Engineering Practicum
Project Report
Self-Guided Ballistic Missile

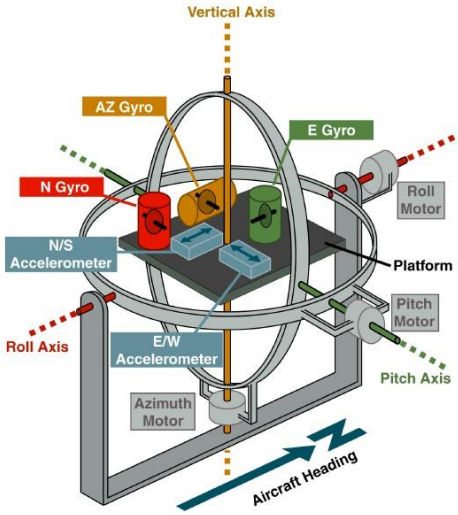
Name of the students:
Sasank Gupta (2101ME60)
Nishant Rao (2101ME45)
Vinay Sobarad (2101ME79)
Nishant M. (2101ME83)
Yashraj (2101ME84)

Mentor:
Dr. Somnath Sarangi

INERTIAL NAVIGATION SYSTEM

An inertial navigation system (INS) is a sophisticated navigation tool used primarily in vehicles, aircraft, and ships to determine their position, orientation, and velocity without relying on external references such as landmarks, GPS signals, or radio beacons. Instead, it relies on internal sensors and computational algorithms to track its motion relative to an initial starting point.

Here is a breakdown of its components and how it works:



Inertial Measurement Unit (IMU): The core of an INS is the IMU, which typically consists of three orthogonal gyroscopes and three orthogonal accelerometers. Gyroscopes measure angular velocity, indicating how the vehicle is rotating, while accelerometers measure specific force, indicating how the vehicle is accelerating.

Gyroscope: Gyroscopes detect the rate of rotation around each of the three axes (roll, pitch, and yaw). They do this by measuring the Coriolis effect, where a mass in motion experiences a force perpendicular to its velocity when the reference frame is rotating.

Accelerometer: Accelerometers measure the vehicle's acceleration along each of the three axes. By integrating these measurements over time, the system can calculate velocity and position changes.

Computational Unit: The data from the IMU is processed by a computational unit, typically a microprocessor or digital signal processor (DSP). This unit applies sophisticated algorithms to integrate the sensor data over time, compensating for errors and drift in the measurements.

Initial Alignment: Before the INS can be used for navigation, it must undergo an initial alignment process to establish its starting position and orientation. This can be done manually or automatically using external references such as GPS or known landmarks.

Dead Reckoning: Once the initial alignment is complete, the INS continuously updates its position and orientation by integrating the sensor measurements over time. This process is known as dead reckoning, and it involves calculating changes in position and velocity relative to the starting point.

Error Correction: Despite its precision, INS readings are subject to errors and drift over time due to factors such as sensor noise, temperature changes, and mechanical vibrations. To mitigate these errors, modern INS systems often incorporate additional sensors or external references, such as GPS or magnetometers, to periodically correct the estimated position and orientation.

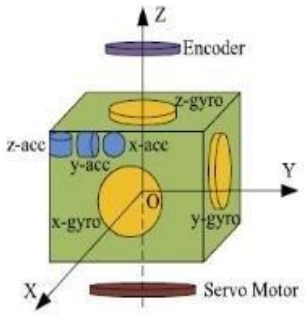
Integration with External Sensors: In many applications, an INS is used in conjunction with other navigation systems, such as GPS or radar, to provide complementary data and improve overall accuracy and reliability.

In summary, an inertial navigation system is a critical tool for navigating vehicles, aircraft, and ships, providing continuous position, orientation, and velocity information based on internal sensors and computational algorithms. Its ability to operate independently of external references makes it particularly useful in environments where GPS signals may be unavailable or unreliable.

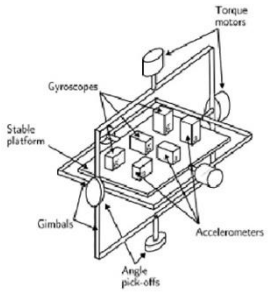
Types of Constrained Gyros:

1) Rate Gyros (Measures Absolute Angular Velocity)
2) Rate Integrating Gyros (Measures Absolute Angular Displacement)

Rate gyros are widely used to generate stabilizing signals in vehicle control systems, whereas **Rate-integrating** gyros are used to fix the reference in navigation and attitude control systems.



The rate-integrating gyro has been subjected to extremely intensive and extensive engineering development to bring its performance characteristics to remarkable levels while maintaining its small size and weight and great resistance to rugged environments. This has been accomplished by painstaking attention to minute mechanical, thermal, and electrical details which would be completely negligible in less sophisticated devices. A floated rate-integrating gyro, which incorporates several clever design features necessary for high performance. The inner gimbal is in the form of a hollow cylinder containing the gyro wheel and is mounted in precision pivot jewel bearings. A small radial gap between this cylinder and its housing is filled with damping fluid to form the damper B. Temperature feedback-control systems regulate electric heaters to maintain damping-fluid viscosity constant. This link must be carefully applied to minimize convection currents in the damping fluid, which exerts spurious torques on the inner gimbal.



COMPONENTS:

- 3-volt button battery
- Rotary Encoder
- 3V DC motor 48000 rpm
- Accelerometer
- Azimuthal Gimbal
- Inner Gimbal
- Outer Gimbal
- Extra Mass for Centre of Gravity
- Arduino Nano
- Damper (Foam, fluidic medium)
- Ball bearing

CALCULATIONS

All the references are taken from the book Measurement Systems (Ernest O Doebelin | Dhanesh N Manik)

In the inertial gyroscope, a beryllium ball has a 1 cm diameter and is rotated with the help of a motor up to 146000 rpm.

For us it was difficult to achieve that rpm in a micro DC motor, so we tried to find an alternate solution to get an angular momentum and size of mass nearer to the original.

