1.0 Scope

This report outlines the design and implementation of outer-loop control algorithms for Manta using a simulation-based approach. The scope includes developing autonomous takeoff, trajectory tracking, and obstacle avoidance strategies within the PX4-ROS-Gazebo environment. Key tasks involve integrating ROS-based controllers with PX4 via MAVROS, validating control logic using simulated sensors in SITL, and ensuring the framework is adaptable for future deployment on the Manta. In addition, this report covers project management strategies, economic and social factors, and bio-inspiration concept.

Revision History

Version	Date	Description
A	07/04/2025	Initial Release
B	08/04/2025	Final Report Revised with Peer Suggestions

Table of Contents

2.0 Nomenclature

UAV	*Unmanned Aerial Vehicle*
GNC	*Guidance, Navigation and Control*
PID	*Proportional Integral Derivative*
ROS	*Robot Operating System*
SITL	*Software in the Loop*
HITL	*Hardware in the Loop*
GPS	*Global Positioning System*
IMU	*Inertial Measurement Unit*
EKF	*Extended Kalman Filter*
ENU	*East-North Up*
NED	*North-East Down*
XML	*Extensible Markup Language*
SDF	*Simulation Description Format*
PD	*Proportional Derivative*
k_{att}	*Attractive scaling factor*
k_{rep}	*Repulsive scaling factor*
α	*Step-size*
ρ_o	*Influence radius*
q_{new}	*Updated position*
q_{old}	*Current Position*
LiDAR	*Light Detection and Ranging*
APF	*Artificial Potential Field*

3.0 Background and Motivation for GNC Development

During the Fall term, initial efforts focused on evaluating guidance, navigation, and control (GNC) strategies using conventional fixed-wing aircraft models such as the Sky Hogg and Fly Beaver in MATLAB. These models helped in developing a foundational understanding of aircraft dynamics, actuator limitations, and control surfaces. However, those conventional aircraft models lacked key features: they offered limited real-time support, no closed-loop sensor integration, and incompatible Software-In-The-Loop (SITL) support. Thus, they were not suitable for developing and testing advanced outer-loop control strategies.

To address these constraints, a more interactive and modular simulation environment using the Robot Operating System (ROS) combined with PX4 and Gazebo was adopted. This setup provided the necessary infrastructure to implement and validate GNC algorithms in a flexible, high-fidelity setting.

4.0 ROS Introduction

The Robot Operating System (ROS) is an open-source middleware framework designed to support the development of robotic applications through a modular, distributed architecture. It provides a communication infrastructure that enables seamless data exchange between different software components, referred to as nodes. These nodes can be developed in multiple programming languages, primarily C++ and Python, allowing for flexible integration of algorithms and hardware interfaces. C++ was used for this project.

It operates as a peer-to-peer network where nodes communicate by publishing and subscribing to named message channels called topics. For example, a sensor node may publish data such as IMU (Inertial Measurement Unit) or GPS (Global Positioning System) readings to a topic, while a control node subscribes to receive and process that data. In addition to topics, ROS also provides services for synchronous communication and actions for long-duration tasks that require feedback and result handling.

This modular and distributed design makes ROS highly suitable for algorithm development. In this project, ROS was used to integrate external controllers, manage sensor data, and

interface with the PX4 flight control system, enabling real-time simulation for autonomous flight within a realistic virtual environment [1].

4.1 Concepts

ROS is built on several key concepts that define how different components of system interact [1]:

- **Nodes**: Individual processes that perform specific tasks, such as reading sensor data, running control algorithms, or sending commands.
- **Topics**: Named communication channels used to send messages between nodes. Nodes can publish to or subscribe from topics to share real-time data.
- **Messages**: Structured data exchanged over topics. These define the format of information being transmitted, such as position, velocity, or sensor readings.
- **Services**: Used for request-response communication between nodes. A service is called when a task must return a result immediately, such as checking the system status.
- **Actions**: Like services but used for longer tasks that require continuous feedback and a final result, such as navigating to a waypoint or landing.

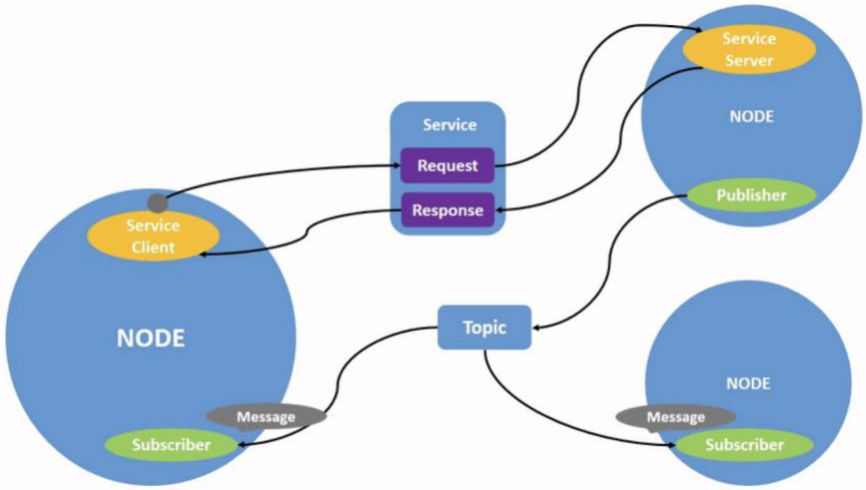


Fig.1: ROS Subscriber - Publisher Model [1]

Figure 1 illustrates the ROS publish–subscribe communication model, where nodes interact through topics to exchange real-time data such as sensor inputs and control outputs.
Understanding these concepts is crucial, as for GNC implementation ROS topics are used to transmit sensor data and control commands between the PX4 flight controller and custom algorithms.

