THESIS PROPOSAL

Evolutionary Algorithm Framework for Controller Design

Author
Anuar Santoyo Alum
Tsinghua University
2023.05.15

Contents

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 (EA) 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 EAs 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 todays 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 controlling 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 Dr. Li, will be trained to strive for hovering control. 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 (more detailed explanation can be found in section 4.2). 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.

2 Literature Research

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 [3, 4, 5, 6, 7, 2, 8, 9, 10, 11, 12, 13],[14, 15, 16, 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 (FC) controller, another well established controller, received notorious attention [3, 5, 6, 2, 19, 10, 12, 15]. 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 position 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 [21, 2, 3]. 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 [22, 23, 24, 14]. 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 (detailed explanation in section 4.2) 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.

3 Theoretical Background

3.1 Covariance Matrix Adaptation (CMA)

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is a robust method for optimizing complex, non-linear, non-convex, multimodal, and possibly noisy objective functions. It is particularly effective in high-dimensional continuous optimization problems.

The core idea behind CMA-ES is to adapt the search distribution during the optimization process based on the samples' success history. It does so by updating not only the distribution mean but also the covariance matrix, which represents the shape and orientation of the search distribution. This allows CMA-ES to learn and exploit the underlying structure of the optimization problem.

Here's a simplified overview of how CMA-ES works:

1. Initialization: A population of candidate solutions is initialized with a multivariate normal distribution with a given mean and covariance matrix.

   2. Evaluation:Each individual in the population is evaluated based on the objective function.

   3. Selection: The top individuals are selected based on their fitness scores.

   4. Update The mean and the covariance matrix of the search distribution are updated based on the selected individuals. The update of the covariance matrix allows the algorithm to learn the problem structure.

   5. Reproduction: A new population is sampled from the updated distribution and the process returns to step 2.

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.

Keep in mind, though, that while CMA-ES is robust and powerful, it is also computationally intensive, especially in very high-dimensional spaces, due to the complexity of updating and storing the covariance matrix. It might not be the best choice for problems where evaluations are very expensive or the dimensionality is extremely high.

3.2 Feedforward Controller (FFC)

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.

The feedforward controller is designed based on a model of the system and aims to predict changes that will affect the system, enabling it to take corrective action in advance.

Here's a simplified way of how it works:

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.

3.3 Feedback Controller

A feedback controller, also known as a closed-loop control system, is a type of control system that adjusts its input based on the output's deviation from a desired or set value. It's called "feedback" because it uses information about the system's output to influence the system's input.

Here's a simplified explanation of how it works:

1. Setpoint: The desired value for the system output is defined. This is often referred to as the setpoint.

2. Measurement: The actual output of the system is measured.

3. Error Calculation: The difference between the actual output (measured value) and the desired output (setpoint) is calculated. This difference is called the error.

4. Correction: The controller uses this error to determine how to adjust the system input to bring the output closer to the setpoint. This could be achieved using various methods, such as proportional, integral, and derivative control (collectively known as PID control).

5. Actuation: The controller applies the calculated adjustment to the system input. This influences the system's output.

6. Repetition: Steps 2 to 5 are repeated continually, creating a feedback loop that allows the system to respond dynamically to changes and disturbances.

The key advantage of feedback control is its ability to correct for disturbances and modeling inaccuracies, even when these are not known in advance. This makes feedback control systems robust and widely applicable in various fields, such as temperature control in ovens, speed control in cars, and even in financial and economic systems.

However, one limitation of feedback control is that it can only react after an error has occurred, which might be too late in some cases. Also, poorly designed feedback controllers can lead to oscillations or instability in the system. Therefore, they need to be carefully

designed and tuned for each specific application.

3.3.1 Proportional Integral Derivative (PID)

A Proportional-Integral-Derivative (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.

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.

3.3.2 Fuzzy Logic (FC)

A fuzzy logic controller is a type of control system based on fuzzy logic, a mathematical framework that handles reasoning that is approximate rather than precise. Unlike traditional binary sets (where variables may be true or false), fuzzy logic variables may have a truth value that ranges between 0 and 1, allowing for a more nuanced, human-like interpretation of data.

Here's a simplified explanation of how a fuzzy logic controller works:

1. Fuzzification: Inputs to the controller are expressed in terms of linguistic variables using fuzzy sets. For example, instead of saying a room is 22°C, which is a precise value, we might say the room is "warm", which is a fuzzy value. The controller converts the crisp numerical input into a fuzzy quantity.

2. Rule Evaluation: Based on the fuzzy inputs, the controller applies a set of if-then rules that are stored in a rule base. These rules are designed to capture the behavior of the system. For instance, a rule for a temperature control system might be: "If the room is 'warm', then turn the air conditioner 'on'." Each rule gives a fuzzy output.

3. Aggregation: The controller combines the fuzzy outputs from all rules into a single fuzzy set. This process may involve finding the maximum value of the outputs, their union, or other operations, depending on the inference mechanism being used.

4. Defuzzification: The fuzzy output is converted back into a crisp value to make it interpretable for the end user or to provide a control signal to another part of the system. There are several methods to do this, such as the centroid method, where the center of gravity of the fuzzy set is calculated.

Fuzzy logic controllers are particularly useful in systems where the model of the system is complex, unknown, or hard to formulate in precise terms. They are widely used in various applications including household appliances, automotive subsystems, and industrial process control. They are designed to handle uncertainty and vagueness, similar to human decision making.

3.3.3 Neural Network (NN)

A neural network controller is a type of control system that utilizes artificial neural networks (ANNs) to control dynamic systems. The controller is designed to learn from the data it's given, identify complex patterns, and make decisions accordingly.

Here's a simplified explanation of how a neural network controller works:

1. Learning Phase: The controller is trained using a set of input-output data. This data represents the system's behavior. The controller adjusts the weights of the neural network during training to minimize the difference between the actual output and the output predicted by the neural network.

2. Control Phase: Once the controller is trained, it can take in new input data and predict the appropriate output based on its learned weights. This output is then used to control the system.

Neural network controllers can be categorized into two types:

- Direct Neural Controllers: Here, the neural network is trained to act as a controller. It directly takes the error (difference between setpoint and actual output) and possibly its derivatives as input and outputs the control signal.

