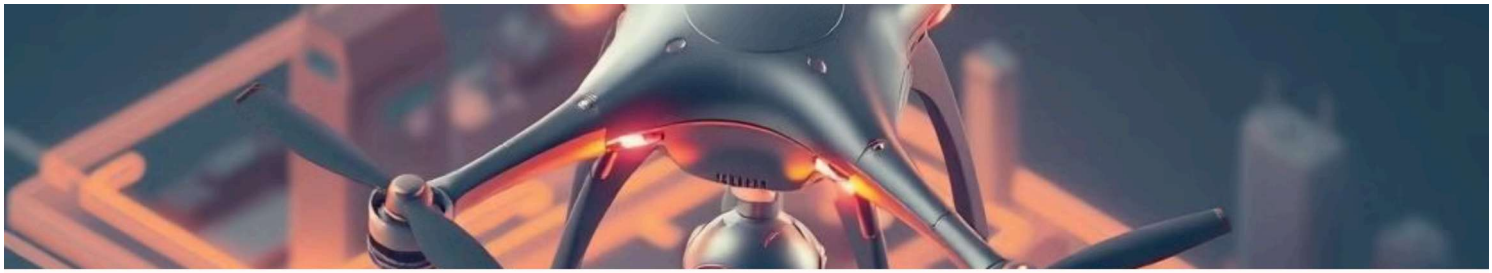


Drone Generation with Catia API

The objective of this project is to develop software solution capable of automating the design and generation of Cyborg Surveillance Drones utilizing the CATIA API (PyCatia). The software integrates a dynamic Graphical User Interface (GUI), which allows users to input key design parameters. These inputs are then processed to algorithmically generate corresponding drone components as CATIA V5 R21 part files. This approach leverages parametric design automation reduce manual intervention.

Name: Md Mashfiq Khan

ID: 30173

Software Demonstration

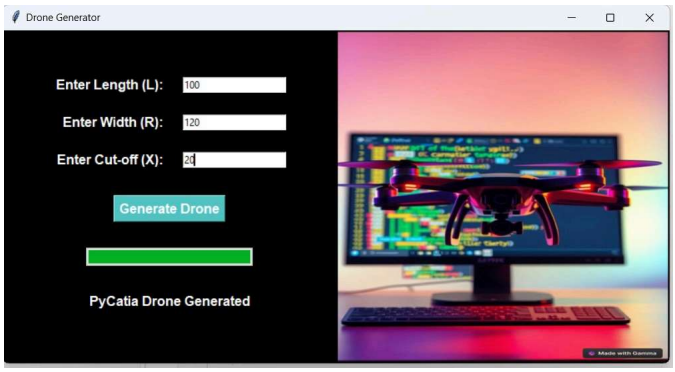


Figure: Front-end

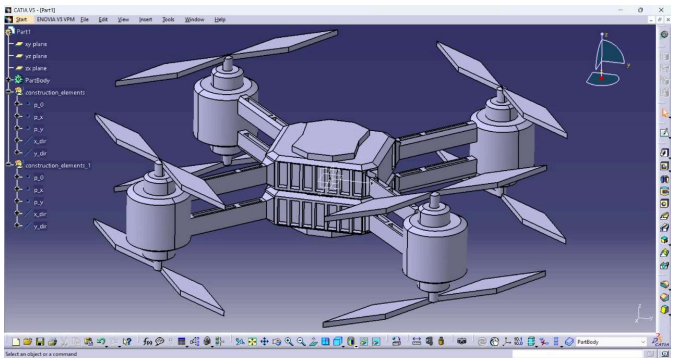


Figure: Back-end

Initialization

Initialization of CATIA Part Document

The `initialize_catia_part` function is central to setting up a structured environment for automating tasks in CATIA using Python. This function ensures a clean, controlled workspace by creating a new CATIA

Part document, free from any interference caused by previously opened documents. Here's a step-by-step explanation of its functionality:

1. Close Any Open Documents

The function begins by accessing the active CATIA application instance and checking for any open documents. If found, these documents are closed to ensure a clean workspace, avoiding potential conflicts from previously opened files.

