Floating high rise in Rotterdam's City Waters: A Spatial and Structural Feasibility Study

CTB3000: Bachelor Thesis

Arend Jan Jongen



Delft University of Tech

TU Delft

Floating high rise in Rotterdam's City Waters: A Spatial and Structural Feasibility Study

by

Arend Jan Jongen

5478499

Jongen

Supervisor:	P.A. Korswagen Eguren
Supervisor:	P.C.J. Hoogenboom
Project Duration:	April, 2025 - June, 2025
Faculty:	Faculty of civil engineering and geosciences, Delft

Cover:	Luchtfoto van Rotterdam met zicht op De Zalmhaven by Gerhard van Roon/Kunst en Vliegwerk RP (Modified)
Style:	TU Delft Report Style, with modifications by Daan Zwaneveld

TU Delft

Preface

This thesis marks the conclusion of my Bachelor of Science in Civil Engineering at Delft University of Technology. The project is centered on finding a feasible solution to the housing shortage in the Netherlands by incorporating floating high-rise buildings into the housing stock.

I've been interested in both building engineering and offshore engineering since the start of the bachelor, and I was looking for a topic where these two fields come together. The idea of using the inner waters of cities as potential building ground fascinates me, especially in the context of sustainable urban growth. This research also ties into another interest of mine: tackling complex problems in urban environments, where many different factors need to be considered for a solution to actually work.

This project challenged me to combine structural thinking with a more design oriented approach. It also gave me the chance to apply a lot of the theory I've learned during the bachelor to one specific, real world problem, and to bring all those pieces together into a single product.

I would like to sincerely thank my supervisors, Paul Korswagen Eguren and Pierre Hoogenboom, for their guidance and feedback throughout the project. Their support was very helpful in shaping this thesis.

I hope this research contributes, even if just a little, to the ongoing search for innovative and climate adaptive housing solutions and maybe encourages others to explore less conventional paths within civil engineering.

Arend Jan Jongen
Delft, June 2025

Summary

This thesis investigates the feasibility of large-scale floating housing in Rotterdam, focusing on modular buildings of up to twelve storeys. As urban space becomes increasingly scarce and housing demand continues to rise, floating structures offer a promising solution for high-density living in harbor zones such as Maashaven. Using a parametric modelling approach, the study evaluates the structural performance, hydrostatic stability, and environmental response of various platform and building configurations under realistic wind and wave conditions.

The results show that terraced platforms perform best, allowing tall structures to remain within stability and comfort limits. Wind-induced heeling is the dominant limiting factor, while wave moments remain moderate due to the sheltered location. The dynamic behavior remains inconclusive, as only natural frequencies were analyzed. None of the buildings have all their modes outside the critical motion sickness range, but since accelerations were not quantified, no firm conclusions can be drawn on this aspect. A combination of ribbed steel frames and concrete floating platforms is shown to be structurally viable.

The study also highlights key uncertainties. Motion comfort has not been fully assessed, and the legal classification of the structure—whether it qualifies as a building or a vessel—remains unresolved, significantly affecting the applicable design standards. Despite these limitations, the thesis concludes that floating high-rises are a realistic addition to Rotterdam's housing strategy, provided that further work is done on dynamic comfort and regulatory integration.

Contents

Preface i

Summary ii

1 Introduction 1
- 1.1 Research questions . . . 2
- 1.2 Methodology . . . 2
- 1.3 Structure of report . . . 2

2 Area 3
- 2.1 Rijnhaven & Maashaven . . . 3
- 2.2 Waalhaven . . . 4
- 2.3 Eemhaven . . . 4
- 2.4 Maas . . . 5
- 2.5 Conclusion . . . 5

3 Frame 7
- 3.1 General design principles . . . 7
- 3.2 Terraced building . . . 8
- 3.3 Building Parameters . . . 8

4 Platform 10
- 4.1 Buoyancy . . . 10
- 4.2 Design Principles . . . 11

5 System 12
- 5.1 Rib and Portal Alignment . . . 12
- 5.2 Equilibrium . . . 13
- 5.3 Metacenter and Metacentric Height . . . 13
- 5.4 Data . . . 14

6 Stability 15
- 6.1 Load Combinations . . . 15
  - 6.1.1 Wind Load . . . 15
  - 6.1.2 Wave Load . . . 16
  - 6.1.3 Combined Effects . . . 16
  - 6.1.4 Moment and Heel Angle . . . 17
- 6.2 Natural Frequency . . . 18
  - 6.2.1 Heave: reference SDOF system . . . 18
  - 6.2.2 Mass distribution for roll and pitch . . . 19
  - 6.2.3 Iterations . . . 21