- Indirect Neural Controllers: In this case, the neural network is first trained to identify or model the plant (system to be controlled). Once the plant model is trained and validated, a control law (like PID or state feedback) is designed based on the identified model.

Neural network controllers can handle non-linear systems and do not necessarily require a mathematical model of the system, which makes them versatile and applicable to a variety of complex systems. However, they require sufficient and representative training data, and it can be challenging to interpret their internal workings due to their "black box" nature.

Applications of neural network controllers span across various fields such as robotics, process control, and autonomous vehicles.

4 Experiment Design

In the following chapter a detailed explanation of the experiments settings will be presented. First of all the materials needed for the experiments will be described in the section 4.1. Following and explanation of the methodology and the steps taken will be described in the section 4.2.

4.1 Materials

4.1.1 Batbot

The Batbot is a robotic bat developed by Dr. Li reference needed. Its design is inspired on one of the most agile flying creatures to have existed. Its structure is mostly made out of carbon fiber with an elastic membrane emulating the skin from the wings of the bat. It is composed of one DC motor and 6 servo motors. The DC motor actuates the flapping motion of the wings. The one servo is used to control the folding and unfolding motion of the wings and another to control the shoulder rotation of the wings, as both the folding/unfolding and shoulder rotation are uniquely linked to the flapping wing motion, the only parameter needed to control the two mentioned servos and the flapping wings is the the input voltage w, which is applied to the dc motor. The angles commanded to both servo motors are determined by the angle of the wing ϕ, which is measured using a magnetic encoder. The value of ϕ cannot be directly related to w due to many dynamic and kinematic factors, such as the initial position of the wing when the voltage w is applied. Nevertheless, the speed in which ϕ changes can be directly related to the flapping frequency f and at the same time is proportional to the input voltage w, therefore $w \sim f$. The position of a hind leg is controlled using two servo motors, one servo motor determines the angle of the hind leg on the horizontal plane, and the second servo motor the angle with respect to the vertical plane. The horizontal plane is the one spanned through the x and y axes, and the vertical plane is the one spanned by the x and z axes, where the x axis shows in the nose-ward direction of the Batbot, the y axis points in the direction of the left wing and the z axis in the direction of the back of the Batbot. In figure 1 a visual representation of the axes definition with respect to the Batbot is provided. Therefore, the parameters needed to determine the position of both hind legs are: the horizontal angle of the left hind leg l_h, the vertical angle of the left hind leg l_v, the horizontal angle of the right hind leg r_h, the vertical angle of the left hind right r_v . With this, all the parameters needed to command the Batbot can be describe with the command vector $\vec{q}$, defined as:

$$\vec{q} = \begin{pmatrix} w \\ l_h \\ l_v \\ r_h \\ r_v \end{pmatrix} \quad , \vec{q} \in \mathbb{R}^5 \tag{1}$$

All the motors are controlled using the micro-controller *STM32 Bluepill*. The wireless module *ESP8266* is connected to the micro-controller using serial communication,. It is used to receive data from the computer to enable wireless communication, crucial for a flying robot. Is a more detailed from the Batbot needed for the proposal? For example angles rage of the motors, battery technical data, measurements, weight of the Batbot, etc...

4.1.2 Real Bat Flight 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*

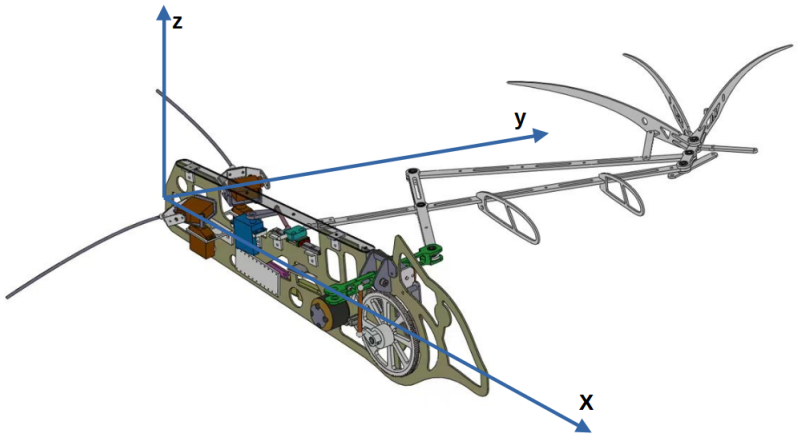
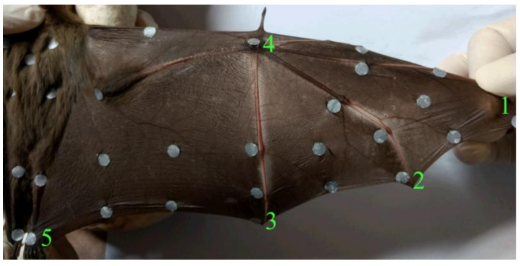
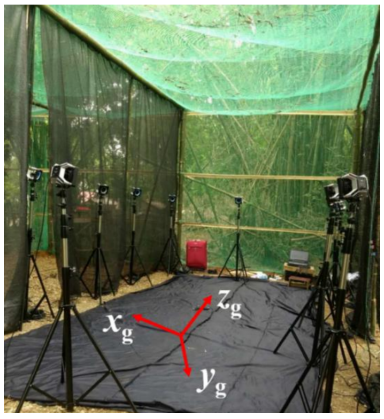


Figure 1: Batbots axes



(a) Markers on bat wing and angles definition from Dr. Sudeeps dissertation



(b) Measurement setting to measure bat flight data using "Optitrack"

Figure 2: Real bat data measurements from Dr. Sudeeps dissertation

were studied during flight to reconstruct the wing kinematics and to identify the primary relationship of kinematic parameter in straight and maneuvering flight. The data obtained by Dr. Sudeep was captured using the "OptiTrack" system (figure 2b) and pasted markers on the body (figure 2a), wings and legs of the bat. The results of his dissertation and the flight data obtained 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. This will be incredibly useful when designing the initial control algorithms.

