SUMMER ENGINEERING RESEARCH INTERNSHIP FOR US STUDENTS (SERIUS)

PROJECT REPORT

Developing digital twin of interacting systems in complex sea scenarios

Gautam Makheja

Department of Electrical And Computer Engineering, College of Design and Engineering, National University of Singapore

Abstract

This paper explores the application of digital twin technology in modeling the predictability of offshore assets in varying weather conditions, with a focus on the integration of Unity for simulation development. In order to master Unity and its C# programming fundamentals, a hands-on sample game was created by the name of "Kitchen Chaos Delivery Manager." This approach facilitated a deeper understanding of game mechanics and software functionalities.

Additionally, the research examined the implementation of PID (Proportional-Integral-Derivative) controllers and dynamic positioning technologies, crucial for maintaining stability in maritime operations. The study emphasized the theoretical foundations and practical applications of these control systems, illustrating their importance in addressing challenges faced by offshore assets. Outcomes included a robust simulation, demonstrating vessel movement through effective control mechanisms, highlighting the significance of advanced technologies in enhancing operational safety and efficiency. This contribution to knowledge underlines the relevance of integrating digital twin methodologies and control systems in the maritime industry, paving the way for future innovations in offshore asset management.

Preface And Acknowledgement

This research report aims to enhance the controllability and predictability of critical FLNG (Floating Liquefied Natural Gas) operations, contributing to the advancement of knowledge in this field. The research conducted in this project has been supported by the Controls and Simulation Lab at the National University of Singapore.

First and foremost, sincere gratitude is extended to Professor Shuzhi Sam Ge and Dr. Liang Xiaoling for their oversight and guidance in project design. Their valuable insights and expertise have been key in shaping the direction of this research. Their expertise, patience, and encouragement have been invaluable in overcoming challenges and achieving significant progress.

Acknowledgement is given to family members for their unwavering encouragement and belief. Their constant support has been a source of motivation throughout the journey at the National University of Singapore, far from home.

Lastly, gratitude is expressed to the academic community and all individuals who have dedicated their time and effort to advancing the field of offshore operations. Especially Mr. Siqi Zhang, who undertook the project last year and was able to build the foundation for the Digital Twin model that I am currently developing.

1.Introduction

The aim of digital twin technology is to essentially model the predictability of offshore assets in varied weather conditions, which includes a variation in terms of the sea and wind conditions. This report talks about my learning of Unity technology, by building a sample game, in order to master the very basics of the Unity language, followed by my exploration of maritime technology and how it can be modeled by different models such as the PID controller technology and the dynamic positioning technology.

Firstly, I am embarking on an exploration of Unity by following a comprehensive tutorial to build a sample game. This hands-on approach allows me to grasp the overall functionalities and intricacies of the Unity platform. By engaging in this practical exercise, I aim to familiarize myself with the Unity development environment, understand its scripting language, and acquire essential skills in game development. This foundational knowledge will be instrumental as I progress to developing the digital twin in Unity.

In parallel, I am delving into various technologies, with a particular focus on PID controller technology and dynamic positioning technology. Understanding these technologies involves studying their theoretical underpinnings, and the mathematical models that drive their functionality. This learning phase is crucial as it equips me with the knowledge needed to implement these technologies effectively in dynamic systems. In this report, I aim to synthesize my knowledge in the **simplest** way possible, so that the information is lucid and comprehensive.

Finally, I am implementing PID controller technology and dynamic positioning technology on the digital twin model built by my predecessor, Mr. Siqi Zhang. By applying what I have learned, I aim to create robust control systems that can be integrated into the current digital twin model of the ship vessels, such that it can accurately model the movement of the ship. This phase not only reinforces my understanding but also allows me to innovate and contribute to the field of control systems with tailored implementations.

2. Learning Unity by Building a Sample Game - Kitchen Chaos Delivery Manager

In order to learn Unity efficiently, I followed the Unity tutorial of Kitchen Chaos on Code Monkey [1]. This tutorial was followed meticulously to create the best game possible. To put it simply, Kitchen Chaos Delivery Manager is an engaging and fast-paced simulation game where players take on the role of a delivery manager for a bustling kitchen. The **objective** is to manage the preparation and delivery of food orders efficiently while dealing with various challenges that arise in a chaotic kitchen environment.

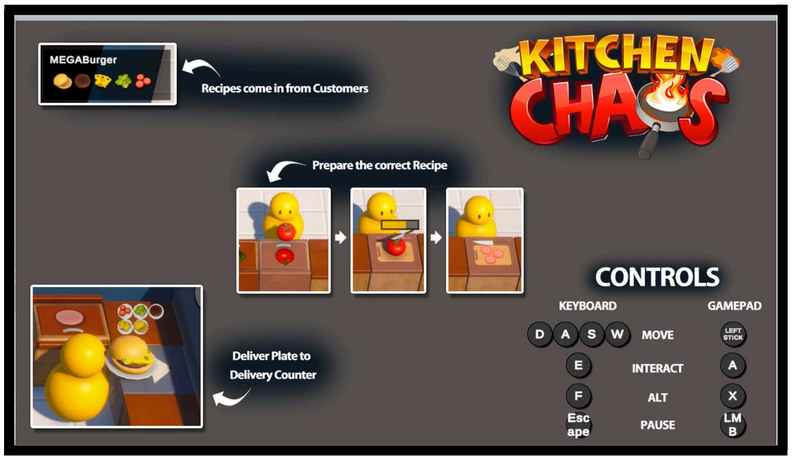


Fig 1. Quick Image at the start of the Game describing how to operate the Game

Here is a quick overview of the controls in order to play the game:

- **Orders:** Customers will appear with order requests displayed as "MEGABurger" with ingredients like tomatoes, cheese, etc.
- **Preparation:** The player, represented by a yellow character, will need to collect the required ingredients and prepare the order. This could involve cutting, slicing, and assembling items.
- **Delivery:** Once the order is complete, the player needs to deliver it to the "Delivery Counter" to satisfy the customer.

