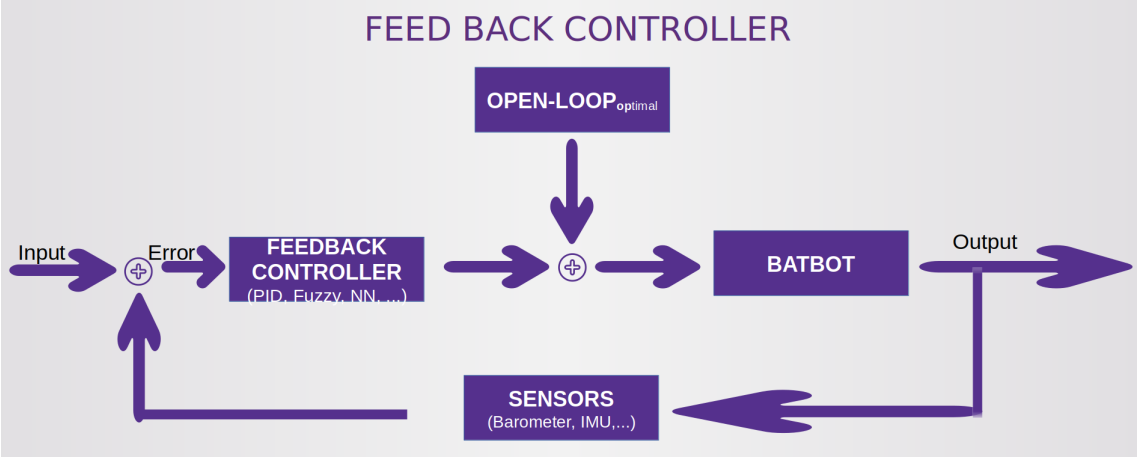


Figure 4.31 Feedback controller diagram

In contrast with the feed forward control pursued in the static test, in the dynamic test a feedback control is needed, as the Batbot will now be susceptible to perturbations and inaccuracies. Similar to the feed forward control, analysing, designing and testing such a controller is one tasks of this thesis. As in this case a feedback controller will be developed, sensors that indicate the Batbot of their current state will be equipped to it. Such sensors could be, for example, barometers to calculate the height of the Batbot, inertial measurement units to obtain the position and current movement of the Batbot, magnetometers to be aware of the Batbots orientation, between others possible options. There are many possibilities of constructing a feedback controller. We decided to implement the robust and well known PID controller.

4.10.2 Proportional Integral Derivative

A PID controller is a type of feedback controller widely used in industrial control systems. The PID controller continuously calculates an error value as the difference between a desired setpoint and a measured process variable. The controller attempts to minimize this error over time by adjusting the process control inputs.[41]

The PID controller is called such because it's composed of three parts:

1. Proportional Control (P): The proportional term produces an output value that is proportional to the current error value. The proportional response can be adjusted by multiplying the error by a constant Kp, known as the proportional gain constant.

2. Integral Control (I): The integral term is proportional to both the magnitude of the error and the duration of the error. It sums up past errors to calculate its output. The integral response is calculated by multiplying the integral of the error over time by a constant Ki, known as the integral gain constant.

3. Derivative Control (D): The derivative term is proportional to the rate of change of the error over time. This means the controller output is influenced by how quickly the error is changing. The derivative response is calculated by multiplying the derivative of the error by a constant Kd, known as the derivative gain constant.

The three constants (Kp, Ki, Kd) are tuned to achieve the best performance, and their optimal values depend on the nature of the system and the specific application.

The PID controller is popular because it is generally effective and relatively simple to understand and implement, although tuning the constants can sometimes be complex. It's widely used in various applications, such as temperature control, speed control, and process tuning in many industries.

4.10.3 IMU JY90 Sensor

The JY90X (IMU) series is a sophisticated module designed for measuring attitude angles, Figure 4.32. It captures raw data from multiple sensors, including three-axis accelerometers, gyroscopes, and magnetometers. Utilizing an advanced proprietary algorithm for attitude dynamics along with a high dynamic Kalman filter, it achieves accurate real-time estimations of three-dimensional attitude angles.[43]

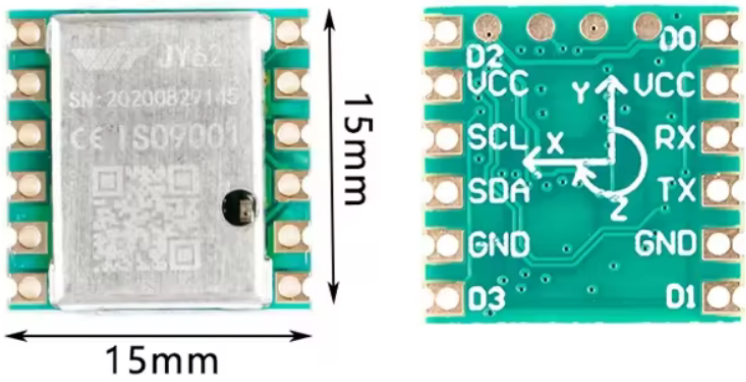


Figure 4.32 IMU JY90 Sensor[43]

Furthermore, the module incorporates magnetic field sensor data into the Z-axis heading calculations to address the gyroscopic drift typically observed in 6-axis systems. This integration ensures long-term stability of the heading angle outputs.

The Kalman filter is employed in the JY90X series to accurately estimate the Batbot's attitude by fusing data from multiple sensors. The integration of these updates ensures continuous refinement of the Batbot's attitude estimation, compensating for potential inaccuracies and sensor noise, thereby enhancing the stability and reliability of the Batbot's

operational parameters. This technology was then implemented to create the final version of the Batbot.

In the latest iteration, the Batbot retains its mechanical design from the previous version, with a strategic shift towards enhancing its electronic components. A pivotal enhancement was the replacement of the MPU6050 accelerometer and gyroscope withthe focusing on mechanical enhancement IMU-JY901 sensor.

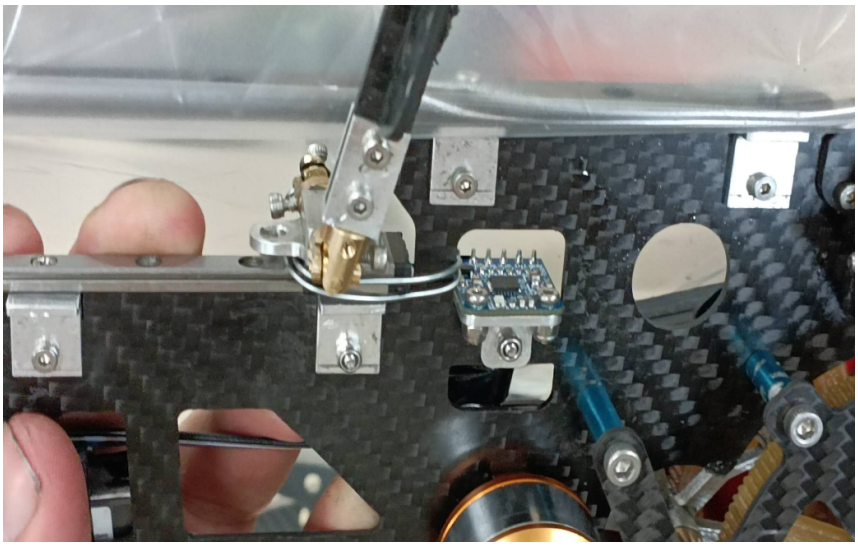


Figure 4.33 Positioning of MPU 5060accelerometer sensor in Batbot

Figure 4.34 Batbot version 5, 2 gears and tail membrane.

The IMU-JY901 sensor distinguishes itself by offering comprehensive sensory capabilities, including acceleration, gyroscope, magnetic earth direction, altitude, and temperature measurements. Crucially, it employs the Cayley algorithm, adept at integrating sensor data to produce reliable pitch and roll angles under dynamic conditions. This capability directly addresses the limitations previously encountered with the MPU6050 sensor, setting a solid foundation for advanced control mechanisms.

4.10.4 Application of PID Control for Flight Stabilization in Bio-inspired Batbot

The introduction of the IMU-JY901 sensor enabled the development of a robust PID control strategy, tailored to enhance the Batbot's maneuverability and stability in forward flight.

Our approach leverages two distinct PID controllers, one for managing the pitch and the other for controlling the roll of the Batbot. Initiation of flight begins with the Batbot in a neutral position, allowing for real-time adjustments via a control remote. This setup facilitates precise manipulation of the Batbot's orientation, enhancing its flight dynamics. The calibration process involved iterative testing to fine-tune the PID controllers' K values, focusing sequentially on the P, D, and I gains. Initial tests ensured the Batbot's capability for sustained flight, followed by open-air trials to observe and adjust the Batbot's behavior dynamically. This meticulous calibration process was crucial for identifying the optimal settings for both roll and pitch control, significantly improving the Batbot's stability and responsiveness.

The flight stability of a bio-inspired Batbot is crucial for its performance and operational efficiency. This section discusses the implementation oPID controllers to regulate the bat's flight attitude, specifically focusing on pitch and roll adjustments using leg movements. The PID controllers are designed to respond to feedback from an IMU-JY901 sensor, which offers accurate pitch and roll angle measurements enhanced by a Kalman filter.

For effective flight stabilization, two separate PID controllers are employed: one for pitch control and another for roll control. The initialization parameters for these controllers are as follows:

- Pitch PID Controller: Proportional gain (k_p) is set to -0.75, with integral (k_i) and derivative (k_d) gains set to 0.
- Roll PID Controller: Proportional gain (k_p) is set to -1.5, with integral (k_i) and derivative (k_d) gains also set to 0.

The neutral states of pitch and roll are dynamically set through input from a radio controller, allowing for real-time adjustments to the bat's desired flight posture. The actual pitch and roll angles, measured by the IMU-JY901 sensor, are fed into their respective PID controllers. The output of these controllers is then used to adjust the neutral pitch and roll, resulting in compensated pitch and roll angles. These adjustments directly influence the

vertical orientation of the bat by modifying the y-axis angle (y_θ) through leg movements.

The PID control logic for leg movement adjustments can be represented as follows:

$$y_{\theta_{new}} = y_{\theta_{neutral}} + PID_{pitch}(pitch_{measured}) \tag{4.17}$$

$$\text{Leg}_{right} = y_{\theta_{new}} + PID_{roll}(roll_{measured}) \tag{4.18}$$

$$\text{Leg}_{left} = y_{\theta_{new}} - PID_{roll}(roll_{measured}) \tag{4.19}$$

Following the implementation of PID controllers for flight stabilization, extensive experimentation has been conducted to validate the enhanced capabilities of the bio-inspired Batbot. After several trials, we achieved a significant milestone in one successful experiment where the Batbot not only maintained a controlled flight but also gained approximately 1 meter of altitude and flew for approximately 10 meters. This achievement, however, was constrained by the length of the safety net used during the experiments. In future studies, conducting experiments within larger, safer environments could potentially yield even more impressive results, showcasing the full capabilities of the Batbot's flight dynamics and further improvements in altitude gain, snapshots of the experiment can be seen in Figure 4.35.

Figure 4.35 Successful PID experiment, achieving approx 10 meters of straight controlled flight

4.11 Results and Conclusion

The comprehensive experiments conducted with the Batbot have yielded significant insights into the optimal parameters for controlling this complex robotic system. Through meticulous testing and analysis, we have been able to refine the Batbot's operational parameters, leading to a substantial improvement in its mechanical design. These enhancements are crucial for ensuring the Batbot's functionality aligns closely with the intricate

dynamics of biological bat flight.

Moreover, the deployment of an evolutionary algorithm system has proven exceptionally beneficial. This advanced approach has not only validated its efficacy but has also significantly reduced the time required to identify optimal parameters. Traditional methods, constrained by the complexity of the task, would likely have failed to uncover these parameters or would have demanded considerably more time and resources. The evolutionary algorithm has demonstrated its capacity to navigate through complex parameter spaces efficiently, achieving results that were previously unattainable.

Additionally, this research has laid a solid foundation for the future development of the Batbot. The integration of feedforward and feedback controllers developed throughout these experiments promises to enhance the Batbot's adaptability and performance. This hybrid control system is anticipated to synergize the strengths of both control mechanisms, fostering a more robust and responsive navigation capability in the Batbot.

In conclusion, the findings from these experiments not only advance our understanding and capabilities in robotic flight but also pave the way for future innovations in autonomous aerial robotics. The continued development of the Batbot is expected to contribute significantly to the field, pushing the boundaries of what is possible in robotic mimetic of biological flight.

CHAPTER 5 BIO-MIMICKING OPTIMIZATION RESULT

Understanding the complex biomechanics of bat leg movements is pivotal for advancing the development of flapping-wing robots. Due to the practical challenges associated with directing live bats in controlled experiments, robotic simulations provide a valuable platform for these studies. The Batbot allows for precise control over hind leg movements, offering insights into their functional roles during flight.

5.1 Bat Data

In the dissertation Kinematic Analysis of a Bat Flight Based on Membrane Wing Motion Reconstruction by Dr. Sudeep Kumar Singh, several short-nosed fruit bat Cynopterus sphinx were studied during flight to reconstruct the wing kinematics and to identify the primary relationship of kinematic parameter in straight and maneuvering flight[42] . The data obtained by Dr. Sudeep was captured using the ”OptiTrack” system, Figure 5.2, and pasted markers on the body as seen in Figure 5.3, wings and legs of the bat. The results of his dissertation and the flight data obtained, an example can be seen in Figure 5.1 can give us useful insight from which we can develop our controller algorithms. Specially the analysis of the aerodynamic forces on a bat body Dr. Sudeep was able to realize using the gathered data.

5.2 Synchronized Wing and Leg Movement Control

In bio-inspired robotics, achieving synchronized and naturally fluid motion patterns between different locomotive appendages, such as wings and legs, is paramount for mimicking the locomotion of biological counterparts. This chapter details the development and implementation of a control mechanism designed to orchestrate the synchronized flapping of wings and the corresponding movement of legs in a robot. The mechanism employs a phase-shift control strategy, allowing for the adjustable synchronization between the cyclical wing flapping and leg movements. For this experiments the coordinate system found in Figure 5.4 were used.

The fundamental aspect of the control mechanism is the cyclic nature of the movement patterns, defined by a parameter cyc_pi, which ranges from 0 to 2π. This parameter

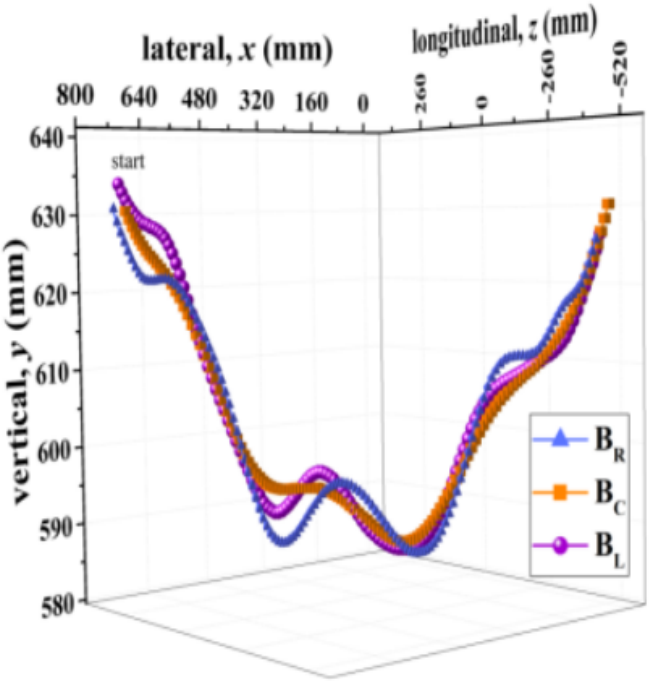


Figure 5.1 Example of the data obtained of the flight path of a bat.[42]

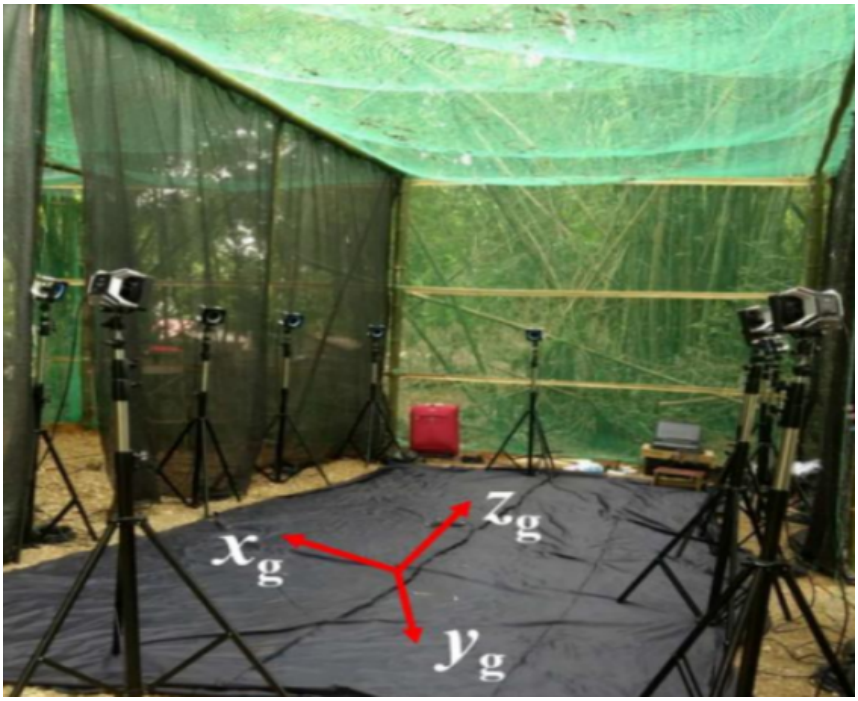


Figure 5.2 Optitrack device used to measure the trajectory of the flying bats.[42]

represents the progression through a complete cycle of movement, from the downstroke to the upstroke of the wing beat, and back to the initial position.