2. Create a New Part Document

The command `documents.add("Part")` initializes a new, blank part document in CATIA. This document serves as the foundation where geometrical and structural components can be defined and manipulated.

3. Retrieve Essential References

Key references needed for CAD operations are collected to simplify subsequent processes. These include:

- **`application`** : The reference to the CATIA application.
- **`document`** : The active document in which the part is created.
- **`part`** : Represents the part object where design elements are introduced.
- **`partbody`** : The main body of the part, serving as the workspace for solid geometry creation.
- **`sketches`** : A collection of sketches contained within the part body.
- **`hybrid_bodies`** : Hybrid bodies for handling complex, non-standard shapes.
- **`hsf`** and **`shpfac`** : Hybrid Shape Factory and Shape Factory for generating 3D shapes and features.
- **`selection`** : Tool used for selecting objects within the document.
- **`plane_XY`** , **`plane_YZ`** , **`plane_ZX`** : The three main reference planes used for sketching and positioning geometries in 3D space.

4. Set the Active Work Object

The function sets the `PartBody` as the active work object using `part.in_work_object = partbody` . This ensures that any actions, such as creating sketches or applying features, will be executed within the main part body, maintaining a structured and organized design process.

5. Return References

The function concludes by returning a dictionary containing all the essential references, making them easily accessible for future operations and ensuring that subsequent tasks can be performed without needing to retrieve each reference individually.

Purpose of Initialization

To get this Toolbar in code mode

Front-End Development 🚀

The **Front-End** of this project delivers an interface, enabling users to input key parameters to automate the Cyborg Surveillance Drone design. Developed using **Tkinter**, it focuses on efficiency and user experience, allowing for seamless interaction.

Layout and Structure 📐

The interface is divided into an input section on the left and an image display on the right. The **Tkinter window** (`root`) is sized at 800x400, offering a clean and functional layout. The main inputs include `Length (L)`, `Width (R)`, and `Cut-off (X)`, entered via **Entry** widgets, with labels providing instructions.

A **Generate Drone** button triggers the **generate_drone()** function, which processes and validates inputs, ensuring proper ranges for `L` (30-285), `R` (30-285), and `X` (20-30).