So we decided to use mild steel (ρ = 7.85 gm/cc) instead of a beryllium ball (ρ = 1.85 gm/cc)

Since we can achieve a rotation speed of 48000 rpm. So instead of using a mid-steel ball, we decided to use a mild steel disk/chip

Assumptions:

a) The disk is rotating with an angular speed of 48000 rpm.
b) The disk is perfectly symmetric.

Let object 1 be a Beryllium ball and object 2 be a Mild Steel Disk.

Beryllium ball	Mild Steel Disk
1. Shape: Sphere 2. Diameter: 1 cm 3. ρ = 1.85 gm/cc	1. Shape: Disk 2. Diameter:? (Need to be calculated) 3. Thickness:? (Need to be calculated) 4. ρ = 7.85 gm/cc

Total Angular Momentum of the Core:

$I_1\omega_1 = I_2\omega_2 = 2/5(m_1R_1^2\ \omega_1)$

$= 2/5(\rho_1V_1R_1^2\ \omega_1)$

$= 3901.6\ gcm^2/s$

$= 4000\ gcm^2/s$ (Approximately)

Keeping the RPM constant

D(cm)	h(cm)	L(gcm^2/s)
1.2	0.3	2400
1.2	0.4	3200
1.3	0.3	3300
1.3	0.4	4400
1.4	0.2	2960
1.4	0.3	4440
1.5	0.2	3900

Reason for the disk-shaped mass of inertia in Gyroscope:

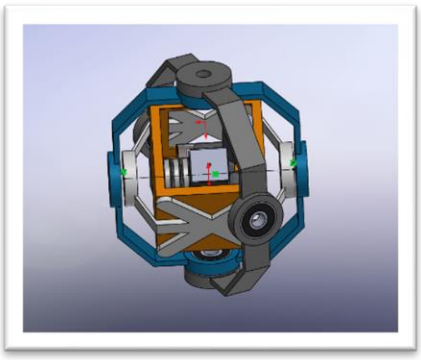
a. Disk has higher Inertia compared to Spere.
b. Space adjustments through smaller thicknesses.

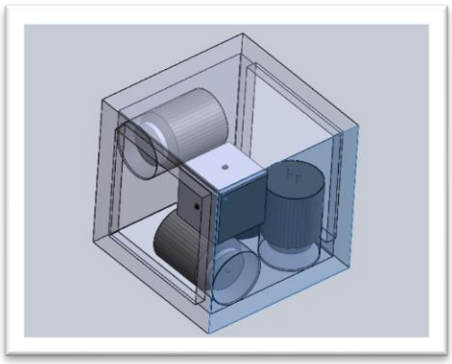
Problems of having disk-shaped mass:

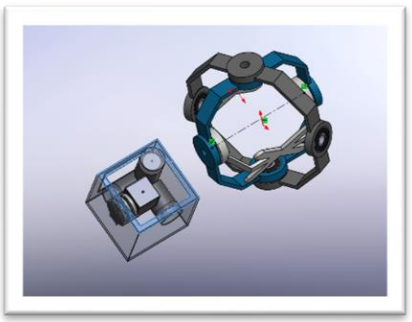
a. Higher value of radius increases the vibration.
b. Hole/ Drill at the centre can be different from the geometrical centre causing asymmetricity which increases the amplitude of vibration.
c. Surfaces should be polished.

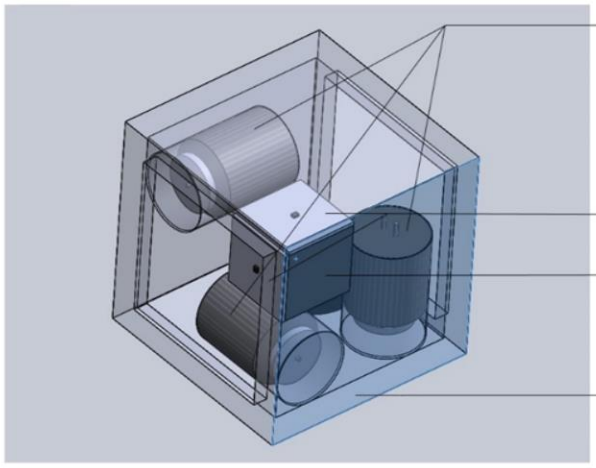
Seeing the Above problems we decided to take diameters/heights pairs of size 1.3/0.4, 1.4/0.3, 1.5/0.2(All values are in cm).

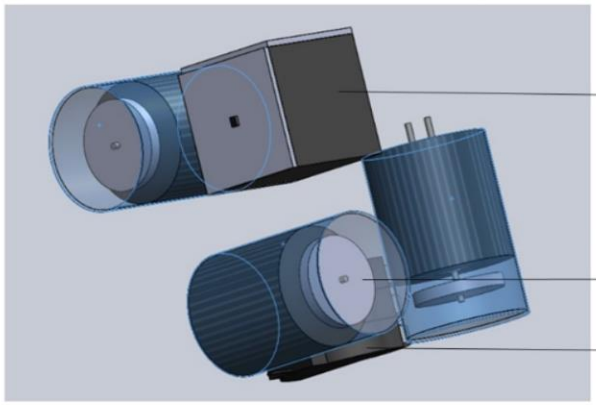
Structure of Gyroscope:











All the gyroscopes and accelerometers are integrated orthogonally to each other

Sources of Error:

1. High vibration due to asymmetric structure and integration of disk/chip to the DC motor.
2. Lags and leads in the digital system.
3. White noise error.
4. Low PPR of encoder.
5. Low damping coefficient of Damper.
6. Misalignment in orthogonality of three gyroscopes.

Manoeuvring of Missile

Introduction:

In recent years, reinforcement learning (RL) has emerged as a powerful framework for training autonomous systems, including missile guidance systems. By leveraging RL, missile guidance systems can autonomously learn optimal control strategies to navigate complex environments and achieve mission objectives, such as hitting a defined target point while avoiding collisions. This report focuses on the implementation and performance evaluation of RL in a simulated missile guidance scenario, where the objective is to guide a missile from a launch point to a target point while maximising proximity to the target and minimising the risk of collision.