To introduce a phase shift between the wing and leg movements, a variable named $delay$ is utilized. This delay adjusts the phase of the leg movement cycle relative to the

Figure 5.3 Markers used to measure bat flight trajectory

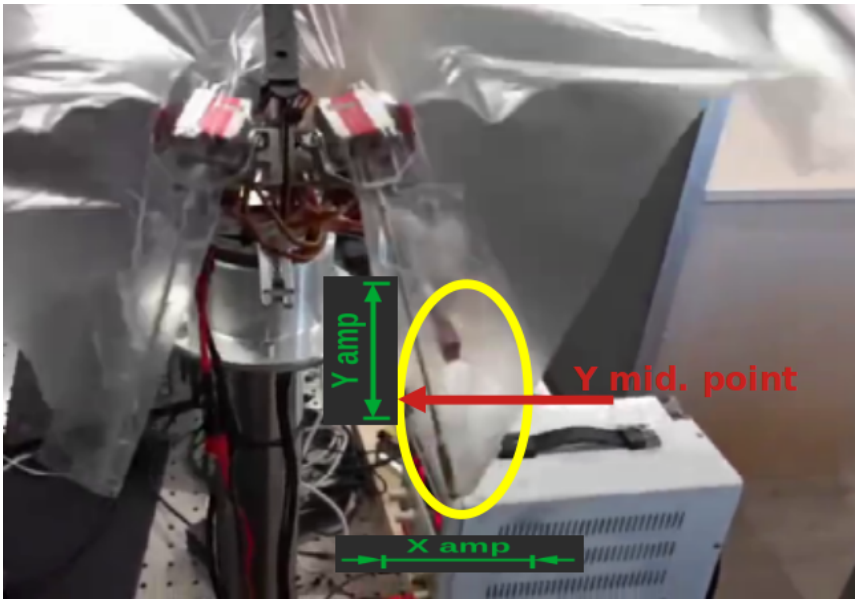


Figure 5.4 Definition of x amplitude, y amplitude and y neutral state

wing flapping cycle, allowing for precise control over the synchronization of these two motions.

A pivotal function, $is\_leg\_upward(cyc\_pi, range\_start)$, is introduced to ascertain whether the legs should be moving upward or downward at any point in the cycle. This determination is based on whether the current phase of the cycle, adjusted for the phase shift, falls within a specific range. The function's algorithm can be mathematically represented as follows:

$$\text{is\_leg\_upward} =$$

$$\begin{cases} \text{True, if (range\_start} \leq \text{cyc\_pi} < \text{range\_end)} & (\text{range\_start} < \text{range\_end}) \\ \text{True, if (cyc\_pi} \geq \text{range\_start} \vee \text{cyc\_pi} < \text{range\_end)} & (\text{range\_start} \geq \text{range\_end}) \\ \text{False, otherwise} \end{cases}$$

where:

$$range\_start = delay \tag{5.1}$$

$$range\_end = range\_start + \pi \tag{5.2}$$

Based on the output of the is_leg_upward function, the angles for the leg's x and y directions are adjusted accordingly. The x direction angle, x_theta, is set to a predefined amplitude, x_amp, if the leg is moving upward, and to 0 otherwise. The y direction angle, y_theta, is calculated as a function of the cosine of the phase-shifted cycle parameter, reflecting the natural oscillatory motion of the leg:

$$y_\theta = y_{\text{mid}} - y_{\text{amp}} \cdot \cos(\text{cyc\_pi} + \text{delay}) \tag{5.3}$$

Finally, the calculated angles and the determined state of leg folding are passed to the *move* function, which actuates the motors to achieve the desired positions and states for both wings and legs, thus completing the cycle of synchronized movement.

5.3 Experimental Setup and Parameters

Using data from Dr. Sudeep's dissertation as a reference[42] , the dimensions of the Batbot were tailored to mimic those of the actual bat, Figure 5.7. The real bat's wingspan of 440 millimeters and leg length of 40 millimeters were scaled to the robot with a wingspan of 1446 millimeters. Using a proportional scaling method, the leg length for the Batbot was calculated to be 130 millimeters, maintaining the anatomical ratio.

All experiments were conducted using a consistent input voltage of 11.6 volts, and each test had a duration of 10 seconds to ensure sufficient data for robust statistical analysis. This duration captures multiple flapping cycles, allowing for reliable averaging of the forces exerted by the Batbot.

5.4 Data Collection and Analysis

Force measurements were taken in the z-direction (lift) and y-direction (thrust), with initial and final measurements potentially capturing incomplete cycles. To address this,

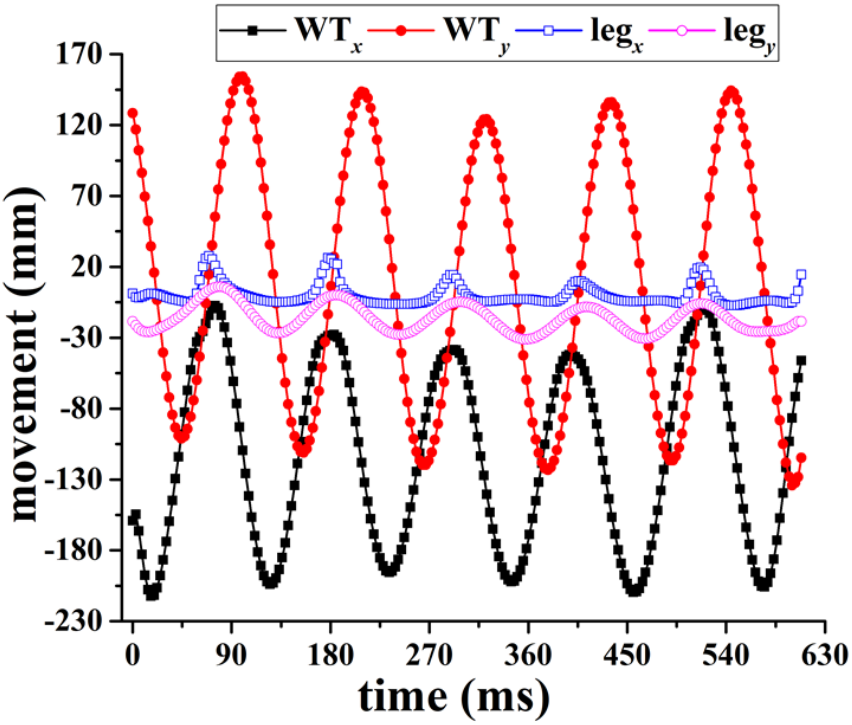


Figure 5.5 Data from real bat, general movements.

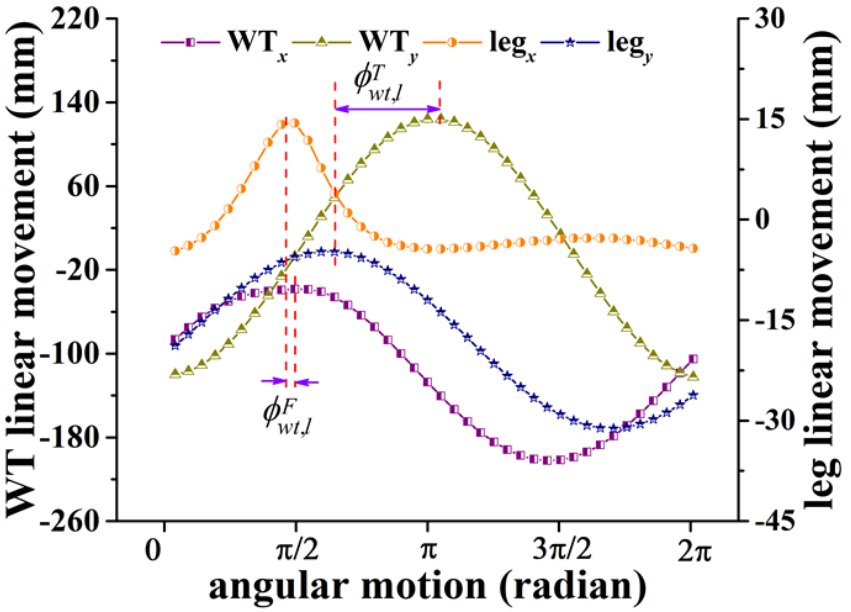


Figure 5.6 Data from real bat, phase shift.

peak analysis was employed to identify complete flapping cycles, discarding data outside the first and last detected peaks.

The trajectory of the resulting force vector throughout the flapping cycles was plotted, with z-force on the vertical axis and y-force on the horizontal axis. This plot aids in understanding the dynamic behavior of the forces involved, similar to Figure 4.18.

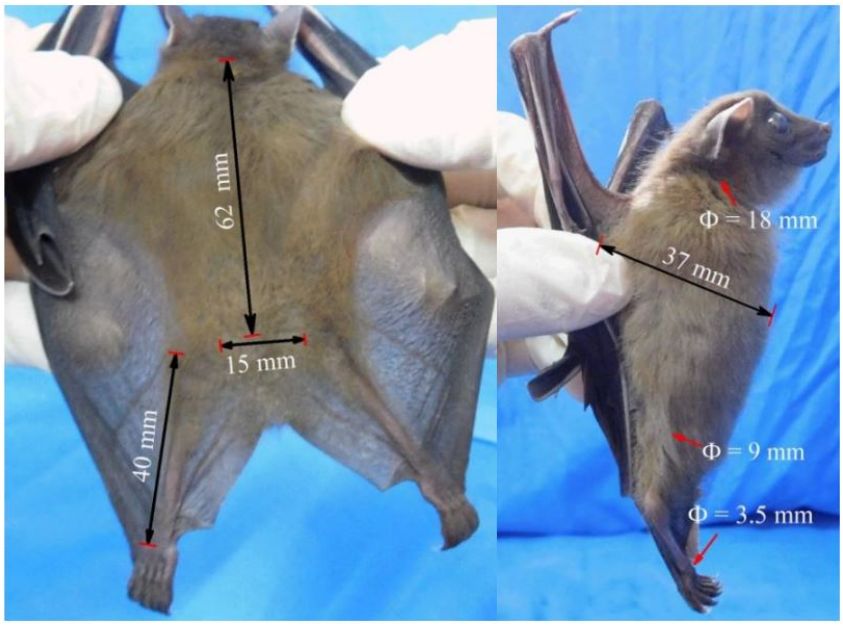


Figure 5.7 Measurements from bat

The average forces in the z and y directions were computed, from which the vector of the resulting force was determined. Using the Euclidean distance, the magnitude of this force and its direction (via the arc tangent) were calculated. Additionally, the frequency of the flapping cycles was derived from the average distance between peak intervals, providing a quantitative measure of the Batbot's operational rhythm.

5.5 Metrics

The force measurements were analyzed to compute the average forces in the z-direction (lift) and y-direction (thrust). The relationship between force components and their influences on flight stability and direction were extensively studied using vector analysis:

$$\text{Resultant Force} = \sqrt{F_z^2 + F_y^2} \tag{5.4}$$

$$\text{Direction} = \arctan\left(\frac{F_z}{F_y}\right) \tag{5.5}$$

The frequency of the wingbeats is a critical parameter for understanding the aerodynamics of the Batbot. It is calculated as the inverse of the average period duration, which is the time between consecutive peaks in the force data. Let T denote the average period

duration, the frequency f can be calculated using:

$$f = \frac{1}{T} \tag{5.6}$$

This frequency calculation helps in quantifying the rhythmic patterns of the Batbot's wings, correlating the mechanical movements to the aerodynamic forces generated during flight.

To determine the average period duration T, we identify the time intervals between each detected peak in the force measurement data. If $t_1, t_2, ..., t_n$ represent the times at which peaks occur, the average period duration is given by:

$$T = \frac{\sum_{i=1}^{n-1}(t_{i+1} - t_i)}{n-1} \tag{5.7}$$

where n is the total number of peaks detected. Substituting this back into the formula for frequency gives us:

$$f = \frac{n-1}{\sum_{i=1}^{n-1}(t_{i+1} - t_i)} \tag{5.8}$$

This approach provides a comprehensive method for analyzing the temporal dynamics of the Batbot's flight, linking force measurements to wingbeat frequency, which is pivotal for optimizing flight control and stability.

The series of experiments conducted provided significant insights into the complex dynamics of bat flight, particularly the role of hind leg movements. Through the application of bio-inspired robotics and evolutionary algorithms, this research not only enhances our understanding of natural flight mechanisms but also guides the development of more efficient and agile robotic designs.

5.6 Experiments

This section presents a series of experiments designed to explore the biomechanical roles of hind legs in bat flight and replicate these dynamics using a robotic model, referred to as Batbot. By systematically varying the movements and parameters of the Batbot's hind legs, the experiments aim to elucidate their impact on flight mechanics, such as lift and thrust generation.

5.6.1 Experiment 1: Simulation of Bat Hind Leg Movement

In this experiment, our objective was to accurately simulate the real bat's hind leg movements. A significant challenge we encountered involved the unit of measurement; data from the real bat was provided in millimeters, whereas our robotic bat (bat bot) operates in terms of angular displacement.

To address this, we converted the linear movement data into angular data using trigonometric principles. The known parameters were as follows:

- Lateral amplitude of hind leg movement: 16 mm
- Vertical amplitude of hind leg movement: 10.5 mm
- Length of the hind leg: 40 mm

Considering the hind leg as the hypotenuse of a right triangle, with lateral and vertical movements representing the opposite sides, we employed the arcsine function to calculate the angular amplitudes:

$$\theta_{\text{lateral}} = \sin^{-1}\left(\frac{16\,\text{mm}}{40\,\text{mm}}\right) \approx 23.5^\circ \tag{5.9}$$

$$\theta_{\text{vertical}} = \sin^{-1}\left(\frac{10.5\,\text{mm}}{40\,\text{mm}}\right) \approx 15^\circ \tag{5.10}$$

During the upstroke phase, the legs opened to $2 \times 23.5^\circ = 47^\circ$ and returned to 0° in the downstroke. The vertical movement synchronized with the wing movements, with the legs moving 15° upward from the original position during each cycle.

5.6.2 Experiment 2: Evaluating the Impact of Hind Leg Movement

The goal of the second experiment was to ascertain the influence of hind leg movements on our bat bot's performance. To isolate the effect of the hind legs, we performed the following procedure:

We fixed the hind legs in a static position, allowing only the folding and unfolding functions of the bat bot to operate. This setup was intended to provide insights into the actual role of hind leg movements.

Results Comparison:

- **Experiment 1:** Demonstrated a force output of 0.96 Newtons at an angle of 23.51° and a frequency of 1.4 Hz.
- **Experiment 2:** With the hind legs immobilized, the force output decreased to 0.9 Newtons, at an angle of 22.22° and a reduced frequency of 0.33 Hz.