7 Discussion 22
- 7.1 Interpretations . . . 22
- 7.2 Implications . . . 22
- 7.3 Limitations and Recommendations . . . 23
  - 7.3.1 Structural refinement and heel-induced moment shifts . . . 23
  - 7.3.2 Platform utilisation and buoyancy strategy . . . 23
  - 7.3.3 Building geometry and optimisation . . . 24
  - 7.3.4 Vibration comfort and RMS accelerations . . . 24
  - 7.3.5 Freeboard and overtopping risk . . . 25
  - 7.3.6 Inner harbour wave modelling . . . 25

7.3.7 Financial feasibility 25

8 Conclusion **26**
8.1 Area *Research question 1* 26
8.2 Frames *Research question 2* 26
8.3 Platform sizing and configuration *Research question 3* 26
8.4 Stability performance *Research question 4* 27

References **28**

A Background **30**
A.1 The Past and Present 30
A.2 Very large floating structures 31
A.3 Legislation 31
A.4 Serviceability Limit 31

B Added Mass **33**

C IPE profiles **35**

D frame code square **36**

E Frame code for terrace **39**

F Platform code **44**

G System evaluator **47**

H Moment and heel angle code **56**

I Natural frequencies code **59**

J Code verification **63**

1

Introduction

All over the world, urbanization is on the rise. Since 2007, more people are living in urban than in rural areas (Ritchie et al., 2024). However, in the Netherlands, urbanization has been high for a long time and is not the main contributor to the now existing housing crisis. Compared to 1960, the population in the Netherlands has grown by 49.2%, whereas the number of households has increased by 147.1%. This is mostly due to the explosive increase in single-person households of 560% (Netherlands, n.d.).

According to Langen (2025), the problem is not the availability of new homes but the type of housing that is being built. With the rise of more single-person households, demand is shifting toward smaller homes, yet supply has stayed almost the same in recent years. In the next 10 years, the largest municipalities and the bordering municipalities will see a population increase (Statistiek, 2022). Due to insufficiently reliable data from CBS, nothing can be said about household increase per municipality. If the assumption is made that household size will remain constant, the demand for housing will also increase the most in these municipalities.

In Rotterdam, for example, less than 10% of the 86.000 people looking for social housing received one (Keete, 2024). For a single-person household, the upper income limit is 47.000 euros, while the median income is 39.000 euros (CBS, 2023). In total, 60% of the city's population is eligible for social housing. This makes it all the more important to ensure that more social housing is being constructed in the city of Rotterdam.

However, with the increasing price and scarcity of undeveloped land in the city of Rotterdam. Realizing affordable housing in the areas most desired is getting harder and harder. Building outside of the dikes is also getting increasingly more risky with climate change (Noorderkwartier, 2025). Increasing the sea levels and more severe rainfall giving higher river discharges. For example places like the Noorderei-land already flood every few years. Causing damages to properties and nuisance to inhabitants living there.

As a result urban planners and architects alike need to explore alternative housing solution to grow the available housing stock. A possible solution to the before mentioned problems is floating homes. One type of floating construction is the pontoon type that relies on buoyancy and not foundations of any type (Wang & Tay, 2011). This type of construction offers a flexible and climate-resilient alternative that could help meet the growing demand for housing in flood-prone urban areas like Rotterdam.

The objective of this thesis is:
Determine the technical and spatial feasibility of large buoyancy based floating platforms for apartment buildings in Rotterdam's inner city waters and harbors.

1.1. Research questions

The before stated objective gives us the following main research question:
What are the spatial and structural requirements for realizing large buoyant floating apartment buildings in Rotterdam's inner city waters and harbors to expand the city's housing capacity?
To answer the main research question the question in sub-divided into smaller questions being:

1. What are the spatial boundaries, including the available depth, width, and length of Rotterdam's inner city waters and harbors, and what height limitations apply to ensure large floating apartment buildings fit into the existing urban fabric?
2. What structural frame configurations are suitable for apartment buildings on buoyant floating platforms, considering modularity, load transfer?
3. What platform dimensions and configurations can accommodate the constraints identified in Sub-question 1 and support the selected building configurations from Sub-question 2?
4. Do the resulting floating configurations meet the hydrostatic and dynamic stability performance criteria as identified in the literature, under both regular and storm conditions relevant to Rotterdam's harbors?

1.2. Methodology

To answer the main research question, four sub-questions are addressed through a structured approach, each with its own tailored method.

The first sub-question concerns the spatial boundaries of Rotterdam's inner city waters and harbors. This is addressed through a geographic scan using Google Maps and Google Earth to identify suitable locations and measure local dimensions. Water depth data is obtained from bathymetric maps, while height restrictions are estimated by comparing the built environment in the immediate surroundings. For each location, width, depth, length, and estimated allowable building height are determined and visualized.

