JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

ORNITHOPTER

AKASH BR[1], Dr. DEVRAJ VERMA C[2], SAI UJWAL V[3], S.CHANDRASHEKAR[4]

[1]Student, Computer science and engineering, Jain University, Bangalore
[2]Professor, Department of CSE-AI, FET Campus,Jain University, Karnataka, India
[3] Student, Computer science and engineering, Jain University, Bangalore
[4]Student, Computer science and engineering, Jain University, Bangalore

Abstract - A robot that uses its wings to fly is referred to as an ornithopter, and its components are designed after the ease of flying of birds and insects. Nevertheless, since its systems are so complex, it has been difficult to attain this achievement. To stay in a state of continuous flight, they modify their set angles, flap rate, and tail position.

Key Words: Flapping-wing unmanned aerial vehicles, biomimicry, random forests, machine learning, reinforcement learning, artificial intelligence, ornithopters, and navigation algorithms

1.INTRODUCTION

But the unmanned ornithopter is an interesting project in itself. Modern unmanned ornithopters give kids great educational experiences and plenty of enjoyment for hobbyists. In 2007, most people misunderstood what they saw as robotic dragonflies as the American government spying on citizens. Whether or not the situation was that way, ornithopters can be employed to carry cameras and other payloads. In biological field research, they have been employed to scare birds away from airport runways, and they could be employed to carry various things from one end of town to the other someday.

Humans have been fascinated by flapping-wing flight for centuries, and truly, it is a challenging field of flight. It is still a relatively unexplored branch of flight today. Insects, birds, and bats were the major sources of inspiration for the first efforts at flappingwing flight. These biological systems have the low Reynolds number, unstable flows that no man-made system can produce.

1.1 Bio-mimicry System

Bio-mimicry is the attempt to duplicate an organism's best qualities. Airflow around the airplane's airfoil-shaped wings is typically put to use by the aircraft, and it may only be varied to a substantial degree at high speeds (the high Reynolds number regime). Nature provides us with flyers capable of completely managing airflow around their wings and even producing their own buoyancy while they fly within a low Reynolds number regime, a peaceful, effectively stagnant airflow regime.

Birds (or ornithopters, as they have been named by researchers as well) and insects are the two broad categories of natural flyers. Insects require more sophisticated flight kinematics to allow them to fly and hover in a flow regime with an extremely low Reynolds number. The current article provides an overview overview of the type of animal utilized in mimicry and its significance as seen in the work of previous authors.

1.2 Scope Of Artificial Intelligence & Machine Learning In Ornithopter

Unlike quadcopters and airplane drones, ornithopters need to stabilize their bodies in wind by flapping wings and positioning their tails in a way that wind never disrupts their flight. An ornithopter can be trained to move its own body by employing machine learning to position its wings and tail in a way that is necessary for optimal flight. In a fight, this will stop the Ornithopter from tilting in any direction with wind force and crashing.

2. LITERATURE REVIEW

2.1 Design, control and application of quadcopter

Ostajic Gordana

Stankovski, Steven

Tejic Branislav

Dkic, Nicola

Unmanned aerial vehicles, or quadcopters, are highly versatile. The design of a quadcopter system and its possible applications will be demonstrated in this paper. We will show and explain the quadcopter building model, basic components with block

diagram, hovering stability, dimensions, and description of basic movements. Experimental demonstration of control algorithms with phases will also be done. After an overview of existing military and civilian applications, possible future ones will be suggested.

Advantages: Quadcopters can hover in the air for quite some time.

The disadvantage is that the quadcopter is unable to fly in strong winds.

2.2 Morphing Aircraft Technology - New Shapes for Aircraft Design

"Weisshaar" and Terrence A.

It came out in 2006.