These findings suggest that hind leg movements significantly contribute to the aerody-

namic performance of the bat bot, influencing both the force produced and the operational frequency.

5.6.3 Experiment 3: Analyzing the Effect of Static Wing and Leg Positions

In the third experiment, our objective was to further explore the dynamics of the bat bot by immobilizing all articulated movements. Given the insights from the previous experiments regarding hind leg mobility, we aimed to evaluate the impact of maintaining both the legs and wings in a fully extended, static position throughout the test.

Experimental Setup:

- The legs were kept fully stretched, eliminating any leg movement.
- The wings were also held in a completely extended position, with no folding or unfolding occurring during the experiment.

Results: The bat bot demonstrated a force output of 0.89 Newtons, at an angle of 22.32° and a frequency of 1.7 Hz. These results suggest that the absence of both leg and wing movements affects the operational dynamics of the device, though the impact on aerodynamic efficiency appears to be minimal. This points out that the hind-leg movement plays a much more important role than the folding mechanism.

5.6.4 Experiment 4: Investigating the Effect of Lateral Amplitude Variations

In the fourth experiment, our focus shifted to understanding the influence of varying lateral amplitudes on the performance of the bat bot. This experiment was an extension of the first, featuring similar conditions with the reactivation of both folding and unfolding of the wings, along with movements in the X and Y directions.

- We replicated the conditions from Experiment 1 with one significant modification: we systematically varied the lateral amplitude.
- Movement in the Y direction was kept synchronous with the wing bit, maintaining a consistent amplitude of 15°.
- A total of ten test sets were conducted, with lateral amplitudes ranging from 0° to 90°.

The Id of each test and the lateral amplitude testes can be found in the Table 5.1, as well as the resulting metrics

Table 5.1 Results of varying lateral amplitudes Batbot performance

ID	Lateral Amplitude	Force (Newtons)	Angle (Degrees)	Frequency (Hz)
1	10°	0.950144	21.030838	1.633416
2	20°	0.975389	22.316757	1.647427
3	30°	0.981344	22.427117	1.646936
4	40°	0.983461	22.557174	1.650880
5	50°	0.990144	23.166601	1.655432
6	60°	0.978789	21.339109	1.662857
7	70°	0.994345	24.581319	1.673474
8	80°	0.984389	22.443446	1.679016
9	90°	1.001941	23.304924	1.663707

5.6.5 Experiment 5: Assessing the Impact of Phase Shifts on Movement Synchronicity

Experiment 5, inspired by the findings of Experiment 2, aimed to investigate the effects of phase shifts between the movements of the hind legs and the wing beats on the bat bot's performance. The experiment specifically focused on how delays in the synchronization influence the dynamic forces generated by the bat bot.

Experimental Setup:

- The amplitudes were set as follows: X amplitude at 47° and Y amplitude at 15°, identical to the settings in Experiment 2.
- We introduced a variable 'delay', representing the phase shift in the hind legs' movements relative to the wing beats. This delay varied from 0 to 2π, encompassing a full cycle of potential phase shifts.
- Values from 0 to π were considered as the hind legs lagging behind the wing beats, whereas values from π to 2π were considered as the hind legs moving ahead of the wing beats.

Objective: To determine how varying the synchronization (phase shift) affects the force, frequency, and angular dynamics of the bat bot.

Results: The results, illustrating the relationship between the delay in leg and wing movements and the corresponding outputs in force, frequency, and angle, are summarized in the table 5.2.

Table 5.2 Results of varying phase shift between wing and hind-leg flapping Batbot performance

ID	Lateral Amplitude	Force (Newtons)	Angle (Degrees)	Frequency (Hz)
1	$1 * \frac{2\pi}{10}$	0.932566	28.507024	1.610714
2	$2 * \frac{2\pi}{10}$	0.824917	28.742679	1.533171
3	$3 * \frac{2\pi}{10}$	0.807341	30.053851	1.509684
4	$4 * \frac{2\pi}{10}$	0.872650	24.605598	1.612093
5	$5 * \frac{2\pi}{10}$	0.910096	19.072184	1.666386
6	$6 * \frac{2\pi}{10}$	0.947108	19.750115	1.683218
7	$7 * \frac{2\pi}{10}$	0.986386	21.155423	1.677177
8	$8 * \frac{2\pi}{10}$	1.005331	24.620348	1.669380
9	$9 * \frac{2\pi}{10}$	0.980961	25.045745	1.658976

5.6.6 Experiment 6: Optimization Using Evolutionary Algorithms

The sixth and final experiment of this chapter aimed to evaluate the effectiveness of evolutionary algorithms in optimizing control strategies for the Batbot hind-leg movement. This method, developed as part of the thesis, holds potential for broad application across various types of robots, especially given current technological needs.

Objective: The primary objective was to optimize a set of parameters to maximize the force generated by the Batbot, thereby demonstrating the utility of evolutionary algorithms in robotic control.

Parameters to Optimize:

- **X Amplitude:** The angle of the hind leg's upstroke movement during flapping, ranging from 0° to 90°.
- **Y Amplitude:** The amplitude of the Y-axis movement, ranging from 0° to 60°.
- **Y Neutral Position:** The starting angle for amplitude calculations during the hind leg's flapping cycle, influencing the range between 15° and 45°, based on a neutral angle of 30°.
- **Delay:** The phase shift or delay in hind leg flapping relative to wing flapping, varying from 0 to 2π.

Optimization Approach: To address the evolutionary algorithm's default tendency to minimize the score function, we defined the score as the negative of the force output. Thus, by minimizing the score function (making it as negative as possible)

$$\text{Score Function} = -(\text{Resultant Force}) \tag{5.11}$$

5.7 Analysis of Experimental Results

In this section, we analyze and compare the results from the various experiments conducted, emphasizing the critical role different parameters play in force generation by our robotic bat model.

5.7.1 Analysis of Experiment 1: Synchronous Bat Mimic Motion

Experiment 1 served as our benchmark with a thrust value of 0.88 Newtons and a lift of 0.38 Newtons, leading to a total force of 0.96 Newtons. The dominance of thrust over lift in this experiment establishes a baseline for evaluating other experiments.

5.7.2 Analysis of Experiment 2: Impact of Static Hind Legs

The immobilization of the hind legs led to reduced thrust and lift, with a total force of 0.89 Newtons. This reduction demonstrates the significant role of hind leg motion in enhancing aerodynamic forces, contradicting initial assumptions that a stretched membrane might increase force.

5.7.3 Analysis of Experiment 3: Effects of Static Wing and Legs

Maintaining static positions for both wings and legs resulted in force outputs similar to Experiment 2, confirming that while hind leg movements are crucial, the folding and unfolding of wings alone do not significantly alter lift and thrust. We can see a comparison of experiment 1 2 and 3 in Figure 5.8, were we can see 2 things that stand out, first that in all metric the baseline experiment 1 showed a better performance, showing that the role of mimicking the hind-leg movement plays and important role in force generation. We can also see that in all cases the predominant factor of the resulting force was the thrust generated, this can also be seen in Figure 5.9.

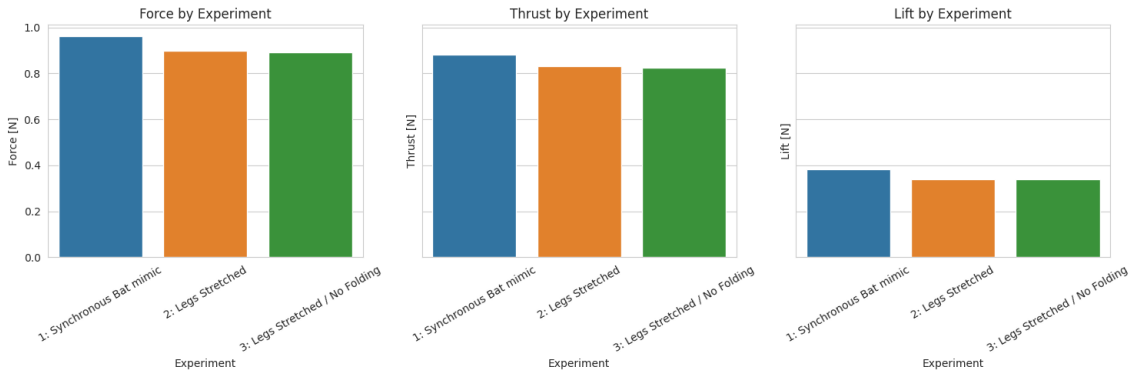


Figure 5.8 Thrust, Lift and Force comparison of experiments 1,2 and 3

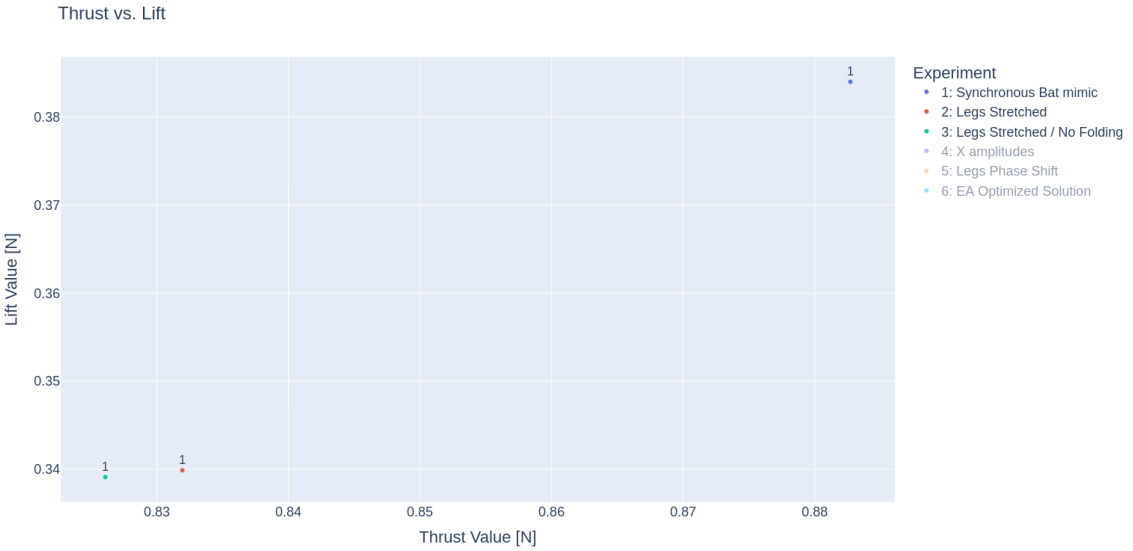


Figure 5.9 Force vector of Experiments 1, 2 and 3

5.7.4 Analysis of Experiment 4: Lateral Amplitude Variations

Varying the lateral amplitude showed that larger amplitudes enhance thrust, with the maximum force of 1.0 Newtons, Figure 5.11 achieved at 90°. This suggests a direct correlation between increased amplitude and force, albeit with diminishing returns at higher angles. This is clearer represented in the bar plot found in Figure 5.10, where higher angles show an ascending force. The horizontal red line show the reference, where we can see that some test have achieved a better performance. In Figure 5.11, we can also see that the variability of the resulting forces is relatively small as all the results lie close to each other.

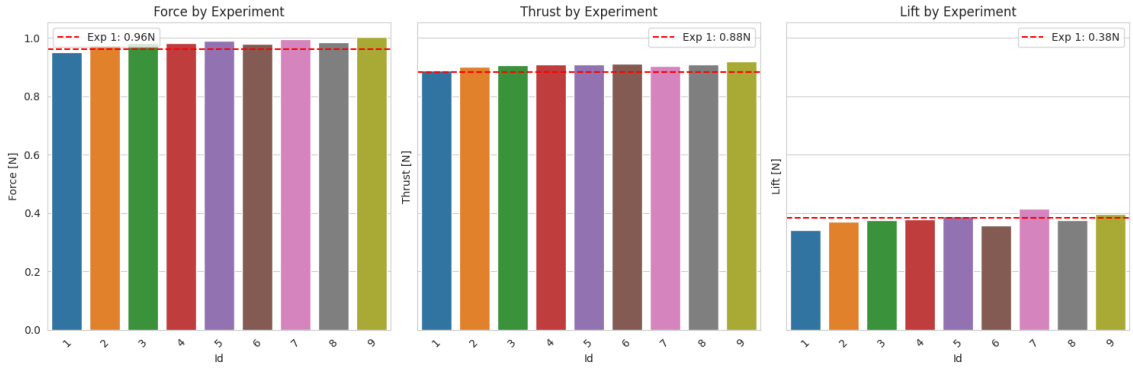


Figure 5.10 Thrust, Lift and Force comparison of experiment 4 with base line Experiment 1

5.7.5 Analysis of Experiment 5: Phase Shift Implications

Adjusting the delay in hind leg movements relative to wing beats highlighted the importance of synchronization, seen in the large variance seen in Figure 5.13. Delays be-

Figure 5.11 Force vectors of 10 test done in experiment 4

tween 0 and π (representing a lag) consistently resulted in poorer performance, while shifts from π to 2π (indicating leading movements) enhanced force outputs. This non-linear relationship suggests complex dynamics between synchronization and force production found in Figure 5.13.

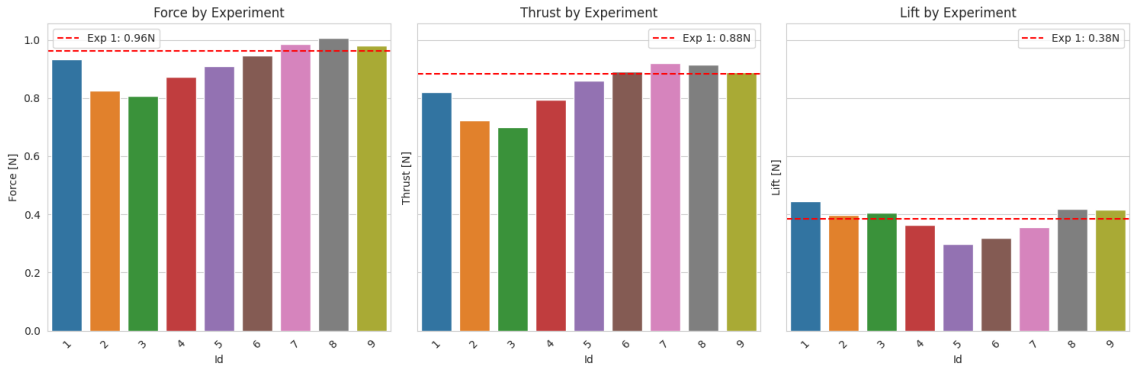


Figure 5.12 Thrust, Lift and Force comparison of experiment 5 with base line Experiment 1

5.7.6 Correlation Analysis and Further Insights

A correlation analysis was conducted to quantitatively assess the relationships among the measured variables as seen in Figure 5.14:

- Thrust showed a strong correlation (0.97) with overall force, underscoring its primary role in force generation.
- Lift displayed a minimal correlation, emphasizing its lesser impact.
- Frequency had a surprisingly high correlation (0.82) with force, indicating that higher frequencies, facilitated by reduced wind resistance through optimal leg