5.0 PX4 and Its Integration with ROS

PX4 is an open-source flight control software (autopilot) stack that provides guidance, navigation, and control algorithms for autonomous vehicles. In this project, PX4 was chosen for its real-time simulation support, compatibility with the Gazebo simulator, and its integration with ROS [2].
The PX4 firmware allows offboard control through ROS using MAVROS, enabling user to send high-level commands and receive telemetry data within the ROS ecosystem. This modular and open-source design makes PX4 particularly suitable for testing autonomous flight algorithms [3]. Details on the system-level integration of PX4, ROS, and Gazebo are provided in Section 6.5. Section 6 outlines the simulation environment, vehicle models, and world configurations used.

6.0 Simulation Tools

Simulation was an essential component for guidance, navigation and control (GNC), enabling the development and verification of control algorithms in a virtual environment. The simulation setup used PX4's SITL capability in conjunction with Gazebo and ROS. This allowed the PX4 flight stack to operate as it would on actual hardware while interfacing with simulated sensors and environments [4].
Gazebo was selected as the primary simulation platform due to its robust physics engine, seamless ROS integration, and capability to emulate realistic environments and sensors. The SITL mode runs the PX4 firmware natively on the host computer and establishes communication with Gazebo using a UDP connection, allowing actuator commands and sensor feedback to be exchanged in real time [5].

6.1 Gazebo

Gazebo is a 3D robotics simulator that provides realistic modeling of physical interactions, including gravity, collisions, and sensor noise. It supports a wide range of robot models and includes plugins for commonly used sensors such as cameras, IMUs, LiDAR, and GPS. Gazebo's seamless integration with ROS enables data from simulated sensors to be published directly to ROS topics, making it possible to test control algorithms without modification [5].

Gazebo served as the primary simulation platform, replicating a controlled virtual environment where the PX4 Iris UAV model could be tested. The simulator was used to evaluate takeoff, trajectory tracking, and obstacle avoidance behavior.

6.2 UAV and Sensor Model

The PX4 SITL package provides a UAV model of the 3DR Iris, which is commercially available multi-copter [6]. This model was chosen simply because of its modular and flexible nature.

Fig.2. 3DR Iris Drone [6]

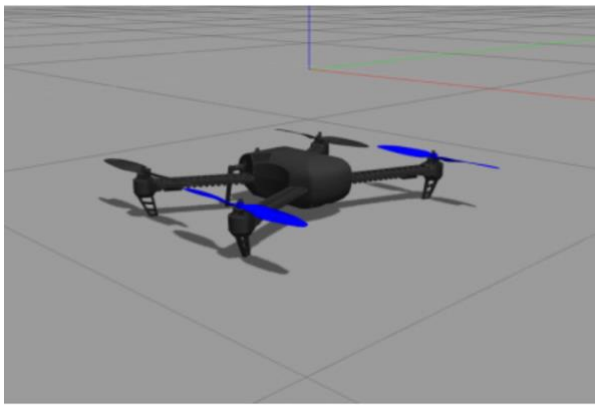


Fig.3. PX4 Iris in Gazebo [7]

This quadrotor model comes with a pre-configured sensor suite and is widely used in PX4 SITL environments due to its simplicity and compatibility with ROS-based tools.

The key onboard sensors emulated in the model include [2]:

- IMU (Inertial Measurement Unit): Simulates accelerometer and gyroscope data for attitude control.
- Magnetometer: Provides magnetic heading data.
- Barometer: Supplies pressure-based altitude information.
- GPS: Simulates global positioning data (latitude, longitude, altitude).

Sensor readings were made accessible through ROS interfaces using MAVROS, enabling external controllers to receive real-time pose, velocity, and altitude data. Internally, PX4 processed this data using the EKF2 (Extended Kalman Filter) estimator in the local NED frame [2].

6.3 World Model

In Gazebo, a world model represents the simulated environment where robots, sensors, and objects interact. It is defined using a world file, which is an XML-based format following the Simulation Description Format (SDF) specifying various elements of the environment, such as terrain, lighting, physics properties, and static or dynamic objects [8]. For this project, an empty world was used initially to validate basic behaviors such as takeoff and trajectory tracking. Additional objects and obstacles were later added for APF-based testing. Figure 4 shows an empty world in Gazebo.

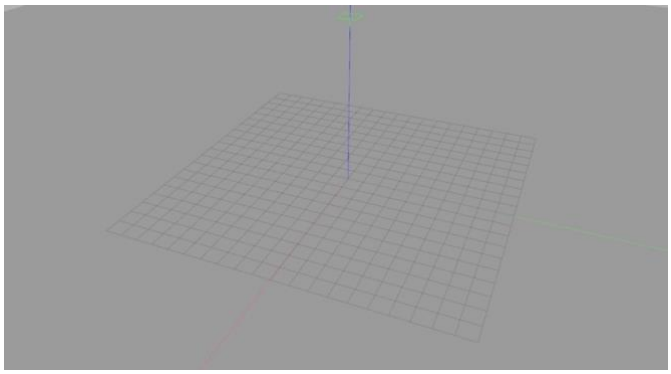


Fig.4. Empty world in Gazebo [8]

6.4 Reference Frames

One challenge in simulation was to align reference frames between different systems. Gazebo uses the East-North-Up (ENU) coordinate frame, while PX4 operates in the North-

East-Down (NED) frame. To remedy this difference, a transformation must be applied when commanding for position and velocity [9].