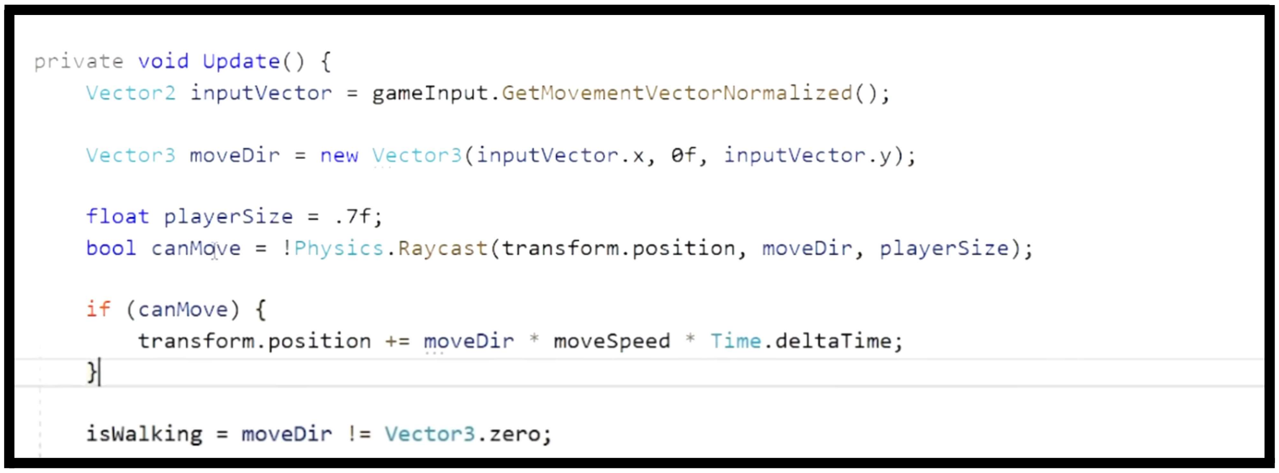
Controls:

- WASD: Move the character around the kitchen.
- E: Interact with objects like ingredients, counters, etc.
- F: Possibly used for specific actions like chopping or grabbing items.

- Esc: Escape the game.
- Alt: Might pause the game or access settings.

Over the course of developing the game, I learnt the core basic of C# logic and also the mechanics of how to use the Unity software. Here are some key points, wherein I believe that I could show you how I was able to develop my code during these parts.

2.1 Detecting and Resisting Movement when encountering Barriers



```
private void Update() {
    Vector2 inputVector = gameInput.GetMovementVectorNormalized();

    Vector3 moveDir = new Vector3(inputVector.x, 0f, inputVector.y);

    float playerSize = .7f;
    bool canMove = !Physics.Raycast(transform.position, moveDir, playerSize);

    if (canMove) {
        transform.position += moveDir * moveSpeed * Time.deltaTime;
    }

    isWalking = moveDir != Vector3.zero;
```

Fig 2. Detecting Movement is often done through the use of Boolean Variables

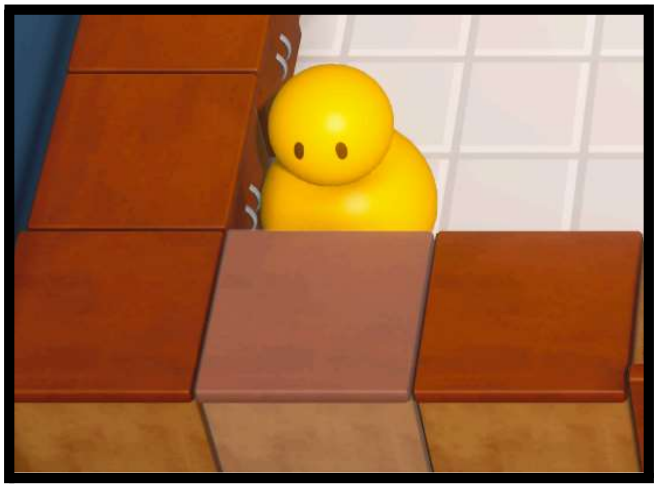
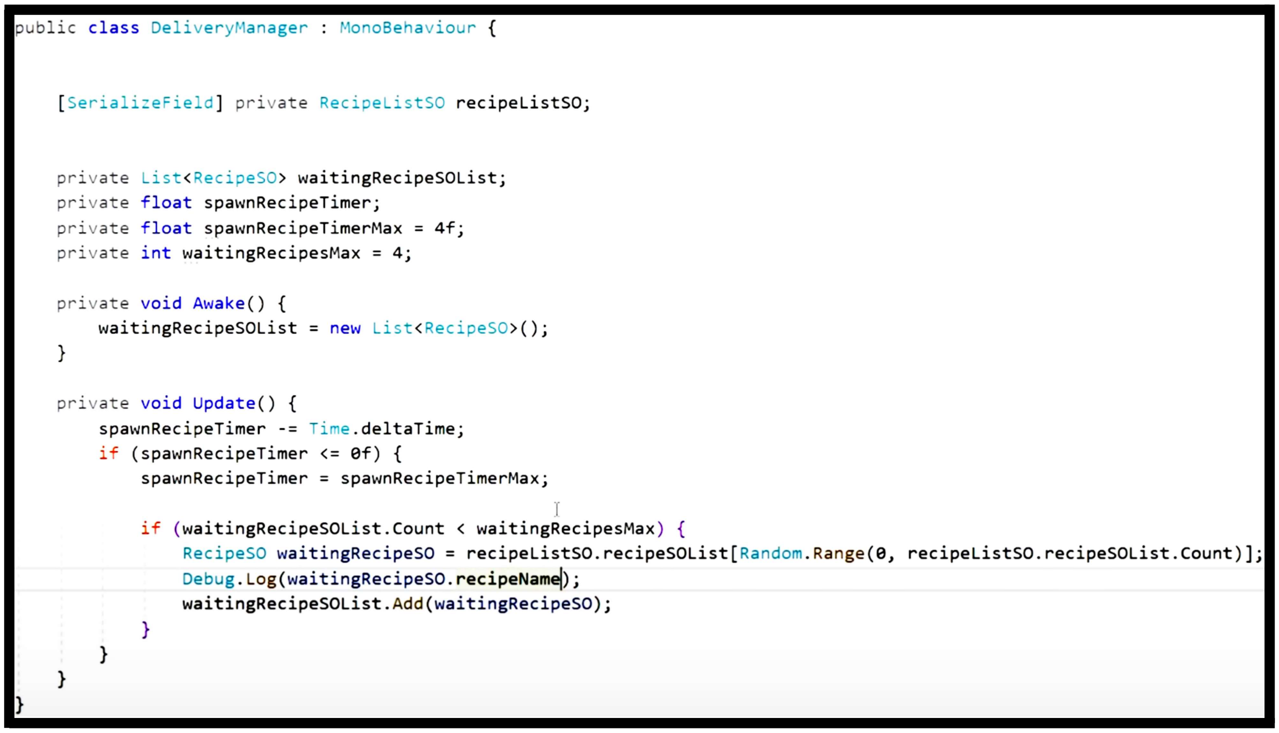