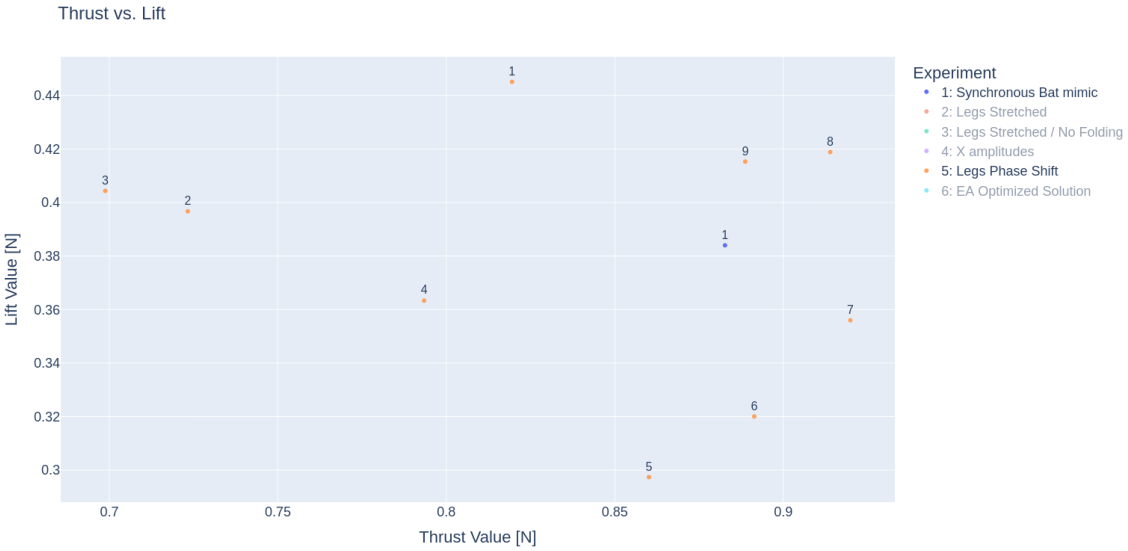


Figure 5.13 Force vectors of 10 test done in experiment 5

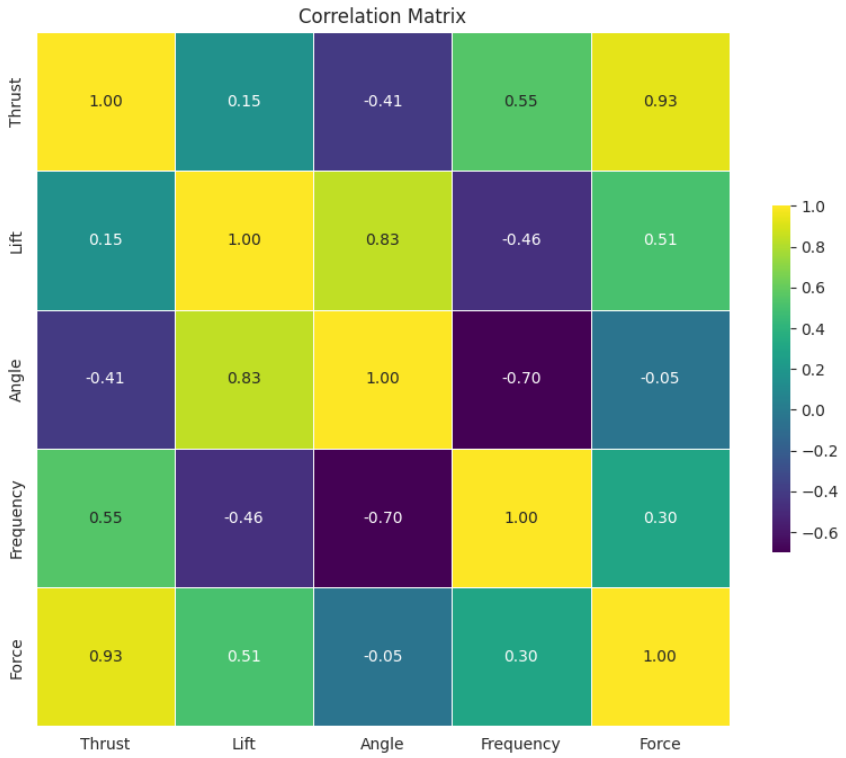


Figure 5.14 Correlation matrix of all test conducted in experiment 1-5

movements, are crucial for maximizing force.

Frequency Analysis: Upon plotting frequency against force, a clear collinearity was observed, Figure 5.15. This relationship was unexpected given the consistent power supplied to the bat bot's motor, suggesting that variations in leg movement and resistance play a significant role in achieving higher frequencies.

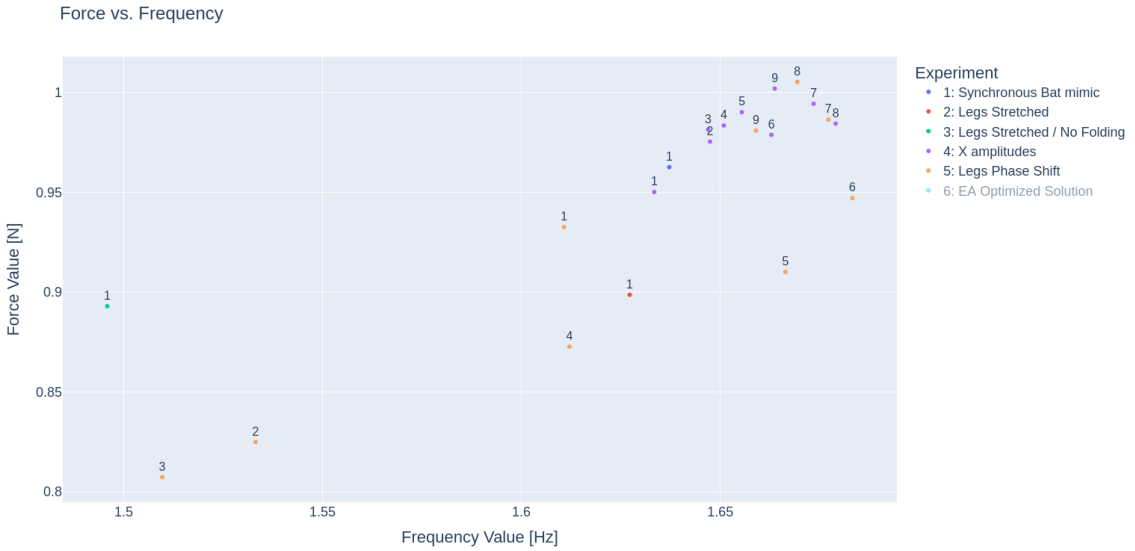


Figure 5.15 Force depending on frequency of all test conducted in experiment 1-5

The comprehensive analysis reveals that optimizing hind leg movement to reduce wind resistance is a pivotal strategy for enhancing force generation. This insight directs future experiments to focus on maximizing frequency through aerodynamic efficiency rather than merely increasing the force through direct mechanical inputs.

5.8 Results of Optimization Experiments

This section discusses the outcomes of the optimization experiments conducted using an evolutionary algorithm to enhance the performance of the Batbot. The algorithm aimed to find optimal parameters that maximize the force generation by adjusting various mechanical movements of the robot's legs.

5.8.1 Evolutionary Algorithm Configuration

The optimization process was conducted over 25 generations, with each generation consisting of 10 trials. Each experiment lasted 6.5 seconds, accumulating to a total experimental time of approximately 27 seconds across all trials and generations.

5.8.2 Results Analysis

The force generated by the Batbot improved significantly over the course of the experiments, starting from an initial average force of 0.95 N and achieving a maximum force of 1.11 N. The best-performing parameters were observed in the final generation, specifically in experiment number 9, which recorded a force of 1.117 N. The parameters for this

experiment were:

- X total amplitude: 49.6˚
- Y amplitude: 20.3˚
- Neutral Y angle: 24˚
- Delay: 5.24 radians

5.8.3 Comparative Analysis

The optimized parameters led to an improvement in lift generation rather than thrust, Figure 5.16, which contributed significantly to the increase in overall force by approximately 16% compared to the baseline established in the Bio-Mimicking Experiment 1, as clearly seen in Figure 5.17

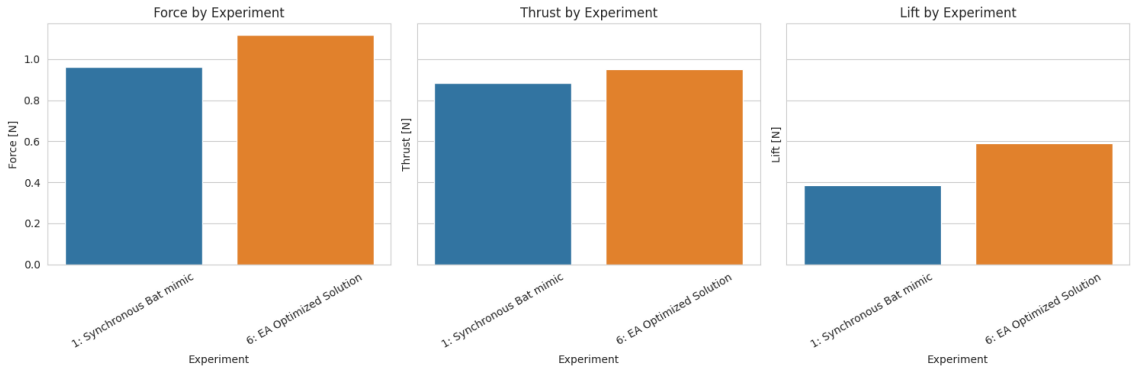


Figure 5.16 Force, Thrust and Lift forces of optimal solution found compared with Experiment 1, a clear improvement of lift can be appreciated

Figure 5.17 Force vector where the optimized results shows clear superiority compared to all other tests

5.8.4 Frequency and Force Correlation

Interestingly, the optimized setup achieved high force outputs at relatively low flapping frequencies, Figure 5.18, suggesting that the leg movements optimized by the algorithm enhance lift more effectively than merely reducing wind resistance. This finding diverges from earlier experiments where higher frequencies typically correlated with greater force generation.

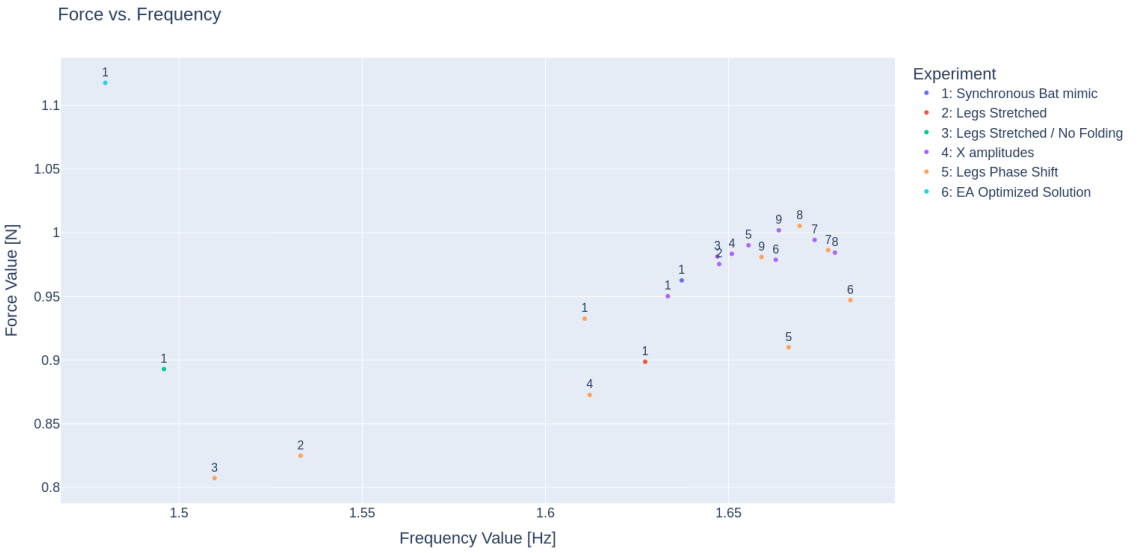


Figure 5.18 Force depending on frequency of all test, where optimal result shows a clear higher force, even by low frequency

5.8.5 Covariance and Parameter Convergence

The resulting parallel coordinates plot from the evolutionary algorithm indicated a strong convergence towards optimal parameters:

- X amplitude: 0.55 (49.6˚ after interpolation)
- Y amplitude: 0.3 (20.3˚after interpolation)
- Y neutral angle: 0.8 (24˚ after interpolation
- Phase shift: 0.8 (5.24 after interpolation)

5.8.6 Comparison with Biological Data

Remarkably, the parameters that yielded the highest force in the robotic model closely resemble those observed in real bat flight dynamics, as seen in Figure 5.5 and 5.6. This similarity validates the effectiveness of the evolutionary algorithm and underscores its potential in replicating and understanding biological locomotion mechanisms. A comparison of the values found by the CMA-ES and the real bat data can be seen in table

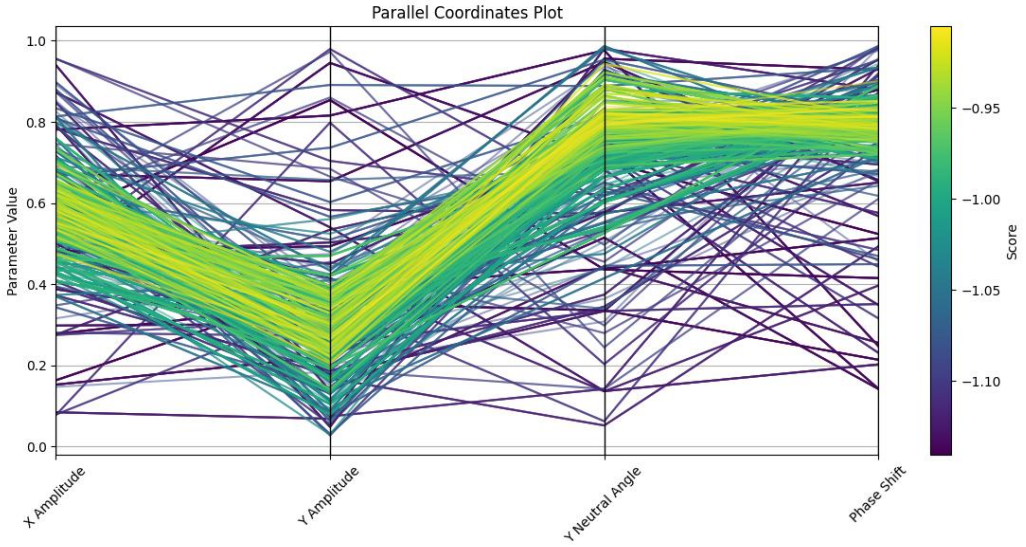


Figure 5.19 Results of made experiments and found optimal parameter.

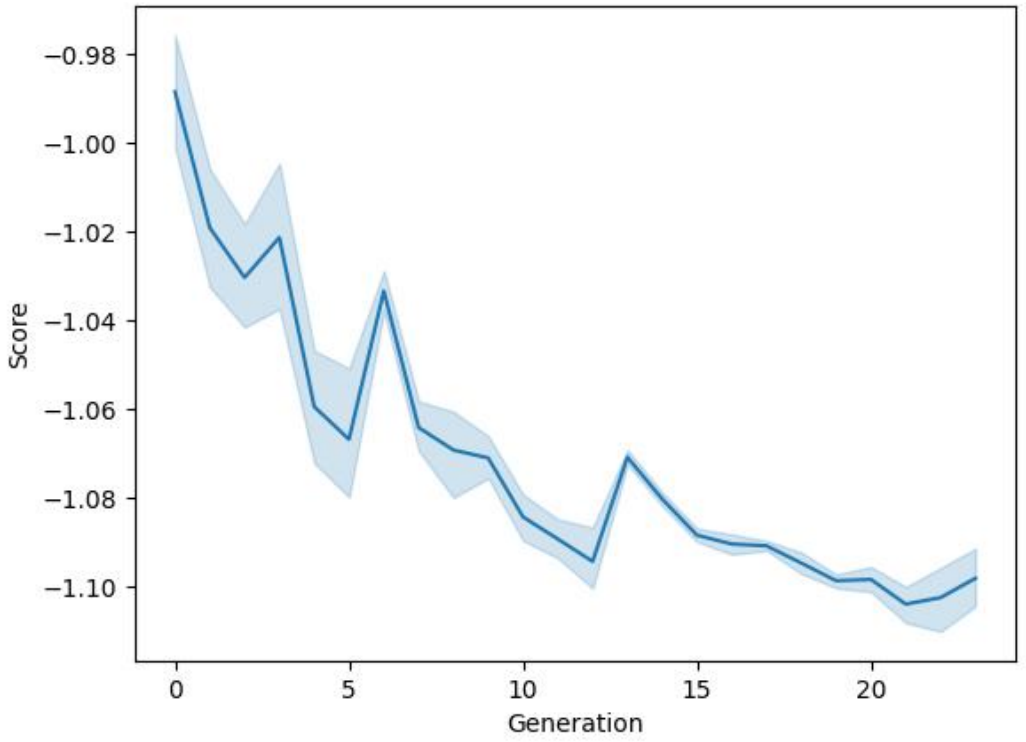


Figure 5.20 Score improvement during each generation.

5.3.