Image Handling 🖼️

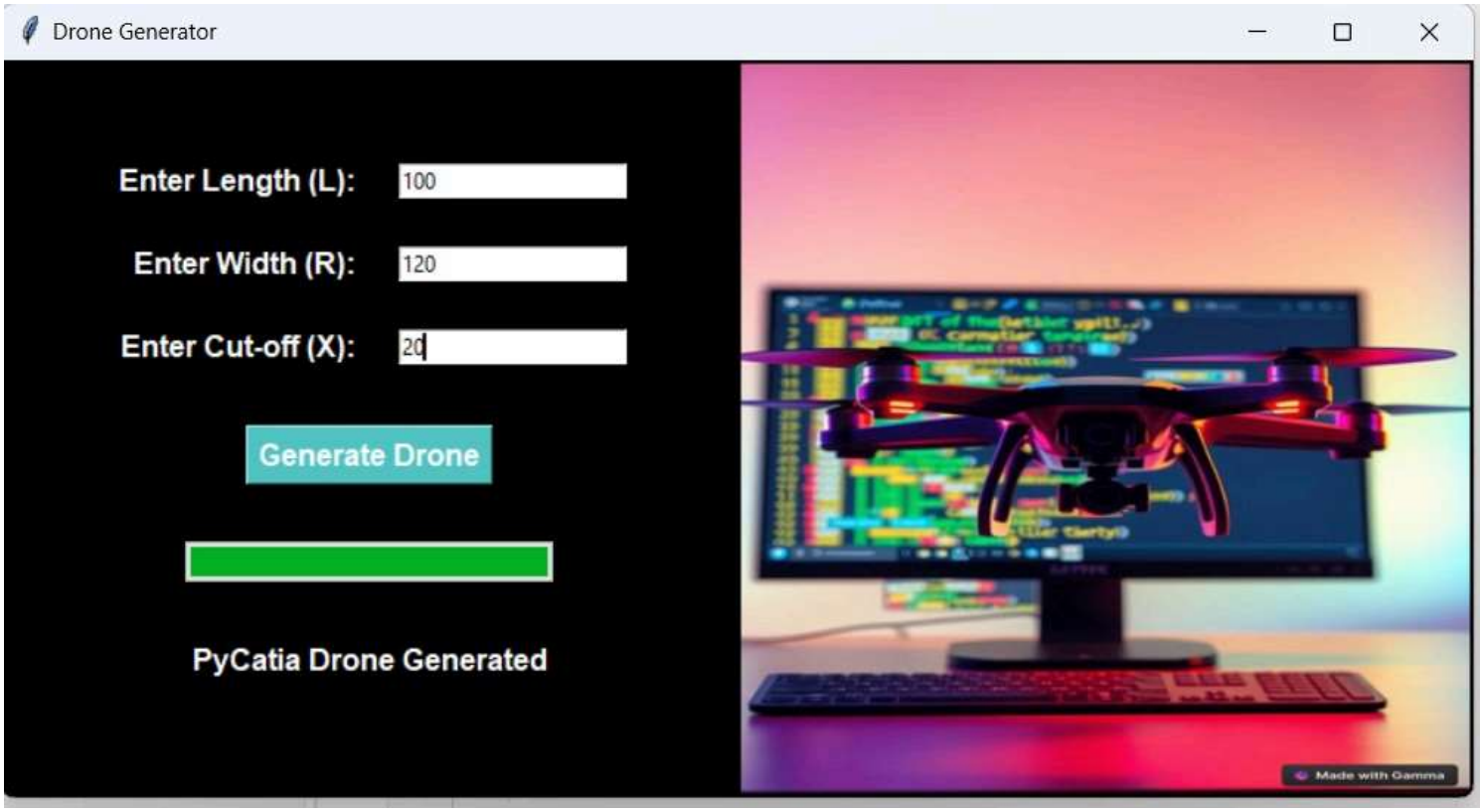
The **load_custom_image()** function manages the display of dynamic images. Using **Pillow**, the images are resized to fit the display frame while maintaining quality through the **LANCZOS** algorithm. This function ensures real-time feedback, enhancing the interactivity of the GUI.

Progress and Status Updates 📊

A **progress bar** visually tracks the generation process, while the **status label** informs users of key updates, like the completion of the drone generation.

Key Functions and Variables ⚙️

- **`generate_drone()`** : Handles user inputs, converts them to integers, and validates their ranges. It updates the progress bar and manages the logic for generating the drone part.
- **L, R, X**: Global variables holding the length, width, and cut-off parameters.
- **output_list**: Stores the validated parameters for further processing.



Back-end (Design Implementation)

Padding the Chastity and Applying Chamfers 🚀

Padding the Chastity 🛠️

This process creates a custom 3D pad from a sketch on the XY plane, driven by parameters that define the shape and size, followed by extrusion into a 3D part.

Key Variables:

- `L` **(Length)**: Horizontal dimension.
- `R` **(Width)**: Vertical dimension.
- `X` **(Offset)**: Adjusts specific points.
- `H` **(Height)**: Extrusion distance.

Process Overview:

1. **Sketch Creation**: The sketch is created on the XY plane, with points like `p1 = (L, R - X)` defining the shape's boundaries.
2. **Closed Loop**: Lines are drawn between points, forming a closed-loop using CATIA's 2D tools.

3. **Extrusion**: The sketch is extruded into a 3D pad with height `H` using `shpfac.add_new_pad()` and regular orientation. The part is updated with `document.part.update()`.

Edges and Chamfer Application ✨

Chamfers are applied to smooth edges, improving the design's aesthetics and function.

Key Variables:

- `G` **(Chamfer Length)**: Chamfer length.
- `Chamfer Angle` : Chamfer angle (default 30°).
- `Selection Tool` : Used to select edges for chamfering.

Process Overview:

1. **Edge Selection**: The user selects 16 edges, which are stored in `edge_references`.
2. **Chamfer Initialization**: The chamfer is applied to the first edge using `shpfac.add_new_chamfer()`, based on the set length, angle, and tangential propagation.
3. **Remaining Edges**: Additional edges are chamfered in sequence for uniformity.
4. **Finalizing**: The document is updated to integrate the chamfered edges using `document.part.update()`.