Fig 3. When encountered by a kitchen table, the player is stopped from moving forward.

One key aspect of learning Unity was using the different modules; for example, in the aforementioned image, I have used the Physics module to observe whether or not the Player is able to move in the specified direction. The Physics.Raycast operation determines whether or not the ray projected by the transform.position coordinate (in this case the player) will collide with an object in the direction of the vector moveDir, and within the distance determined by playerSize. Thus, if the player does not collide with any obstacle along the way, then it means that the player is free to move forward by the duration of time he has pressed that key multiplied by the moving speed of the player.

2.2 Storing the Serialised List of Recipes for the Customer



```
public class DeliveryManager : MonoBehaviour {


    [SerializeField] private RecipeListSO recipeListSO;


    private List<RecipeSO> waitingRecipeSOList;
    private float spawnRecipeTimer;
    private float spawnRecipeTimerMax = 4f;
    private int waitingRecipesMax = 4;

    private void Awake() {
        waitingRecipeSOList = new List<RecipeSO>();
    }

    private void Update() {
        spawnRecipeTimer -= Time.deltaTime;
        if (spawnRecipeTimer <= 0f) {
            spawnRecipeTimer = spawnRecipeTimerMax;

            if (waitingRecipeSOList.Count < waitingRecipesMax) {
                RecipeSO waitingRecipeSO = recipeListSO.recipeSOList[Random.Range(0, recipeListSO.recipeSOList.Count)];
                Debug.Log(waitingRecipeSO.recipeName);
                waitingRecipeSOList.Add(waitingRecipeSO);
            }
        }
    }
}
```

Fig 4. Waiting and adding the Recipes for the Player whenever the timer counts down

Another key highlight of the game is where the recipes are spawned every few seconds to generate the next recipe for the player to develop and serve. In the above figure, a new list is created in the private function Awake(), but in the Update() Section, it can be observed that whenever the spawnRecipeTimer goes below 0, it generates a new recipe. Moreover, it can also be observed that whenever the recipe list count is below 4 (waitingRecipesMax), a new recipe is generated to ensure the fact that the length of the list of recipes always remains at 4.

3. Vessel Movement with PID Controller Technology

A PID controller is a type of technology used in many control systems to maintain a desired level of performance [2]. This performance is often related to operations including, but not limited to speed, pressure, temperature, and other variables, allowing the ability for a variable to maintain that set value and compensate for the changes in the environment. "PID" stands for Proportional, Integral, and Derivative, which are the three components used in the controller. PID controllers continuously monitor the difference between the output value named a process variable (PV) and setpoint (SP) and play their role using proportional, integral, and deviation mechanisms. A practical example of this PID control system is found in controlling the steering system for ships, this technology developed in the 1920s.

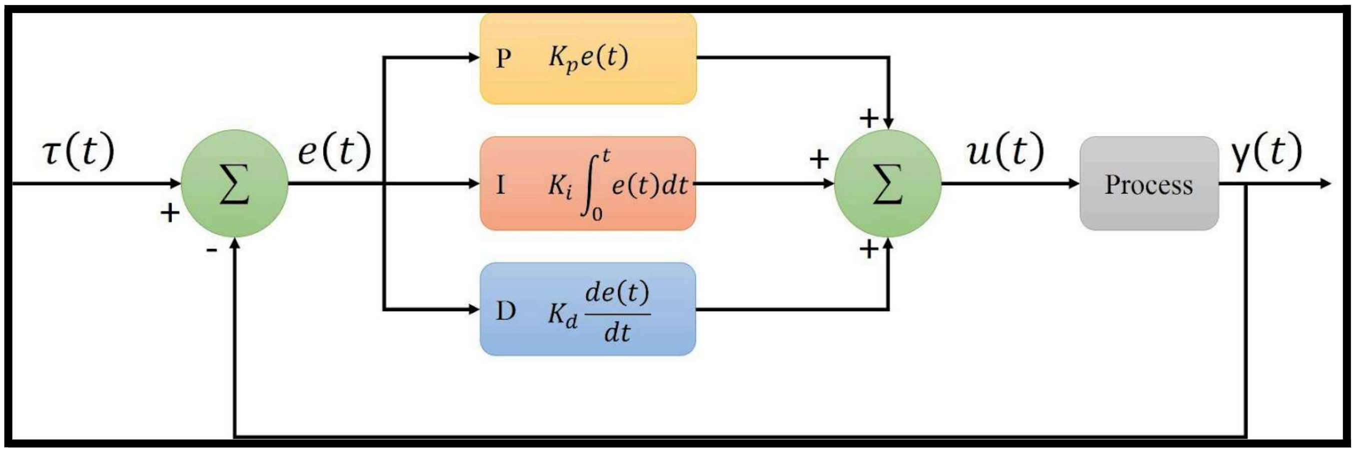


Fig 5. τ(t) is the setpoint or desired value while y(t) is the value of the variable known as a process variable (PV) [2].