Table 5.3 Comparison of parameters from real bat and CMA-ES

Parameter	**Real Bat Data[42]**	**Optimal Value from CMA-ES**
x amplitude	47°	49.6°
y amplitude	23.5°	20.3°
phase shift	5.1	5.24

5.8.7 Conclusion

The optimization experiments have not only demonstrated the potential of evolutionary algorithms in fine-tuning bio-inspired robotic systems but have also provided new insights into the functional role of hind leg movements in bat flight. The alignment of optimized parameters with those of real bats suggests that the bio-inspired approach to robot design can lead to more accurate replication of natural flight mechanics, offering opportunities for further research and application in robotics.

The series of experiments conducted provided significant insights into the complex dynamics of bat flight, particularly the role of hind leg movements. Through the application of bio-inspired robotics and evolutionary algorithms, this research not only enhances our understanding of natural flight mechanisms but also guides the development of more efficient and agile robotic designs.

CHAPTER 6 CONCLUSION AND OUTLOOK

This thesis delves into the potential of evolutionary algorithms, particularly the Covariance Matrix Adaptation Evolution Strategy (CMA-ES), to enhance robotic controller strategies in dynamic and complex environments where traditional simulations may lack precision or efficiency. Focused on the practical application and training of robots, this research highlights the development of a robot capable of mimicking natural bat leg movements, which demonstrated remarkable resemblance to real bat movements due to the optimization of hind leg controllers using the CMA-ES.

The use of evolutionary algorithms in this thesis shows substantial promise for advancing robotic control strategies. By engaging these algorithms in real-world training scenarios, the limitations inherent in simulations are circumvented, thus providing a truer representation of dynamic interactions between robots and their environments. In particular, the CMA-ES was pivotal not only for optimizing flight control parameters but also for refining the mechanical design of the Batbot, leading to the significant achievement of a mechanically capable Batbot designed to hover.

Moreover, the development of both feedforward and feedback controllers has been integral to the iterative mechanical design improvement process. The experiments conducted illuminated not only the learning capabilities enabled by the CMA algorithm but also the mechanical and physical limitations that required subsequent refinements.

Despite the challenges, the research revealed meaningful progress toward enabling the Batbot to learn optimal controlling strategies, highlighting the necessity for robust robot design due to the wear and tear experienced during repeated experiments. This foundational work paves the way for future investigations into more sophisticated robotic tasks, leveraging the enhanced mechanical and electronic capabilities of the latest Batbot design.

The thesis sets a robust framework for the future improvement of controlling algorithms for the Batbot, building upon the new design and optimization strategies developed. Future research could extend the methodologies and systems established here to other complex robotic behaviors, harnessing the adaptability and effectiveness of the CMA algorithm. This not only validates the use of optimization algorithms in robotic development but also emphasizes the importance of a systematic approach to identifying and rectifying

design limitations.

In conclusion, this work has significantly advanced our understanding of the application of evolutionary algorithms in robotic control, providing a solid foundation for the ongoing evolution of robotic capabilities. With the groundwork laid by this thesis, future endeavors can look to further push the boundaries of what robotic systems can achieve, promising a future of more versatile, efficient, and capable robotic systems.

REFERENCES

[1] Slowik A, Kwasnicka H. Evolutionary algorithms and their applications to engineering problems[J/OL]. Neural Computing and Applications, 2020, 32(16): 12363-12379. https://doi.org/10.1007/s00521-020-04832-8.

[2] Grefenstette J J. Evolutionary algorithms in robotics[C]//1994.

[3] Seng T L, Bin Khalid M, Yusof R. Tuning of a neuro-fuzzy controller by genetic algorithm [J/OL]. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 1999, 29 (2): 226-236. DOI: 10.1109/3477.752795.

[4] Gundogdu O. Optimal-tuning of pid controller gains using genetic algorithms[J]. Journal of Engineering Sciences, 2005, 11: 131-135.

[5] Fan L, Joo E M. Design for auto-tuning pid controller based on genetic algorithms[C/OL]// 2009 4th IEEE Conference on Industrial Electronics and Applications. 2009: 1924-1928. DOI: 10.1109/ICIEA.2009.5138538.

[6] Hoffmann F. Evolutionary algorithms for fuzzy control system design[J/OL]. Proceedings of the IEEE, 2001, 89(9): 1318-1333. DOI: 10.1109/5.949487.

[7] Kwok D, Sheng F. Genetic algorithm and simulated annealing for optimal robot arm pid control [C/OL]//Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence. 1994: 707-713 vol.2. DOI: 10.1109/ICEC.1994.349971.

[8] Vikhar P A. Evolutionary algorithms: A critical review and its future prospects[C/OL]//2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC). 2016: 261-265. DOI: 10.1109/ICGTSPICC.2016.7955308.

[9] Back T, Emmerich M, Shir O. Evolutionary algorithms for real world applications [application notes][J/OL]. IEEE Computational Intelligence Magazine, 2008, 3(1): 64-67. DOI: 10.1109/MCI.2007.913378.

[10] Narvydas G, Simutis R, Raudonis V. Autonomous mobile robot control using fuzzy logic and genetic algorithm[C/OL]//2007 4th IEEE Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications. 2007: 460-464. DOI: 10.1109/IDAACS.2007.4488460.

[11] Neenu T, Poongodi P. Position control of dc motor using genetic algorithm based pid controller [J]. Lecture Notes in Engineering and Computer Science, 2009, 2177.

[12] Sheroz K, Salami A, Adetunji L, et al. Design and implementation of an optimal fuzzy logic controller using genetic algorithm[J/OL]. Journal of Computer Science, 2008, 4. DOI: 10.3844/jcssp.2008.799.806.

[13] Bagis A. Determination of the pid controller parameters by modified genetic algorithm for improved performance.[J]. J. Inf. Sci. Eng., 2007, 23: 1469-1480.

[14] Iruthayarajan M W, Baskar S. Evolutionary algorithms based design of multivariable pid controller[J/OL]. Expert Systems with Applications, 2009, 36(5): 9159-9167. https://www.sciencedirect.com/science/article/pii/S0957417408008920. DOI: https://doi.org/10.1016/j.eswa.2008.12.033.

[15] Kim S H, Park C, Harashima F. A self-organized fuzzy controller for wheeled mobile robot using an evolutionary algorithm[J/OL]. IEEE Transactions on Industrial Electronics, 2001, 48 (2): 467-474. DOI: 10.1109/41.915427.

[16] Schultz A. Learning robot behaviors using genetic algorithms[Z]. 1999.

[17] da Silva Assis L, da Silva Soares A, Coelho C J, et al. An evolutionary algorithm for autonomous robot navigation[J/OL]. Procedia Computer Science, 2016, 80: 2261-2265. https://www.sciencedirect.com/science/article/pii/S1877050916308808. DOI: https://doi.org/10.1016/j.procs.2016.05.404.

[18] Willjuice Iruthayarajan M, Baskar S. Covariance matrix adaptation evolution strategy based design of centralized pid controller[J/OL]. Expert Systems with Applications, 2010, 37(8): 5775-5781. https://www.sciencedirect.com/science/article/pii/S0957417410000709. DOI: https://doi.org/10.1016/j.eswa.2010.02.031.

[19] M. Mucientes A B S B, D.L. Moreno. Design of a fuzzy controller in mobile robotics using genetic algorithms[J]. Applied Soft Computing, 2007, 7: 540–546.

[20] Lu W, Yang J, Liu X. The pid controller based on the artificial neural network and the differential evolution algorithm[J/OL]. Journal of Computers, 2012, 7. DOI: 10.4304/jcp.7.10.2368-2375.

[21] Karlra P, Prakash N. A neuro-genetic algorithm approach for solving the inverse kinematics of robotic manipulators[C/OL]//SMC'03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme - System Security and Assurance (Cat. No.03CH37483): volume 2. 2003: 1979-1984 vol.2. DOI: 10.1109/ICSMC.2003.1244702.

[22] Corne D, Lones M A. Evolutionary algorithms[M/OL]//Handbook of Heuristics. Springer International Publishing, 2018: 1-22. https://doi.org/10.1007%2F978-3-319-07153-4_27-1. DOI: 10.1007/978-3-319-07153-4_27-1.

[23] Yoshida A, Kanagawa S, Wakasa Y, et al. Pid controller tuning based on the covariance matrix adaptation evolution strategy[C]//2009 ICCAS-SICE. 2009: 2982-2986.

[24] Sivananaithaperumal S, Subramanian B. Design of multivariable fractional order pid controller using covariance matrix adaptation evolution strategy[J/OL]. Archives of Control Sciences, 2014, 24. DOI: 10.2478/acsc-2014-0014.

[25] Sheikhan M, Ghoreishi S A. Application of covariance matrix adaptation–evolution strategy to optimal control of hepatitis b infection[J/OL]. Neural Computing and Applications, 2013, 23 (3): 881-894. https://doi.org/10.1007/s00521-012-1013-3.

[26] Pedersen G K, Butz M V. Evolving robust controller parameters using covariance matrix adaptation[C/OL]//GECCO '10: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation. New York, NY, USA: Association for Computing Machinery, 2010: 1251–1258. https://doi.org/10.1145/1830483.1830708.

[27] Zhu H, Yu J, Gupta A, et al. The ingredients of real-world robotic reinforcement learning[A]. 2020. arXiv: 2004.12570.

[28] Asafuddoula M, Ray T, Sarker R. An adaptive differential evolution algorithm and its performance on real world optimization problems[C/OL]//2011 IEEE Congress of Evolutionary Computation (CEC). 2011: 1057-1062. DOI: 10.1109/CEC.2011.5949734.

[29] Pascalis Trentsiosa D G, Mario Wolf*a. Overcoming the sim-to-real gap in autonomous robots [J]. Procedia CIRP, 2022, 109: 540–546.

[30] Wu P, Escontrela A, Hafner D, et al. Daydreamer: World models for physical robot learning [A]. 2022. arXiv: 2206.14176.

[31] Guo S, Lin J, Wöhrl T, et al. A neuro-musculo-skeletal model for insects with data-driven optimization[J/OL]. Scientific Reports, 2018, 8(1): 2129. https://doi.org/10.1038/s41598-018-20093-x.

[32] Bahlman J W, Swartz S M, Breuer K S. Design and characterization of a multi-articulated robotic bat wing[J/OL]. Bioinspiration Biomimetics, 2013, 8(1): 016009. https://dx.doi.org/10.1088/1748-3182/8/1/016009.

[33] Ghanbari A, Mottaghi E, Qaredaghi E. A new model of bio-inspired bat robot[C/OL]//2013 First RSI/ISM International Conference on Robotics and Mechatronics (ICRoM). 2013: 403-406. DOI: 10.1109/ICRoM.2013.6510141.

[34] Hoff J, Syed U, Ramezani A, et al. Trajectory planning for a bat-like flapping wing robot[C/OL]//2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2019: 6800-6805. DOI: 10.1109/IROS40897.2019.8968450.

[35] Woodward M A, Sitti M. Multimo-bat: A biologically inspired integrated jumping–gliding robot[J/OL]. The International Journal of Robotics Research, 2014, 33(12): 1511-1529. https://doi.org/10.1177/0278364914541301.

[36] Ramezani A, Shi X, Chung S J, et al. Lagrangian modeling and flight control of articulated-winged bat robot[C/OL]//2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2015: 2867-2874. DOI: 10.1109/IROS.2015.7353772.

[37] Ramezani A, Shi X, Chung S J, et al. Bat bot (b2), a biologically inspired flying machine[C/OL]//2016 IEEE International Conference on Robotics and Automation (ICRA). 2016: 3219-3226. DOI: 10.1109/ICRA.2016.7487491.

[38] Sui T, Zou T, Riskin D. Optimum design of a novel bio-inspired bat robot[J/OL]. IEEE Robotics and Automation Letters, 2022, 7(2): 3419-3426. DOI: 10.1109/LRA.2022.3146536.

[39] Hedenström A, Johansson L C. Bat flight: aerodynamics, kinematics and flight morphology [J/OL]. Journal of Experimental Biology, 2015, 218(5): 653-663. https://doi.org/10.1242/jeb.031203.

[40] Hansen N. The cma evolution strategy: A tutorial[A]. 2023. arXiv: 1604.00772.

[41] Karl Johan Astrom R M M. Feedback systems, an introduction for scientists and engineers[M]. v2.11b ed. n the United Kingdom: Princeton University Press 6 Oxford Street, Woodstock, Oxfordshire OX20 1TW: Princeton Univeristy Press, 2012.

[42] Singh S K. Kinematic analysis of a bat flight based on membrane wing motion reconstruction [D]. Beijing, China: Tsinghua University, 2022.

[43] Alibaba. Product description of the jy90x[EB/OL]. [2024-04-15]. https://www.alibaba.com/product-detail/JY901B-9-Axis-Accelerometer-Gyroscope-MPU9250_1600437996406.html.