Multi-mission aircraft dramatically altered externally in form to suit a variable mission environment during flight are called morphing aircraft. This adds capabilities beyond that which would otherwise be unattainable without a change of form. Design of high-performance, aerodynamically configured, shape-morphing aircraft with wings capable of radically changing shape and performance in flight is the objective of morphing efforts. Morphing aircraft, as new mission responsibilities or missions are assigned to them, are relatively more competitive compared to conventional aircraft. This essay will discuss the history of morphing aircraft, outline a recently concluded DARPA program, and cite key enabling technologies for morphing.

Benefit: These planes are faster than other flying drones and can be utilized in the military.

A disadvantage of these planes is that they are expensive and difficult to repair. They are also piloted.

2.3. Ornithopter flight simulation based on flexible multi-body dynamics

Horst Baier, Horst T. Pfeiffer, Andreas T. Pfeiffer, Jun-Seong Lee

2010 saw the publication of

To model the flight of an ornithopter (flapping-wing aircraft), in this paper, a fluid-structure interaction method, a new flapping-wing aerodynamic model, and flexible multi-body dynamics are presented.

From a finite element model created in the finite element analysis software ANSYS to the multi-body dynamics software MSC.ADAMS, the flexible parts of an ornithopter simulated model were utilized.

Flapping wing aerodynamics were modeled with an improved version of modified strip theory.

Advantage: It describes the fundamentals of a bird's flight.

A limitation is that it lacks a control system as it is a basic model.

3.DESIGN

3.1. Reinforcement algorithm For Improved Flight performance

Step 1: Initialize the RL algorithm with a list of potential actions like changing the angle of attack, wing shape, or wing flapping frequency.

Step 2: Modify the RL algorithm's estimate of what reward each action predicts based on seeing the reward of that action.

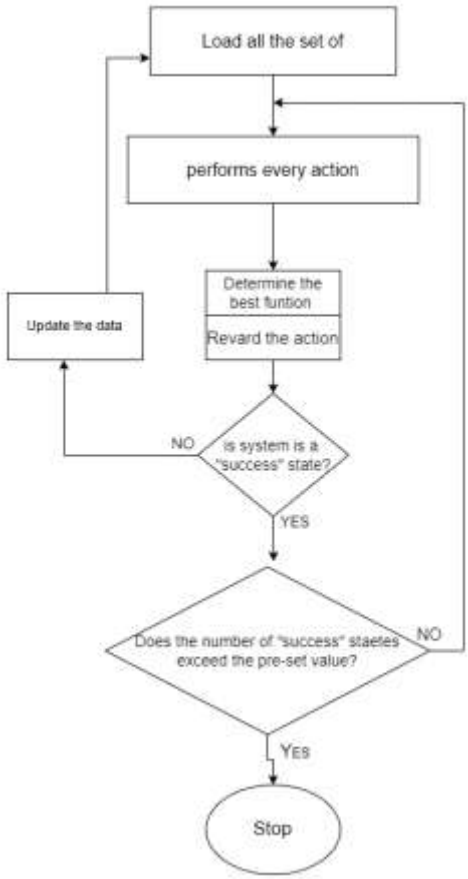
Step 3: Select the action with the highest estimated payoff in the present circumstance based on the revised estimates.

Step 4: Document the reward that is attained by modifying the behavior of the ornithopter according to the chosen action.

Step 5: Repeat steps 4 and update the RL algorithm's estimates based on the reward seen.

Step 6: until, as displayed in figure 3.1, the algorithm has converged toward the optimal behaviour.

Step 7: Implement the learned RL algorithm within the ornithopter for autonomous flight management and improvement in flight capabilities.



3.2. autonomous navigation algorithm

Step 1: Perception: Autonomous vehicles make use of a combination of sensors such as GPS, LIDAR, RADAR, and cameras to perceive the surroundings. Perception algorithms interpret this sensor data to identify landmarks, obstacles, and objects nearby.

Step 2: Localization: Localization algorithms calculate the vehicle's position relative to its environment. They make estimates of vehicle position from sensor readings and a map of the environment.