4.1.3 Static Test-Bench

The static test-bench plays a very important role in the hovering learning process. It manages to enable emulation of Batbot hovering in a safe environment, which is a great

concern and drawback of training a robot in the real world. The test-bench, as seen in figure 3, consists of 6 force sensors (represented in yellow) and a DC motor to conveniently change the orientation of the Batbot. The sensors are connected to the data acquisition system *BSQ-JN-P8*, which is directly connected to the computer. Using Python and the python package *serial*, communication between the computer and the data acquisition system is possible at a baud-rate of 115200 using the Modbus communication protocol. Further analysis will be needed to determine if the actual configuration/choice of sensors is adequate or could be optimized. Are more detailed measurements needed or the math to calculate, lift, thrust and torque?

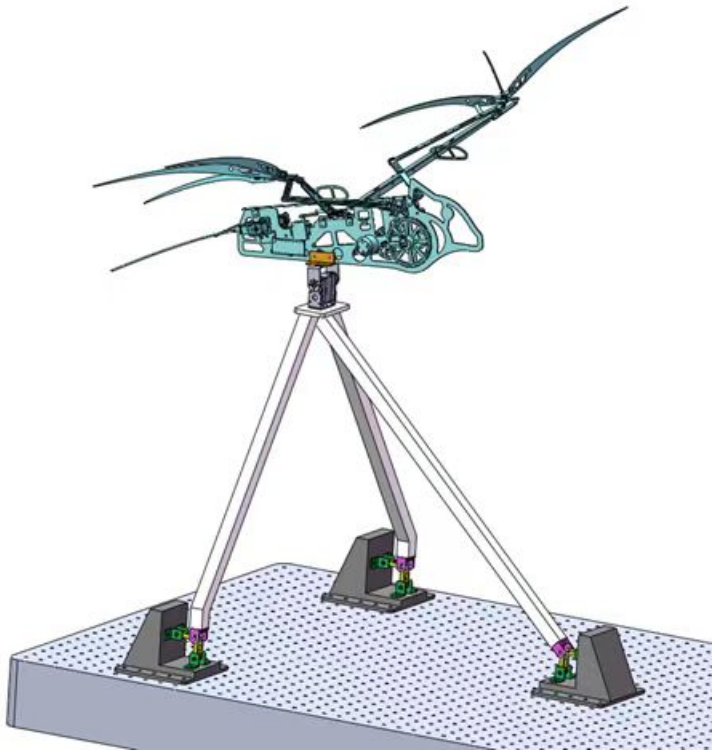


Figure 3: CAD of static test-bench

The sensors measurements can be defined as the function vector

$$\vec{s}_{stat} = s_i(t), \quad i \in \{1, .., 6\} \tag{2}$$

where $s_i(t)$ are the measurements obtained from the sensor i depending on time t.

4.1.4 Feed Forward Control

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 a function of the parameter to optimize θ and the wing angle ϕ. Therefore, the feedforward controller FFC is defined as follows:

$$FFC(\theta, \phi) = \vec{q} \tag{3}$$

This means, that FFC returns the command vector from equation 1.

We would like to present an example to clarify the FFC. Let us assume that using the analysis obtained from the real bat data, we can conclude that the flapping of the hind legs and the wings is linked, so that when the wings go up, so do the hind legs, all at the same vertical angle. Another assumption we can make is that the hind legs only flap in the vertical direction and stay at a constant angle with respect to the horizontal plane. For this reason, we select the following parameters to be ones needed to optimize: w to control the flapping frequency and the optimal horizontal angle for both hind legs α, so that $\theta = [w, \alpha]$. With all these (strong) assumptions we can now define our example feedforward controller $FFC_{example}$ as:

$$FFC_{example}(\theta, \phi) = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \phi \\ \theta_2 \\ \phi \end{pmatrix} = \begin{pmatrix} w \\ \alpha \\ \phi \\ \alpha \\ \phi \end{pmatrix} = \vec{q} \tag{4}$$

Correctly defining the feed forward controller and its respective optimization parameters θ is crucial to obtain. For this reason, a major task of this thesis is the analysis of real bat data as well as several FFC design tests. The process to find the optimal optimization parameters θ_{opt} will be in depth explained in section 4.2.4. Once θ_{opt} has been found, we define our optimal feedforward controller as follows:

$$FFC_{opt}(\phi) := FFC(\theta_{opt}, \phi) \tag{5}$$

Where FFC_{opt} only depends on the wing angle ϕ as the parameters θ will now be fixed to be θ_{opt}

4.1.5 Fitness Static Test

The fitness function of the static test $F_{stat}(\vec{s}_{stat})$ is used to measure how close is the Batbot to achieve hovering in the static test-bench, it has as an input the sensor measurements of a test and it returns a fitness score $fitness$. It manages to accomplish this using the measurements of the sensors $\vec{s}_{stat}$ from equation 2. First of all, the neutral state of the sensors (the force measured when the Batbot is unmounted) must be calculated in order to calibrate the sensors. This was done with several experiments, where different forces and

torques were applied at the several intervals and then the neutral states were measured. From this tests it could be concluded that the neutral states of the sensors always remained under a tolerable range. In figure 4 we can see the resulting voltage measurements in the sensors from an experiment where random forces and torques were manually applied.