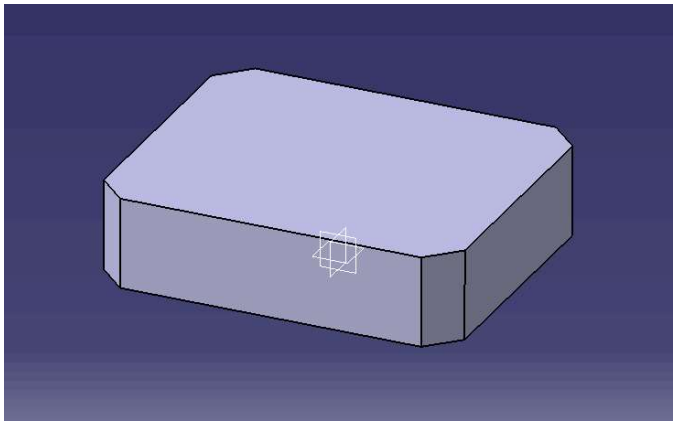


Figure: Padding

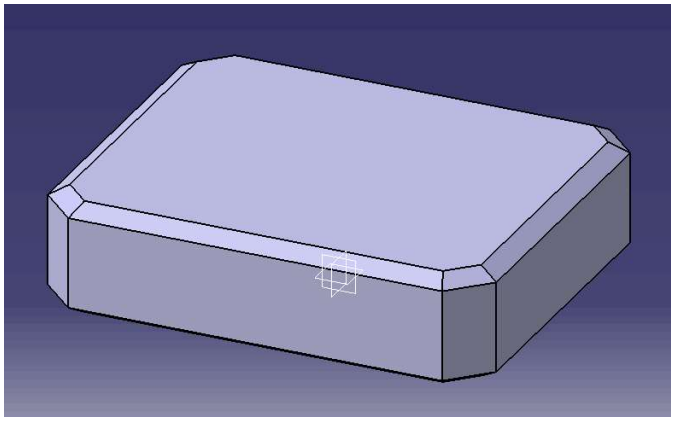


Figure: Chamfer

Rotor Pad Operations 🛠️

Cleaning and Preparing Surface References

This process begins by cleaning the surface reference from JSON format to define the sketch planes for geometry operations. The `remove_selection_rsur_section()` function removes unnecessary prefixes like `Selection_RSur`, making the surface name usable in CATIA.

Key Variables:

- **`input_string`** : Raw surface reference from JSON.
- **`cleaned_string`** : Processed surface name ready for CATIA.

Once cleaned, **`part.create_reference_from_name()`** creates the surface reference for the sketch. This forms the foundation for further geometry.

Process Overview:

1. **Surface Cleanup**: The surface reference is cleaned for CATIA operations.
2. **Sketch Creation**: A new sketch, `"box"` , is created on the cleaned surface, defined by points like `p1` , `p2` , etc.
3. **Extruding the Sketch**: The sketch is extruded into a 3D pad with **`shpfac.add_new_pad()`** , ensuring correct orientation via **`catRegularOrientation`** .

Chamfering Edges Based on Selection

After creating the pad, chamfers are applied to smooth the edges for a refined finish.

Key Variables:

- **`edge_names`** : JSON-stored edge references.
- **`chamfer_length`** and **`chamfer_angle`** : Length and angle of the chamfer.

Process Overview:

1. **Cleaning Edge References**: **`remove_selection_section()`** strips `Selection_REdge` , making the edges suitable for CATIA.
2. **Chamfer Application**: The first edge is chamfered with **`shpfac.add_new_chamfer()`** , using predefined chamfer length and angle.
3. **Chamfer Propagation**: Additional edges are added to the chamfer operation for uniformity.
4. **Finalizing**: The chamfer is rendered with **`document.part.update()`** .

Pocket Creation on Defined Surfaces

This step creates a pocket on a defined surface, allowing for material removal.

Key Variables:

- **`pocket_depth`** : Depth of the pocket.
- **`offset_x`** , **`offset_y`** : Adjustments to pocket placement.