Step 3: Chartin A chart of the environment is created by mapping algorithms and utilized by the vehicle in planning the route.
Coordinates of landmark points, barriers, and other points of interest can be marked on the chart.

Step 4: Path Planning: Path planning algorithms calculate the best path along which the car must travel to reach its destination. They consider the environment, the position of the car, and whether there must be any obstacles.

Step 5: Management

To stay on the planned course and steer clear of any hindrances, control algorithms modulate the vehicle's speed, direction, and other parameters.

Step 6: Decision-Making As can be seen in figure 3.2, decision-making algorithms use a variety of criteria, including time, safety, and energy efficiency, in making more abstract decisions on where to go, what to do, and how to do it.

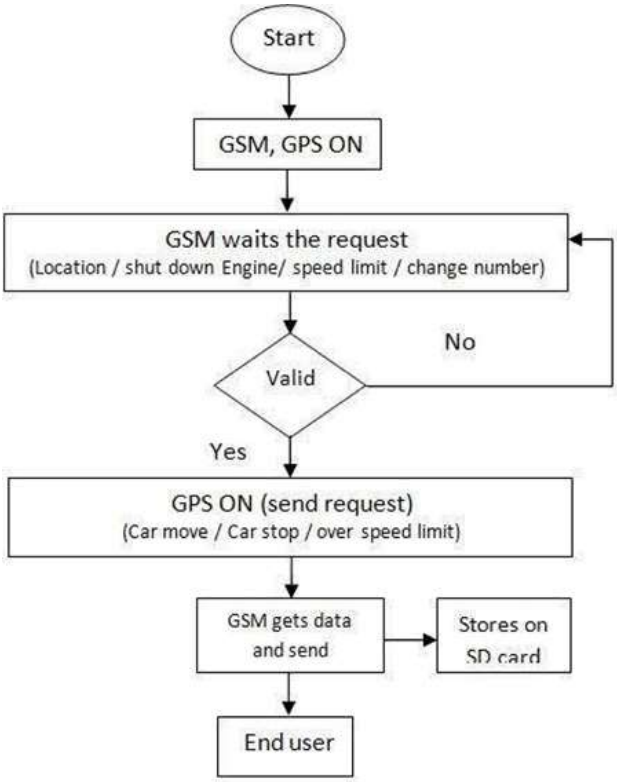


Figure 3.2 Navigation algorithm

3.3.

Figure 3.1 Reinforment algorithm

Random Forest algorithm

Step1:Collect sensor and instrument readings from the ornithopter, such as accelerometers, gyroscopes, and cameras.

Step 2: Remove noise, outliers, and missing values by preprocessing the data.

Step 3: Split the data into training and testing sets.

Step 4: Set the parameters of the random forest algorithm, including the number of trees and tree depth.

Step 5: Train the random forest classifier on the training data, with the data features as input and the target outputs as labels.

Step 6:Use the trained model to make predictions on the test data.

Step 7: Compare the actual and expected outputs to determine the performance of the model.

Step 8: Adjust the random forest algorithm's parameters and repeat the training and testing process if the result is not satisfactory.

Step 9: Use the model learned to predict new data collected by the ornithopter.