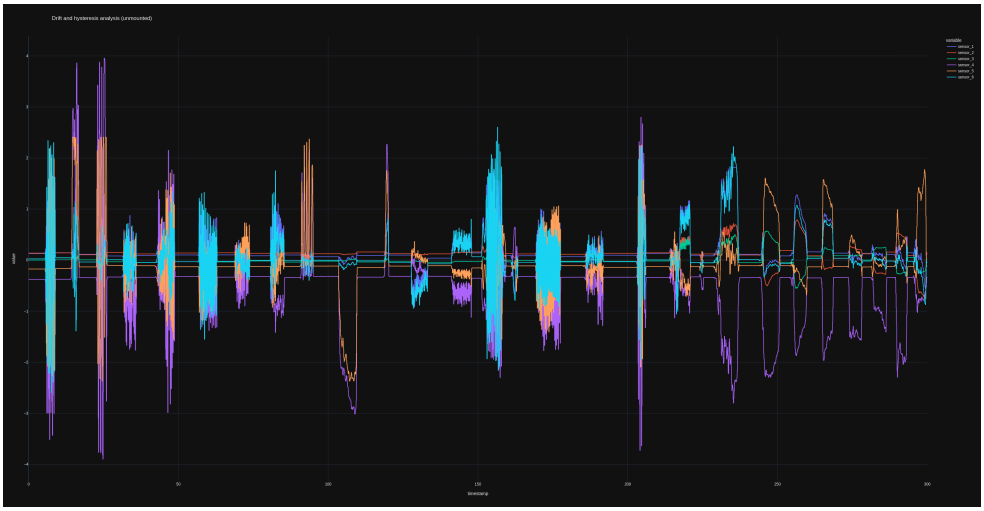


Figure 4: Sensor neutral state experiment

Using this data, the neutral states (the value from the horizontal lines in figure 4) were obtained. Their distribution can be seen in the density plots in figure 5. Here it can be observed that the neutral state values follow a distribution similar to a normal distribution with small variance. Therefore, the mean of the distributions can be considered as the neutral state of the sensors. The resulting values can be found in table 1. This values are result of an initial experimentation, through the thesis this values will have to be corroborated using the Batbot to exert the forces instead of the manual approach taken in this experiment, nevertheless this values are sufficient for initial experimentation.

Sensor	Neutral State (Volts)
1	0.103034
2	0.126191
3	-0.016099
4	-0.337834
5	-0.127773
6	-0.016209

Table 1: Sensor neutral states (negative values represent pushing and positive pulling forces)

Once the neutral states are know the static fitness function can be designed. There are several plausible ideas that will need testing, the most promising ideas for a fitness function could be to simply calculate the mean squared error between the sensor measurement and its neutral state. This would incentivize the Batbot to reduce the force exerted on the sensors, testing might be needed to assure if special weighting to each sensor will be necessary or not. Other ideas might involve using the sensor measurements to calculate the forces and

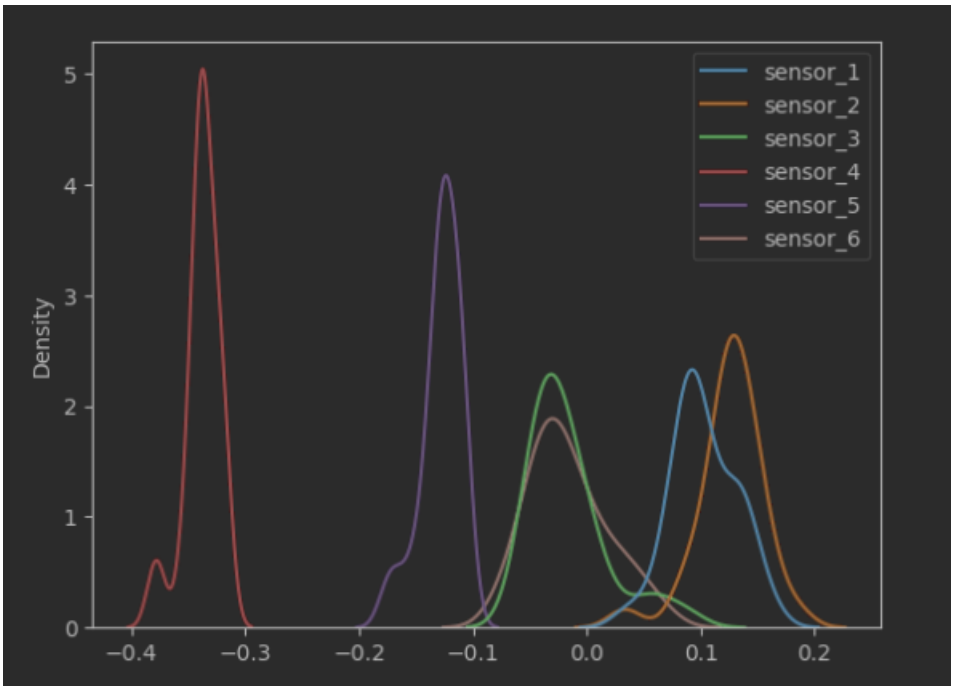


Figure 5: Sensors neutral state distributions

torques the Batbot exerts and incentivize to reduce those to zero. If reducing to zero is not entirely possible, a force that oscillates with a mean value of zero would also represent hovering, as it would mean that the Batbot oscillated around the same point, given that the oscillations are small enough.

4.1.6 Dynamic Test-Bench

The goal of the dynamic test-bench is to allow the Batbot to train to hover in a realistic way, without being attached to a static structure as in the static test-bench. Nevertheless, the safety of the robot must be assured. The design and development of such a test-bench is a major task of the thesis. Detailed ideas are yet to be obtained, nonetheless a basic idea has already been realized and will be here explained.

In the dynamic test-bench, the Batbot will hang from some sort of string, connected to the string there will be a force sensor that would enable us to measure if the Batbot is actually hovering or just hanging, logically the string sensor must measure a neutral state to assume hovering. Another important part to be measured is the position of the Batbot in the 3D space, for this several types of sensors and configurations can be used, such as: an array of ultrasonic distance sensor, a ladar sensor, camera aided with computer vision among others. The measurements of this test-bench, which is yet to be determined, will be represented in the function vector $\vec{s}_{dym}$.

4.1.7 Feed Back Control

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 of the main 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. The state measurements obtained by the integrated sensors can be denoted in the vector $\vec{b}$, where the wing angle ϕ will be an element of the state measurements. It is important not to mix the state measurements $\vec{b}$, which denote the state of the Batbot and are used for the feedback loop, with the dynamic sensor measurements $\vec{s}_{dyn}$, which will be used to calculate the fitness score in the dynamic test.