APPENDIX A SUPPLEMENTARY CONTENT

A.1 Figures

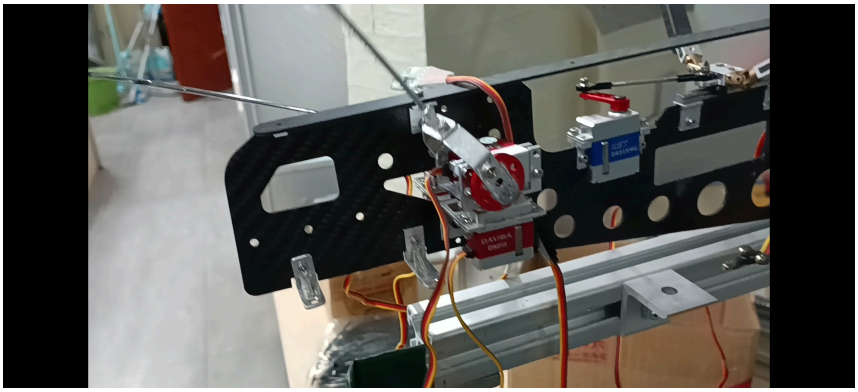


Figure A.1 2 DOF leg installation

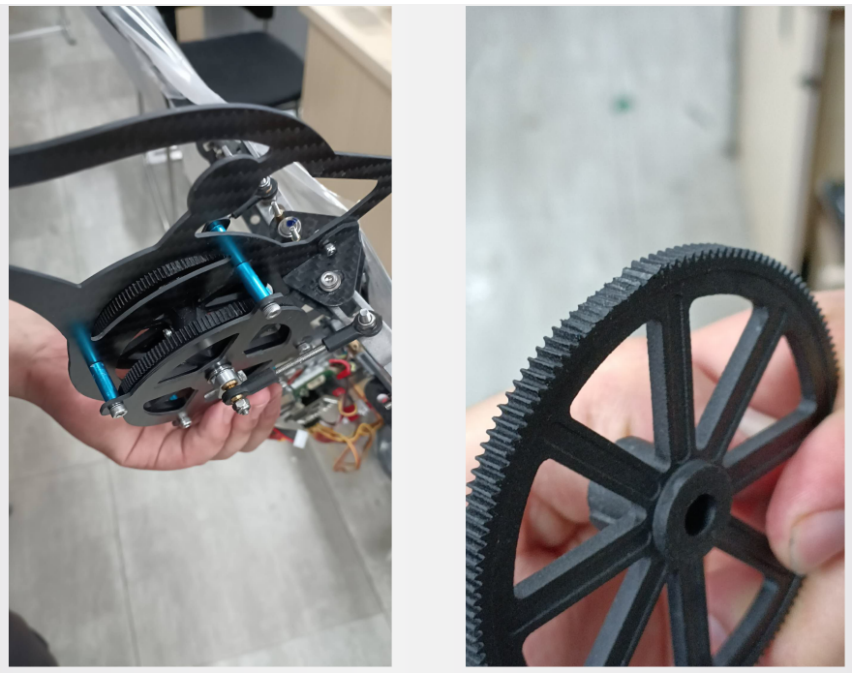


Figure A.2 Gear failure example

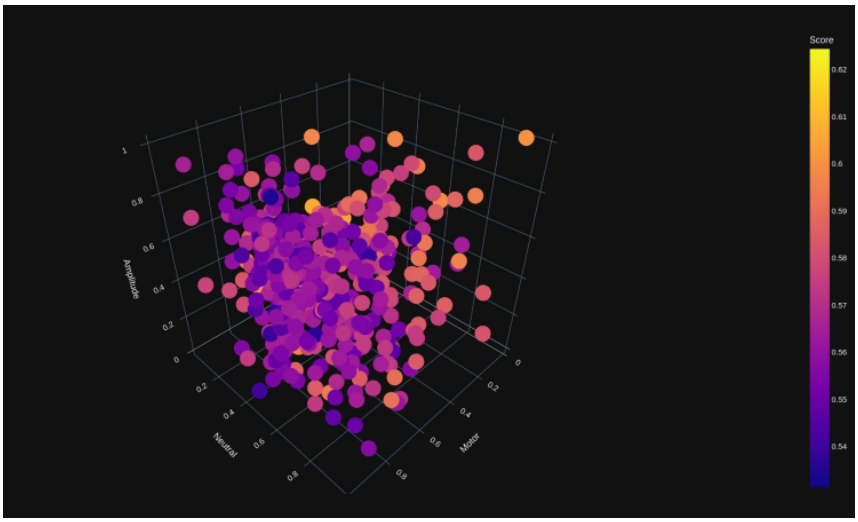


Figure A.3 Optimization of motor, angle and amplitude, with clear convergence

Figure A.4 Magnet installation in shoulder for wing angle measurement

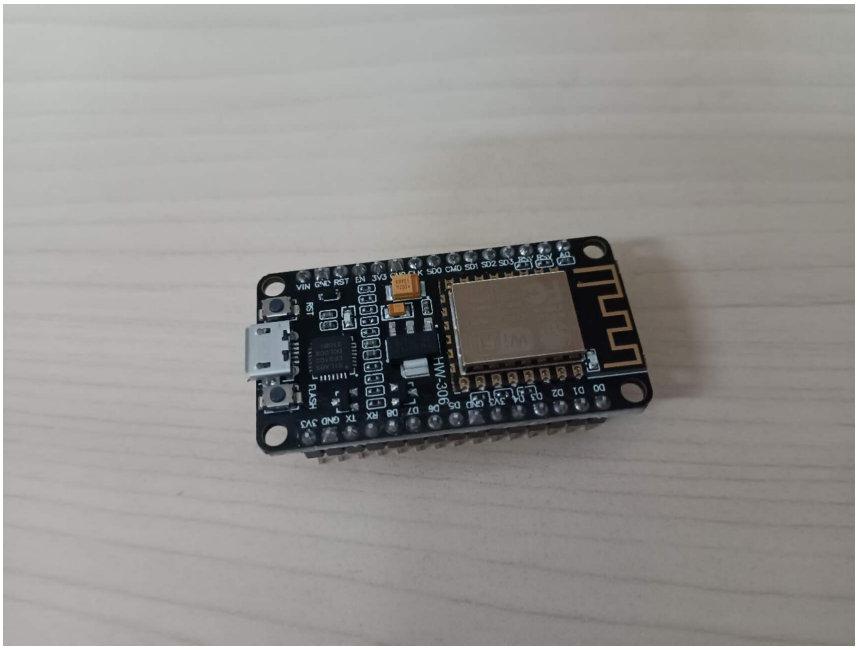


Figure A.5 NODEMcu

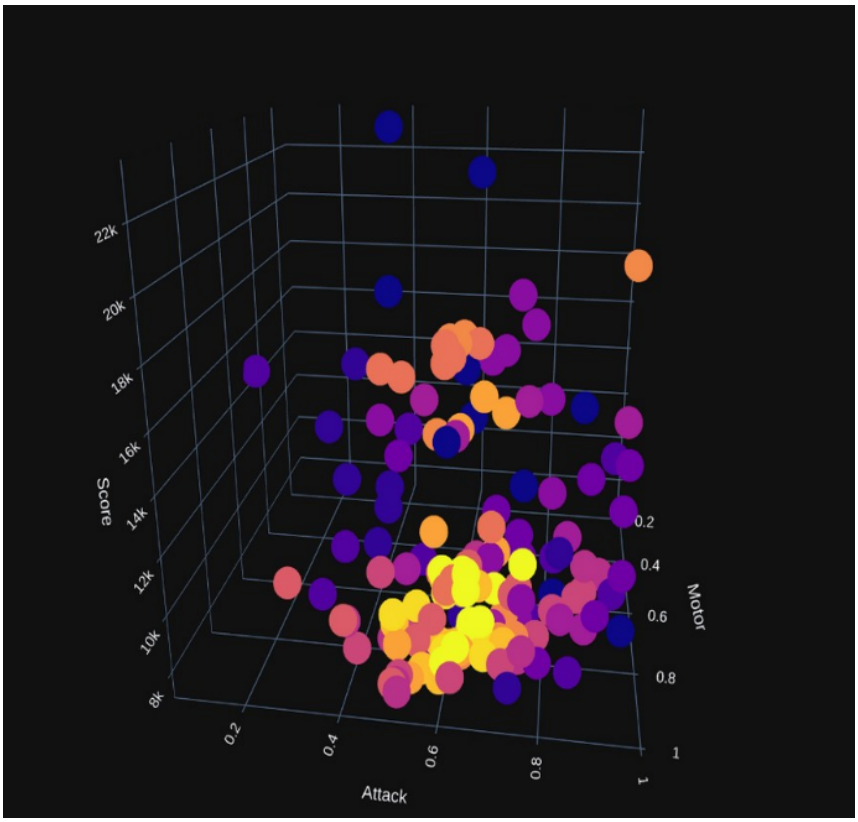


Figure A.6 Example of influence of noisy data in optimization strategy

Figure A.7 Stm32 initial tests

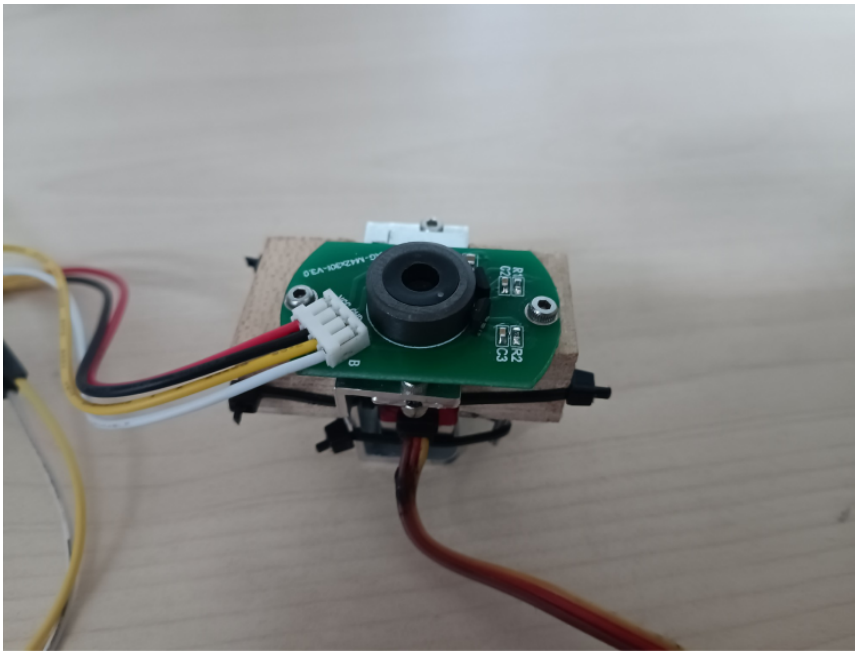


Figure A.8 Servo rotation velocity measurement

A.2 Code

Listing A.1 Python code for CMA-ES

```
import matplotlib.pyplot as plt

from utils import *
import numpy as np
from cmaes import CMA
import pickle
import seaborn as sns
import plotly.express as px
from pandas.plotting import parallel_coordinates

# Connection details
daq_port = "/dev/ttyUSB2" # Find port using !python -m
   ↪ serial.tools.list_ports
command_port = "/dev/ttyACM0"

# Sensor calibration
calib = pd.read_csv('/home/anuarsantoyo/
   ↪ PycharmProjects/Batbot/analysis/forces/231116/
   ↪ data/calib_corrected.csv', index_col=0).iloc[:,0]


# Start the CMA optimizer # TODO: double check
pop_size = 10
n_generation = 20
save_directory = "experiments/optimizer_batbotV2
   ↪ /231118/test1/" # TODO: dir
load = False
if load:
    file = open(save_directory+'optimizers/optimizer_17.
       ↪ pickle', 'rb')
    loaded_file = pickle.load(file)
```

```
    optimizer = loaded_file['optimizer']
    generation_0 = loaded_file['last_generation'] + 1
    file.close()
    results = pd.read_csv(save_directory+'results.csv')
else:
    optimizer = CMA(mean=np.array([0.5 for i in range(5)
      ↪ ]), # TODO: dim
                sigma=0.5,
                population_size=pop_size,
                bounds=np.array([[0, 1] for i in range
                  ↪ (5)])) # TODO: dim
    results = pd.DataFrame(columns=['Generation',
                                    'Id',
                                    'Score',
                                    'Leg␣x',
                                    'Leg␣y',
                                    'Amplitude␣x',
                                    'Amplitude␣y',
                                    'Ellipse']) # TODO: dim
    generation_0 = 0

# df to plot scores
scores_plot = []
df_dict_list = []


# Loop for all generations
for generation in range(generation_0, generation_0+
  ↪ n_generation):
    print("\n")
    print("*"*50)
    print(f"Generation␣{generation+1}/{generation_0+
      ↪ n_generation}")
```

```
print("*"*50)
print("\n")

solutions = []
for i in range(optimizer.population_size):
    print(f"Test: {i+1}/{pop_size}")
    x = optimizer.ask()
    # motor, leg_x, leg_y, leg_x_amplitude,
    ↪ leg_y_amplitude, ellipse_angle = x
    leg_x, leg_y, leg_x_amplitude, leg_y_amplitude,
    ↪ ellipse_angle = x # TODO: dim
    motor = 1

    cmd = (motor, leg_x, leg_y, leg_x_amplitude,
    ↪ leg_y_amplitude, ellipse_angle)

    command_batbot_V2(cmd, command_port)
    time.sleep(1) # To allow the Batbot to reach the
    ↪  attack angle and flapping speed
    measurements = read_measurements_df_6axis(port=
    ↪ daq_port, duration=5) - calib
    score = fitness_avg_force(measurements, plot=
    ↪ True)
    print(f"Result: {score}")
    print("-" * 10)
    print("\n")
    time.sleep(1)

    measurements.to_csv(save_directory + f"
    ↪ measurements/{generation}_{i}({round(score
    ↪ ,2)}).csv", index=False)
    solutions.append((x, score))
```

```
    df_dict_list.append({'Generation': generation,
                         'Id': i,
                         'Score': score,
                         'Leg␣x': leg_x,
                         'Leg␣y': leg_y,
                         'Amplitude␣x': leg_x_amplitude,
                         'Amplitude␣y': leg_y_amplitude,
                         'Ellipse': ellipse_angle}) #
                           ↪ TODO:dim

optimizer.tell(solutions)
df = pd.DataFrame(df_dict_list)
results = pd.concat([results, df], ignore_index=
  ↪ True)
results.to_csv(save_directory + "results.csv",
  ↪ index=False)
with open(save_directory+f"optimizers/optimizer_{
  ↪ generation}.pickle", "wb") as file:
   pickle.dump({'optimizer': optimizer, '
     ↪ last_generation': generation}, file)

# Visualization
variables = results.drop(['Generation', 'Id', '
  ↪ Score'], axis=1).columns

fig, (ax1, ax2) = plt.subplots(1, 2, gridspec_kw={'
  ↪ width_ratios': [1, 0.1]}, figsize=(10, 6))
# Plot the covariance matrix on ax1
sns.heatmap(optimizer._C, annot=True, cmap='
  ↪ coolwarm', cbar=True, square=True,
        xticklabels=variables, yticklabels=
          ↪ variables, ax=ax1)
# Plot the mean values on ax2
```

```
sns.heatmap(optimizer._mean[:, np.newaxis], annot=
    ↪ True, cmap='vlag', cbar=False, yticklabels=
    ↪ variables, ax=ax2)
ax2.set_title('Mean')
ax2.set_xlabel('')
ax2.set_yticks([]) # Hide the y-axis ticks
plt.tight_layout()
plt.show()

sns.lineplot(data=results, x="Generation", y="Score
    ↪ ")
plt.savefig(save_directory + "score.jpg")
plt.show()

plt.figure(figsize=(12, 6))
parallel_coordinates(results.drop(['Generation', '
    ↪ Id'], axis=1), class_column='Score', colormap=
    ↪ 'viridis', alpha=0.5)
plt.legend().set_visible(False)
plt.title('Parallel Coordinates Plot')
plt.ylabel('Parameter Value')
plt.xticks(rotation=45)
plt.tight_layout() # Ensures that all elements of
    ↪ the plot fit well
# Add color bar
sm = plt.cm.ScalarMappable(cmap='viridis',
                           norm=plt.Normalize(vmin=results
                               ↪ ['Score'].min(), vmax=
                               ↪ results['Score'].max()))
sm.set_array([]) # You need to set the array for
    ↪ the scalar mappable even if not used
plt.colorbar(sm, label='Score')
```

```
    plt.savefig(save_directory + "results.jpg")
    plt.show()

# Dummy data creation for demonstration purposes
# Assuming 'results' is a DataFrame with a 'Score'
    ↪ column which we want to use as class column
np.random.seed(0)
results = pd.DataFrame({
    'Parameter A': np.random.rand(10),
    'Parameter B': np.random.rand(10),
    'Parameter C': np.random.rand(10),
    'Score': np.random.randint(1, 100, 10)
})

# Plotting parallel coordinates with a colormap and
    ↪ adding a colorbar
plt.figure(figsize=(12, 6))

# Create the parallel coordinates plot
parallel_coordinates(results, class_column='Score',
    ↪ colormap='viridis', alpha=0.5)

# Hide the legend if not needed
plt.legend().set_visible(False)

# Set the title and labels
plt.title('Parallel Coordinates Plot')
plt.ylabel('Parameter Value')
plt.xticks(rotation=45)

# Ensure that all elements of the plot fit well within
    ↪  the figure
plt.tight_layout()
```

```
# Add color bar
sm = plt.cm.ScalarMappable(cmap='viridis', norm=plt.
   ↪ Normalize(vmin=results['Score'].min(), vmax=
   ↪ results['Score'].max()))
sm.set_array([]) # You need to set the array for the
   ↪ scalar mappable even if not used
plt.colorbar(sm, label='Score')

# Show the plot
plt.show()
```