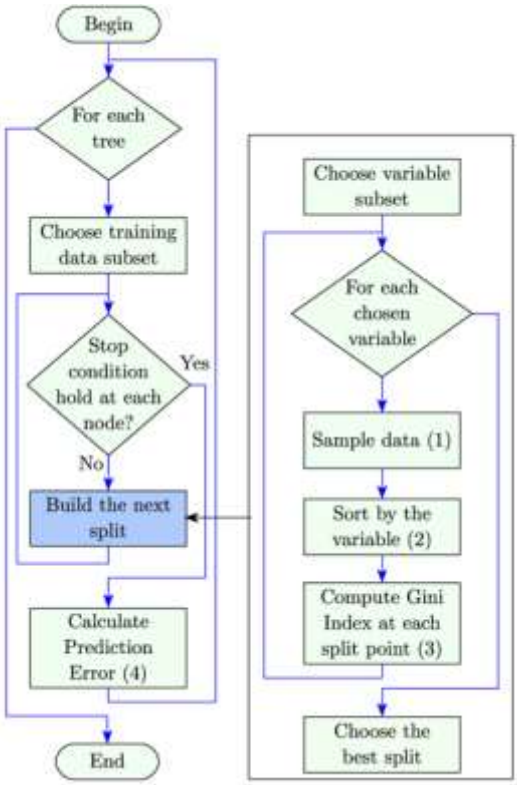


Figure 3.3 Random Forest algorithm

ARCHITECTURE DIAGRAM:

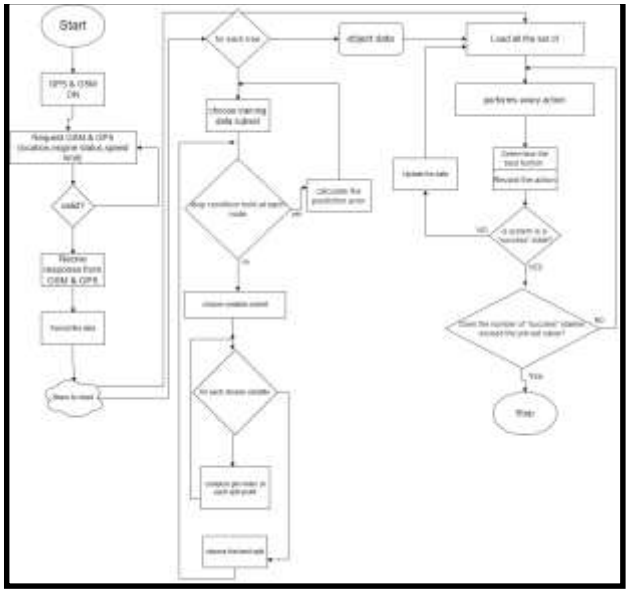


Figure 3.4 Combination of Algorithms

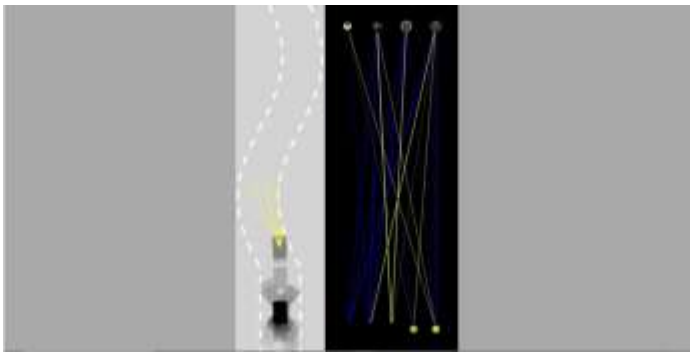
3.1 , 3.2 and 3.3

Navigation algorithm is utilized by the master program to determine the final destination of the ornithopter and to provide the cloud platform and the user with the real-time location of the ornithopter. As soon as the end destination is determined, the ornithopter initiates object detection using the Random Forest algorithm to prevent collisions with objects nearby. Information is passed on to the small memory to compute the flight path, regulate the flight path by modifying the wing and tail angles, and utilize the Reinforcement algorithm to learn from mistakes. All of this is terminated as soon as the ornithopter arrives at its final destination. (See figure 3.4.)

4 TESTING

By applying testing techniques, a system or a component is tested to check if it is meeting the requirements. Defects, faults, or incomplete requirements other than the actual requirements can be detected through system testing. Testing techniques are the best practices adopted by the testing team to test the software developed against some standards. These techniques assist in ensuring the overall quality of the software or product, i.e., its usability, security, and functionality.