There are many possibilities of constructing a feedback controller with their respective parameters to optimize ψ. One idea could be implemented to robust and well known PID controller and choose ψ to be the gains of the controller. Another idea could be to use a simple neural network that receives as an input the value of the Batbot state $\vec{b}$ and as an output gives values that alter the motors controlling, in this case ψ would be the weights and biases of the neural network. A further option could be to use a fuzzy controller and try to optimize the membership functions or the rule base system. In any of he cases, we can define our feedback controller FBC , as follows:

$$FBC(\psi, \vec{q}) := \vec{c} \tag{6}$$

Where $\vec{c}$ is the correction vector, this vector denotes the correction that has to be applied to the command vector $\vec{q}$, obtained from FFC_{opt}, to correct the Batbot to remain in a hovering position. In order to obtain the complete control, we first generate a command vector $\vec{q}$ from FFC_{opt} and correct it depending on the measured state of the Batbot $\vec{b}$, using FBC. Hence, the complete control scheme looks as follows:

$$FFC_{opt}(\phi) + FBC(\psi, \vec{b}) = \vec{q} + \vec{c} = \hat{\vec{q}} \tag{7}$$

Where $\hat{\vec{q}}$ denotes, the corrected command vector. Similar to the equation 5 once the optima feedback controller parameters ψ_{opt} are found (how this is done is in detail explained in section 4.2.5), the optimal feedback controller can be defined as:

$$FBC_{opt}(\vec{q}) := FBC(\psi_{opt}, \vec{q}) \tag{8}$$

Which would finally describe our optimal controller as:

$$FFC_{opt}(\phi) + FBC_{opt}(\vec{q}) = \hat{\vec{q}} \tag{9}$$

Were the corrected command $\hat{\vec{q}}$, and the wing angle ϕ are the only information needed.

4.1.8 Fitness Dynamic Test

The goal of the dynamic fitness function $F_{dym}(\vec{s}_{dym})$ is to calculate how close the Batbot is to hovering. Using the measurements of $\vec{s}_{dym}$, which include information of the position of the Batbot in the 3D space throughout a test and the measurement of the string the Batbot

hangs from, we can use both of these measurement to incentivize the Batbot to reduce the force of the string sensor to zero as a primary task and then to reduce its position in the 3D space to the desired hovering position. Different combinations of these basic ideas can be tested, such as using different weighting for the string or the position penalization, or developing a multi-phase training were first the string penalization is of relevance and through several iterations the position penalization starts playing a more important role.

4.2 Methods

Training a control algorithm for a robot in the real world, as previously discussed, has some great advantages, specially for complex systems. Nevertheless, it also brings with it some drawbacks that must be mentioned. A mayor disadvantage is the time needed to test one proposed solution. For this reason, learning a control algorithm from scratch can require many iterations to converge. To address this issue we decided to divide the training process in 3 phases. The goal of each phase is used as a starting point for the next phase. By subdividing the goal in sub-goals we provide the Batbot with a clearer learning direction, which will save it several iteration compared to if it were to learn it by itself from scratch. Another strategy used to reduce the needed training is to use heuristics in form of real bat flight data. This would allow us to embed in the controlling algorithm information obtained from real bats, which could potentially reduce the amount of parameters to optimize and the needed training iterations with it. Therefore using real bat data flight and a 3-phased training approach we try to overcome the limitations encountered when training control algorithms on a real robot.

In the following subsection, the detailed steps of the experiments will be explained. Here, we would like to provide a general overview of the 3 phased process we propose. This will hopefully provide the reader with a better understanding of the general idea, so that when reading the detailed steps of the 3 phases a relationship with the overall goal can be easily found. The first phase of the experiments is *Flight dynamic analysis*. In this phase, an analysis of real bat flight data as well as experiments on the static test bench are taken to obtain an understanding of how can we best control the Batbot, or better said generate a good design of a FFC. The second phase, *Static Test*, uses CMA and the static test-bench to find the optimal parameters θ_{opt} from which FFC_{opt} can be defined. The last phase, *Dynamic Test* uses the found FFC_{opt}, which gives an open-loop hovering command to the Batbot, to optimize the FBC (meaning to find ψ_{opt}) using the dynamic test bench.

The goal of this thesis is to analyze the optimization process of the controllers, to conclude if using the proposed framework actually improves the performance of the controller strategies. To achieve this the fitness functions F_{stat} and F_{dym} are crucial, as they can be used to measure the performance of the solutions proposed by the CMA algorithm throughout the experimentation, these are at the same time used to optimize the CMA to generate better new generations. The thesis will conclude with and extensive analysis, result and discussion of said results to confirm or deny the efficacy and feasibility of the proposed framework.

4.2.1 Training Cycle General Workflow

The proposed framework design positions the Batbot in a test-bench were measurements can be taken using sensors. A computer running the CMA EA 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 CME EA 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 6.

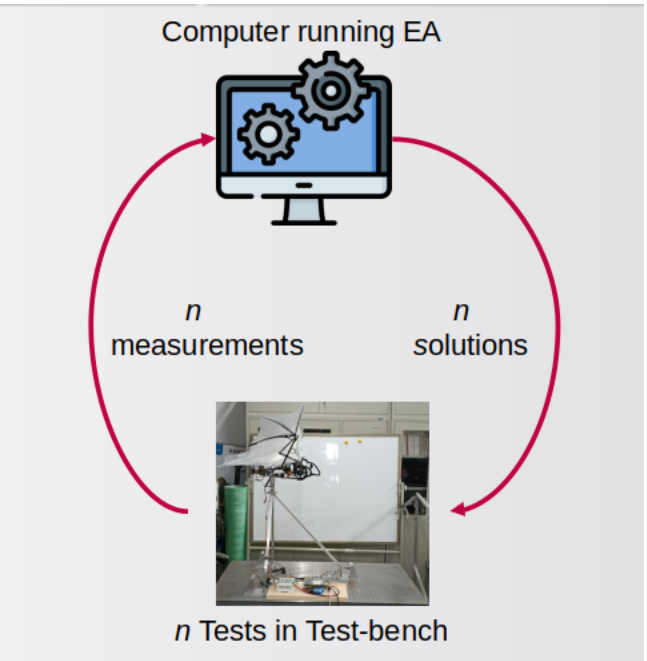


Figure 6: General workflow of training cycle

4.2.2 Proof of Concept Prototype

The training cycle explained in 4.2.1 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