Q-learning, a foundational algorithm in RL, holds particular relevance for autonomous aircraft control. It offers a systematic approach to learning optimal control strategies by estimating the expected future rewards associated with different actions in a given state. By iteratively updating these action-value estimates based on observed outcomes, Q-learning enables aircraft control systems to make informed decisions in real time, navigating the complexities of flight operations effectively.

Objective of the Report:

The RL algorithm employed for missile guidance is Q-learning, a model-free RL technique suited for scenarios with discrete state and action spaces. In this context, Q-learning enables the missile guidance system to learn optimal control policies by iteratively updating Q-values associated with state-action pairs. The core idea is to maximise cumulative rewards, where rewards are based on the proximity of the missile to the target point.

Background

Explanation of Q-learning Algorithm Adapted for Missile Guidance:

Q-learning, a foundational algorithm in RL, is employed to train the missile guidance system. Adapted for missile guidance, Q-learning enables the system to learn optimal control policies by iteratively updating the Q-values associated with state-action pairs. The Q-value represents the expected cumulative reward for taking a particular action (such as adjusting trajectory) in a specific state (such as current position and velocity), and following the optimal policy thereafter. The Q-learning update equation facilitates this iterative learning process by updating Q-values based on observed transitions in the environment.

The core idea behind Q-learning is the Bellman equation, which expresses the relationship between the Q-values of state-action pairs and the Q-values of subsequent states. The Q-value update rule, known as the Q-learning update equation, is defined as follows:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$$

Where:

$Q(s, a)$ is the Q-value of state-action pair (s, a).

α is the learning rate, determining the weight of new information. r is the immediate reward received after taking action a in state s. γ is the discount factor, balancing immediate and future rewards. s' is the subsequent state resulting from taking action a in state s. a' is the next action chosen according to the optimal policy.

The Q-learning algorithm iteratively updates Q-values based on observed transitions in the environment, gradually converging towards the optimal Q-function that maximises long-term rewards. By learning the optimal Q-values, the missile guidance system can make informed decisions in real-time, selecting actions that maximise safety, efficiency, and performance during flight.

Description of the Simulated Aircraft Environment:

The simulated missile environment represents the dynamics of missile flight, including its position, velocity, and orientation. The objective is to navigate the missile from a defined launch point to a target point while avoiding collisions. Actions available to the missile include adjustments to its trajectory, such as changes in velocity and orientation angles. The reward system is designed based on the distance between the missile's current position and the target point: the missile receives a positive reward if it reduces the distance to the target and a negative reward if it increases the distance. Additionally, a substantial negative reward is given if the missile crashes, prompting the simulation to restart.

Importance of Autonomous Control in Aviation and the Role of Reinforcement Learning:

Autonomous control plays a pivotal role in modern aviation, offering numerous benefits such as improved safety, operational efficiency, and mission flexibility. By automating routine tasks, assisting pilots in decision-making, and responding swiftly to dynamic environments, autonomous systems
enhance the overall effectiveness and reliability of flight operations.

Reinforcement learning holds immense potential for advancing autonomous control in aviation by
enabling aircraft to learn and adapt to complex scenarios without explicit programming. RL algorithms like Q-learning empower aircraft control systems to autonomously acquire optimal control strategies
through interaction with the environment, facilitating adaptive behaviour and intelligent decision-making in diverse flight conditions.

Code Overview

Brief Description of the Provided Script:

The provided script serves as the implementation framework for simulating missile control using Q-learning. It orchestrates the interaction between the environment, reinforcement learning algorithms, and control logic to facilitate autonomous flight simulation. The script manages the training process, data logging, and model evaluation, providing a comprehensive platform for experimenting.

Q-learning is integrated into the missile guidance simulation as follows:

1. Agent Initialization: The RL agent representing the missile guidance system is initialised with parameters such as learning rate, discount factor, and exploration rate.
2. State-Action Representation: States are represented by the current position and velocity of the missile, while actions correspond to adjustments in trajectory.
3. Reward Calculation: Rewards are computed based on the change in distance between the missile's current position and the target point. Positive rewards are given for reducing distance, while negative rewards are incurred for increasing distance or crashing.
4. Q-value Update: The Q-values associated with state-action pairs are updated using the Q-learning update equation, taking into account the observed rewards and subsequent states.
5. Training Loop: The missile guidance system undergoes training iterations, during which it interacts with the environment, learns from experiences, and updates its control policy to maximise rewards.

Main Script:

Imports and Setup:

- **Libraries**: The script imports necessary libraries such as **socket**, **time**, **os**, **numpy**, and **matplotlib.pyplot** for general functionalities, numerical operations, and plotting.

- **Custom Classes**: It imports custom classes relevant to aircraft simulation, including the Q-learning agent (**QDoubleDeepLearn**), environment (**JSBSimEnv**), and scenarios (**deltaAttitudeControlScene**).

Experiment Setup:

- **Parameters**: Experiment parameters are defined, including the number of epochs (**n_epochs**), steps per epoch (**n_steps**), number of actions (**n_actions**), and number of states (**n_states**). Other parameters such as discount rate (**gamma**), learning rate (**lr**), and exploration rate (**epsilon**) are also specified.

- **Environment Configuration**: The script sets up configurations related to the simulation environment, such as starting position, velocity, control ranges, etc. These configurations define the initial conditions and constraints of the aircraft simulation.

Functions:

- **log(i_epoch, i_step, reward, logList)**: This function logs various metrics and information during the simulation, including the state, action, reward, current epsilon, control, etc. It provides detailed insights into the simulation process at each step.

- **step(i_step, done, reward, oldState)**: Executes a single step of the simulation, interacting with the environment and updating the Q-values based on the observed transitions. It handles error handling for connection issues and NaN states.

- **epoch(i_epoch)**: Orchestrates a complete epoch of the training process, consisting of multiple steps, logging, and updating the Q-values. It calls the **step()** function iteratively for each step of the epoch.

Execution Loop:

- The main execution loop iterates through the specified number of epochs, calling the **epoch()** function to run each epoch of the training process. During each epoch, the Q-learning algorithm interacts with the aircraft simulation environment, learns from experiences, and updates its Q-values accordingly.