Figure 4 - Testing The Object



Types of Testing Techniques:

4.1 Black Box Testing:

Software testing that focuses on the way a system works and not on how it works internally is referred to as "black box testing." For testers, inputs and outputs of the system are more important than the software's implementation or code. To detect faults and confirm that the system is working as expected, they employ a variety of techniques, including boundary value analysis, equivalency partitioning, and decision table testing. Black box testing assists in detecting errors that can be caused by incorrect or unforeseen inputs, making the system reliable and operational from the perspective of the end-user.

4.2 White Box Testing:

White box testing, or structural testing or clear box testing, is a technique for testing software that tests the internal structure of a system. The code, its use, and the system's response to this usage are all of interest to testers. They employ techniques such as statement coverage, branch coverage, and path coverage to make sure that all lines of code are executed and all possible outcomes are covered. Bugs due to coding errors or unintended interactions among different system components can be identified with the help of white box testing. It also assists the developer in making sure that the system is strong and reliable.

4.3 Unit Testing:

Unit testing is a form of software testing where individual units or parts of a system are tested separately from the system as a whole. Ensuring that each unit or part is working as intended and meets its design specifications is the objective. The tests are applied to short code segments, such as functions or methods, and are usually automated. By detecting defects early in the development cycle, unit testing can avoid wasting time and money to repair problems later. It also serves to raise the overall quality and maintainability of the system by ensuring each module is reliable and robust.

4.4 Integration Testing:

To guarantee that several system modules or parts function correctly, software testing referred to as "integration testing" brings together and evaluates all of them simultaneously.

Identification of errors in how various modules communicate with each other and verifying that the system overall complies with design requirements are the aims. Integration testing may be performed in a number of modes such as top-down, bottom-up, or a combination of the two. The tests may examine a number of attributes of the system's architecture and may either be automatic or manual. Integration testing is a critical phase in ensuring a system is reliable and working before it is released to customers.

4.5 System Testing:

As part of a software testing procedure called system testing, the entire integrated system is tested to make sure it satisfies both functional and non-functional specifications. Ensuring the system works as expected in general and that each component works as expected when used together is the objective. A system may be subjected to a range of tests, including functional, performance, security, usability, and compatibility tests. The tests may be done manually or automatically in any number of configurations to verify that the system is operating in an assortment of configurations. There is a need for system testing to verify the system is dependable and suitable for user requirements.

4.6 Acceptance testing:

Acceptance testing, which is a form of software testing, ensures the system satisfies the acceptance and needs of the stakeholders. The aim is to ensure the system fulfills end-users' expectations and is deployable. Acceptance testing, which might include functional, performance, usability, and security testing, might be conducted by the stakeholders or a specific testing team. In order to make sure that the system works as it should, the tests can be executed manually or automatically in a variety of configurations. Acceptance testing is an important phase in guaranteeing customer satisfaction and minimizing the risk of problems after deployment.

4.7 Regression Testing:

Regression testing, a software test, is utilized to check if modifications or extensions to a system do not unknowingly influence its current functionality. The aim is to ensure that the system continues to perform as designed after modifications have been made. System testing, integration testing, and unit testing are among the layers on which regression testing may be applied. The tests can either be automated or manual, and most of the time, they consist of re-running the system's operation to ensure that no defects have been introduced. Regression testing is critical in ensuring the quality and reliability of the system over time without any negative impacts on users or business operations.

4.8 Performance testing:

Performance testing is the act of monitoring the system's performance under a range of load conditions to make certain that it fulfills its performance criteria. Identifying any performance bottlenecks, for example, those introduced by resource usage, scalability, or delayed response times, is the objective. Performance testing can be performed through many different methods, for example, load testing, stress testing, endurance testing, and spike testing. To simulate real-world conditions, the tests are executed manually or automatically and in a variety of configurations. To make sure that the system is capable of handling the anticipated load and offer a satisfactory user experience in normal as well as high load circumstances, performance testing is necessary.