The conversion between ENU and NED frames can be handled using a rotation matrix:

$$R_{NED}^{ENU} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \quad (1)\ [9]$$

6.5 ROS with Gazebo Simulation for PX4

The simulation setup was built entirely on PX4's Software-in-the-Loop (SITL) configuration. This mode allows the PX4 autopilot to run on a host computer while interacting with the Gazebo simulator to model the drone's physical dynamics and sensor feedback. Communication between PX4 (flight control) and Gazebo (simulator) is handled by the simulator_mavlink.cpp interface, which relays MAVLink messages to and from the simulated environment.

To bridge PX4 with the ROS ecosystem, MAVROS is used as an intermediary. MAVROS translates MAVLink messages into ROS-compatible topics and services, allowing ROS nodes to publish commands and subscribe to flight telemetry. For example, ROS streams position or velocity setpoints to PX4, while PX4 returns real-time feedback such as vehicle state for monitoring and control purposes.[10].

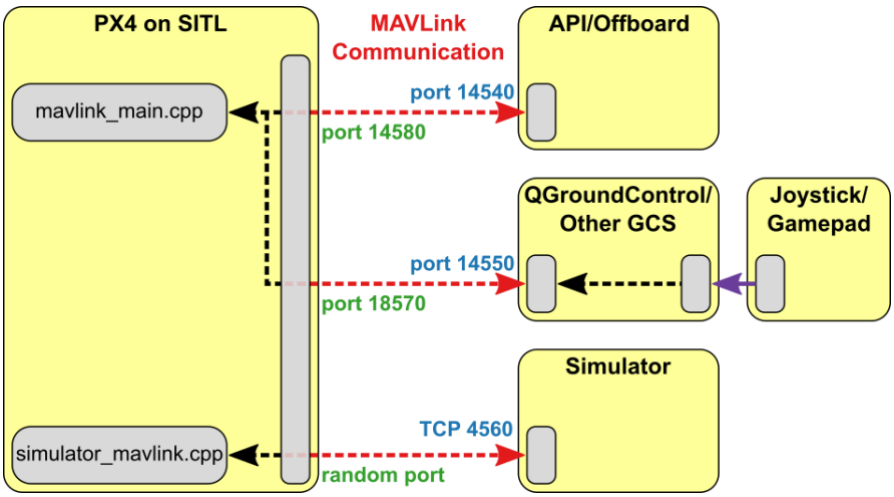


Fig.5. ROS with Gazebo Simulation for PX4 [10]

Figure 5 illustrates the data flow among PX4, MAVROS, ROS nodes, and the Gazebo simulator. This fully virtual environment facilitates testing of autonomous GNC algorithms with realistic sensor models and flight behavior.

7.0 Take off Simulation

The take-off phase was implemented using a ROS node that commands the PX4 drone to ascend and hover at a predefined altitude of 2m. This control is achieved by entering offboard mode, where external commands override the autopilot's internal flight logic.

To begin the takeoff, a ROS node first sends commands to arm the vehicle and switch to offboard mode. In this context, arming refers to enabling the PX4 Iris's motors, allowing it to accept and execute control commands.

It then continuously publishes position setpoints that specify a target altitude. These updates are sent at a frequency higher than 2 Hz to maintain offboard control, as required by PX4's safety logic.

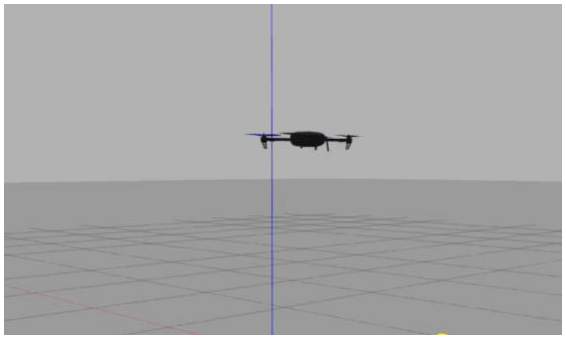


Fig.6. Drone at a take-off height of 2 m

The messages and services used in this process are summarized in Table 1. They include state monitoring, mode switching, arming services, and the position setpoint publisher, which together enable autonomous vertical flight. [11]

Table 1: ROS Interfaces Used for Take-Off Simulation [11]

Interface Type	Description
Vehicle State Monitor	Monitors current PX4 mode and arming status
Offboard Mode Service	Switches the drone to offboard mode
Arming Service	Arms Iris for flight
Position Setpoint Publisher	Continuously sends desired position to PX4 via MAVROS

8.0 Trajectory Controller

Following successful take-off and hover, a trajectory tracking controller was implemented to command the drone to follow a predefined path in simulation. Similar to take-off, by sending velocity commands to PX4 in offboard mode. This outer loop determined the drone's desired linear and angular velocities based on its current state relative to the target trajectory.

The system used the drone's real-time position, received via MAVROS, to compute the velocity vector needed to reduce the positional error between the drone and the next waypoint. A Proportional-Derivative (PD) control law was applied to determine these velocity commands, which were published to PX4 through MAVROS interfaces. This allowed the drone to adjust its motion smoothly and maintain a stable trajectory in response to dynamic changes.

To ensure reliable state estimation, the drone's position was inferred from simulation feedback, and the controller relied on constant pose updates from the MAVROS local position topics. Although a Kalman Filter was available for sensor fusion in PX4, this simulation setup did not require additional estimation, as the SITL was for noise-free feedback.