The value of the process variable (PV) is calculated as y(t) : e(t) = τ(t) – y(t), and the goal of this technology is to maintain the PV as close as possible to the setpoint variable (SV). In order to maintain the PV as close as possible to SV, the control system technology known as PID controller is used.

Here's a simple breakdown:

1. **Proportional (P)**: This part of the controller reacts to the current error. The error is the difference calculated as PV - SV. At a specific moment in time, if there is no error in the system, no action will be taken by the controller. Another important fact to note is that this variable is independent of time, as it will only respond to large differences every specific time period, and thus this is considered first-order
2. **Integral (I)**: This part looks at the accumulated error over time, by summing it all. If there's a persistent, small error, the integral component will add up these small errors and adjust the output to eliminate them. This variable is also known to be second-order, as it accumulates the memory of the past changes and tries to diminish these changes over time.
3. **Derivative (D)**: This part predicts future errors based on the rate of change of the error. It looks at how quickly the error is changing and adjusts the output to prevent the error from changing too rapidly. Derivative action enhances control performance around setpoint conditions because it acts as the rate of change of the controller output.

The overall mathematical calculation of the PID controller can be viewed below as follows:

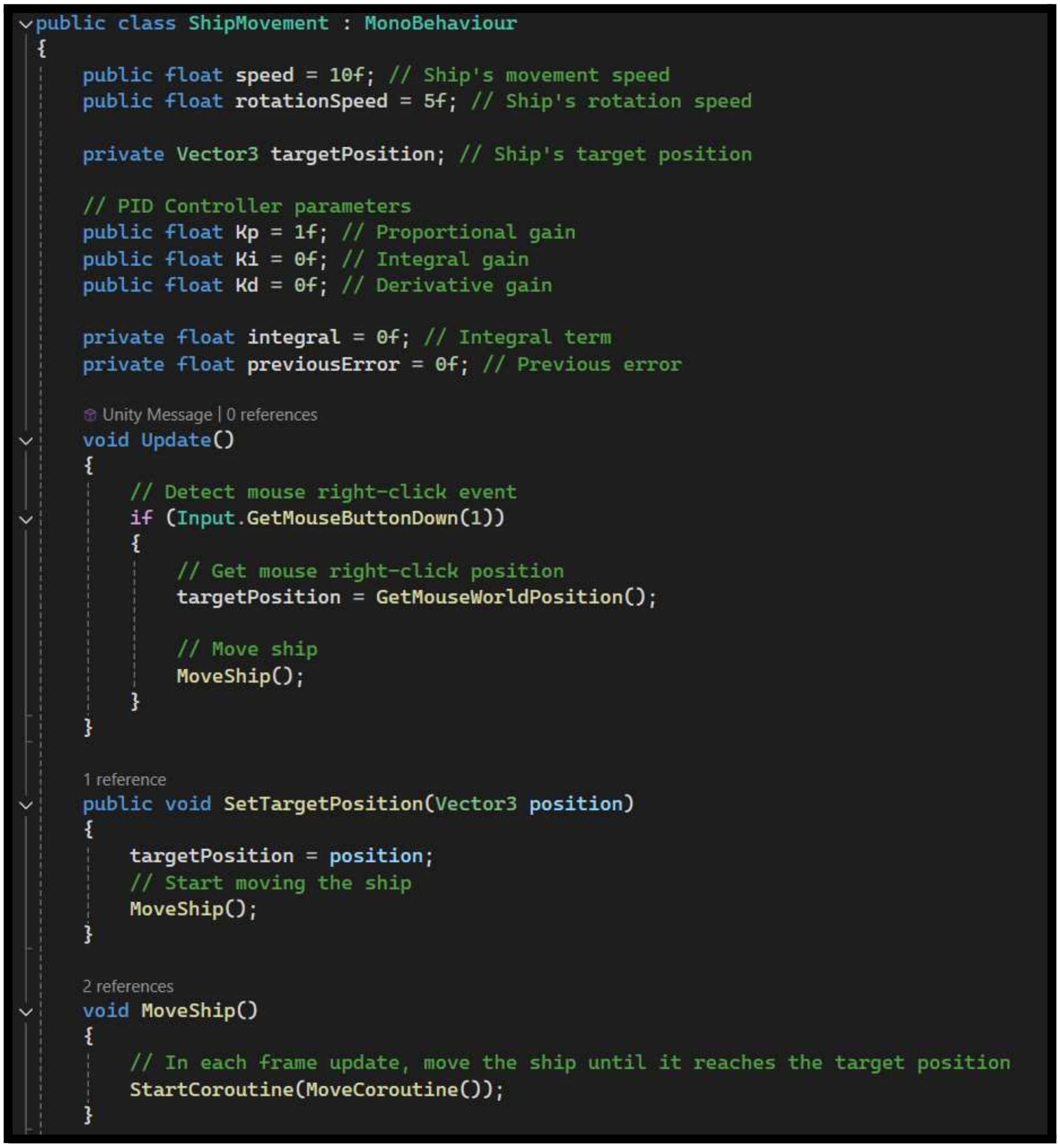
$$\boxed{u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt}}$$

e(t) - error

Kp - coefficient of deviation (Proportional term)