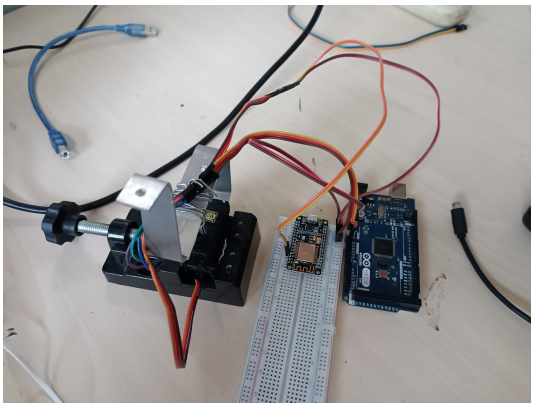


Figure 7: Prototype connections

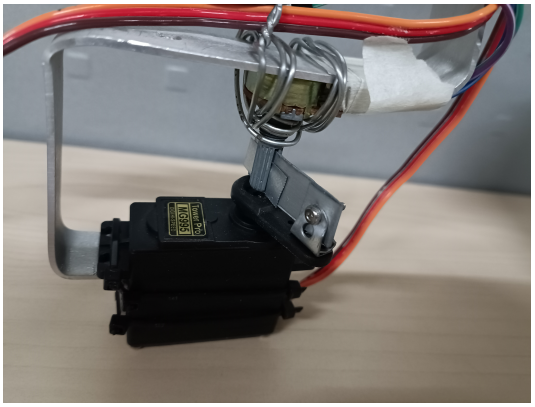


Figure 8: Prototype configuration

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 7 the prototype, NodeMCu and Arduino can be seen and 8the 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 able to find the optimal solution. In the figures 9,10,11,12 and 13 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 optimal solution. The sin curved plotted is the comparison between the best solution of the generation with the optimal solution.

4.2.3 Phase 1: Flight Dynamic Analysis

The first phase, is the only of the three where actually no training is involved. The main goal of this phase is to obtain a better understanding of the effects certain commands have on the Batbot. In this stage, the analysis of the real bat flight data (4.1.2) could be used to design better flight commands. From the results of this analysis we can better understand

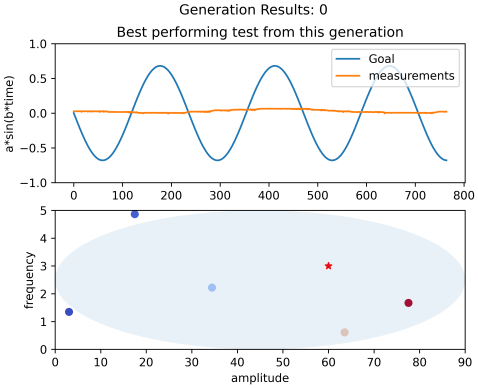


Figure 9: Generation 0

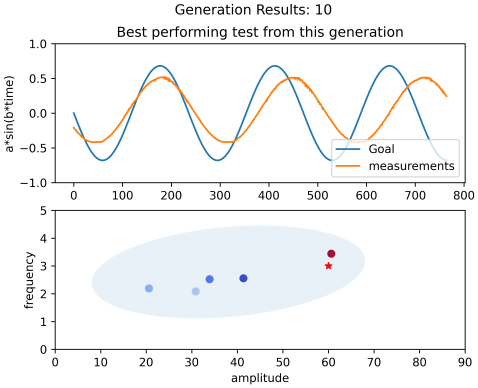


Figure 10: Generation 10

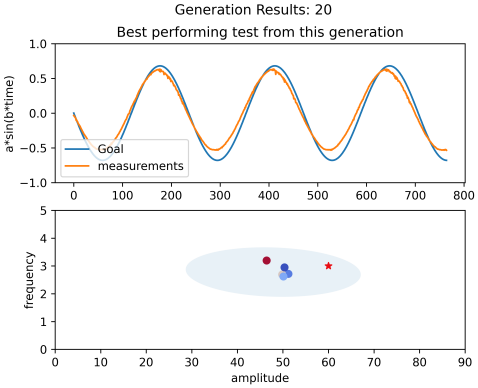


Figure 11: Generation 20

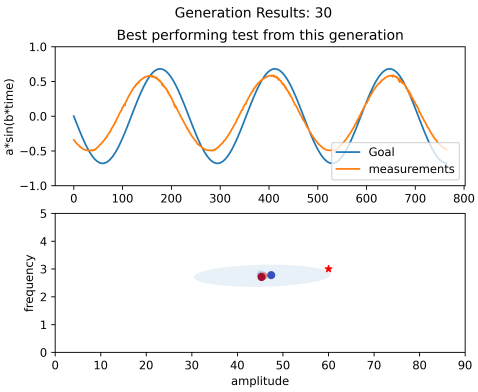


Figure 12: Generation 30

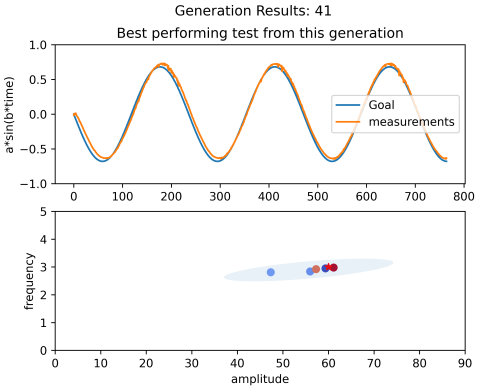


Figure 13: Generation 41

which specific motor commands should be made to generate a force or torque in a desired direction. To achieve this, the Batbot is positioned on a static test-bench, in detail explained in section 4.1.3, where the Batbot can be commanded to realize different flight maneuvers from which the resulting forces and torques can be calculated using the static test-bench sensors. After the analysis is completed, FFC can be design which will then be optimized in the static test phase.