Listing A.2 Python code for utils.py

```
import matplotlib.pyplot as plt
import serial
import numpy as np
import urllib.request
from matplotlib.patches import Ellipse
import struct
from scipy.signal import savgol_filter, find_peaks
import minimalmodbus
import time
import pandas as pd
from scipy.interpolate import interp1d
import math

sensors_col = ['Fx', 'Fy', 'Fz', 'Mx', 'My', 'Mz']
#calib = pd.read_csv('/home/anuarsantoyo/
   ↪ PycharmProjects/Batbot/analysis/forces/231115/
   ↪ data/df_calib.csv').mean()
#calib['Time'] = 0


def command_batbot_V2(command, port):
```

```
    """
    This function takes a command and sends it to the
      ↪ Batbot.
    :param solution: list of commands [motor,
      ↪ attack_angle, neutral_state, amplitude]
    :param port: port of the wireless uart module.
    :return: None
    """
    motor, leg_x, leg_y, leg_x_amplitude,
      ↪ leg_y_amplitude, ellipse_angle = command
    motor = np.interp(motor, [0, 1], [260, 270])
    leg_x = np.interp(leg_x, [0, 1], [40, 120])
    leg_y = np.interp(leg_y, [0, 1], [30, 150])
    leg_x_amplitude = np.interp(leg_x_amplitude, [0,
      ↪ 1], [0, 40])
    leg_y_amplitude = np.interp(leg_y_amplitude, [0,
      ↪ 1], [0, 60])
    print(f"Sending␣command␣to␣batbot␣"
          f"leg_x:␣{leg_x}\n"
          f"leg_y:␣{leg_y}\n"
          f"leg_x_amplitude:␣{leg_x_amplitude}\n"
          f"leg_y_amplitude:␣{leg_y_amplitude}\n"
          f"Ellipse:␣{ellipse_angle}")
    ser = serial.Serial(port=port, baudrate=115200,
      ↪ bytesize=8, parity='N', stopbits=1)
    ser.write(str.encode(f'{motor},{leg_x},{leg_y},{
      ↪ leg_x_amplitude},{leg_y_amplitude},{
      ↪ ellipse_angle}'))
    print("Sent!\n")

def command_batbot_mimic(command, port):
    """
    This function takes a command and sends it to the
```

```
        ↪ Batbot.
    :param solution: list of commands [motor,
        ↪ attack_angle, neutral_state, amplitude]
    :param port: port of the wireless uart module.
    :return: None
    """
    x_amp, y_amp, y_mid, delay = command
    x_amp = np.interp(x_amp, [0, 1], [0, 90])
    y_amp = np.interp(y_amp, [0, 1], [0, 60])
    y_mid = np.interp(y_mid, [0, 1], [0, 30])
    delay = np.interp(delay, [0, 1], [0, 2*np.pi])
    print(f"Sending␣command␣to␣batbot␣"
          f"x_amp:␣{x_amp}\n"
          f"y_amp:␣{y_amp}\n"
          f"y_mid:␣{y_mid}\n"
          f"delay:␣{delay}")
    ser = serial.Serial(port=port, baudrate=115200,
        ↪ bytesize=8, parity='N', stopbits=1)
    ser.write(str.encode(f'{x_amp},{y_amp},{y_mid},{
        ↪ delay}'))
    print("Sent!\n")

def fitness_mse(measurements):
    """
    Calculates the fitness as the MSE, with the idea of
        ↪  trying to reduce to zero the sensors instead
        ↪ of just it's mean.
    The sensor 1 which has shown to have the largest
        ↪ values is just to slice the df from the first
        ↪ peak to the last
    valley.
    :param measurements: df of data provided by
        ↪ read_measurements_df()
```

```
    :return: score
    """
    y = measurements.sensor_1
    peaks, _ = find_peaks(y, height=0)
    df_sliced = measurements.iloc[peaks[0]: peaks[-1]]
    return (df_sliced.drop('timestamp', axis=1) ** 2).
      ↪ mean().sum()

def peak_slice_interpolate(measurements, args = {'
  ↪ peak_height':0, 'peak_distance':10}):
    """
    Interpolates sensor data between the first and last
      ↪  detected peaks and c
    :param measurements: (pandas.DataFrame) A DataFrame
      ↪  containing sensor data.
    :return: Interpolated data after slicing from first
      ↪  to last peak
    """
    df = measurements.copy()
    # Find peaks (adjust parameters as discussed
      ↪ previously)
    peaks, _ = find_peaks(df['Fy'], height=args['
      ↪ peak_height'], distance=args['peak_distance'])
    df_sliced = df.iloc[peaks[0]:peaks[-1] + 1].
      ↪ reset_index(drop=True)
    # Now create the new time series with 1000 points
    new_time = np.linspace(df_sliced['Time'].min(),
      ↪ df_sliced['Time'].max(), 1000)
    # Initialize a new DataFrame to store the
      ↪ interpolated values
    interpolated_df = pd.DataFrame(index=new_time)

    # Interpolate each sensor column
```

```
    for column in df_sliced.columns:
        if column != 'Time':
            # Create the interpolation function
            interp_func = interp1d(df_sliced['Time'],
                ↪ df_sliced[column], kind='linear',
                ↪ fill_value='extrapolate')
            # Apply the interpolation function to the new
                ↪  time series
            interpolated_df[column] = interp_func(
                ↪ new_time)

    # Reset the index to make 'Time' a column again
    interpolated_df.reset_index(inplace=True)
    interpolated_df.rename(columns={'index': 'Time'},
        ↪ inplace=True)
    return interpolated_df, peaks

def fitness_avg_force(measurements, plot=False, args =
    ↪  {'peak_height':0, 'peak_distance':10}):
    """
    Interpolates sensor data between the first and last
        ↪  detected peaks and calculates the mean
    of the interpolated 'Fy' and 'Fz' columns to
        ↪ compute a score as the Euclidean norm.

    :param measurements: (pandas.DataFrame) A DataFrame
        ↪  containing sensor data with at least 'Time',
                                'Fy', and 'Fz' columns.

    :return: score: (float) The Euclidean norm of the
        ↪ mean values of 'Fy' and 'Fz' from the
                    interpolated data.
    """
```

```
df = measurements.copy()
interpolated_df, peaks = peak_slice_interpolate(df,
    ↪ args)

# interpolated_df now contains 1000 interpolated
    ↪ data points based on the 'Time' column.

Fy, Fz = interpolated_df.mean()[['Fy', 'Fz']]
score = np.sqrt(Fy ** 2 + (Fz - 5.95) ** 2)

if plot:
    # Create the base line plot
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize
        ↪ =(10, 4))
    # Plot 'Fz' column
    ax1.plot(df['Time'], df['Fz'], label='Fz')
    ax1.plot(df['Time'], df['Fy'], label='Fy',
        ↪ zorder=1)
    # Plot the peaks
    ax1.scatter(df['Time'][peaks], df['Fy'][peaks],
        ↪ color='red', s=10, label='Peaks', zorder=5)
    # Adding title and labels
    ax1.legend()
    ax1.set_title('Peaks␣found')
    ax1.set_xlabel('Time')
    ax1.set_ylabel('Force')
    # Setting up a simple plot with equal scaling on
        ↪ the axes
    # Setting the aspect of the plot to be equal.
    ax2.set_aspect('equal', adjustable='box')

    # Drawing a simple line for demonstration
    ax2.plot(interpolated_df.Fy, interpolated_df.Fz,
```

```
            ↪ zorder=1)
        ax2.scatter(Fy, Fz, c='r', marker='*')
        ax2.scatter(0, 0, c='g', marker='+')
        ax2.arrow(0, 0, Fy, Fz, head_width=0.3,
            ↪ head_length=0.3)

        # Setting labels for the axes
        ax2.set_xlabel('Fy')
        ax2.set_ylabel('Fz')
        ax2.set_title(f'Score:␣{round(score,␣2)}')

        # Display the plot
        plt.show()

    return score

def fitness_avg_biomimic(measurements, plot=False,
    ↪ args = {'peak_height':0, 'peak_distance':10}):
    """
    Interpolates sensor data between the first and last
        ↪ detected peaks and calculates the mean
    of the interpolated 'Fy' and 'Fz' columns to
        ↪ compute a score as the Euclidean norm.

    :param measurements: (pandas.DataFrame) A DataFrame
        ↪ containing sensor data with at least 'Time',
                                'Fy', and 'Fz' columns.

    :return: score: (float) The Euclidean norm of the
        ↪ mean values of 'Fy' and 'Fz' from the
                    interpolated data.
    """
    df = measurements.copy()
```

```
interpolated_df, peaks = peak_slice_interpolate(df,
    ↪ args)
peak_times = df.loc[peaks, 'Time']
periods_mean = peak_times.diff().dropna().mean()
average_frequencies = 1 / periods_mean
# interpolated_df now contains 1000 interpolated
    ↪ data points based on the 'Time' column.

Fy, Fz = interpolated_df.mean()[['Fy', 'Fz']]
force = np.sqrt(Fy ** 2 + Fz ** 2)
angle = np.degrees(np.arctan(Fz/Fy))

if plot:
    # Create the base line plot
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize
        ↪ =(10, 4))
    # Plot 'Fz' column
    ax1.plot(df['Time'], df['Fz'], label='Fz')
    ax1.plot(df['Time'], df['Fy'], label='Fy',
        ↪ zorder=1)
    # Plot the peaks
    ax1.scatter(df['Time'][peaks], df['Fy'][peaks],
        ↪ color='red', s=10, label='Peaks', zorder=5)
    # Adding title and labels
    ax1.legend()
    ax1.set_title('Peaks␣found')
    ax1.set_xlabel('Time')
    ax1.set_ylabel('Force')
    # Setting up a simple plot with equal scaling on
        ↪ the axes
    # Setting the aspect of the plot to be equal.
    ax2.set_aspect('equal', adjustable='box')
```

```
        # Drawing a simple line for demonstration
        ax2.plot(interpolated_df.Fy, interpolated_df.Fz,
            ↪ zorder=1)
        ax2.scatter(Fy, Fz, c='r', marker='*')
        ax2.scatter(0, 0, c='g', marker='+')
        ax2.arrow(0, 0, Fy, Fz, head_width=0.3,
            ↪ head_length=0.3)

        # Setting labels for the axes
        ax2.set_xlabel('Fy')
        ax2.set_ylabel('Fz')
        ax2.set_title(f'Force:␣{round(force,␣2)},␣Angle:␣
            ↪ {round(angle,␣2)}')

        # Display the plot
        plt.show()
    #return force
    return Fy, Fz, force, angle, average_frequencies


def twos_complement(value, bits):
    """Compute the 2's complement of int value."""
    if value & (1 << (bits - 1)):
        value -= 1 << bits
    return value


def read_measurements_df_6axis(port='/dev/ttyUSB0',
    ↪ duration=10, calibration=False):
    '''
    This function reads the measurements from a 6-axis
        ↪ force/torque sensor connected via Modbus RTU.
    It collects data for a specified duration and
```

```
    ↪ returns a pandas DataFrame with the
    ↪ measurements.

The DataFrame includes time-series data for force
    ↪ and torque measurements along and around
the X, Y, and Z axes. If a calibration setting is
    ↪ applied, the function can also return
    ↪ calibrated data.

:param port: str, optional
    The serial port to which the sensor is connected
        ↪ . Default is '/dev/ttyUSB0'.
:param duration: int or float, optional
    The duration in seconds for which to collect
        ↪ data from the sensor. Default is 10 seconds
        ↪ .
:param calibration: bool

:return: pandas.DataFrame
    A DataFrame containing the time-series data for
        ↪ the 6-axis measurements. Each force
        ↪ measurement
    is in Newtons and each torque measurement is in
        ↪ Newton-meters, with appropriate scaling
        ↪ applied.

The returned DataFrame has the following columns:
- 'Time': The time points of the measurement in
    ↪ seconds.
- 'Fxyz': The force measurement along the XYZ-axis
    ↪ in Newtons.
- 'Mxyz': The torque measurement around the XYZ-
    ↪ axis in Newton-meters.
```

```
'''

# Setting up the Modbus RTU connection
instrument = minimalmodbus.Instrument(port, 1)
instrument.serial.baudrate = 115200
instrument.serial.parity = minimalmodbus.serial.
   ↪ PARITY_NONE
instrument.mode = minimalmodbus.MODE_RTU

# Store measurements for each of the 6 sensors
measurements = [[] for _ in range(6)]
time_points = []

end_time = time.time() + duration # Measure for
   ↪ duration time

while time.time() < end_time:
    # Reading 12 registers starting from 2560 (0
       ↪ x0A00 in hex), which equals 24 bytes
    response = instrument.read_registers(2560, 12,
       ↪ functioncode=3)

    # Extracting the 6 quantities from the response
       ↪ and convert using two's complement
    quantities = [twos_complement((response[i] <<
       ↪ 16) | response[i + 1], 32) for i in range(0,
       ↪  len(response), 2)]

    # Append measurements for each sensor
    for q_values, q in zip(measurements, quantities):
        q_values.append(q)

    time_points.append(time.time() - (end_time -
```

```
            ↪ duration)) # Record the measurement time

        time.sleep(0.003) # Interval of 0.1 seconds

    # Convert data to a DataFrame for easy plotting
        ↪ with Plotly Express
    labels = ['Fx', 'Fy', 'Fz', 'Mx', 'My', 'Mz']
    df = pd.DataFrame({'Time': time_points})
    for label, data in zip(labels, measurements):
        df[label] = data
    df['Mx'] = -df['Mx']
    df['Mz'] = -df['Mz']
    df[['Mx', 'My', 'Mz']] = df[['Mx', 'My', 'Mz']] /
        ↪ 10000
    df[['Fx', 'Fy', 'Fz']] = df[['Fx', 'Fy', 'Fz']] /
        ↪ 100

    if calibration:
        return df#-calib
    else:
        return df

def get_angle_degrees(x, y):
    """
    Calculate the angle from the positive x-axis to the
        ↪ point (x, y) in degrees.

    :param x: x-coordinate of the point
    :param y: y-coordinate of the point
    :return: angle in degrees between 0 and 360
    """
    angle_radians = math.atan2(y, x) # Angle in radians
    angle_degrees = math.degrees(angle_radians) #
```

```
        ↪ Convert to degrees
    if angle_degrees < 0:
        angle_degrees += 360 # Normalize to 0-360 range
            ↪ if negative
    return angle_degrees
```

Listing A.3 MicroPython code for PID controller of Batbot

```
import pyb
import os
import math
from machine import I2C,SoftI2C
import time
import jy901
import as5048a
import pid
import pca9685
import pca9685_servo
import micropython
from micropython import const
import reciever
import servo_angle

micropython.opt_level(3)

#iic接口及iic设备初始化
iic1=SoftI2C(scl='X9',sda='X10',freq=150000)
imu=jy901.JY901(iic1)
pca=pca9685.PCA9685(iic1)
servos=pca9685_servo.Servos(pca)

reciever_UART=pyb.UART(3)
reciever_UART.init(115200)
recv_decoder=reciever.Reciever_decoder()
```

```
recv_channels=[0]*16



RL_x = 0
RL_y = 1
LL_x = 2
LL_y = 3
FLAPPER = 4
legx_open = 90
legx_closed = 80


def linear_interp(x, x0, y0, x1, y1): # if x is x0
   ↪ then give y0 back, if x1 then y1
    y = y0 + (x - x0) * (y1 - y0) / (x1 - x0)
    return max(min(y1,y),y0)

class PID:
    def __init__(self, Kp, Ki, Kd, setpoint,
       ↪ sample_time):
        self.Kp = Kp
        self.Ki = Ki
        self.Kd = Kd
        self.setpoint = setpoint
        self.prev_error = 0
        self.integral = 0
        self.sample_time = sample_time

    def compute(self, input_value):
        # Calculate error
        error = self.setpoint - input_value
```

```
        # Proportional term
        P = self.Kp * error

        # Integral term
        self.integral += error * self.sample_time
        I = self.Ki * self.integral

        # Derivative term
        D = self.Kd * (error - self.prev_error) / self.
            ↪ sample_time

        # Total output
        output = P + I + D

        # Save error for next loop
        self.prev_error = error

        return output


class MovingAverageFilter:
    def __init__(self, window_size):
        self.window_size = window_size
        self.values = []

    def update(self, value):
        if len(self.values) >= self.window_size:
            self.values.pop(0)
        self.values.append(value)
        return sum(self.values) / len(self.values)


# PID stuff
```

```
sample_time = 0.01
k_roll_old = 0
k_pitch_old = 0

#window_filter_x = MovingAverageFilter(10) # Example
   ↪ window size
#window_filter_z = MovingAverageFilter(10) # Example
   ↪ window size
time.sleep(1)

while True:
   time.sleep(0.01)
   recv_nbytes=reciever_UART.any()
   if recv_nbytes!=0:

      #initilaze pid if new k value is found
      k_roll_new = recv_channels[5]
      k_pitch_new = recv_channels[6]

      if k_roll_new != k_roll_old:
         k_roll_old = k_roll_new
         kp = k_roll_old/1342 # Normalize 0-1
         pid_roll = PID(Kp=-1.5*kp, Ki=0, Kd=0,
            ↪ setpoint=-90, sample_time=sample_time)
         print("Changed␣roll␣k:␣", kp)

      if k_pitch_new != k_pitch_old:
         k_pitch_old = k_pitch_new
         kp = k_pitch_old/1342 # Normalize 0-1
         pid_pitch = PID(Kp=-0.75*kp, Ki=0, Kd=0,
            ↪ setpoint=-20, sample_time=sample_time)
         print("Changed␣pitch␣k:␣", kp)
```

```
recv_data=reciever_UART.read(recv_nbytes)
for i in recv_data:
    recv_decoder.passin(i)
recv_channels=recv_decoder.get_channels_data()
use_pid = recv_channels[8] == 0
motor_raw = recv_channels[2] #0-1342 down-up
motor = linear_interp(motor_raw, 0, 230, 1342,
    ↪ 400)
pca.duty(FLAPPER, motor)

pitch_raw = recv_channels[1] #0-671-1342 up-
    ↪ middle-down
roll_raw = recv_channels[3] #0-671-1342 left-
    ↪ middle-rigth

pitch_neutral = linear_interp(pitch_raw, 1342,
    ↪ 60, 0, 120)
roll_neutral = linear_interp(roll_raw, 1342,
    ↪ -30, 0, 30)
#print(pitch_neutral, roll_neutral)

# Read the sensor value
sensor = imu.data_read()
sensor_pitch = sensor['Pth']
sensor_roll = sensor['Rol']
#sensor_pitch = window_filter_x.update(sensor['
    ↪ Pth'])
#sensor_roll = window_filter_z.update(sensor['
    ↪ Rol'])
#print(sensor_mpu['AcZ'])

# Compute PID output
```

```
        pitch = pitch_neutral
        roll = roll_neutral
        if use_pid:
            pitch += int(pid_pitch.compute(sensor_pitch))
            roll += int(pid_roll.compute(sensor_roll))

        # Control the actuator
        y_theta = pitch
        LL_y_angle = (y_theta + roll)
        RL_y_angle = 165 - (y_theta - roll)

        x_theta = legx_closed
        RL_x_angle = x_theta
        LL_x_angle = 170 - x_theta

        servos.position(RL_x, RL_x_angle)
        servos.position(LL_x, LL_x_angle)
        servos.position(LL_y, LL_y_angle)
        servos.position(RL_y, RL_y_angle)
```