QLearn Class:

- **Initialization**:
  - Initialises the Q-learning agent with parameters such as the number of states (**n_states**), number of actions (**n_actions**), discount factor (**gamma**), learning rate (**learningRate**), exploration rate (**epsilon**), and other parameters relevant for training and exploration.
  - Initialises the Q-table with zeros to store Q-values for state-action pairs.
  - Set up the DQN agent (**model**) for training using the **DQNAgent** class.

- **selectAction(state, episode, n_epochs)**:
  - Selects an action based on the current state, episode number, and total number of epochs.
  - Implements an epsilon-greedy strategy for exploration and exploitation.
  - Decays the epsilon value over time to balance exploration and exploitation.

- **learn(state, action, reward, new_state, done)**:
  - Updates the Q-values based on the observed transition in the environment.
  - Stores the transition tuple **(state, action, reward, new_state, done)** in the replay memory.
  - Trains the DQN agent (**model**) based on the replay memory.

- **archive(epoch)**:
  - Archives the model and replay memory periodically for tracking training progress and potential future use.

DQNAgent Class:

- **Initialization**:
  - Initialises the DQN agent with parameters such as the number of inputs (**inputs**), number of outputs (**outputs**), learning rate (**learningRate**), replay memory size (**replayMemSize**), batch size (**batch**), discount factor (**gamma**), update rate for target network (**update**), etc.
  - Creates the main neural network model (**model**) and the target network model (**targetModel**) using Keras.

- Loads pre-trained model and replay memory if specified.

- **createModel()**:
  - Creates the neural network model architecture using Keras Sequential API.
  - Consists of the input layer, hidden layers with ReLU activation, and an output layer for Q-values.

- **updateReplayMemory(transition)**:
  - Updates the replay memory with the transition tuple **(state, action, reward, new_state, done)**.

- **train(done)**:
  - Trains the main neural network model using mini-batch gradient descent.
  - Samples transition from the replay memory and update the Q-values using the Bellman equation.
  - Updates the target network periodically.

- **getQs(state)**:
  - Queries the main neural network model for Q-values given a state.

Implementation Details:

Q-Learning Principles for Missile Control

In this implementation, Q-learning principles are applied to control missile behaviour by training an agent to make decisions in a simulated environment. Q-learning is a model-free reinforcement learning technique where an agent learns to make decisions by maximising the expected cumulative reward.

States, Actions, and Rewards in Simulation

- **States:** In the missile guidance simulation, the state space represents the current state of the missile, encompassing parameters crucial for navigation. This includes the missile's position, velocity, orientation, and potentially other relevant factors such as fuel level or engine status. These state variables are encoded into a format understandable by the Q-learning agent, facilitating informed decision-making.
- **Actions:** The action space comprises the available manoeuvres or control inputs that the missile can execute to adjust its trajectory. These actions may involve alterations in the missile's velocity, orientation angles, or other parameters critical for navigation. The agent selects actions based on its current state and the learned Q-values, aiming to guide the missile toward its target point while avoiding collisions or crashes.
- **Rewards:** Rewards serve as feedback to the agent, evaluating the effectiveness of its actions in achieving the mission objective. In the missile guidance simulation, rewards are defined based on the distance between the missile's current position and the target point. Positive rewards are granted for actions that reduce this distance, indicating progress toward the target. Conversely, negative rewards are imposed for actions leading to an increase in distance or collisions, discouraging undesirable behaviours and promoting efficient navigation towards the target point.

Management within the Missile Simulation Context:

State Representation: In the missile guidance problem, states are represented as vectors or arrays containing essential numerical values describing the current state of the missile. These values encompass parameters such as the missile's position relative to the target point, its velocity, orientation angles, and potentially other relevant factors like fuel level or engine status. The state representation must encapsulate sufficient information to enable the agent to make informed decisions regarding the missile's trajectory towards the target.

Action Selection: The agent in the missile guidance scenario selects actions based on its current state and the Q-values associated with each action. During the training process, the agent explores different actions to understand their consequences and updates its Q-values accordingly. Exploration is guided by an epsilon-greedy strategy, striking a balance between exploring new actions and exploiting learned knowledge to guide the missile effectively towards the target point while avoiding undesirable outcomes such as crashes or deviations from the intended trajectory.

Reward Design: Rewards are meticulously designed to incentivize the agent to exhibit behaviours conducive to successfully hitting the target point while avoiding crashes or deviations. Positive rewards are bestowed upon actions that result in the missile moving closer to the target point, signifying progress towards the mission objective. Conversely, negative rewards or penalties are applied to actions leading to an increase in distance from the target point or collisions, discouraging undesirable behaviours and guiding the agent towards optimal decision-making.

Training and Learning: The agent learns from its interactions with the simulated environment by updating its Q-values based on observed transitions. These transitions comprise the agent's experiences, including its current state, the action selected, the resulting reward, and the subsequent state. Leveraging the Bellman equation, the agent incrementally refines its policy over time, gradually improving its ability to navigate the missile towards the target point while minimising the risk of crashes or deviations from the intended trajectory.

Conclusion

The implementation of RL in missile guidance simulation demonstrates its potential for enabling autonomous decision-making and precise navigation in complex environments. By effectively managing states, actions, and rewards, the RL agent learns to guide the missile towards the target point while avoiding crashes. Future research can explore more sophisticated reward structures and scalability techniques to address real-world challenges, ultimately enhancing the effectiveness and reliability of autonomous missile guidance systems.

ANALYSIS OF AERODYNAMICS OF THE MISSILE

The making of fluid flow simulations that pass the surface of the missile using CFD software was conducted to determine the aerodynamic characteristics of the missiles described in the form of simulations. Process of research consists of the initial study phase, and CFD simulation phase. The output and parameters of this research that are to be seen are pressure contours, velocity contours, lift and drag, which in each parameter is carried out by α of 0˚, 5˚, 10˚, and 15˚. The nature of the air flow around the missile when simulated has different characteristics of velocity and pressure in each variation α given. The change in α is directly proportional to the value of pressure. The speed value changes relative to each change α.