The second sub-question focuses on the structural configuration of apartment buildings on floating platforms. This is investigated through parametric design using Python, where two steel building configurations are evaluated. Structural weight is estimated using rule of thumb values to ensure conservative load assumptions. Floor systems, façades, and roof contributions are also included to obtain a complete and realistic mass estimate.

The third sub-question examines what platform dimensions and configurations can accommodate both spatial constraints and the selected building types. This is based on buoyancy principles, using Archimedes' law to determine the minimum submerged volume required for each configuration. The pontoons follow modular rule of thumb designs.

The fourth and final sub-question assesses whether the selected floating configurations meet hydrostatic and dynamic stability requirements. First, hydrostatic stability is analyzed using the metacentric height (GM), based on the location of the center of buoyancy and center of gravity. Next, dynamic stability is evaluated by calculating external moments and resulting heel angles due to combined wind and wave loads. These are compared against maximum allowable values found in literature. Lastly, natural frequency calculations are performed to determine whether the structure's response avoids resonance ranges known to cause discomfort for occupants.

1.3. Structure of report

The structure of the report is as follows. Chapter 2 investigates the available locations in Rotterdam's inner city waters and harbors, providing spatial constraints that are essential for the platform design. Chapter 3 introduces two apartment building configurations, along with their design rules and structural parameters. Chapter 4 addresses pontoon design, using spatial data from Chapter 2 to define dimensions. This chapter also explains the principles of buoyancy and modular platform construction. Chapter 5 combines the building and platform into a single floating system and performs an initial hydrostatic stability assessment using the metacentric height (GM). Chapter 6 then assesses dynamic stability by evaluating environmental moments, resulting heel angles, and the system's natural frequency. The report concludes with a discussion of the findings and the final conclusions.

2

Area

To determine the feasible width, length, and height of both the platform and the building, a detailed analysis of potential locations for floating semi-high-rise structures will be carried out. This study focuses on the areas identified by *Uitvoeringsagenda 1.0* (2025) as promising for synergy between the city and the harbor. The selected areas include the Rijnhaven, Maashaven, Waalhaven, Eemhaven, and the Maas River. These locations have been chosen due to their close proximity to Rotterdam's highly valued city center.

2.1. Rijnhaven & Maashaven

Figure 2.1 shows the water depths in both the Rijnhaven and the Maashaven. The black area in the Maashaven indicates the site of the Nelson Mandela Park, which is currently under construction ("Award winning Nelson Mandela Park masterplan", 2022). In the Rijnhaven, the depths are significantly lower due to its inactive use as a harbor. Since 2021, a three-story floating office has been located here (Rijnmond, 2021). Access to the harbor is restricted by a bridge, limiting the size of incoming platforms to a maximum width of 25 meters. The north side of the Rijnhaven features the tallest buildings among all the harbors studied, with the New Orleans tower reaching 45 stories (Bouwwereld, 2010). On the opposite side, most buildings are approximately 10 stories tall. On the Maashaven side closest to the Rijnhaven, buildings are slightly lower, generally ranging between 6 and 8 stories. On the other side of the Maashaven stands the Maassilo, an event venue slightly taller than 10 stories. The normative depth of the Maashaven is -6 meters. The maximum width of platforms is 120 meters, determined by the width of the harbor entrance to the Maas. However, to enable new use of the Maashaven, the existing moorings for inland shipping would need to be relocated. One option is to move them closer to Maasvlakte 2, which would improve the integration between inland and maritime transport.

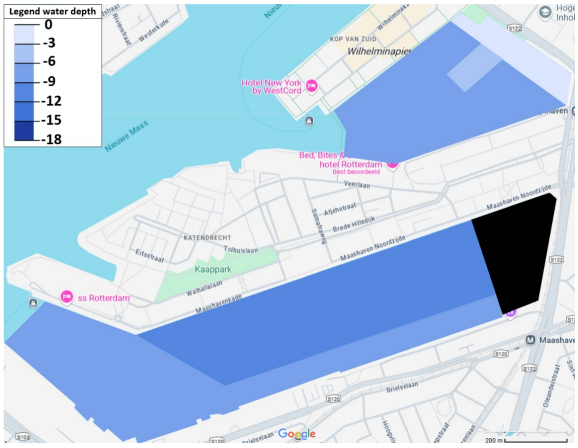


Figure 2.1: Overview of Rijnhaven and Maashaven with depth

2.2. Waalhaven

In the Waalhaven, around 75% of the addresses in the surrounding neighborhood are registered as having an industrial function (“Buurt Eemhaven (gemeente Rotterdam) in cijfers en grafieken”, 2016). Some of these businesses are space intensive and may be difficult to relocate, as can be observed via satellite imagery on Google Maps. These larger companies are primarily located on the western side of the harbor. In contrast, the eastern side is home to smaller logistics firms, which are more easily relocated. On the western side, the number of stories cannot be meaningfully compared to the rest of the urban area, as there is hardly any residential or mixed-use development. However, on the eastern side, the nearby neighborhood of Oud-Charlois has an average building height of approximately four stories. The narrowest point in the harbor is 120 meters, and the normative depth is -6 meters, as most branches of the harbor range between -6 and -9 meters in depth. As seen in figure 2.2 only the southern part of the harbor falls short of these depth