The controller was tested on a circular trajectory, where the drone was commanded to follow the trajectory in the XY-plane. Figure 7 shows the circular trajectory where green is desired while red is actual trajectory. The performance demonstrated accurate tracking, responsive control, and stable flight behavior, validating the control loop before introducing external disturbances such as obstacles

Table 2: Parameters used

Takeoff Height	1.0 m
Radius	1.0 m
Velocity	1.0 m/s
Proportional Gain	0.9
Derivative Gain	0.3

Table 3: ROS Topics for Trajectory Tracking [12]

/mavros/local_position/pose	Provides real-time position and orientation
/mavros/state	Monitors current flight mode and status
/mavros/setpoint_velocity/cmd_vel	Publishes desired linear and angular velocity

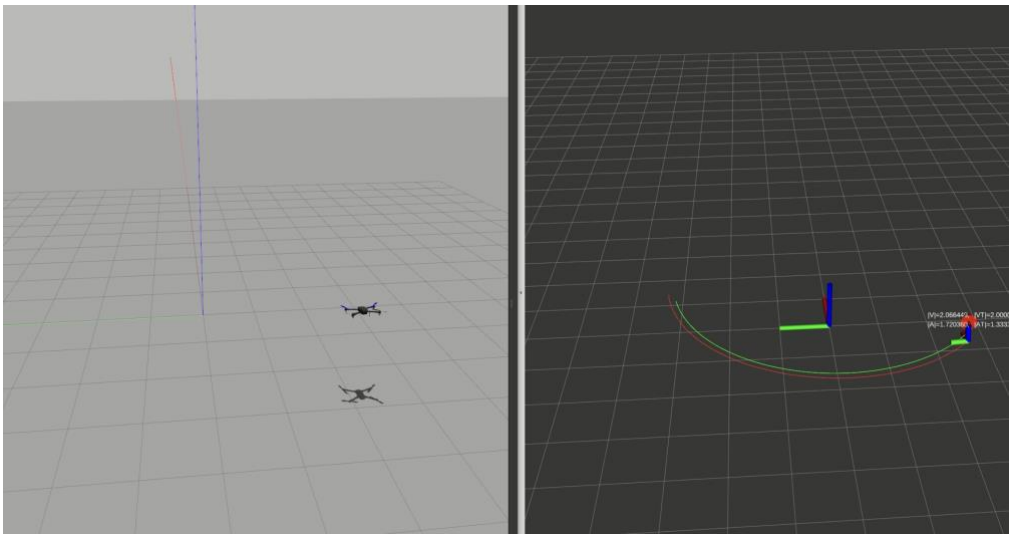


Fig.7: Circular Trajectory Controller

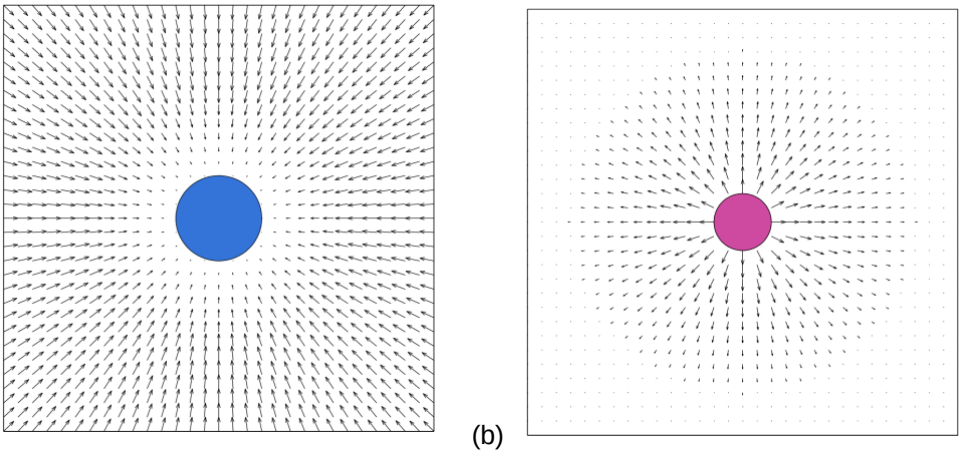
9.0 Obstacle Avoidance Algorithm

Obstacle avoidance is a critical component of autonomous navigation, especially in dynamic environments. A reactive method known as the Artificial Potential Field (APF) algorithm was implemented to facilitate real-time obstacle avoidance. The APF approach enables the vehicle to navigate toward its target while avoiding collisions with obstacles by simulating the environment as a continuous potential field. Given Manta's requirement to operate autonomously in low-visibility conditions and land on unprepared terrain, APF offers a lightweight, reactive solution for local obstacle avoidance without relying on detailed maps or extensive onboard computation.

9.1 Artificial Potential Field (APF)

The Artificial Potential Field method models the navigation environment as a continuous scalar field in which the UAV experiences virtual forces. These forces are derived from potential functions [13]:

- Attractive potentials draw the vehicle toward a predefined goal.
- Repulsive potentials push the vehicle away from obstacles within a certain threshold distance.



(a) (b)

Fig.8 (a) Goal attractive and (b) obstacle repulsive potential field [13]

As illustrated in Figures 8 the goal is modeled as a sink generating an attractive field, whereas obstacles act as sources of repulsive potential. The resulting force vector at each point in space is computed as the negative gradient of the combined potential field:

$$F(q) = -\nabla U(q) \qquad (2)\ [13]$$

where $F(q)$ is the artificial force acting at position and $\nabla U(q)$ is the gradient of the potential field.

9.2 Implementation for Point-Mass Navigation

To validate the APF algorithm, a MATLAB-based simulation was implemented for a point mass navigating in a 2D environment. The agent was required to move from a start position at (50, 350) to a goal at (400, 50) while avoiding two rectangular and two circular obstacles.