Process Overview:

1. **Surface Reference Cleanup**: Cleans the surface for pocket sketching.
2. **Sketching the Pocket**: A rectangle is sketched using points, with offsets applied for accurate placement.
3. **Pocket Creation**: The sketch is extruded into a pocket with `shpfac.add_new_pocket()`, using the defined depth and `catInverseOrientation`.
4. **Finalizing**: The pocket is integrated with `document.part.update()`.

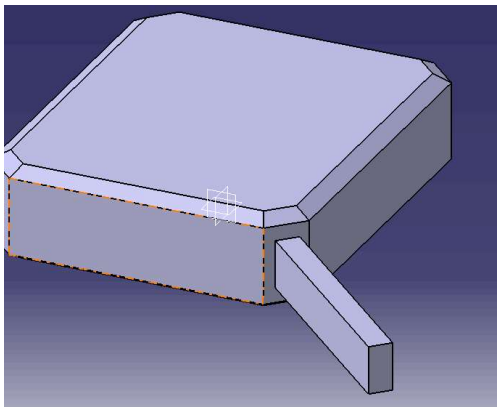


Figure: Rotor Pad

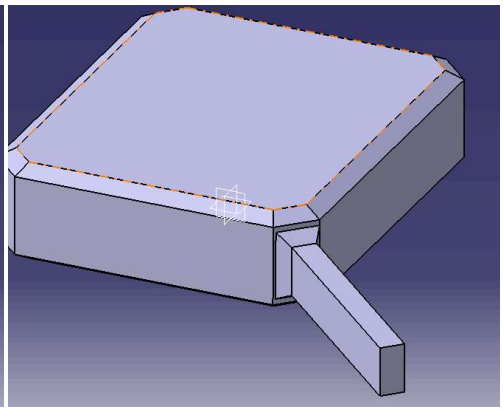


Figure: Chamfer

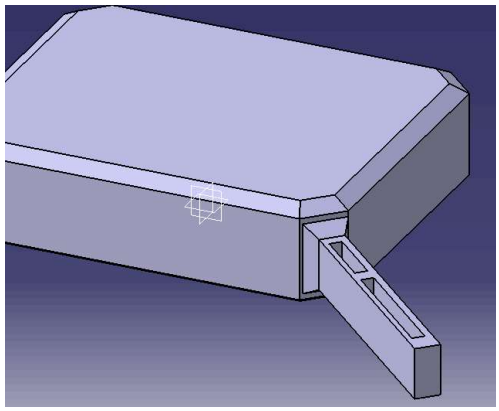


Figure: Pocket

Rotor

Rotor Outer Circle and Chamfer Application

This process automates the creation and refinement of the rotor's outer circle and edge chamfering using JSON references.

Rotor Outer Circle Creation:

1. **Surface Reference**: Loaded from `circle_ref.json` and cleaned using `remove_selection_rsur_section()`.
2. **Sketch Creation**: A circle with a radius of 50 is drawn and extruded into a 3D pad with a height of 45 units.
3. **Part Update**: The part is updated to reflect the rotor geometry.

Chamfering the Edges:

1. **Edge References**: Loaded from `circle-edges.json` and cleaned.
2. **Chamfer Application**: Chamfers with a length of 10 and an angle of 30 are applied.
3. **Part Update**: The model is updated after chamfering.

Rotor Paddle Creation and Chamfering

Rotor Paddle Creation:

1. **Surface Reference**: Loaded from `rotor_paddle_surface.json` and cleaned.
2. **Sketch Creation**: A circular sketch with a radius of 25 is drawn and extruded into a 3D pad of 25 units.
3. **Part Update**: The new paddle is integrated into the model.

Chamfering Rotor Paddle Edges:

1. **Edge References**: Loaded from `rotor-edges.json` for chamfering.
2. **Chamfer Application**: Chamfers of 7 units and an angle of 30 are applied to the edges.
3. **Part Update**: Finalize the chamfering operation and update the part.

Saddle Creation and Chamfering

Saddle Creation:

1. **Surface Reference**: Loaded from `saddle_ref_surface.json` and cleaned.
2. **Sketch Creation**: A circle with a radius of 8 is drawn and extruded into a 3D pad of 25 units.
3. **Part Update**: The saddle is updated into the model.