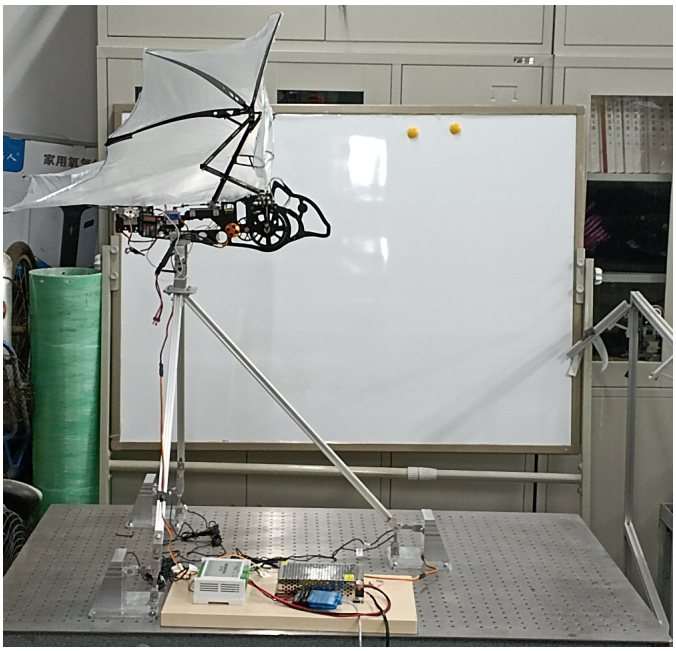


Figure 14: Batbot in static test-bench for flight analysis

4.2.4 Phase 2: Static Test

In the static test, using the results of section 4.2.3, we can position the Batbot in the hovering position on the static test-bench, represented in figure 15. The idea behind this test is to use the sensors of the static test-bench to calculate the forces exerted by the Batbot when executing a flight command, if the sensors are unable to measure any force (even when the Batbot is mounted to the test-bench) we can assume that in an ideal situation the Batbot would be hovering when executing said command, as no forces represent no acceleration in any direction. Therefore, we can use the training cycle explained in 4.2.1 to incentive the Batbot to optimize FFC to reduce the forces exerted on the sensors. This is of course real only in an ideal situation, if after successfully completing the phase 2, the Batbot were to be tested on hovering it would fail, as we have only optimized a feed forward controller, which means that by any small disturbance on the Batbot or any small deviation of the hovering position the Batbot would collapse. To ensure real hovering, a feedback controller is needed that provides the Batbot with current information of its state to compensate any disturbance. Nevertheless, obtaining an optimal feed forward controller on the static test is a great starting point to develop a feedback controller, as the main hovering command would already be learnt, and all that would be left to do is learn how to maintain it in the presence of disturbances.

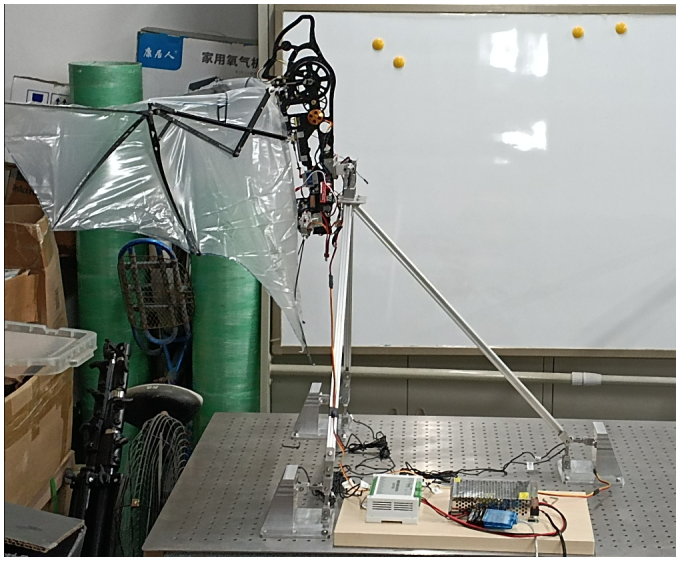


Figure 15: Batbot in static test-bench in hovering position (visual aid, not real hovering position)

4.2.5 Phase 3: Dynamic Test

In the final phase, the knowledge of the last two phases could be applied to optimize a feedback controller for hovering. In this case the Batbot would be positioned on the dynamic test-bench. Were it would be able to freely flight under a constraint space. To achieve this, a feedback controller is optimized using the training cycle explained on section 4.2.1. Both the dynamic test-bench and the feedback algorithm are explained with more detail in section 4.1.6 and 4.1.7 respectively. The theoretical completion of this phase would deliver a Batbot with an optimized feedback controller for hovering.

4.2.6 Analysis

As with many other ML learning approaches, quickly converging to an optimal solution is also highly dependent on the hyper-parameters selected for the ML algorithm. For this reason an analysis of the chosen hyper-parameters with their performance can be made. This is possible thanks to the fitness functions of the static and dynamic test, that let us quantify the performance of a proposed solution. This will allow us to analyze gradual improvement of our controlling algorithm throughout each iteration and phase. From this analysis we will be able to determine if it is possible to develop a controlling algorithm using EA on a real robot.

5 Time Plan

Tasks:

1. Preparatory

    1.1 Create proof of concept prototype.

    1.2 Develop wireless communication between computer and STM32.

1.3 Design and construct static test-bench.

1.4 Enable communication between data acquisition system and computer.

1.5 Write framework that enables general workflow of training cycle in Python.

2. Phase 1: Flight Dynamic Analysis

2.1 Analyse real bat data (from papers and Dr. Sudeep's dissertation).

2.2 Design and program ideas for feedforward controllers in STM32.

2.3 Test feedforward controllers.

2.4 Analyse data to clarify commands needed to exert specific forces and torques on Batbot.

2.5 Optimization of static test-bench if needed.

2.6 Design final feed forward control to be optimized on phase 2.

3. Phase 2: Static Test

3.1 Research on hyper-parameter optimization of CMA-ES.

3.2 Design and development of static fitness function.

3.3 Implementation of training framework.

3.4 Optimization of training framework.

3.5 Definition of optimal feedforward controller found.

3.6 Analysis of performance development of training framework and resulting controller.

4. Phase 3: Dynamic Test

4.1 Design and development of dynamic test-bench.

4.2 Research and analysis of possible feedback algorithm designs.

4.3 Research, selection and implementation of Batbots state sensors (eg. accelerometers, barometers, ...), needed for the feedback loop.

4.4 Implementation and testing of feedback algorithms with state sensors.

4.5 Optimization of algorithm or test-bench if needed.

4.6 Analysis of performance development of training framework and resulting controller.

5. Documentation

5.1 Analysis of obtained data to develop conclusion of performance of the proposed controller training framework.

5.2 Writing of master thesis.

5.3 Thesis defense.

Task	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1.1																							
1.2																							
1.3																							
1.4																							
1.5																							
2.1																							
2.2																							
2.3																							
2.4																							
2.5																							
2.6																							
3.1																							
3.2																							
3.3																							
3.4																							
3.5																							
3.6																							
4.1																							
4.2																							
4.3																							
4.4																							
4.5																							
4.6																							
5.1																							
5.2																							
5.3																							

Table 2: Task timetable with done (red) and to be done (green) markers on respective year week
Just example, real data will be filled after final version of Thesis Proposal

References

[1] A. Slowik and H. Kwasnicka, "Evolutionary algorithms and their applications to engineering problems," *Neural Computing and Applications*, vol. 32, no. 16, pp. 12 363–12 379, Aug 2020. [Online]. Available: https://doi.org/10.1007/s00521-020-04832-8

[2] J. J. Grefenstette, "Evolutionary algorithms in robotics," 1994.