4.9 Security Testing:

Detection of weaknesses and risks in software is achieved by means of security testing. The purpose is to make it a point that the system is safe and can withstand any intrusions or unauthorized usage. Security testing can be accomplished in a variety of manners such as risk analysis, vulnerability scans, and penetration testing. The tests are either automated or human and may address a broad spectrum of security topics, including network security, application security, data security, and access control.Security testing is vital to ensure the availability, confidentiality, and integrity of the system as well as to detect and fix security problems. Learning, test design, and test execution that happen simultaneously are all part of exploratory software testing. The goal is to rapidly give feedback on new or rapidly evolving features or to detect defects that scripted testing may not catch. Rather than adhering to a pre-defined test plan, exploratory testing employs the tester's subject matter expertise, experience, and imagination to create and run test cases in real time. Testing tools may be utilized to run the tests automatically or manually. Exploratory testing improves the system quality overall, provides higher test coverage, and determines issues early in the development cycle.

5.RESULTS AND DISCUSSION

5.1 Reinforcement Algorithm

1.NeuralNetwork Class: A neural network is initialized with a given architecture (neurons for each layer) using the Class NeuralNetwork.
• Provides methods for changing the network (mutate) and performing out feedforward inference (feedForward).

2. Level Class: Refers to a single layer of neurons in a neural network.
• Randomly initializes the weights and biases.
It performs feedforward computation, where the output is derived by multiplying inputs by weights, adding them with biases, and then passing them through a step function.

3. Initialization: The constructor initializes weights and biases randomly while looping through the provided neuron counts to create levels in a neural network (new NeuralNetwork(neuronCounts)).

4: Feedforward Inference: You call NeuralNetwork to perform feedforward inference. givenInputs, network; feedForward.
• The process forwards outputs from one network level as inputs to the next, looping through each level of the network.
• Inputs are weighted, summed up with biases, and then passed through a step function at each level to produce outputs.

5.Mutation NeuralNetwork: The weights and biases of the network are randomly changed using the mutate(network, amount) method.
As it traverses the levels of the network, it adds random noise to each weight and bias.

6.The Level feedForward:(givenInputs, level) method multiplies inputs by weights and adds biases for every neuron. This is the sixth step in the feedforward process (Level Class).

Neuron biases are used to compare the obtained sums. The neuron fires (output is 1) if it is greater; if not, it does not (output is 0). All things considered, this application offers a simple JavaScript implementation of a feedforward neural network that can make inferences and change.

5.2. Autonomous navigation algorithm

Description of the Code: Description of the Code: Importing Required Libraries:The `Nominatim` class is imported from `geopy.geocoders` at the beginning of the code. Geocoding, i.e., converting addresses to geographic locations, is performed using this class. The `get_lat_long` function is defined in the following way: The function argument `get_lat_long(city)` is the city's name. - The agent name defined by the user (`geoapiExercises`) is utilized to instantiate a `Nominatim` geolocator object. It gets the location information (latitude and longitude) of the given city through this geolocator Finally, it returns a tuple containing the latitude and longitude of the city.

Obtaining Latitude-Longitude and Plugging in the City:The value `"New York"` is assigned to the city variable.The city variable is used as a parameter to the `get_lat_long()` function.The `latitude` and `longitude` variables are assigned using an unpacking of the returned latitude and longitude data.Results Printing:The console prints the latitude and longitude readings. Input: Python city = "New York"By using the name of a city, this Python code fetches its latitude and longitude coordinates through the geopy package.It defines a function get_lat_long(city) to fetch the coordinates for a given city, loads the Nominatim geocoder from geopy.geocoders, and prints the latitude and longitude values that the method yields.

OUTPUT: Latitude: 40.7127281 Longitude: -74.0060152

The output code displays the latitude and longitude coordinates of the city "Hubli" obtained using the get_lat_long function. This allows you to see the city of "Hubli's" exact geographic coordinates (latitude and longitude). The code demonstrates how to utilize the Nominatim geocoder to obtain geographic coordinates (latitude and longitude) for a particular city using the Geopy module in Python. The coordinates of New York City are especially fetched in this case.
Then, the console prints out the readings of latitude and longitude.