Chamfering Saddle Edges:

1. **Edge References**: Loaded from `saddle-edges.json` .
2. **Chamfer Application**: Chamfers with a length of 7 and an angle of 30 are applied.
3. **Part Update**: The chamfering is finalized and applied to the model.

Wing Creation

Wing Geometry Definition:

1. **Surface Reference**: Loaded from `wing-surface.json` and cleaned.
2. **Sketch Creation**: Two mirrored sections of the wing are created using points and lines.

Extrusion (Pad Creation):

1. **Pad Creation**: The wing is extruded into a 3D pad of 9 units.
2. **Part Update**: The wing structure is added to the model.

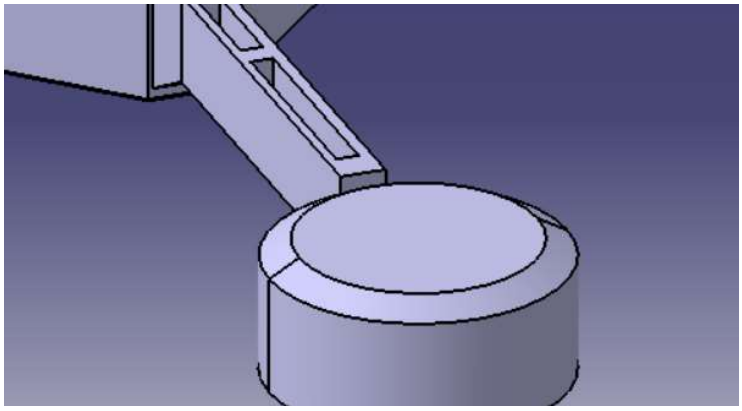


Figure: Rotor Base

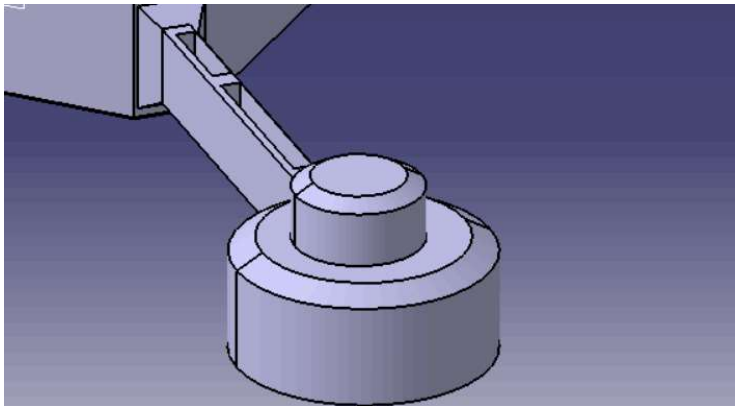


Figure: Rotor Pad

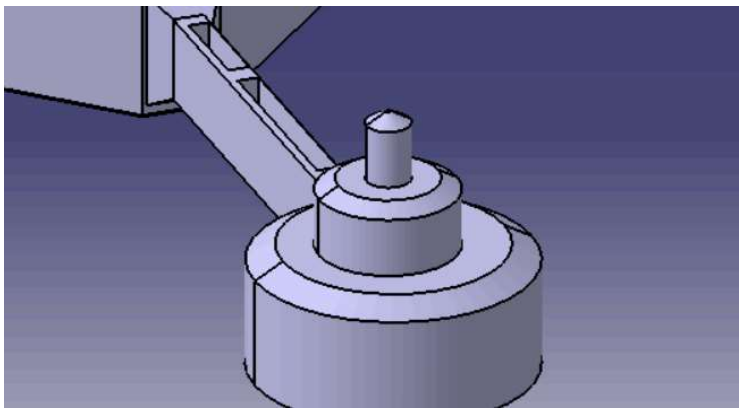


Figure: Rotor Saddle

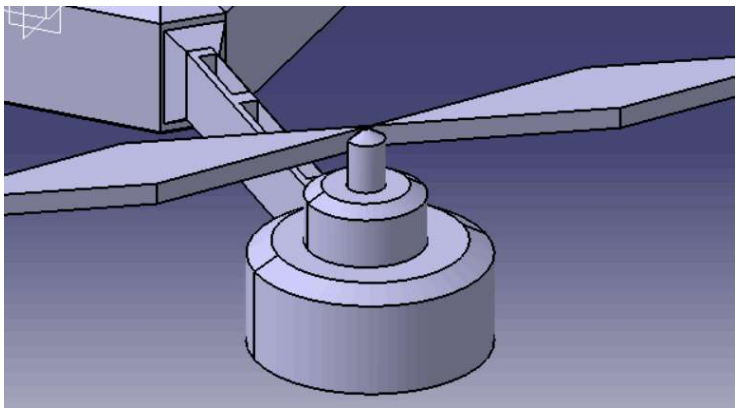


Figure: Rotor Wing

Surface Design and Geometry Operations

Top Surface Pad Creation

This section covers the creation of a top surface sketch and its extrusion into a 3D pad.

- **Surface Reference**: The surface reference is loaded and cleaned from `T.json` and used for sketching.
- **Sketch Creation**: A sketch named `"Top"` is created on the XY plane. Points (`p1` to `p8`) are calculated using scaled values of `L` , `R1` , and `X` .
- **Line Drawing**: Lines are drawn between the points to form a closed shape for the top surface.
- **Extrusion (Pad)**: The sketch is extruded into a pad with a height of `H + 10` using **`shpfac.add_new_pad()`** . The part is then updated to reflect the changes.