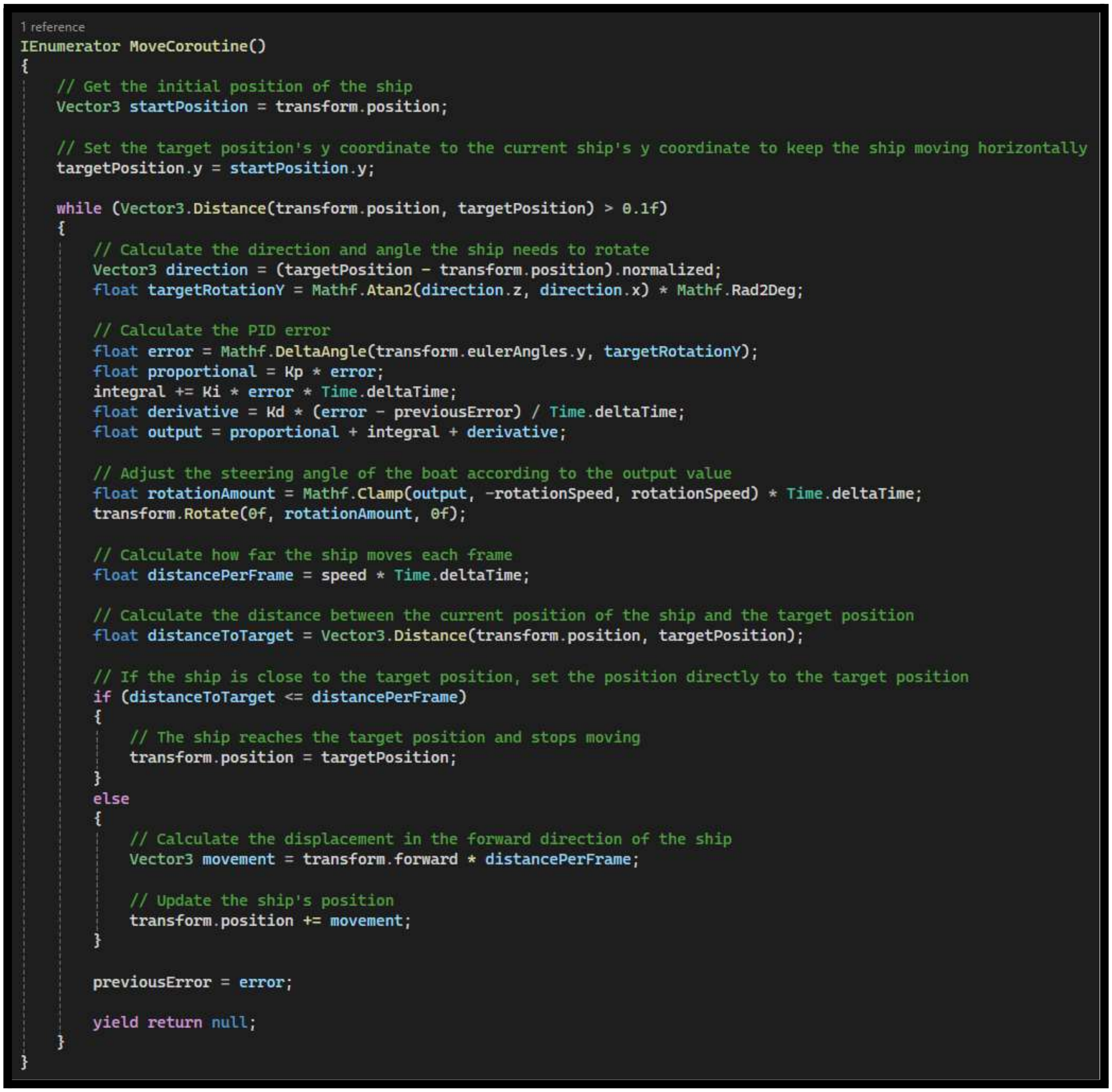
Ki - coefficient of deviation (Integral term)

Below is the implementation of the PID controller technology, which I implemented. Note that my implementation of the PID controller technology differs from Mr. Siqi's version is that when a certain location is right-clicked on the ocean, the ship moves and rotates simultaneously. This is in contrast to the previous version, wherein the ship had to rotate completely first and then move in the direction specified.

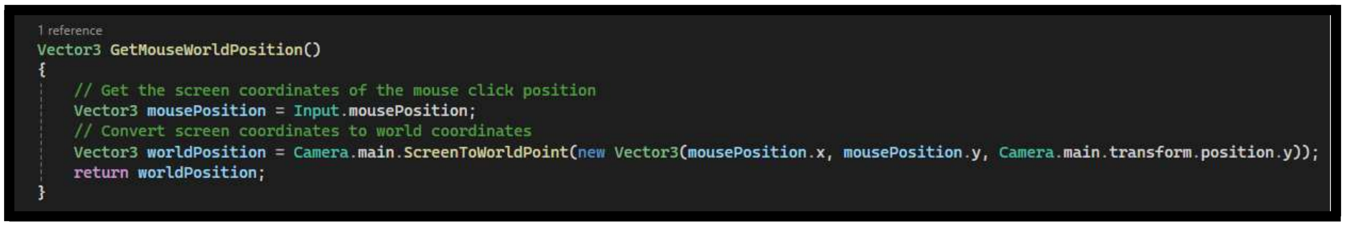


```
public class ShipMovement : MonoBehaviour
{
    public float speed = 10f; // Ship's movement speed
    public float rotationSpeed = 5f; // Ship's rotation speed

    private Vector3 targetPosition; // Ship's target position

    // PID Controller parameters
    public float Kp = 1f; // Proportional gain
    public float Ki = 0f; // Integral gain
    public float Kd = 0f; // Derivative gain

    private float integral = 0f; // Integral term
    private float previousError = 0f; // Previous error

    Unity Message | 0 references
    void Update()
    {
        // Detect mouse right-click event
        if (Input.GetMouseButtonDown(1))
        {
            // Get mouse right-click position
            targetPosition = GetMouseWorldPosition();

            // Move ship
            MoveShip();
        }
    }

    1 reference
    public void SetTargetPosition(Vector3 position)
    {
        targetPosition = position;
        // Start moving the ship
        MoveShip();
    }

    2 references
    void MoveShip()
    {
        // In each frame update, move the ship until it reaches the target position
        StartCoroutine(MoveCoroutine());
    }
```