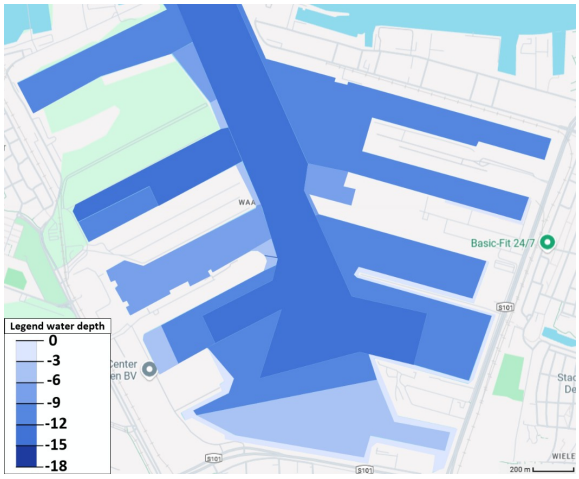


Figure 2.2: Overview of Waalhaven with depth

2.3. Eemhaven

According to (“Buurt Waalhaven (gemeente Rotterdam) in cijfers en grafieken”, 2016), around 66% of the addresses in the Eemhaven neighborhood are registered as having an industrial function. Most of the companies located here are space intensive and rely heavily on water transport by container ships for their operations. The number of stories is not particularly relevant, as the area is not currently considered an urban environment. To truly integrate it into Rotterdam, the entire site would need to be overhauled. As visible in figure 2.3 , the top branch of the harbor is relatively shallow and narrow, making it different from the rest of the harbor. In the main sections, the narrowest point measures approximately 130 meters, and the normative depth is -9 meters, although some of the side cutouts are shallower.



Figure 2.3: Overview of Eemhaven with depth

2.4. Maas

In 2024, the Port of Rotterdam received 91,356 inland vessel visits (Port of Rotterdam, n.d.). A significant portion of these vessels pass through the city center, making this section of the Maas a particularly busy waterway. To accommodate any building of significant size in this area, platforms would need to be narrow and elongated to allow vessel traffic to continue uninterrupted. Due to the high frequency of ship movements, water disturbance is also at its peak here, which would result in greater motion of any floating structures. Additionally, development in the Koningshaven is not feasible, as it is the only location where the Erasmus Bridge can open providing the sole exit route for tall ships such as sailing vessels. Building height on the north side is only limited by the platform's structural capacity, as many surrounding buildings exceed 15 stories allowing new developments to blend well into the urban fabric. On the Noordereiland side, however, heights should not exceed 6 stories to remain consistent with the existing building scale on the island.

2.5. Conclusion

As shown in the histogram in Figure 2.4, depths between −9 and −12 meters are the most common. However, these areas are typically located further from the quay and are therefore not immediately suitable for floating high-rise developments. Utilizing these deeper areas would require expanding into shallower zones closer to the quay.

These shallow zones, however, pose challenges due to the steep underwater incline near the quay wall, which limits their usability. Slightly further from the quay, water depths between −3 and −9 meters are more favorable for platform installation. Once such platforms are established, extending towards depths of −9 to −12 meters becomes feasible as well.

Additionally, building height should correspond to the urban context: taller structures are better suited near the city center, while lower-rise developments are more appropriate further out towards the port area. Finally, the platform width should not exceed 120 meters if the municipality intends to move the platforms with the structures on top. This is because the entrances of the harbors are approximately this width.