9.2.1 Attractive Potential and Force

Attractive potential field is given by: $U_{att}(\rho) = \frac{1}{2} \cdot k_{att} \cdot \rho^2$ (3) [13]

The corresponding attractive force is: $\nabla U_{att}(\boldsymbol{\rho}) = k_{att} \cdot (\boldsymbol{\rho} - \boldsymbol{\rho}_{final})$ (4) [13]

where k_{att} is scaling factor, ρ is the Euclidean norm to the goal, and the $\boldsymbol{\rho}_{final}$ is position of the goal

9.2.2 Repulsive Potential and Force

Repulsive potential field is given by: $U_{rep}(q) = \begin{cases} \frac{1}{2} \cdot k_{rep} \cdot \left(\frac{1}{\rho(q)} - \frac{1}{\rho_o}\right)^2 & : \rho(q) \leq \rho_o \\ 0 & : \rho(q) > \rho_o \end{cases}$ (5)[13]

The repulsive force is: $\nabla U_{rep}(\boldsymbol{q}) = \begin{cases} -k_{rep} \cdot \left(\frac{1}{\rho(q)} - \frac{1}{\rho_o}\right) \frac{\cdot \nabla \rho(q)}{(\rho(q))^2} & : \rho(q) \leq \rho_o \\ 0 & : \rho(q) > \rho_o \end{cases}$ (6)[13]

where k_{rep} is scaling factor, $\rho(q)$ is the distance from the nearest obstacle and ρ_o is minimum threshold distance to avoid obstacle.

9.2.3 Total Potential Field

The net behavior of the agent is governed by the total potential field, which is a superposition of the attractive and repulsive components:

$$U_{att} + U_{rep} = U_{total} \quad (7)\ [13]$$

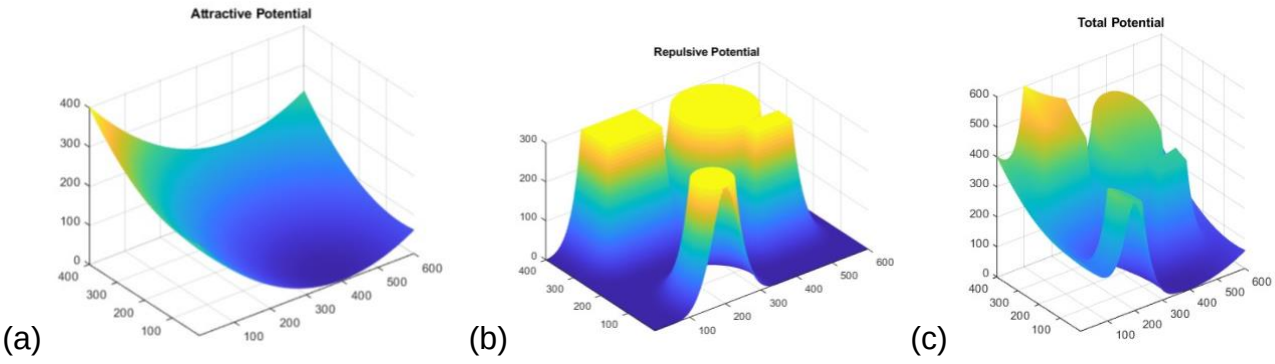


Fig.9: (a) Attractive Potential Field (b) Repulsive Potential Field (c) Total Potential Field in MATLAB

This combined field defines a navigation landscape where the agent naturally flows away from obstacles while being pulled toward the goal. Figure 9(c) shows the 3D visualization of this total potential field.

9.2.4 Gradient Descent and Path Generation

This path planning strategy can be viewed as an optimization problem, where the goal is to minimize the total potential function. This is achieved using an optimization technique called gradient descent.

In this context, gradient descent is used to iteratively update the agent's position by moving in the direction of the steepest decrease in potential:

$$q_{new} = q_{old} - \alpha \cdot \nabla U|q_{old} \qquad (8)\ [13]$$

where, q_{new} is updated position, α is the step-size, ∇U is the gradient of total potential field and q_{old} is the current position.