Fig 6. Setting up placeholder values for the Proportional, Integral, and Derivative gain



```
1 reference
IEnumerator MoveCoroutine()
{
    // Get the initial position of the ship
    Vector3 startPosition = transform.position;

    // Set the target position's y coordinate to the current ship's y coordinate to keep the ship moving horizontally
    targetPosition.y = startPosition.y;

    while (Vector3.Distance(transform.position, targetPosition) > 0.1f)
    {
        // Calculate the direction and angle the ship needs to rotate
        Vector3 direction = (targetPosition - transform.position).normalized;
        float targetRotationY = Mathf.Atan2(direction.z, direction.x) * Mathf.Rad2Deg;

        // Calculate the PID error
        float error = Mathf.DeltaAngle(transform.eulerAngles.y, targetRotationY);
        float proportional = Kp * error;
        integral += Ki * error * Time.deltaTime;
        float derivative = Kd * (error - previousError) / Time.deltaTime;
        float output = proportional + integral + derivative;

        // Adjust the steering angle of the boat according to the output value
        float rotationAmount = Mathf.Clamp(output, -rotationSpeed, rotationSpeed) * Time.deltaTime;
        transform.Rotate(0f, rotationAmount, 0f);

        // Calculate how far the ship moves each frame
        float distancePerFrame = speed * Time.deltaTime;

        // Calculate the distance between the current position of the ship and the target position
        float distanceToTarget = Vector3.Distance(transform.position, targetPosition);

        // If the ship is close to the target position, set the position directly to the target position
        if (distanceToTarget <= distancePerFrame)
        {
            // The ship reaches the target position and stops moving
            transform.position = targetPosition;
        }
        else
        {
            // Calculate the displacement in the forward direction of the ship
            Vector3 movement = transform.forward * distancePerFrame;

            // Update the ship's position
            transform.position += movement;
        }

        previousError = error;

        yield return null;
    }
}
```



```
1 reference
Vector3 GetMouseWorldPosition()
{
    // Get the screen coordinates of the mouse click position
    Vector3 mousePosition = Input.mousePosition;
    // Convert screen coordinates to world coordinates
    Vector3 worldPosition = Camera.main.ScreenToWorldPoint(new Vector3(mousePosition.x, mousePosition.y, Camera.main.transform.position.y));
    return worldPosition;
}
```

Fig 7 & 8. Applying the formulas into code (top), then updating the position of the Ship (below)

4. Vessel Movement with Dynamic Positioning Technology

For many offshore operations it is necessary to keep a vessel at a fixed position, such as when drilling or carrying out refueling. Traditionally this has been done using an anchor. Nowadays, Dynamic Positioning (DP) systems are replacing anchors, as they allow the boat to be tethered to one place, without having the risk of being moved due to any external disturbance from wind or water waves [3].

A Dynamic Positioning system is able to control the position and heading of a vessel by using thrusters that are constantly active and automatically balance the environmental forces (wind, waves, current etc.). Environmental forces tend to move the vessel off the desired position, while the automatically controlled thrust balances those forces and keeps the vessel in position.

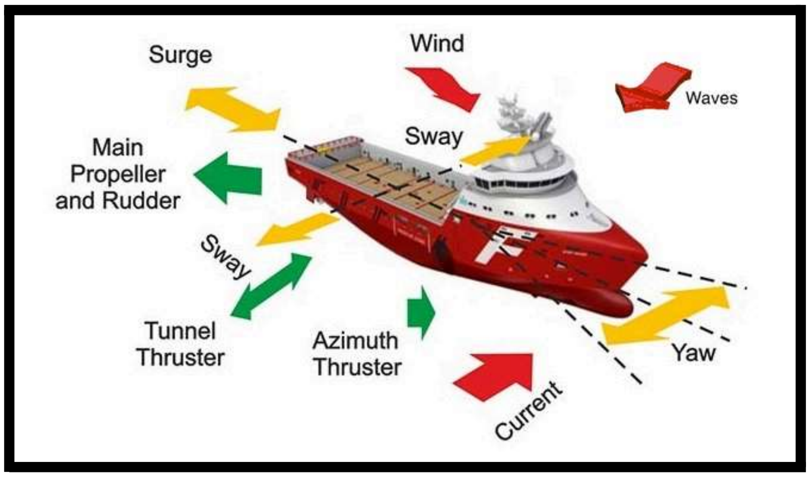
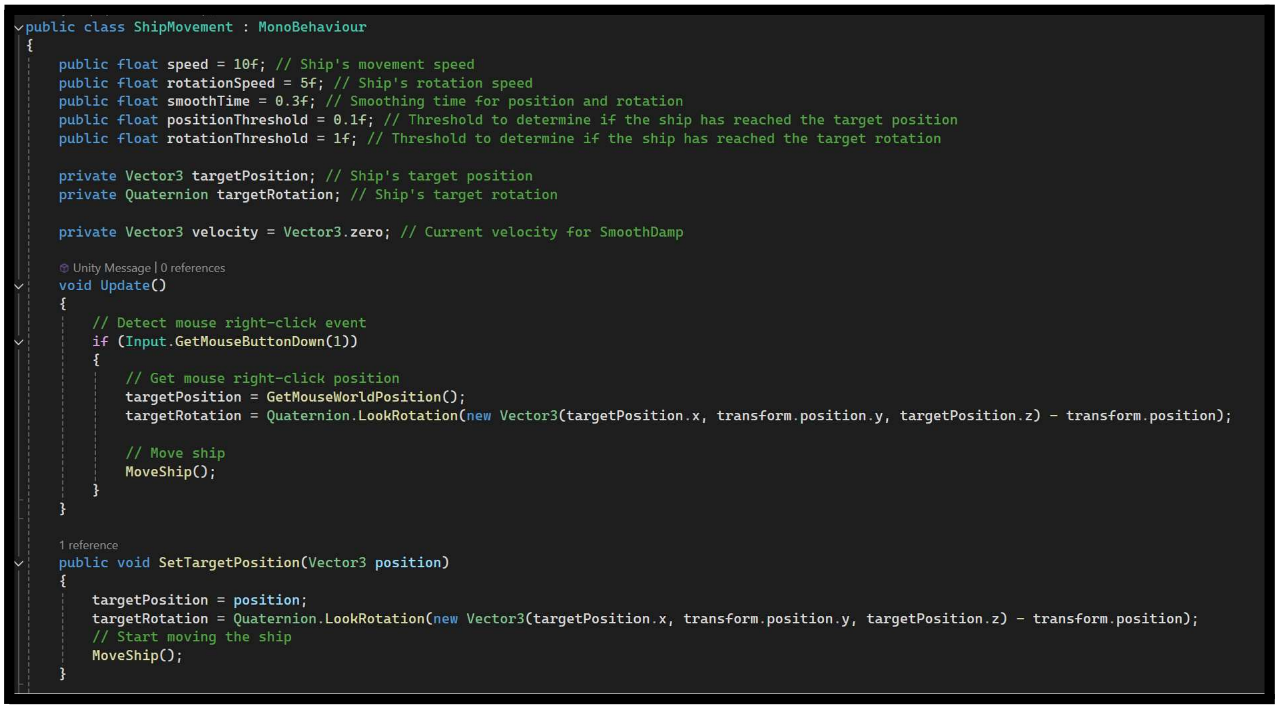