To launch a missile, it should be noted about its aerodynamic characteristics such as lift, drag, and moment so as to facilitate the movement and control of the missile. The aerodynamic characteristics need to be considered because it greatly affects the altitude and distance that can be reached by the missile. In addition, note also the Mach value possessed by the missile when fired with each different speed. The most crucial thing about a missile is its flying/cruising height, where the height must be as low as possible to avoid detection from the radar, but also in its low altitude the missile must not lose the lift. In simulations, missile aerodynamic characteristics can be obtained through coefficients (CL and CD).

The problems that form the basis of the analysis:

The analysis process is only carried out on the speed and pressure experienced by the missile when it is launched.

Only variations in the value of α (with intervals of 5˚ from 0-15˚) are used to see the characteristics of missile aerodynamics.

The objective(s) of this research is:

To know the nature of the airflow around the missile when simulated.

To know the aerodynamic characteristics of missiles that were simulated.

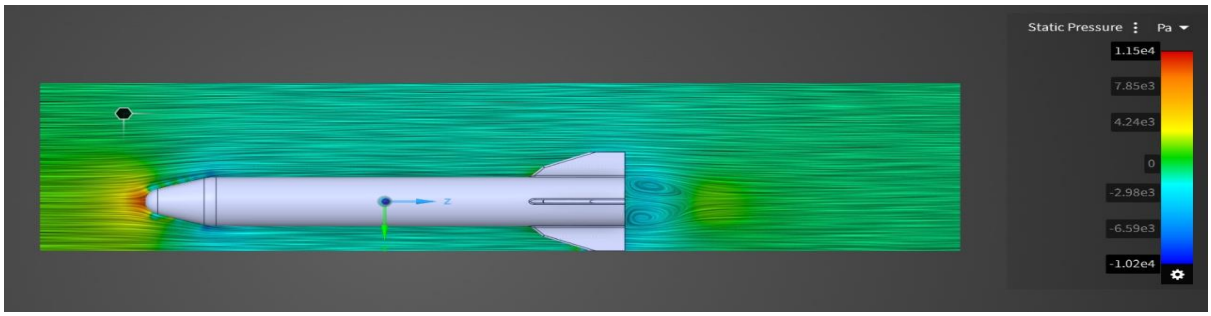
In general, the missile structure consists of:

The nose section is the part of the missile that first hits the fluid while flying so that it experiences the greatest force compared to the other structures. The shape of this part affects the stability of the missile when sliding. The nose cone is made taper because the pointed shape has a maximum speed and can divide the airflow better.

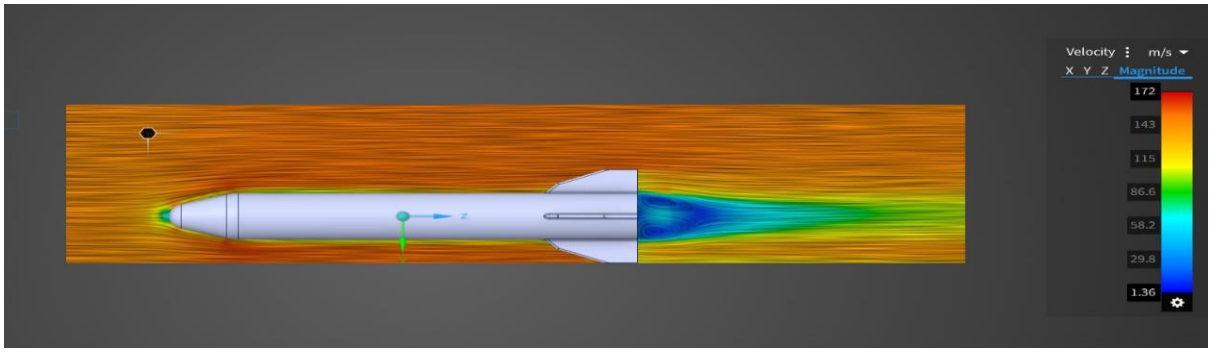
The body is a part of the missile that rubs against the fluid which then flows/passes fluid from the nose cone to the fin. In addition, this section is equipped with a chip that functions as a control system; and a place to put warheads.

Fin, is a fin that functions as a guide to the flow of air from the tip of the missile to its rear and also serves to make the missile movement more stable. Wing/fin is a thing that has a big role in the success of missiles in terms of aerodynamics.

Initial Design:

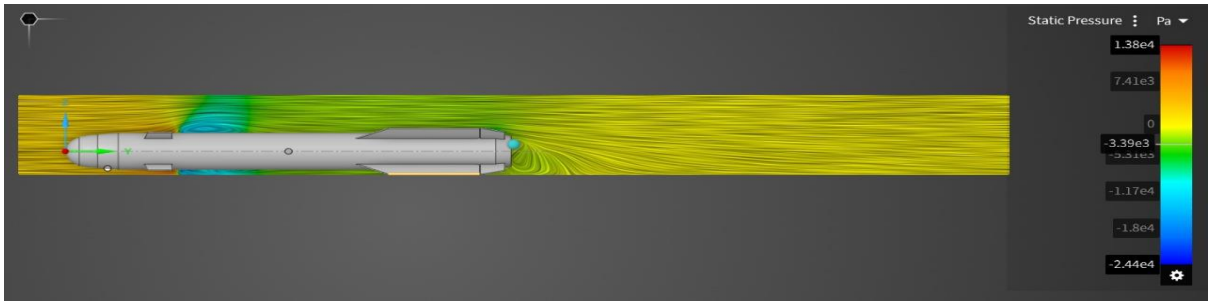


Pressure α = 0 deg

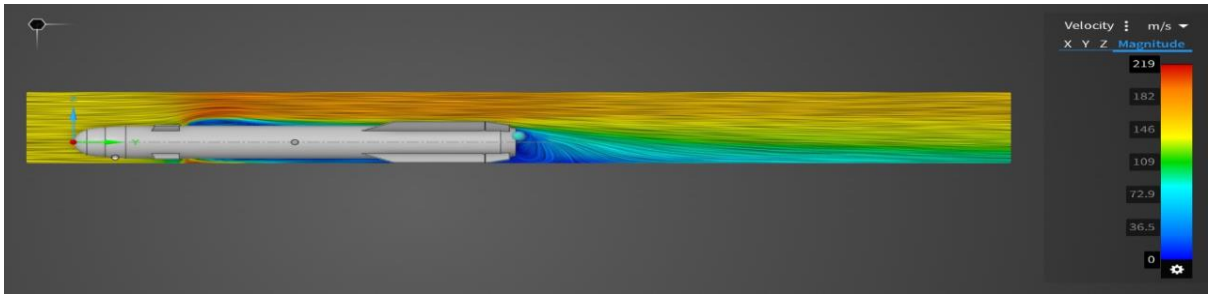


Velocity at α = 0 deg

Final Design:

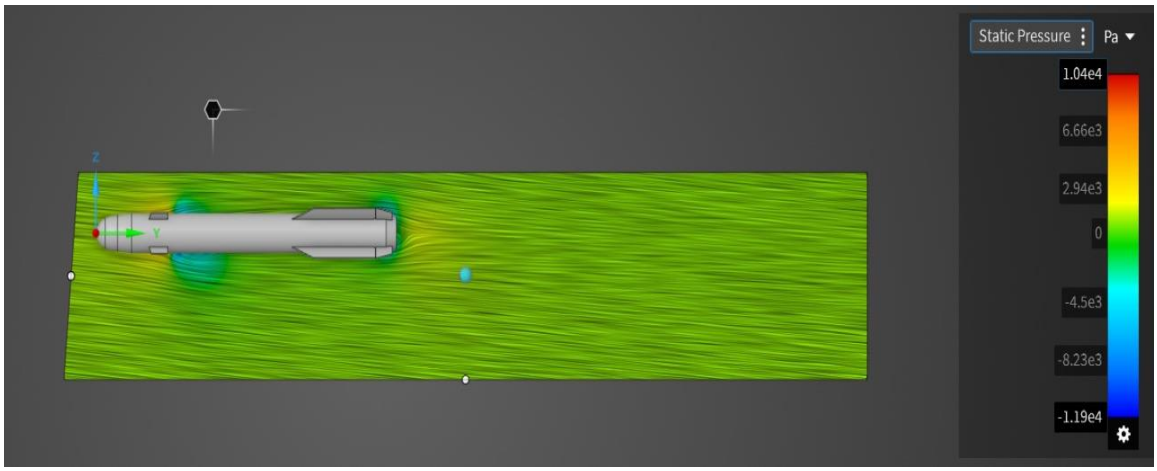


Pressure α = 0 deg

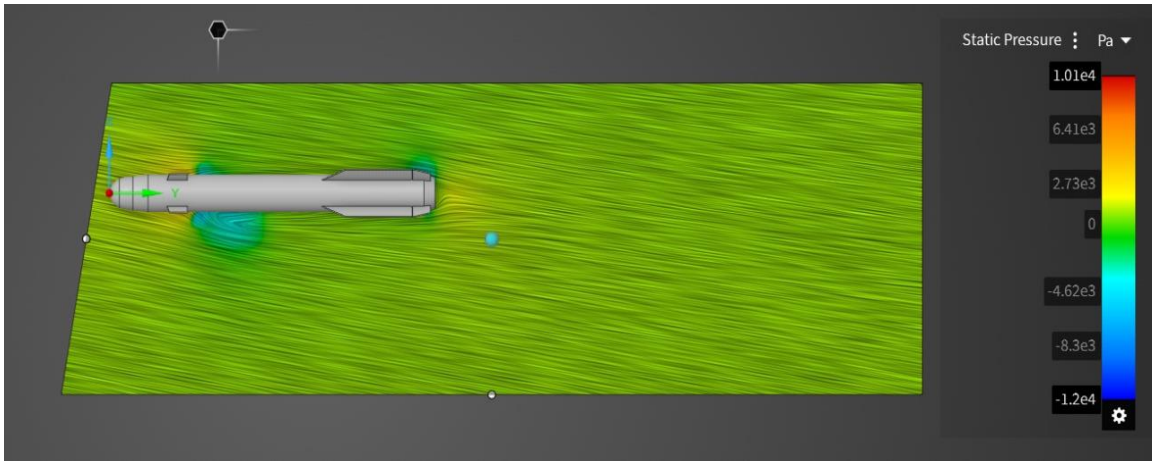


Velocity at α = 0 deg

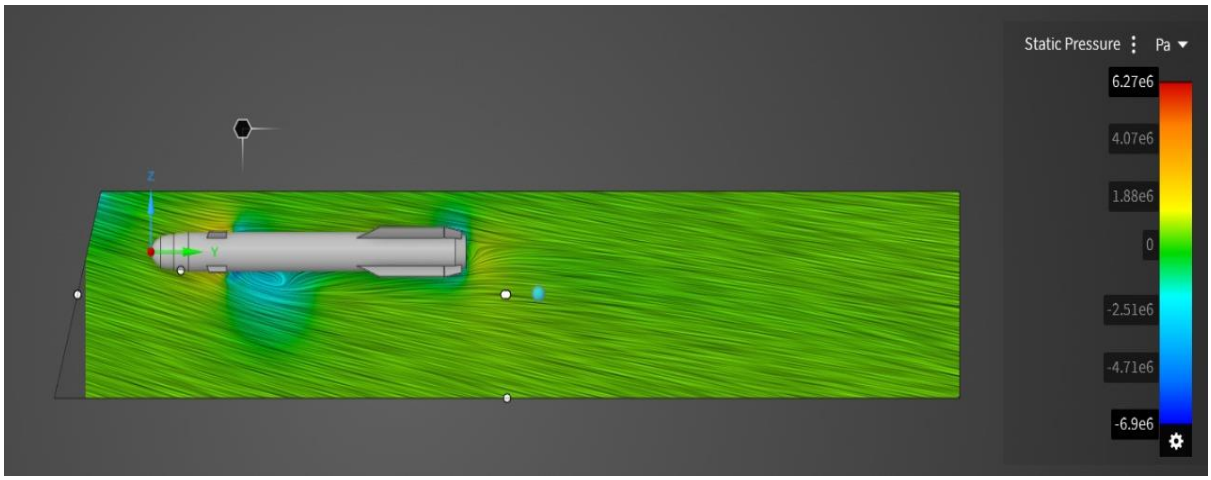
Pressure Contours:



Pressure α = 5 deg

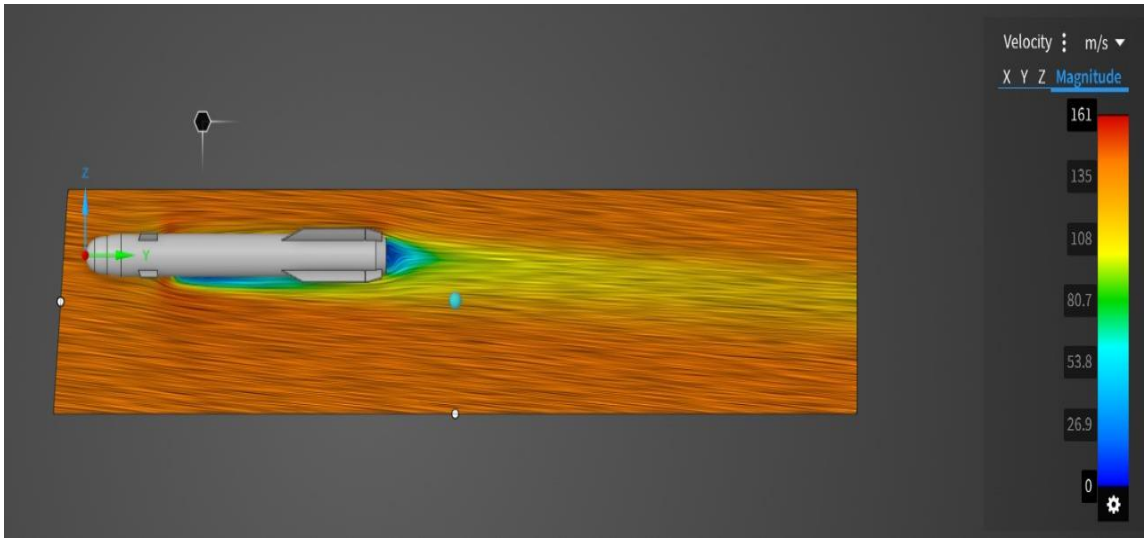


Pressure at α = 10 deg

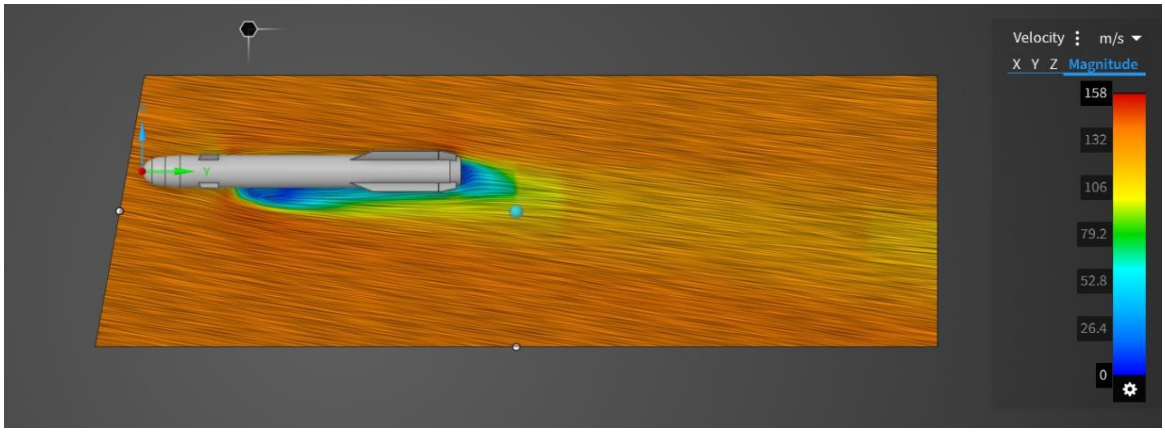


Pressure at α = 15 deg

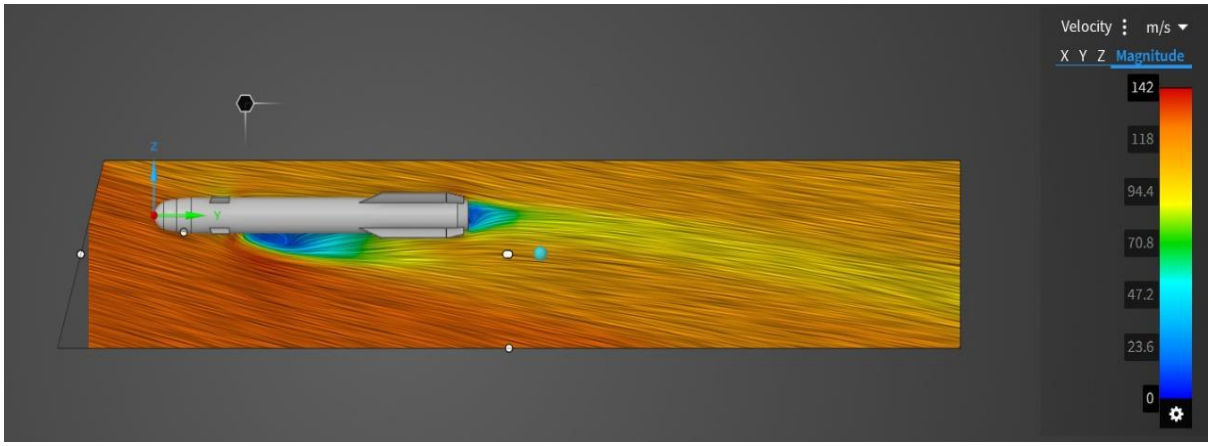
Velocity Contours:



Velocity at α = 5 deg

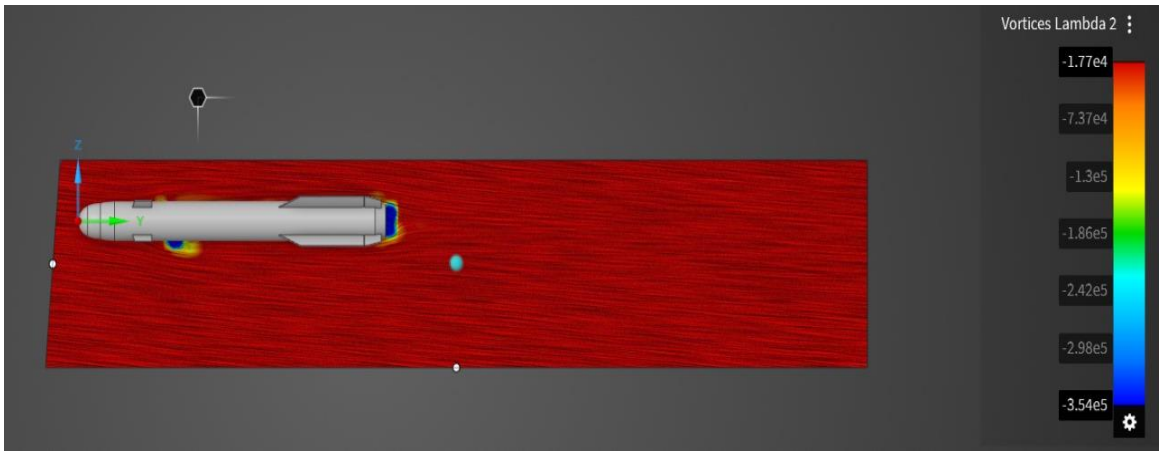


Velocity at α = 10 deg

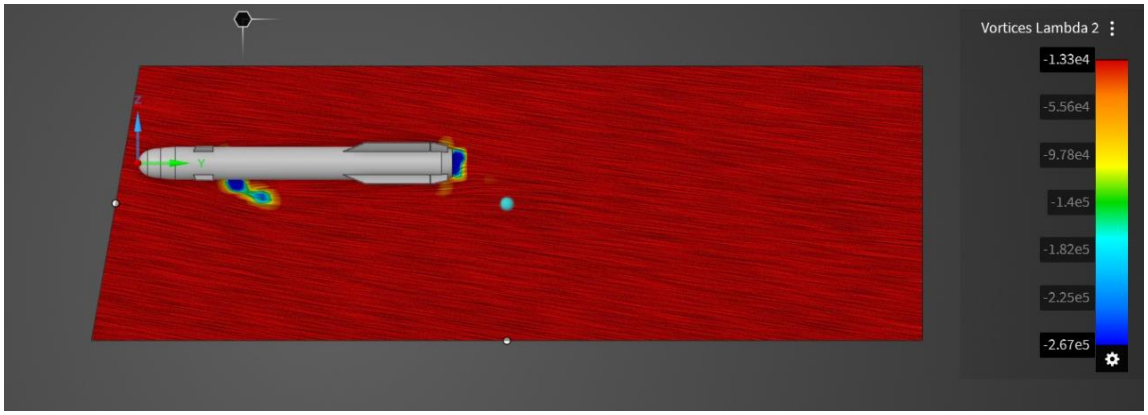


Velocity at α = 15 deg

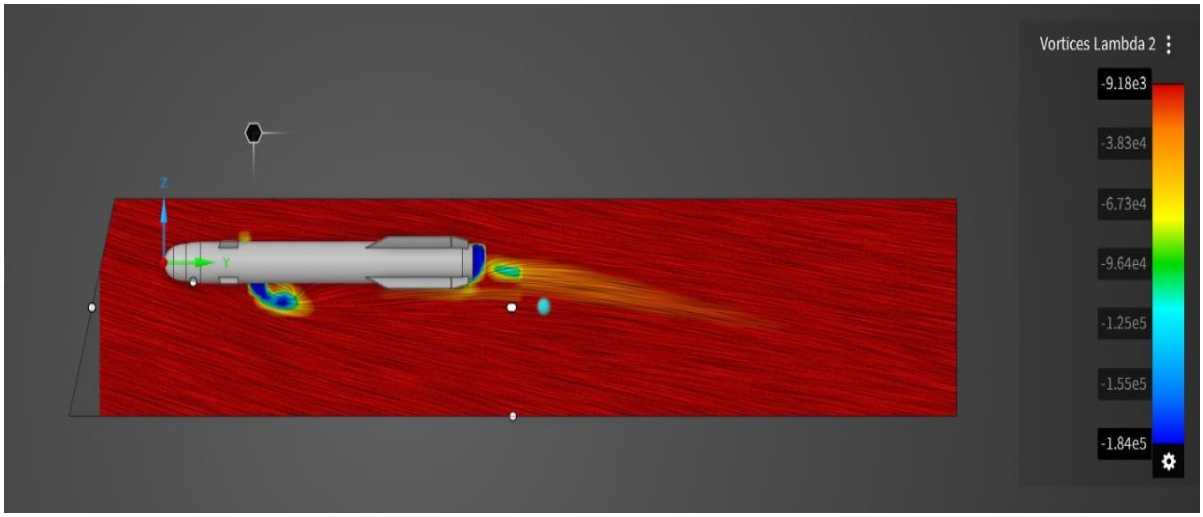
Vortices Contours:



Vortices at α = 5 deg



Vortices at α = 10 deg



Vortices at α = 15 deg

The choice of guidance system will affect the demands on missile manoeuvrability and response, which, in turn, may affect the choice of lifting and control surfaces. The design of domes to protect guidance sensors usually calls for compromises. For example, a hemispherical dome is usually considered the optimum choice for the sensors, but it has high drag. A dome with a high fineness ratio (i.e., length/ diameter ratio) has low drag but tends to degrade the reception of signals by the seeker.