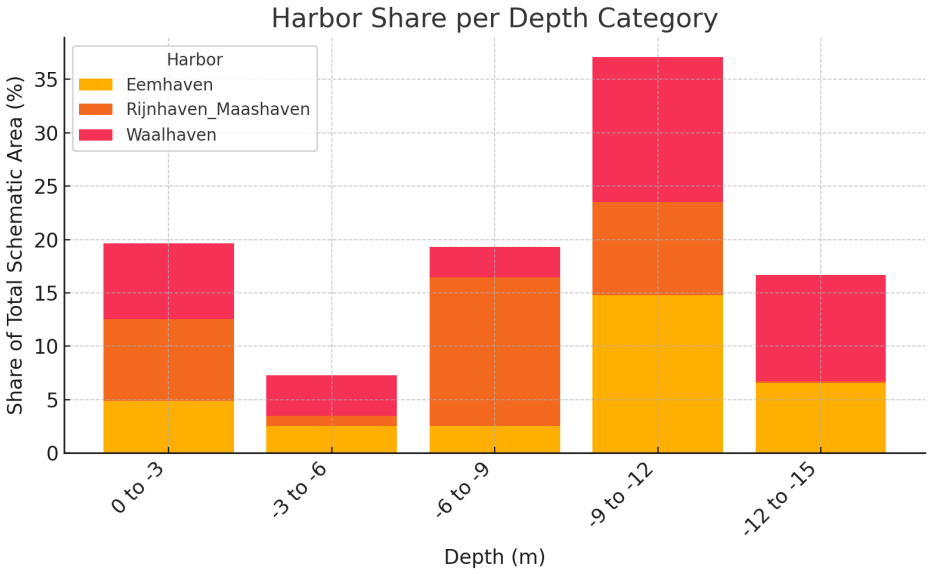


Figure 2.4: Overview of percentage depth over all the harbors

3

Frame

In the following chapter, the structural frame of the building will be examined. This thesis considers two distinct buildings: a rectangular building and a terraced building. The chapter begins with an overview of the general design principles applicable to both types. Subsequently, the chapter concludes with a discussion of the features that are unique to the terraced style building. Finally, the key design parameters common to both configurations are presented.

3.1. General design principles

The primary objective of this study is not to develop an optimized structural frame, but rather to establish a conservative baseline model. Therefore, simplified design principles based on engineering rules of thumb are applied. These rules of thumbs are derived from the Quick Reference (F.A.M. Soons et al., n.d.). In cases where the reference provides a range, the upper bound is consistently selected to ensure that the resulting structural model is conservative, thereby producing a relatively higher total frame weight.

Steel is chosen as the material for the structural frame due to its favorable properties for achieving lightweight and slender constructions. As illustrated in figure 3.1, the structural system is defined by three principal dimensions: length, width, and height. The spans refer to the horizontal elements connecting columns within a portal frame, while the studs connect adjacent portals. For both spans and studs, the following rule of thumb is applied to determine the required profile height:

$$h = \frac{l}{24} \tag{3.1}$$

where h is the required profile height in millimeters, and l is the span length in millimeters.

The calculated height is then rounded up to the next available standard IPE profile from the tabulated values in Eurocode (n.d.) which can also be found in C. For the vertical columns, a more conservative sizing rule is applied, using:

$$h = \frac{l}{10} \tag{3.2}$$

where h is the required profile height in millimeters, and l is the span length in millimeters.

This stricter approach for column sizing ensures greater stiffness and load-carrying capacity in the vertical direction, aligning with conservative design assumptions. The number of studs, spans, and columns is determined through optimization. Longer spans require heavier profiles, while shorter spans allow lighter sections but increase the number of columns and studs. Since studs are placed between each pair of adjacent portal columns, reducing span length leads to more structural elements. The optimization balances this trade-off to minimize total frame weight.

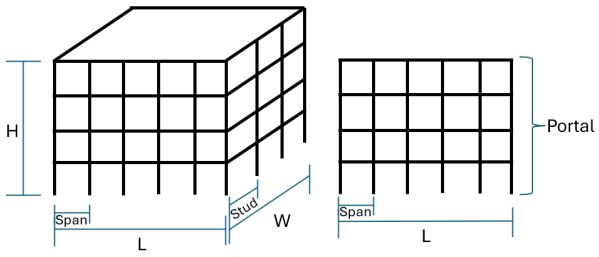


Figure 3.1: View front and 3D view rectangular showing used naming

3.2. Terraced building

The terraced building applies the same design rules as the square configuration. The key distinction lies in the introduction of a *terrace offset*, as illustrated in Figure 3.2, which is varied between 1.0 and 2.0 metres in 0.5 metre increments.

Compared to the square design, the terraced form offers a lower center of gravity, which contributes positively to the system's overall stability, as discussed further in chapter 5.

An optimization is also performed for the number of columns, spans, and studs. However, several additional constraints apply:

- The topmost usable floor span must exceed 8 metres to ensure meaningful usable floor area.
- Only the outermost columns may shift, promoting efficient force distribution.
- When these columns overlap with or move beyond the next inner row, they are removed to reduce structural weight with height.

