```
using UnityEngine;

public class PhysicsDrone : MonoBehaviour
{
    [Tooltip("The model to use for displaying rotations")]
    [SerializeField]
    private Transform model;
    [SerializeField]
    [Tooltip("The maximum rotation angle the model can reach in the x and z axis")]
    private Vector2 maxRotations;
    [Header("Control")]
    [SerializeField]
    private float throttleIncrement = 0.1f;
    [SerializeField]
    private float maxThrust = 200f;
    [SerializeField]
    private Vector2 responsiveness = new Vector2(10f, 10f);

    private float throttle;

    // Input values
    private float yaw;
    private float pitch;

    /// <summary>
    /// Makes it easier for the model rotation to reach its full value
    /// </summary>
    private Vector2 ScaledResponse => responsiveness / 10f;
    /// <summary>
    /// The weight of the drones effect on responsiveness
    /// </summary>
    private Vector2 ResponseModifier => (rb.mass / 10f) * responsiveness;

    private Rigidbody rb;

    /// <summary>
    /// Where the drone will respawn when colliding
    /// </summary>
    private Vector3 startLocation;

    // Start is called before the first frame update
    void Start()
    {
        rb = GetComponent<Rigidbody>();

        startLocation = transform.position;
    }

    private void HandleInputs()
    {
        yaw = Input.GetAxis("Horizontal");
        pitch = Input.GetAxis("Vertical");
    }

    // Update is called once per frame
    void Update()
    {
        // Update/Increase throttle until it reaches max
        throttle = Mathf.Clamp(throttle + throttleIncrement, 0f, 100f);
        HandleInputs();
        HandleModelRotation();
    }
```

```
    private void HandleModelRotation()
    {
        // - 0.5 allows the range of both x and y to fall into negatives,
        // while * 2 makes sure it falls between -1 and 1
        var xRotation = (Mathf.InverseLerp(-(ScaledResponse.x), ScaledResponse.x, rb.velocity.x)
- 0.5f) * 2;
        xRotation *= maxRotations.x;

        var yRotation = (Mathf.InverseLerp(-(ScaledResponse.y), ScaledResponse.y, rb.velocity.y)
- 0.5f) * 2;
        yRotation *= maxRotations.y;

        model.localRotation = Quaternion.Euler(Vector3.forward * -xRotation +
                                               Vector3.right * -yRotation);
    }

    private void FixedUpdate()
    {
        // Constant forward force
        rb.AddForce(transform.forward * (maxThrust * throttle));

        // Input determined forces
        rb.AddForce(transform.right * (yaw * ResponseModifier.x)); // Left Right Movement
        rb.AddForce(transform.up * (pitch * ResponseModifier.y)); // Up Down Movement
    }

    private void OnCollisionEnter(Collision other)
    {
        Respawn();
    }


    /// <summary>
    /// Returns the drone to the initial position when the game started & resets rigidbody speed
    /// </summary>
    private void Respawn()
    {
        transform.position = startLocation;
        rb.velocity = Vector3.zero;
        throttle = 0;
    }
}
```

```
using UnityEngine;

/// <summary>
/// Keep the GameObject at a constant world rotation
/// </summary>
public class RotationLock : MonoBehaviour
{
    [SerializeField]
    private Vector3 rotation = Vector3.zero;

    // What rotation Axis to lock, so that not everything is constant
    [Header("Axis Lock")]
    [SerializeField, Tooltip("X")]
    private bool pitch;
    [SerializeField, Tooltip("Y")]
    private bool yaw;
    [SerializeField, Tooltip("Z")]
    private bool roll;

    void Update()
    {
        // cache the current rotation in eulers
        var eu = transform.rotation.eulerAngles;
        // Depending on the Axis lock values,
      // either set the world rotation to the rotation field or the cached Euler
        transform.rotation = Quaternion.Euler(!pitch ? eu.x : rotation.x,
                                              !yaw ? eu.y : rotation.y,
                                              !roll ? eu.z : rotation.z);
    }
}
```