Listing A.4 MicroPython code for Biomimic optimization for Batbot

```
import pyb
import os
import math
from machine import I2C,SoftI2C
import time
import jy901
import as5048a
import pid
import pca9685
import pca9685_servo
import micropython
from micropython import const
import reciever
```

```
import servo_angle
import utime
import gc


micropython.opt_level(3)

#iic接口及iic设备初始化
iic1=SoftI2C(scl='X9',sda='X10',freq=150000)
pca=pca9685.PCA9685(iic1)
servos=pca9685_servo.Servos(pca)
as5048_cs = pyb.Pin("X4", pyb.Pin.OUT_PP) # For
   ↪ magnetic encoder (wing angle measurement)
as5048_cs.high()
as5048_spi = pyb.SPI(1)
as5048_spi.init(mode=pyb.SPI.MASTER, prescaler=8, bits
   ↪ =8)
mag_sensor = as5048a.AS5048A(as5048_spi, as5048_cs)
uart = pyb.UART(3, 115200) # Wireless uart to send
   ↪ commands.


RL_x = 0
RL_y = 1
LL_x = 2
LL_y = 3
FLAPPER = 4 # from 240
FOLDER = 5 # 40-140:extended-folded
folded = 140
extended = 40

def move(motor, x_theta, y_theta, fold):
    # Control the actuator
```

```
    #y_theta = 0 # -90 : 90
    LL_y_angle = y_theta + 100
    RL_y_angle = 80 - y_theta

    #x_theta = 0 # 0-180
    RL_x_angle = x_theta + 48
    LL_x_angle = 135 - x_theta


    pca.duty(FLAPPER, motor)
    servos.position(FOLDER, fold)
    servos.position(RL_x, RL_x_angle)
    servos.position(LL_x, LL_x_angle)
    servos.position(LL_y, LL_y_angle)
    servos.position(RL_y, RL_y_angle)


class CycCalculation:
    def __init__(self, mag_sensor):
        self.angles = [0] * 200 # Initialize the list of
            ↪  angles with 500 zeros.
        self.mag = mag_sensor # This should be your
            ↪ magnetometer object.
        self.old_angle = self.mag.read_angle() # Store
            ↪ the initial angle read from the
            ↪ magnetometer.
        self.angle_max = max(self.angles) # The maximum
            ↪ angle in the current list of angles.
        self.angle_min = min(self.angles) # The minimum
            ↪ angle in the current list of angles.

    def update(self):
        new_angle = self.mag.read_angle()
```

```
self.angles.append(new_angle)
del self.angles[0]
self.angle_max = max(self.angles)
self.angle_min = min(self.angles)

delta_y1 = self.angles[-1] - self.angles[-2] #
    ↪ Change between first and second point.
delta_y2 = self.angles[-2] - self.angles[-3] #
    ↪ Change between second and third point.
delta_y3 = self.angles[-3] - self.angles[-4]

average_delta_y = (delta_y1 + delta_y2 +
    ↪ delta_y3) / 3

# Since x is uniform, we can assume delta_x as
    ↪ 1.
upward = average_delta_y > 0 # Determine wing
    ↪ beat direction.
cyc = 2 * ((new_angle - self.angle_min) / (self.
    ↪ angle_max - self.angle_min)) - 1 #
    ↪ Normalize cyc value between -1 and 1.

# Calculate cyc_pi based on the direction
    ↪ indicated by 'upward'.
if upward:
    cyc_pi = (cyc + 1) / 2 * math.pi # Map cyc
        ↪ from 0 to 1 to 0 to pi.
else:
    cyc_pi = math.pi * (2 - ((cyc + 1) / 2)) #
        ↪ Map cyc from 0 to 1 to pi to 2*pi.

self.old_angle = new_angle
```

```
        return upward, cyc_pi, cyc

def is_leg_upward(cyc_pi, range_start):
    """
    Check if cyc_pi is within a cyclic range of width
        ↪ pi,
    starting from range_start, considering the circular
        ↪  nature of angles.

    :param cyc_pi: The angle in radians, normalized to
        ↪ [0, 2*pi).
    :param range_start: The start of the range,
        ↪ normalized to [0, 2*pi).
    :return: True if cyc_pi is within the range, False
        ↪ otherwise.
    """

    range_start = range_start % (2 * math.pi)

    # Calculate the end of the range
    range_end = (range_start + math.pi) % (2 * math.pi)

    # Determine if cyc_pi is within the range
    if range_start < range_end:
        return range_start <= cyc_pi < range_end
    else:
        # The range wraps around, so cyc_pi is within
            ↪ the range if it's
        # either greater than the start or less than the
            ↪  end.
        return cyc_pi >= range_start or cyc_pi <
            ↪ range_end
```

```
cyc_calculation = CycCalculation(mag_sensor)
motor = 280
pca.duty(FLAPPER, 200)
time.sleep(1)


pca.duty(FLAPPER, motor)

'''
print('Calculating min/max...')
start_time = utime.ticks_ms()
while utime.ticks_diff(utime.ticks_ms(), start_time) <
   ↪ 3000:
    cyc_calculation.update()
    time.sleep(0.01)
print('Done!')
pca.duty(FLAPPER, 200)
'''
while True:
    response = uart.read() # read command
    if response:
        print(response)
        pca.duty(FLAPPER, motor)
        data_str = response.decode('utf-8')
        numbers = data_str.split(',')
        x_amp, y_amp, y_mid, delay = [float(num) for num
           ↪ in numbers]

        duration = 6500
        start_time = utime.ticks_ms()
        while utime.ticks_diff(utime.ticks_ms(),
           ↪ start_time) < duration:
```

```
            upward, cyc_pi, cyc = cyc_calculation.update
              ↪ () # Calculate cicle of wing
            leg_upward = is_leg_upward(cyc_pi, delay)
            x_theta = x_amp if leg_upward else 0
            fold = folded if upward else extended
            y_theta = y_mid - y_amp*math.cos(cyc_pi +
              ↪ delay)
            move(motor, x_theta, y_theta, fold)
            time.sleep(0.001)

        uart.read() # in case data was sent while in the
          ↪  loop it is deleted
        #pca.duty(FLAPPER, 200) # Turn off flapper
```

Listing A.5 MicroPyhton code for elliptical leg movement optimization

```
import pyb
import machine
import pca9685
import pca9685_servo
import as5048a
import utime
from micropython import const
import math
import time

# Configuration
i2c = machine.SoftI2C("X9","X10",freq=100000) # For
  ↪ PCA
pca = pca9685.PCA9685(i2c)
servos = pca9685_servo.Servos(pca)
as5048_cs = pyb.Pin("X5", pyb.Pin.OUT_PP) # For
  ↪ magnetic encoder (wing angle measurement)
as5048_cs.high()
as5048_spi = pyb.SPI(1)
```

```
as5048_spi.init(mode=pyb.SPI.MASTER, prescaler=8, bits
   ↪ =8)
mag = as5048a.AS5048A(as5048_spi, as5048_cs) #
   ↪ readings from 235-194 -> down-up
uart = pyb.UART(3, 115200) # Wireless uart to send
   ↪ commands.

# Constants
RL_x = 0
RL_y = 1
LL_x = 2
LL_y = 3
FOLDER = 4
FLAPPER = 5
folded = 140
extended = 50


time.sleep(3)
# Motors initialization
print('Initializing...')
servos.position(FOLDER, extended)
pca.duty(FLAPPER, 200)
time.sleep(3)
pca.duty(FLAPPER, 260)
time.sleep(1)
print('Iniitialized!')
# Initialize angle max, min
angles = []

i = 0

print('Calculating␣min/max...')
```

```
start_time = utime.ticks_ms()
while utime.ticks_diff(utime.ticks_ms(), start_time) <
   ↪ 3000:
    i += 1
    time.sleep(0.01)
    if i < 200:
        angle = mag.read_angle()
        angles.append(angle)
    else:
        break

angle_max = max(angles)
angle_min = min(angles)
print('Calculated!')
print('Max angle:', angle_max)
print('Min angle:', angle_min)
pca.duty(FLAPPER, 200)



while True:
    response = uart.read() # read command
    if response:
        print(response)
        data_str = response.decode('utf-8')
        numbers = data_str.split(',')
        motor, leg_x, leg_y, leg_x_amplitude,
           ↪ leg_y_amplitude, ellipse_angle \
            = [float(num) for num in numbers] # Extract
               ↪ command

        pca.duty(FLAPPER, motor)
```

```
duration = 6500 # 7.5 seconds
old_angle = mag.read_angle() # Used to calculate
   ↪ derivative which gives stroke direction
start_time = utime.ticks_ms()

while utime.ticks_diff(utime.ticks_ms(),
   ↪ start_time) < duration: # Run as long as
   ↪ stated experiment duration
   utime.sleep(0.001) # Used to stabiize the
      ↪ loop, if not added time measurement
      ↪ fails.

   new_angle = mag.read_angle()
   angles.append(new_angle)
   del angles[0]
   angle_max = max(angles)
   angle_min = min(angles)

   upward = new_angle > old_angle # Calculate
      ↪ wing beat direction
   cyc = (new_angle - angle_min) / (angle_max -
      ↪ angle_min) # down:0 up:1
   print(cyc)

   if cyc > 0.6:
      fold = extended # after half way up star
         ↪ extending
   elif cyc < 0.15:
      fold = folded # 20% before reaching down
         ↪ start folding
   elif upward:
      fold = folded
   else:
```

```
    fold = extended # down-stroke extend

servos.position(FOLDER, fold)

old_angle = new_angle

if upward:
    pi_cyc = math.pi * cyc # [0,pi]
else:
    pi_cyc = 2 * math.pi - math.pi * cyc # [pi
        ↪ , 2pi]

y_theta = leg_y - leg_y_amplitude * math.cos(
    ↪ pi_cyc) # Calculate angle from body
    ↪ plane to leg in vertical
if y_theta < 30: # Stops angle from exceeding
    ↪  limit values (also allowing straight
    ↪ trajectories in leg)
    y_theta = 30
elif y_theta > 150:
    y_theta = 150
LL_y_angle = y_theta
RL_y_angle = 180 - y_theta # Invert for right
    ↪  leg

# Calculate angle from body plane to leg in
    ↪ vertical
x_theta = leg_x + leg_x_amplitude * math.sin(
    ↪ pi_cyc + 2 * math.pi * ellipse_angle)
if x_theta < 40:
    x_theta = 40
elif x_theta > 120:
    x_theta = 120
```

```
        RL_x_angle = x_theta
        LL_x_angle = 180 - RL_x_angle

        # Command servos
        servos.position(LL_y, LL_y_angle)
        servos.position(RL_y, RL_y_angle)

        servos.position(RL_x, RL_x_angle)
        servos.position(LL_x, LL_x_angle)

        pca.duty(FLAPPER, motor)

    uart.read() # in case data was sent while in the
      ↪ loop it is deleted
    pca.duty(FLAPPER, 200) # Turn off flapper
```

ACKNOWLEDGEMENTS

There are countless people without whom this incredible adventure would have not been a reality, to each one of them I am deeply thankful for being part of my path and for their support in becoming a better version of myself. I am particularly thankful with my Professor Zhao Jingshan, his ability to find the golden spot between offering a wise advice when needed and allowing me the freedom of trying my ideas with liberty to make and learn from my mistakes, incredibly catapulted my development as a scientist. I am not only thankful for his professional mentoring, but also for the utter support he always made me feel, even when dealing with more personal and human topics. I am extremely thankful for the presence of Dr. Li in the lab throughout this years, besides being a role model of hard work and determination, a helping hand whenever needed and occasionally, an involuntary Chinese language teacher, above all of that he proved to be a great friend. I cannot thank my family back at home enough, for the support they have given me to accomplish my dreams. Thank you for all those hours spent on the phone, form which a bit of home managed to get through, and for the unconditional love. I would like to thank my fiance, that since the conception of the idea of coming to Tsinghua University until today, she has shown nothing but complete support. Putting even her personal desires aside for the sake of mine. Finally I would like to thank the friends that became family, with your presence, this year could not have been more enjoyable. Thank you everybody.

在这场不可思议的冒险中，有无数人的陪伴是必不可少的，我深深感谢每一个人，感谢他们成为我人生道路的一部分，以及在我成为更好的自己的过程中给予的支持。我特别感谢赵景山教授，他善于在需要时提供明智的建议与允许我自由尝试我的想法，自由地犯错误并从中学习，这就促进了我作为一名科学家的发展。我不仅想感谢他在专业上的指导，并还想感谢他即使是在处理更加个人和人性的话题时，他总是能让我感受到的绝对支持。我非常感谢这些年来李京虎在实验室的陪伴，他不仅是勤奋和决心的楷模，需要时总是伸出援手，偶尔担任语文老师，可是最重要的是，他证明了自己是一个伟大的朋友。我无法足够感谢我的家人，感谢他们给予我实现梦想的支持。感谢他们为我花那么多时间与我打电话，让我感受到一点家的气息，以及他们无条件的爱。我想感谢我的未婚妻，从我想要去清华大学的念头直到今天，她一直都在支持我。甚至为了我，她将自己的愿望排在最后。最后，我想感谢那些已经成为了家人的朋友们，有了他们的陪伴，这一年才会如此愉快。谢谢大家。

声　明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。

签　名：_____ 日　期：_____

RESUME

个人简历

职业经历

2018 年 12 月至今：柏林工业大学产品开发与机电一体化学院学生助理，参与中国电动巴士项目。

2021 年 5 月至 2021 年 9 月：Panda Insight 数据科学家。

2019 年 10 月至 2020 年 10 月：柏林工业大学放松阻尼研究项目学生助理。

2018 年 9 月至 2019 年 9 月：柏林工业大学力学 I 导师。

2017 年 12 月至 2018 年 11 月：ChronoBank 网站翻译（英语-德语-西班牙语）。

2010 年 8 月至 2013 年 5 月：Instituto Cumbres Cozumel 和 Instituto Gardner 数学与物理私人家教。

教育背景

2022 年 2 月至今：柏林工业大学物理工程科学硕士生，专注于数值计算。

2016 年 4 月至 2022 年 1 月：柏林工业大学物理工程科学学士，专注于机电一体化。

2018 年 4 月至 2018 年 9 月：贝鲁特 Saifi 阿拉伯学院阿拉伯语语言文凭（B1 级）。

2015 年 2 月至 2016 年 1 月：柏林工业大学预科。

2013 年 12 月至 2014 年 7 月：Hartnackschule 德语语言文凭（B2 级）。

2006 年 5 月至 2013 年 5 月：Instituto Cumbres Cozumel 高中毕业证书。

2000 年 5 月至 2006 年 5 月：Instituto Gardner 小学教育。

在学期间完成的相关学术成果

学术论文

暂无

专利

暂无

COMMENTS FROM THESIS SUPERVISOR

暂无

RESOLUTION OF THESIS DEFENSE COMMITTEE

暂无