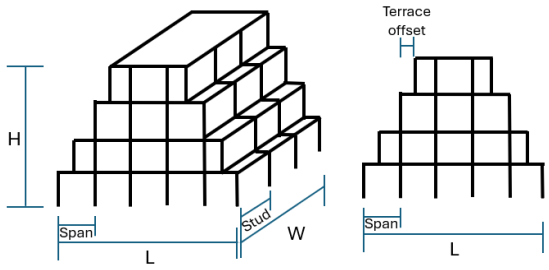


Figure 3.2: View front and 3D view terraced showing used naming

3.3. Building Parameters

The primary building parameters are the total length, width, and height. Both length and width are varied between 15 and 35 metres in increments of 1 meter, allowing for maximum configurational flexibility. The height per story is fixed at 3 metres to comply with the minimum requirements of the Dutch Building Decree, even after accounting for the space needed for installations.

The minimum and maximum number of stories are defined in chapter 2, ranging from 6 to 12 floors. The number of portals is not optimized for structural weight but rather selected to ensure alignment with the platform's ribs, as discussed further in chapter 5. These parameters give for the rectangular building 3087 options and for the terraced 9261 options.

Naturally, the frame is not the sole contributor to the building's total weight. Additional elements such as the facade, floors, and roof are included in the load assessment. For the floors, a live load of 2.5 kN/m^2 and a dead load of 4.35 kN/m^2 are applied, based on values from the F.A.M. Soons et al. (n.d.). The roof is assigned the same dead load, but a slightly lower live load of 1.5 kN/m^2. The facade is modeled with a dead load of 1.47 kN/m^2.

For estimating the center of gravity used in later stability calculations, only the dead and live loads of the floors are considered. This simplification is justified by the fact that the structural frame itself

constitutes approximately 2% of the total building weight, resulting in only a minor deviation from the true center of gravity while significantly reducing computational complexity.

Figure 3.3 shows the total gravitational forces for both the square (left) and terraced (right) building configurations. These forces are evaluated over a range of building heights, lengths, and widths to capture their influence on the total load.

In the plot for the terraced configuration, it is evident that for certain combinations of building dimensions, certain dimension sets become infeasible. This is due to the applied terrace offset, which in this case is set to 1 meter. As a result, shorter buildings no longer satisfy the requirement that the roof must have a minimum usable length of 8 metres. In such cases, no data surface is plotted, indicating that no valid structural configuration could be found within the design constraints.

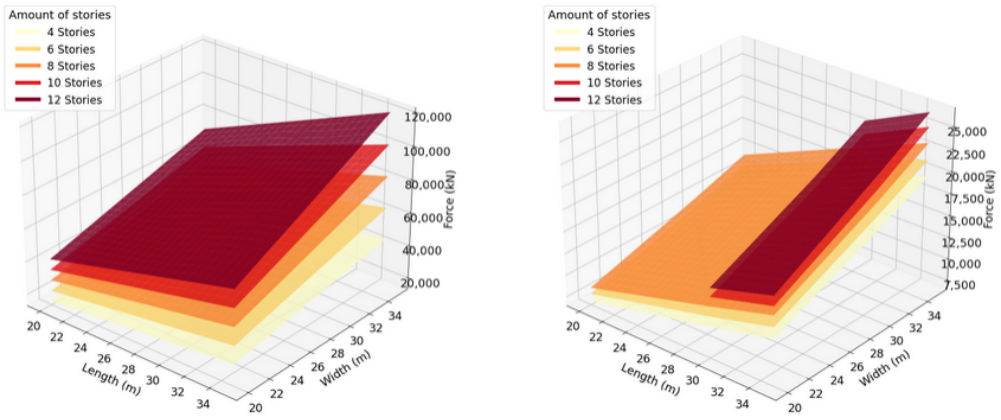


Figure 3.3: Force increase for the square (left) and terraced (right) building

4

Platform

In the following chapter, the floating platform responsible for generating the buoyant force that supports the high-rise structure is introduced. The chapter begins with a background on the physical principle of buoyancy. This is followed by an explanation of the general design principles.

4.1. Buoyancy

Buoyancy is the upward force exerted by a fluid per Archimedes' principle on a body submerged in it. This force results from the displacement of fluid by the submerged volume of the object. A floating object is in equilibrium when the buoyant force equals the gravitational force acting on it. This condition is illustrated in Figure 4.1, where the gravitational force G acts downward through the center of gravity, and the buoyant force B acts upward through the center of buoyancy. In this research, the platform has a square cross-section. Therefore, the center of buoyancy lies at half the submerged depth, as the volume distribution is symmetric.

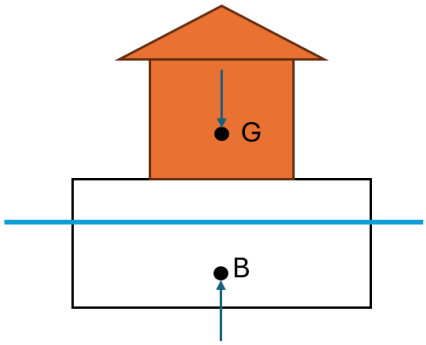


Figure 4.1: Schematization of equilibrium buoyancy and gravity force

The equilibrium condition can be expressed mathematically as:

$$m \cdot g = \rho \cdot g \cdot (L \cdot W \cdot d_{\text{submerged}}) \tag{4.1}$$

where m is the total mass of the system, g is the acceleration due to gravity, ρ is the fluid density (with a value of $1{,}000\ \mathrm{kg/m^3}$ used for water in Rotterdam), L and W are the platform's length and width, and $d_{\text{submerged}}$ is the submerged depth of the platform.