Mirroring Geometry

This section describes the mirroring of 3D geometry to ensure symmetry.

- **First Mirror Operation**: The geometry is mirrored across the **ZX plane** using `shpfac.add_new_mirror(plane_ZX)` to create symmetry along the ZX axis.
- **Second Mirror Operation**: The geometry is mirrored across the **YZ plane** using `shpfac.add_new_mirror(plane_YZ)`, ensuring symmetry along the YZ axis.
- **Part Update**: After each mirror operation, `document.part.update()` is called to apply the mirrored geometry to the model.

Zig Pocket Creation and Rectangular Pattern

This section automates the creation of a "Zig" pocket and applies a rectangular pattern.

Zig Pocket Creation:

1. **Surface Reference**: The surface reference is loaded from `Zig.json`, cleaned, and used for sketching.
2. **Sketch Definition**: A sketch named `"Zig"` is created with four points defining a rectangle.
3. **Pocket Creation**: The sketch is extruded inward to a depth of `-10` using `shpfac.add_new_pocket()`.

Construction Elements:

1. **Points and Directions**: Points (`p_0`, `p_x`, `p_y`) are created using `hsf.add_new_point_coord()`, and lines (`x_dir`, `y_dir`) are defined as references.
2. **Hybrid Body**: A hybrid body named `"construction_elements"` is created to store these shapes.

Rectangular Pattern:

1. **Pattern Application**: A rectangular pattern is applied to the pocket using `shpfac.add_new_rect_pattern()`, creating a grid of copies along the x and y directions.
2. **Part Update**: The part is updated to reflect the changes in the 3D model.