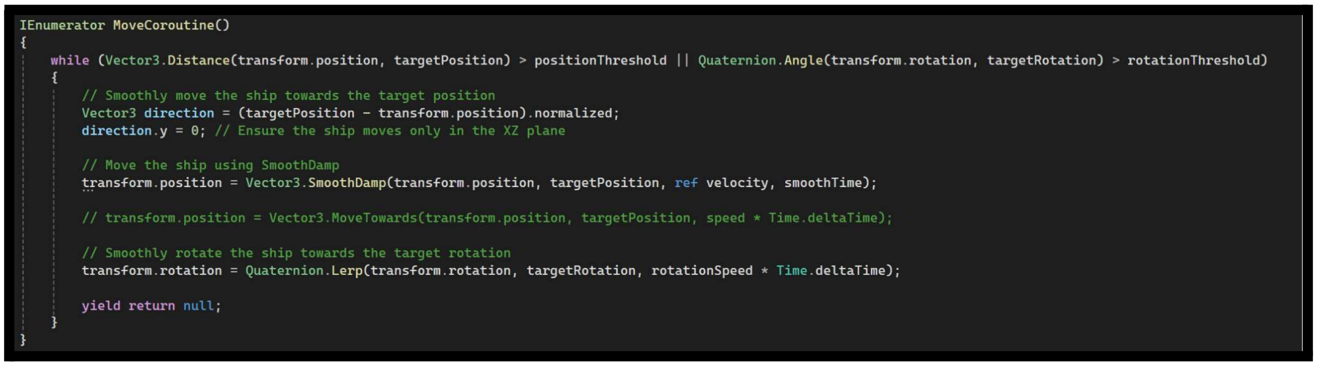
Figure 9: DP systems use a set of thrusters to automatically balance external forces such as wind, wave, current etc [3].

When the vessel moves off the intended position the DP computer will calculate the required thrust which will then be applied by the thrusters in order to maintain the position of the vessel.



```
public class ShipMovement : MonoBehaviour
{
    public float speed = 10f; // Ship's movement speed
    public float rotationSpeed = 5f; // Ship's rotation speed
    public float smoothTime = 0.3f; // Smoothing time for position and rotation
    public float positionThreshold = 0.1f; // Threshold to determine if the ship has reached the target position
    public float rotationThreshold = 1f; // Threshold to determine if the ship has reached the target rotation

    private Vector3 targetPosition; // Ship's target position
    private Quaternion targetRotation; // Ship's target rotation

    private Vector3 velocity = Vector3.zero; // Current velocity for SmoothDamp

    Unity Message | 0 references
    void Update()
    {
        // Detect mouse right-click event
        if (Input.GetMouseButtonDown(1))
        {
            // Get mouse right-click position
            targetPosition = GetMouseWorldPosition();
            targetRotation = Quaternion.LookRotation(new Vector3(targetPosition.x, transform.position.y, targetPosition.z) - transform.position);

            // Move ship
            MoveShip();
        }
    }

    1 reference
    public void SetTargetPosition(Vector3 position)
    {
        targetPosition = position;
        targetRotation = Quaternion.LookRotation(new Vector3(targetPosition.x, transform.position.y, targetPosition.z) - transform.position);
        // Start moving the ship
        MoveShip();
    }
```