Since g appears on both sides of Equation (4.1), it cancels out, leading to a simplified expression based on mass alone. In the context of a floating high-rise, the total mass m includes both the mass of the building and that of the platform itself, as both contribute to the gravitational force that must be counteracted by buoyancy.

4.2. Design Principles

To allow for maximum flexibility in platform placement and construction, a modular system is adopted. These modules are referred to as *pontoons* throughout this chapter. The pontoons follow a set of predefined design rules to ensure structural feasibility and modular adaptability. The material selected for the pontoons is reinforced concrete, as the material density of $2{,}400\ \mathrm{kg/m^3}$ is used in all calculations.

To keep individual pontoons within realistic transport and construction limits, the length is restricted to a maximum of 20 meters. The width can vary between 4 and 8 meters, in discrete steps of 1 meter. The vertical rib thickness is fixed at 30 cm, which has been verified through hand calculations for the most extreme loading case to provide sufficient wall strength. This results in over dimensioning for lighter structures but avoids the need for recalculating wall strength for every configuration.

The slabs at the top and bottom of the pontoons are sized using a rule of thumb from the F.A.M. Soons et al. (n.d.), which relates slab thickness to the shortest horizontal span:

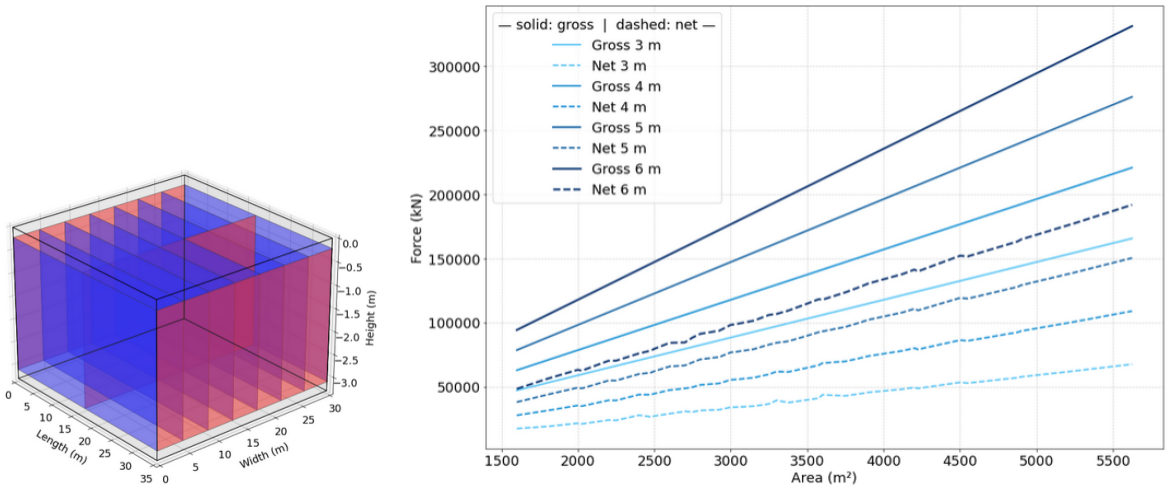
$$h = \frac{l}{24} \tag{4.2}$$

where h is the required slab thickness in millimeters, and l is the shortest horizontal span of the pontoon in millimeters. The resulting value for h is rounded up to the nearest ten millimeters to match standard construction dimensions.

The total height of a pontoon is determined by the required submerged depth, plus an additional freeboard. In this report, a freeboard of 0.3 m is adopted to minimize wave overtopping, even under storm like conditions. This contributes not only to safety, but also to the perceived comfort and security for occupants. While this value is fixed in the current study, further optimization may be possible in future work.

Figure 4.2a shows a 3D visualization of the modular platform, including its individual pontoons. In the design space, both the platform length and width are varied between 40 and 75 meters, in increments of 5 meters. The platform height is determined by the available water depth, as introduced in Chapter 2, and ranges from 3 to 6 meters. This results in a total of 256 platform configurations.

Figure 4.2b illustrates the resulting buoyancy forces. Both the gross and net buoyancy forces are presented, highlighting the increase in total upward force with platform area. These diagrams are essential for understanding the balance between gravitational and buoyant forces as the platform dimensions change.