5.3. Random Forest algorithm

Code description: The code utilizes the scikit-learn, numpy, and pandas packages to create a synthetic dataset. After the dataset has been divided into a training set and a testing set, a Random Forest Regressor model is created and fitted, predictions are calculated on the test set, the performance of the model is checked with Mean Squared Error, and the output is printed.
INPUT: The code given uses a Random Forest Regressor to predict a target variable based on the input features. The initial step is to generate a synthetic dataset with the target variable and the features like "roll," "pitch," "altitude," "airspeed," and "yaw." The dataset is then divided into training and test sets using a 70-30 ratio. The training set is utilized to train the Random Forest Regressor model once it has been built with 100 decision trees. The performance of the model is then assessed using the Mean Squared Error metric after making predictions using the testing set.
The output of the code is the Mean Squared Error figure, which determines the average squared difference between actual and predicted target values. Here, the Mean Squared Error is approximately 864.83. This measure illustrates how well the Random Forest Regressor model predicts the desired variable based on the input data.

6.CONCLUSION

Ornithopters, inspired by the structure and motion of insects and birds, are a major breakthrough in aerial robotics. Its capacity to "flap" its wings makes it more flexible and energy-efficient compared to traditional fixed-wing or rotary drones. We have demonstrated the potential for creating more autonomous and flexible aerial systems through the principles of biomimicry and advanced Artificial Intelligence (AI) and Machine Learning (ML).

The ornithopter is capable of adapting to the environment, enhancing its flight trajectories, and responding to external perturbations such as wind through reinforcement learning. The navigation algorithm provides reliable path planning for optimal motion and localization, and the random forest model aids in sensor-based obstacle avoidance apart from decision-making.

This interdisciplinary design not only enhances the ornithopter's flight control and stability but also extends its real-world applications, including environmental monitoring, military reconnaissance, and search and rescue operations in regions inaccessible to conventional aircraft.

In addition, such a system is of tremendous educational significance as it allows researchers and students to test real-world AI applications in robotics.

Growing autonomy using deep learning-driven visual systems, reducing dependence on remote servers using edge computing, and increasing energy efficiency using solar integration and lightweight materials are a few potential research areas in the future. Ornithopters, with their perfect combination of cutting-edge technology and design elements inspired by nature, can become a key part of future aerial systems with further research.

7. REFERENCES

1. *Nikola Dukic, Branislav Tejic, Gordana Ostojic, and Steven Stankovski. "Quadcopter design, control, and application," Terrence A. Weisshaar, "Morphing Aircraft Technology: New Shapes for Aircraft Design," 2015. Sumaila Musa, "Quadcopter Modeling and Design Techniques," 2006 2017*

2. *Jae-Hung Han, Horst Baier, Andreas T. Pfeiffer, and Jun-Scong Lee, "Ornithopter flight simulation based on flexible multi-body dynamics," Joon Hyuk Park and Kwang-Joon Yoon (2010) "Creating a Biomimetic Ornithopter with Controllable and Sustained Flight," 2008*

3. *Ramesh NG, Balaji J, Dilip Kumar N, and Dharshan L appeared in the 2020 volume of the International Research Journal of Engineering and Technology (IRJET).*

4. *Design and Construction of an Autonomous Ornithopter by Zachary J.on Jackowski, 2009*

5. *Mohamed Lalami, Andre Preumont, Matej Karasek, Alexandre Hua, Yanghai Nan, and "Pitch and Roll control mechanism for a hovering Flapping Wing MAV," 2014*

6. *"Nonlinear Time-Periodic Models of the Longitudinal Flight Dynamics of Desert Locusts Schistocera Gregaria," Graham K. Taylor and Rafal Z. Bikowsi, 2005*

7. *"Dynamics, Stability, and Control Analysis of Flapping Wing Micro-Air Vehicles," Christopher T. Orlowski, 2012.*