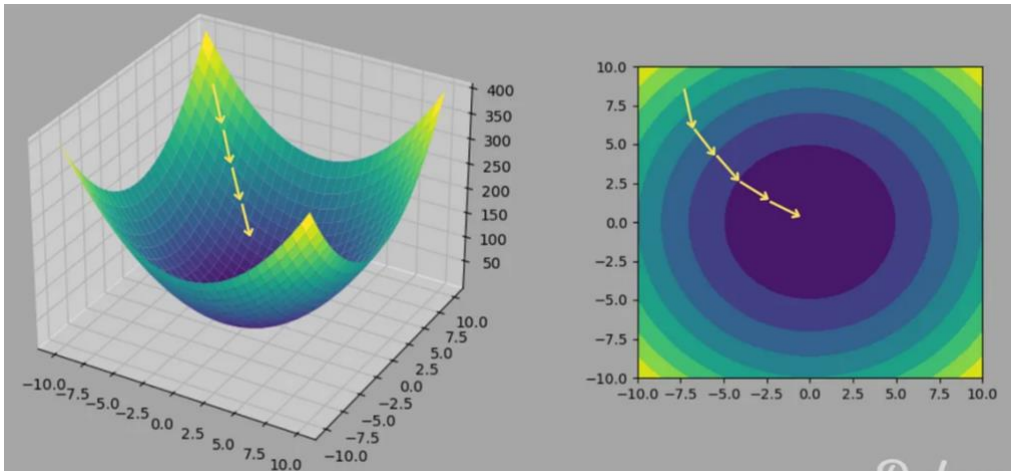


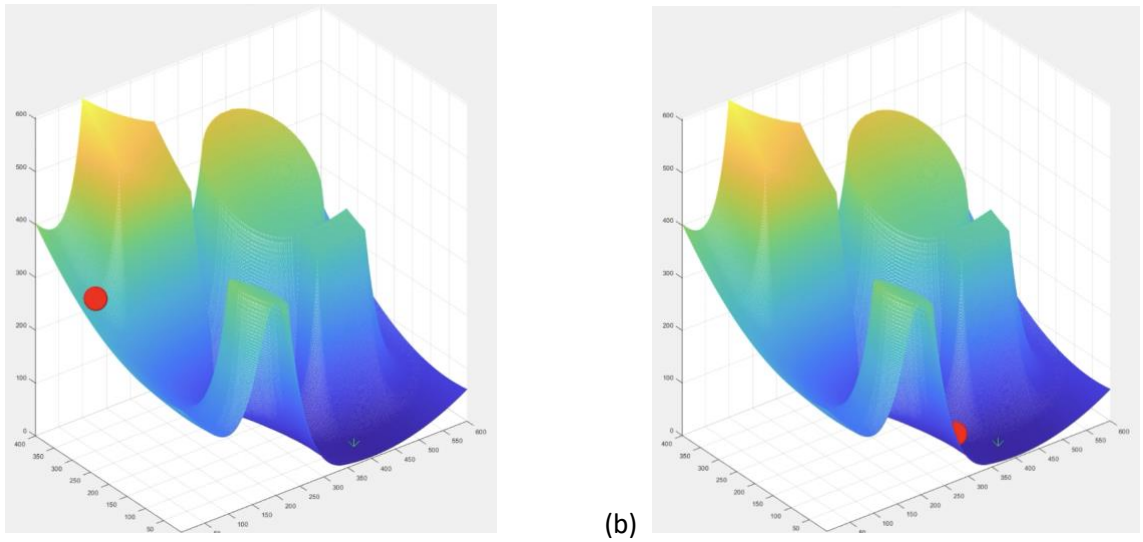
Fig.10: Gradient Descent with 3D and 2D plane [14]

Figure 10 visualizes gradient descent minimizing a potential function where left shows the descent on a 3D surface, while right illustrates the corresponding path over contour lines in 2D.

This technique was then translated into a MATLAB simulation, where a point-mass agent navigated through a 2D environment by minimizing the total potential field in real time, effectively avoiding obstacles while progressing toward the goal. Simulation parameters are given in Table 3

Table 4: Simulation Parameters for APF in MATLAB

Start coordinate.	(50,350)
Goal coordinate	(400,50)
k_{att}	1/700
k_{rep}	1200
ρ_o	1.01
α	3
Obstacles	2x Rectangles, 2x Circle



(a) (b)

Fig.11: APF in MATLAB for a point-mass agent at (a) start position (b) end position

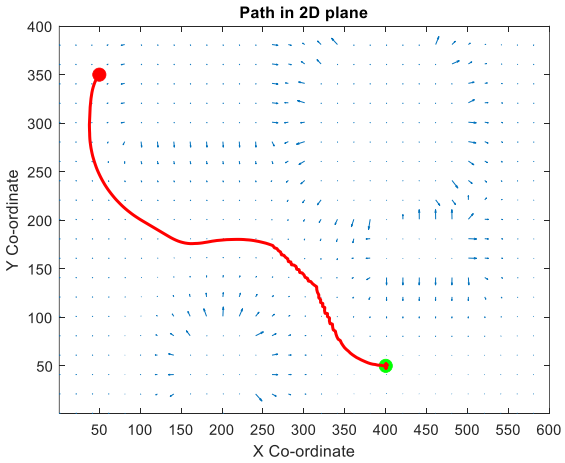


Fig.12: Path planned with APF in MATLAB for a point-mass agent in 2D Plane

Figure 11(b) shows the resulting trajectory as the agent avoids obstacles and reach to the goal, while Figure 12 illustrates trajectory drawn in 2D plane. This implementation served as a foundational validation step before extending APF to PX4 Iris within the ROS-Gazebo environment using real-time sensor feedback.

10.0 Real-Time Obstacle Avoidance with PX4 Iris

In real-world environment, autonomous vehicles must be capable of detecting and responding to nearby obstacles to ensure safe navigation. This requires real-time depth sensing to perceive the surroundings accurately. To facilitate spatial awareness for reactive algorithms like APF to generate avoidance maneuvers, LiDAR sensor was used. This section describes how APF was implemented using LiDAR data within the PX4-ROS simulation framework to achieve real-time obstacle avoidance on the Iris quadrotor.

10.1 Sensor Setup

LiDAR provides accurate distance measurements and is well-suited for local obstacle detection due to its depth sensing technology. The Gazebo simulation used the iris_rplidar drone (pre-configured) model, which includes a 2D LiDAR sensor mounted at the center of mass. The sensor published range data in a 360° sweep, simulating realistic perception for avoidance task.

10.2 Implementation of APF in ROS

The Artificial Potential Field algorithm was implemented as a custom ROS node that subscribes to LiDAR data and computes repulsive velocity commands based on the distance to nearby object. The attractive component was omitted in this setup, and the focus was on purely reactive avoidance behavior. The drone maintained a stationary position until an obstacle moved into proximity, at which point the APF node issued velocity commands to move away.

The system used MAVROS to relay these commands to PX4 in offboard mode. The drone's local position was monitored via MAVROS, and the repulsive velocity commands were published continuously to maintain active control. The APF computation was based on

previously defined parameters in Section 8, such as repulsive gain k_{rep}, influence radius ρ_o, and step size α.

10.3 Test Setup and Results

In addition to the ROS topics listed earlier in Table 3 for trajectory tracking, the difference in this setup was the use of the /scan topic, which provided real-time LiDAR data necessary for computing repulsive forces. This additional sensor input enabled the drone to perform reactive obstacle avoidance based solely on proximity.

Table 5: Simulation Parameters for APF in ROS

k_{rep}	1.20
ρ_o	1.01
α	3.00

A single spherical obstacle was included in the Gazebo environment and was initially placed at the same position as the drone. To simulate a dynamic avoidance scenario, the obstacle was manually moved toward the drone using keyboard commands. As the object approached, the drone began to react, generating appropriate repulsive velocity commands to steer away from the obstacle in real time. The drone consistently moved away from the obstacle when necessary and returned to hover once the influence of obstacle was cleared. This experiment validated the integration of perception with LiDAR and control using MAVROS and set the foundation for future work involving path planning in dynamic environments.