[3] T. L. Seng, M. Bin Khalid, and R. Yusof, "Tuning of a neuro-fuzzy controller by genetic algorithm," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 29, no. 2, pp. 226–236, 1999.

[4] O. Gundogdu, "Optimal-tuning of pid controller gains using genetic algorithms," *Journal of Engineering Sciences*, vol. 11, pp. 131–135, 01 2005.

[5] L. Fan and E. M. Joo, "Design for auto-tuning pid controller based on genetic algorithms," in *2009 4th IEEE Conference on Industrial Electronics and Applications*, 2009, pp. 1924–1928.

[6] F. Hoffmann, "Evolutionary algorithms for fuzzy control system design," *Proceedings of the IEEE*, vol. 89, no. 9, pp. 1318–1333, 2001.

[7] D. Kwok and F. Sheng, "Genetic algorithm and simulated annealing for optimal robot arm pid control," in *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, 1994, pp. 707–713 vol.2.

[8] P. A. Vikhar, "Evolutionary algorithms: A critical review and its future prospects," in *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*, 2016, pp. 261–265.

[9] T. Back, M. Emmerich, and O. Shir, "Evolutionary algorithms for real world applications [application notes]," *IEEE Computational Intelligence Magazine*, vol. 3, no. 1, pp. 64–67, 2008.

[10] G. Narvydas, R. Simutis, and V. Raudonis, "Autonomous mobile robot control using fuzzy logic and genetic algorithm," in *2007 4th IEEE Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*, 2007, pp. 460–464.

[11] T. Neenu and P. Poongodi, "Position control of dc motor using genetic algorithm based pid controller," *Lecture Notes in Engineering and Computer Science*, vol. 2177, 07 2009.

[12] K. Sheroz, A. Salami, L. Adetunji, A. Alam, M. Salami, S. Hameed, A. Hasan, and M. Islam, "Design and implementation of an optimal fuzzy logic controller using genetic algorithm," *Journal of Computer Science*, vol. 4, 10 2008.

[13] A. Bagis, "Determination of the pid controller parameters by modified genetic algorithm for improved performance." *J. Inf. Sci. Eng.*, vol. 23, pp. 1469–1480, 09 2007.

[14] M. W. Iruthayarajan and S. Baskar, "Evolutionary algorithms based design of multivariable pid controller," *Expert Systems with Applications*, vol. 36, no. 5, pp. 9159–9167, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417408008920

[15] S. H. Kim, C. Park, and F. Harashima, "A self-organized fuzzy controller for wheeled mobile robot using an evolutionary algorithm," *IEEE Transactions on Industrial Electronics*, vol. 48, no. 2, pp. 467–474, 2001.

[16] A. Schultz, "Learning robot behaviors using genetic algorithms," 01 1999.

[17] L. da Silva Assis, A. da Silva Soares, C. J. Coelho, and J. Van Baalen, "An evolutionary algorithm for autonomous robot navigation," *Procedia Computer Science*, vol. 80, pp. 2261–2265, 2016, international Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050916308808

[18] M. Willjuice Iruthayarajan and S. Baskar, "Covariance matrix adaptation evolution strategy based design of centralized pid controller," *Expert Systems with Applications*, vol. 37, no. 8, pp. 5775–5781, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417410000709

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

[20] W. Lu, J. Yang, and X. Liu, "The pid controller based on the artificial neural network and the differential evolution algorithm," *Journal of Computers*, vol. 7, 10 2012.

[21] P. Karlra and N. Prakash, "A neuro-genetic algorithm approach for solving the inverse kinematics of robotic manipulators," in *SMC'03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme - System Security and Assurance (Cat. No.03CH37483)*, vol. 2, 2003, pp. 1979–1984 vol.2.

[22] D. Corne and M. A. Lones, "Evolutionary algorithms," in *Handbook of Heuristics*. Springer International Publishing, 2018, pp. 1–22. [Online]. Available: https://doi.org/10.1007%2F978-3-319-07153-$4_27 - 1$

[23] A. Yoshida, S. Kanagawa, Y. Wakasa, K. Tanaka, and T. Akashi, "Pid controller tuning based on the covariance matrix adaptation evolution strategy," in *2009 ICCAS-SICE*, 2009, pp. 2982–2986.

[24] S. Sivananaithaperumal and B. Subramanian, "Design of multivariable fractional order pid controller using covariance matrix adaptation evolution strategy," *Archives of Control Sciences*, vol. 24, 06 2014.

[25] M. Sheikhan and S. A. Ghoreishi, "Application of covariance matrix adaptation–evolution strategy to optimal control of hepatitis b infection," *Neural Computing and Applications*, vol. 23, no. 3, pp. 881–894, Sep 2013. [Online]. Available: https://doi.org/10.1007/s00521-012-1013-3

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

[27] H. Zhu, J. Yu, A. Gupta, D. Shah, K. Hartikainen, A. Singh, V. Kumar, and S. Levine, "The ingredients of real-world robotic reinforcement learning," 2020.

[28] M. Asafuddoula, T. Ray, and R. Sarker, "An adaptive differential evolution algorithm and its performance on real world optimization problems," in *2011 IEEE Congress of Evolutionary Computation (CEC)*, 2011, pp. 1057–1062.

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

[30] P. Wu, A. Escontrela, D. Hafner, K. Goldberg, and P. Abbeel, "Daydreamer: World models for physical robot learning," 2022.

[31] S. Guo, J. Lin, T. Wöhrl, and M. Liao, "A neuro-musculo-skeletal model for insects with data-driven optimization," *Scientific Reports*, vol. 8, no. 1, p. 2129, Feb 2018. [Online]. Available: https://doi.org/10.1038/s41598-018-20093-x