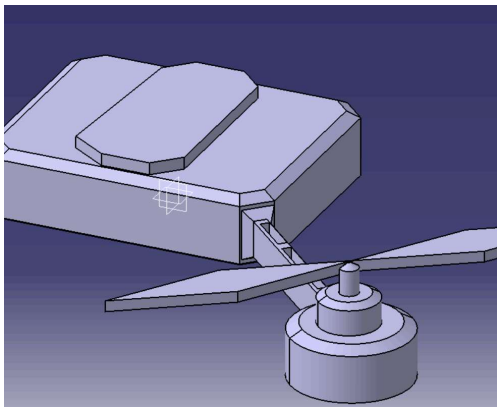


Figure: Cushioning Pad

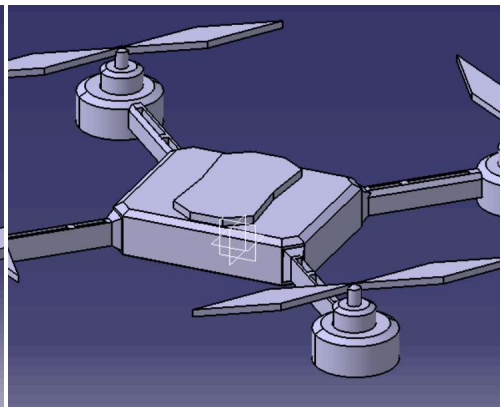


Figure: Mirror

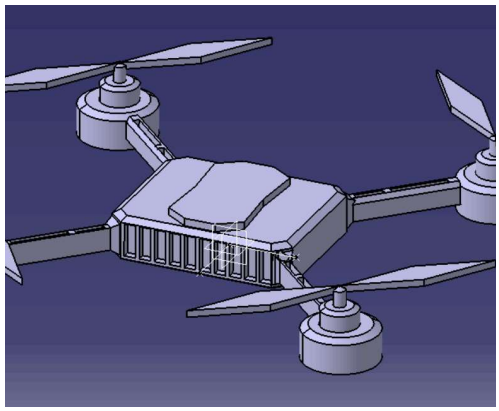


Figure: Fins

Mirror Operation on XY Plane

This section performs a mirror operation to create a symmetrical reflection of the existing geometry across the XY plane.

- **Mirror Operation**: The geometry is mirrored across the **XY plane** using `shpfac.add_new_mirror(plane_XY)`, creating a symmetric copy of the original geometry along the XY axis.
- **Part Update**: The part is updated using `document.part.update()` to integrate the mirrored geometry into the CATIA model.

This ensures that the geometry is symmetrically extended across the XY plane for balanced design.

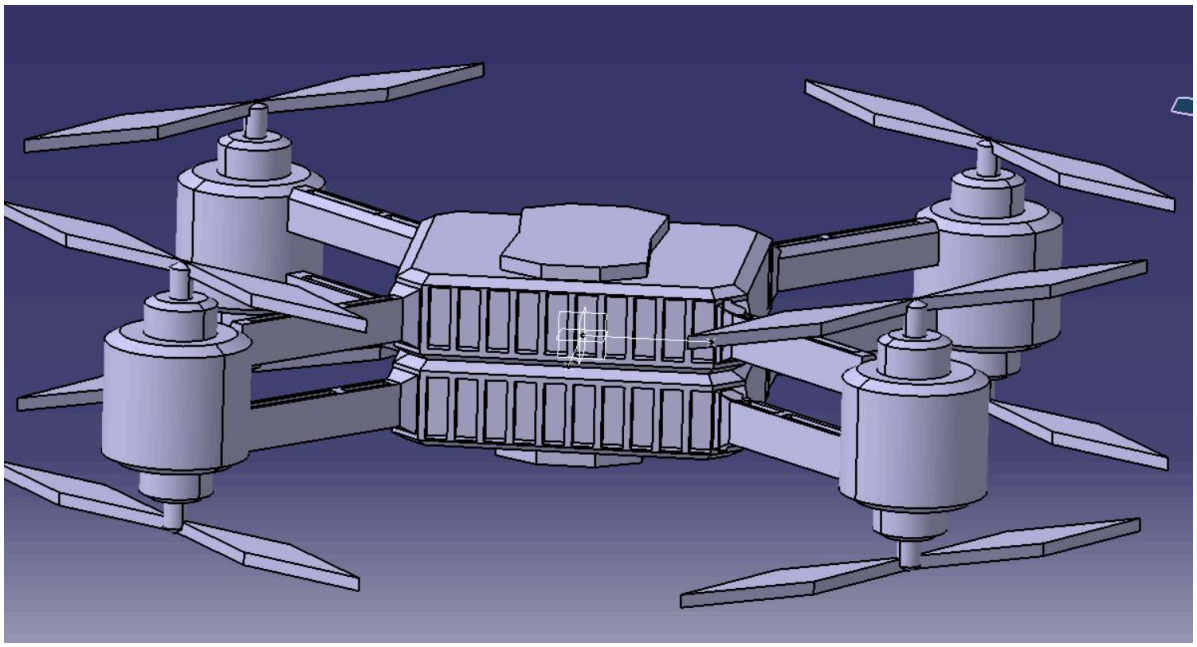


Figure: Final Design