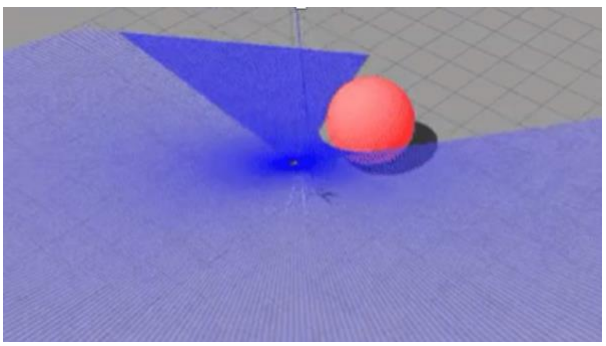


Fig. 13: APF in ROS for PX4-Iris with LiDAR

11.0 Bioinspiration

In this project, the obstacle avoidance behavior was inspired by the flight strategies of insects such as bees and flies, which rely on simple yet effective mechanisms to navigate dynamic environments. These insects do not have global map available to them, instead, they make real-time adjustments to their velocity and heading based on local sensing techniques that mirrors the repulsive behavior modeled in Artificial Potential Field (APF) method. The implementation of APF in this work mimics this biological strategy by continuously pushing the drone away from nearby obstacles using a local LiDAR sensor, much like how an insect might avoid a tree branch mid-flight.

12.0 Project Management

This project followed a structured engineering design process, beginning with research and conceptual design, followed by system integration, simulation, and testing. A Gantt chart was developed at the start of the academic year to define major deliverables and timelines, including design reviews, algorithm development phases, and validation milestones. Tasks were divided into distinct categories: initiation and research, system design, algorithm development, and final deliverables.

During the execution phase, dedicated efforts were invested, with approximately 4 hours per day on weekdays and 7 hours per day on weekends committed to meeting the scheduled milestones. Weekly updates were presented to Lead Engineers for continuous feedback and progress validation. The project concluded with a structured closure phase, which included the compiling all the work and recommendations based on the technical feedback received throughout the term.

Notably, I began the project with no prior experience using the ROS-Gazebo simulation environment. This required an initial learning phase, during which foundational knowledge of ROS, simulation tools and PX4 integration was developed before control strategies/algorithm could be reliably implemented.

13.0 Social and Economic Impact

The implementation of advanced Guidance, Navigation, and Control (GNC) systems in the Manta has significant social and economic implications. From a social perspective, robust GNC systems enhance flight safety and reliability, enabling autonomous operations even in complex or low-visibility environments, which is critical for delivering goods to remote or underserved rural communities. This supports equitable access to essential services, such as healthcare and food distribution. Economically, improved GNC allows for efficient route planning and energy management, reducing operational costs and minimizing battery usage, which extends the service life of key components. Additionally, autonomous emergency landing and precision navigation features decrease the need for extensive ground infrastructure and labor, making aerial logistics more viable and cost-effective for agricultural applications as per the requirement.

14.0 Conclusion and Future Work

In conclusion, this project focused on the development of outer-loop guidance, navigation, and control strategies for the Manta using PX4, ROS, and Gazebo. A modular framework was built around PX4's Software-in-the-Loop environment, enabling autonomous tasks such as takeoff, trajectory tracking, and real-time obstacle avoidance to be implemented and tested. Trajectory tracking was achieved using a PD velocity controller, while obstacle avoidance relied on a repulsive only Artificial Potential Field algorithm.

Although the PX4 Iris model was used in this study, the architecture was designed with flexibility in mind. The same framework can be extended to the Manta by importing its geometry into Gazebo, allowing platform-specific testing and tuning prior to deployment.

Future work will focus on enhancing system autonomy and robustness. This includes incorporating attractive potentials for full APF navigation, implementing global path planning techniques in combination with a computer vision system for improved environmental awareness, transitioning to hardware deployment on the Manta exploring advanced state estimation and sensor fusion methods.

References

[1] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, et al., "ROS: an open-source Robot Operating System," 2009.

[2] "PX4/PX4-Autopilot: PX4 Autopilot Software." [Online] https://github.com/PX4/PX4 -Autopilot

[3] "MAVLink Messaging | PX4 User Guide." [Online] https://docs.px4.io/main/en/middle ware/mavlink.html.

[4] "Simulation | PX4 User Guide." [Online] https://docs.px4.io/main/en/simulation/

[5] Gazebo Simulator. [Online] http://gazebosim.org/

[6] "3DR Iris - the ready to fly UAV Quadcopter," [Online] *Arduino based Arducopter UAV, the open-source multi-rotor*, 2025. https://www.arducopter.co.uk/iris-quadcopter-uav.html

[7] D. Qi, M. Cai, Z. Jiao, X. He, X. Ren, and Y. Hou, "Design and Implementation of Simulation System for Multi-UAV Mission," *Applied Sciences*, vol. 13, no. 3, p. 1490, Jan. 2023, doi: https://doi.org/10.3390/app13031490.