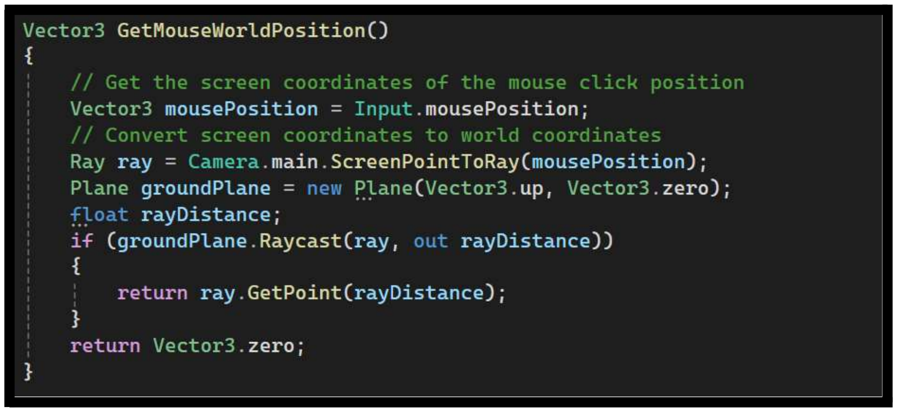
Figure 10: In the above figure, the Quaternion describes the moving vector with the x, y, and z coordinates with the change in angle involved.



```
IEnumerator MoveCoroutine()
{
    while (Vector3.Distance(transform.position, targetPosition) > positionThreshold || Quaternion.Angle(transform.rotation, targetRotation) > rotationThreshold)
    {
        // Smoothly move the ship towards the target position
        Vector3 direction = (targetPosition - transform.position).normalized;
        direction.y = 0; // Ensure the ship moves only in the XZ plane

        // Move the ship using SmoothDamp
        transform.position = Vector3.SmoothDamp(transform.position, targetPosition, ref velocity, smoothTime);

        // transform.position = Vector3.MoveTowards(transform.position, targetPosition, speed * Time.deltaTime);

        // Smoothly rotate the ship towards the target rotation
        transform.rotation = Quaternion.Lerp(transform.rotation, targetRotation, rotationSpeed * Time.deltaTime);

        yield return null;
    }
}
```

Figure 11. The .Lerp and .SmoothDamp functions allow smooth movement of the Ship respectively, which also decreases overall instability.

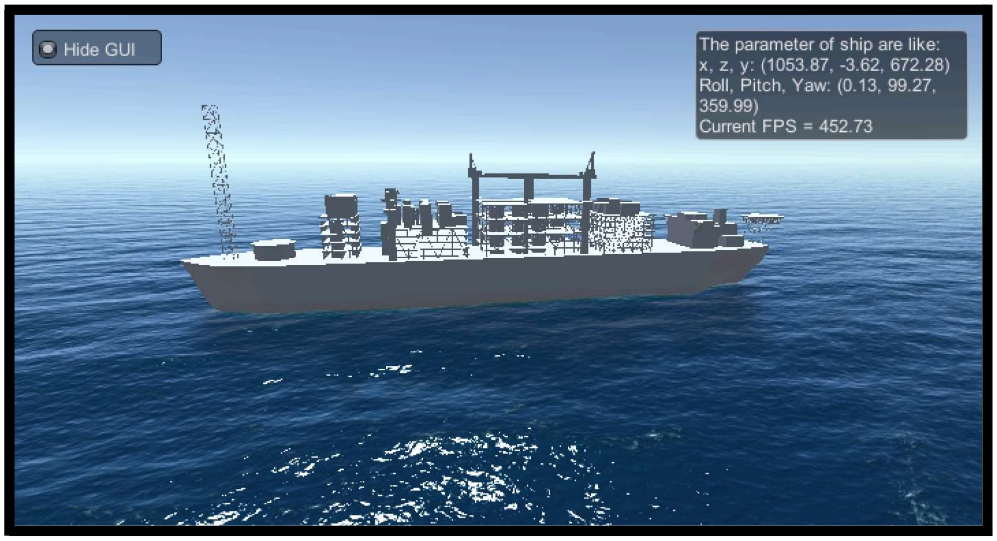
In order to maintain stability of the Ship during Dynamic Positioning, the y-coordinate value of the Ship was set to 0 w.r.t X-Z plane, so that any FLNG processes can proceed smoothly.



```
Vector3 GetMouseWorldPosition()
{
    // Get the screen coordinates of the mouse click position
    Vector3 mousePosition = Input.mousePosition;
    // Convert screen coordinates to world coordinates
    Ray ray = Camera.main.ScreenPointToRay(mousePosition);
    Plane groundPlane = new Plane(Vector3.up, Vector3.zero);
    float rayDistance;
    if (groundPlane.Raycast(ray, out rayDistance))
    {
        return ray.GetPoint(rayDistance);
    }
    return Vector3.zero;
}
```

Figure 12. Moving the ship to the final conditions warranted by the

When any position on the sea is clicked, the ship is automatically smoothly transported to that position, while maintaining stability of the assets on board.



The parameter of ship are like:
x, z, y: (1053.87, -3.62, 672.28)
Roll, Pitch, Yaw: (0.13, 99.27, 359.99)
Current FPS = 452.73

Figure 13. Notice that the Ship maintains its composure despite the wave turbulences that affect the ship.

5. Conclusion

In conclusion, the exploration of digital twin technology through the development of a Unity-based simulation has provided valuable insights into the complexities of maritime operations under varying environmental conditions. By creating the "Kitchen Chaos Delivery Manager" game, I gained foundational skills in Unity and C#, which served as a practical platform to understand game mechanics and programming logic. Furthermore, the integration of PID controller technology and dynamic positioning systems highlighted the importance of robust control mechanisms in maintaining operational stability for offshore assets.

The theoretical understanding of PID controllers, focusing on their proportional, integral, and derivative components, complemented the hands-on experience of implementing these concepts in a dynamic environment. Additionally, the exploration of dynamic positioning systems revealed their critical role in modern maritime operations, allowing vessels to maintain precise positioning without traditional anchoring methods.

Overall, this project not only reinforced my technical skills but also emphasized the significance of integrating advanced control systems in the maritime industry. As I continue to develop my expertise, I am motivated to innovate further and contribute to the evolving landscape of maritime technology, enhancing both safety and efficiency in offshore operations.

6. References

1. Unity Codemonkey. "Kitchen Chaos Course." Unity Codemonkey, https://unitycodemonkey.com/kitchenchaoscourse.php .

2. PLCynergy. "PID Controller." PLCynergy, https://plcynergy.com/pid-controller/ .

3. Offshore Engineering. "What is Dynamic Positioning?" Offshore Engineering, https://www.offshoreengineering.com/dp-dynamic-positioning/what-is-dynamic-positioning/ .