[8]"Gazebo Virtual Worlds - nlamprian," *Nlamprian.me*, 2019. https://nlamprian.me/blog/software/ros/2019/10/06/gazebo-virtual-worlds/

[9] "Using Vision or Motion Capture Systems for Position Estimation | PX4 User Guide." [Online] https://docs.px4.io/main/en/ros/external_position_estimation.html

[10] "ROS with Gazebo Classic Simulation | PX4 Guide (main)," *Docs.px4.io*, 2025. https://docs.px4.io/main/en/simulation/ros_interface.html

[11]"Offboard mode (generic/all frames) ," Offboard Mode (Generic/All Frames) | PX4 Guide (main), https://docs.px4.io/main/en/flight_modes/offboard.html

[12] "Wiki," ros.org, https://wiki.ros.org/mavros

[13] M. Goodrich, "Potential Fields Tutorial." Available: https://phoenix.goucher.edu/~jillz/cs325_robotics/goodrich_potential_fields.pdf

[14] A. singh, "Gradient descent explained: The engine behind AI training," Medium, https://medium.com/@abhaysingh71711/gradient-descent-explained-the-engine-behind-ai-training-2d8ef6ecad6f

Appendix

A: Simulation

Take – Off

```
#include <ros/ros.h>
#include <geometry_msgs/PoseStamped.h>
#include <mavros_msgs/CommandBool.h>
#include <mavros_msgs/SetMode.h>
#include <mavros_msgs/State.h>

mavros_msgs::State current_state;
void state_cb(const mavros_msgs::State::ConstPtr& msg){
    current_state = *msg;
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "offb_node");
    ros::NodeHandle nh;

    ros::Subscriber state_sub = nh.subscribe<mavros_msgs::State>
            ("mavros/state", 10, state_cb);
    ros::Publisher local_pos_pub = nh.advertise<geometry_msgs::PoseStamped>
            ("mavros/setpoint_position/local", 10);
    ros::ServiceClient arming_client = nh.serviceClient<mavros_msgs::CommandBool>
            ("mavros/cmd/arming");
    ros::ServiceClient set_mode_client = nh.serviceClient<mavros_msgs::SetMode>
            ("mavros/set_mode");

    ros::Rate rate(20.0);

    // wait for FCU connection
```

```
while(ros::ok() && !current_state.connected){
    ros::spinOnce();
    rate.sleep();
}
geometry_msgs::PoseStamped pose;
pose.pose.position.x = 0;
pose.pose.position.y = 0;
pose.pose.position.z = 2;

//send a few setpoints before starting
for(int i = 100; ros::ok() && i > 0; --i){
    local_pos_pub.publish(pose);
    ros::spinOnce();
    rate.sleep();
}
mavros_msgs::SetMode offb_set_mode;
offb_set_mode.request.custom_mode = "OFFBOARD";

mavros_msgs::CommandBool arm_cmd;
arm_cmd.request.value = true;

ros::Time last_request = ros::Time::now();

while(ros::ok()){
    if( current_state.mode != "OFFBOARD" &&
        (ros::Time::now() - last_request > ros::Duration(5.0))){
        if( set_mode_client.call(offb_set_mode) &&
            offb_set_mode.response.mode_sent){
            ROS_INFO("Offboard enabled");
        }
        last_request = ros::Time::now();
    } else {
        if( !current_state.armed &&
            (ros::Time::now() - last_request > ros::Duration(5.0))){
            if( arming_client.call(arm_cmd) &&
```

```
                    arm_cmd.response.success){
                    ROS_INFO("Vehicle armed");
                }
                last_request = ros::Time::now();
            }
        }
        local_pos_pub.publish(pose);

        ros::spinOnce();
        rate.sleep();
    }

    return 0;
}
```

TRACKING - Messages

```
std_msgs/Header header

uint8 coordinate_frame
uint8 FRAME_LOCAL_NED = 1
uint8 FRAME_LOCAL_OFFSET_NED = 7
uint8 FRAME_BODY_NED = 8
uint8 FRAME_BODY_OFFSET_NED = 9

uint16 type_mask
uint16 IGNORE_PX = 1 # Position ignore flags
uint16 IGNORE_PY = 2
uint16 IGNORE_PZ = 4
uint16 IGNORE_VX = 8 # Velocity vector ignore flags
uint16 IGNORE_VY = 16
uint16 IGNORE_VZ = 32
uint16 IGNORE_AFX = 64 # Acceleration/Force vector ignore flags
uint16 IGNORE_AFY = 128
```

uint16 IGNORE_AFZ = 256
uint16 FORCE = 512 # Force in af vector flag
uint16 IGNORE_YAW = 1024
uint16 IGNORE_YAW_RATE = 2048

geometry_msgs/Point position
geometry_msgs/Vector3 velocity
geometry_msgs/Vector3 acceleration_or_force
float32 yaw
float32 yaw_rate

B: Gantt Chart

TASK	PROGRESS	START	END
1.0 Initiation / Research			
1.1 Design Specifications and Requirements	15%	9-9-24	10-14-24
1.2 Literature Review	10%	9-24-24	10-29-24
1.3 Bio-Inspiration	5%	9-29-24	11-28-24
2.0 System Design			
2.1 Requirement Analysis	0%	10-22-24	11-13-24
2.2 Conceptual Design	0%	11-10-24	12-12-24
2.3 Detailed Design with Communication Protocols	0%	12-9-24	12-16-24
3.0 Algorithm Development			
3.1 Research on Algorithms	0%	12-15-24	1-5-25
3.2 Develop and Refine Selected Algorithms	0%	1-3-25	1-23-25
3.3 Simulate the Algorithm	0%	1-21-25	3-2-25
3.4 Testing and Validation	0%	1-21-25	3-5-25
4.0 Deliverables			
4.1 First Report	0%	10-7-24	10-27-24
4.2 Final Design Review	0%	10-28-24	11-24-24
4.3 Fall Deliverables	0%	11-25-24	12-5-24
4.4 Technical Presentation	0%	2-24-25	3-3-25
4.5 Winter Design Review	0%	3-3-25	4-4-25
4.6 Winter Deliverables	0%	3-11-25	4-7-25