(a) Platform 3D visualization

(b) Buoyancy and gravity forces

Figure 4.2: Visual representations of the platform geometry and corresponding force diagram

5

System

In the previous chapters, the building and platform components were introduced separately. This chapter considers their integration into a complete structural system. The chapter begins by addressing the alignment criteria between platform ribs and building portals, a necessary condition for feasible system configurations, which is also implemented in the model. The second part focuses on the concept of the metacenter and the associated metacentric height, which are essential for assessing the system's stability. Finally, the chapter presents a selection of feasible building-platform combinations that form complete floating systems.

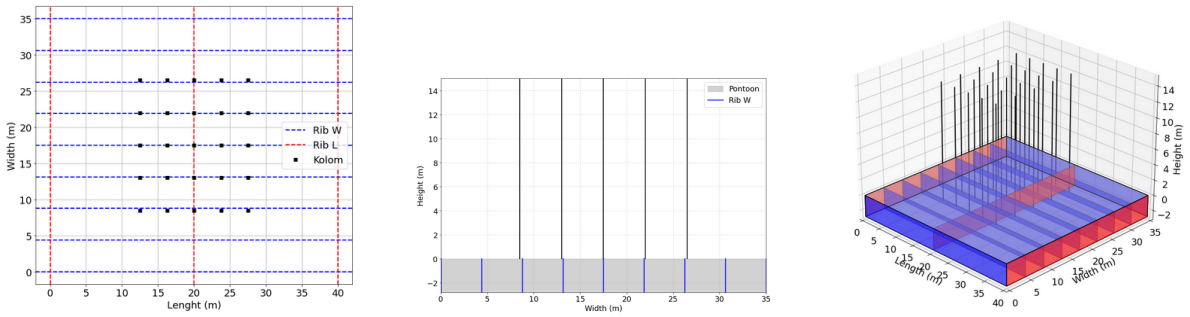
5.1. Rib and Portal Alignment

As introduced in Chapter 3, the aim is to construct a conservative baseline model. This approach also applies when integrating the structural frame with the floating platform. For efficient load transfer between the building and the platform, the optimal configuration places each building portal in direct alignment with a platform rib, ideally centered between two adjacent pontoons. This ensures that vertical loads are symmetrically distributed, with half the force transferred to each neighbouring pontoon. Such a configuration promotes efficient wall design and enables thinner top slabs on the pontoons.

To ensure sufficient opportunities for rib–portal alignment, the number of portals is varied between 3 and 8. These portals are evenly spaced along the building length, after which the building is centered on the platform.

To enforce alignment, a check is implemented in the computational model. A tolerance of 10 cm is introduced to accommodate minor misalignments. This value is justified by the observation that, for typical span lengths, the rule of thumb ($l/24$, with l in millimetres) results in profile height differences of only a few millimetres, typically around 5 mm. Given that standard IPE profiles are available in depth increments of 20 mm to 50 mm (see Appendix C), such small deviations rarely lead to a change in the selected profile. Misalignments within the 10 cm tolerance generally do not significantly impact structural sizing, and may be offset by compensating changes elsewhere in the structure. Tolerances greater than 10 cm, however, may require substantial profile adjustments, leading to an increase in total structural weight.

In Figure 5.1a, several minor misalignments are visible due to the equal spacing of portals, which does not always perfectly coincide with the rib layout. However, since these deviations fall within the defined tolerance, the system is accepted as valid by the model. After the alignment check only 16 of 256 configurations are left over because only those comply with rib spacing of 3–8 portals within the 10 cm tolerance.



(a) System 2D rib and portal alignment visualization

(b) Misalignment in platform and portals

(c) System 3D visualization

Figure 5.1: Visual representations of the geometry

5.2. Equilibrium

A floating structure is in equilibrium when both the resultant force and the resultant moment acting on the system are zero (J.M.J. Journée & W.W. Massie, 2001). In static equilibrium, this condition requires that the center of gravity (G) and the center of buoyancy (B) lie on the same vertical line. This configuration is illustrated in the left part of Figure 5.2.

However, an external force or moment can disturb this equilibrium and introduce a heeling moment. As the structure heels, the submerged volume of the platform is redistributed, causing the center of buoyancy to shift. This shift results in a misalignment between G and B, as shown in the right part of Figure 5.2, generating a restoring moment known as the *righting moment*.

If the righting moment is sufficient to counteract the heeling moment, the structure will return to its original upright position once the disturbance is removed. In this case, the system is considered *stable*. Oppositely, if the heeling moment exceeds the righting capacity of the structure, it will continue to tilt or capsize, and the system is deemed *unstable*.

5.3. Metacenter and Metacentric Height

As discussed in the previous section, the righting moment must always be larger than and opposite to the applied heeling moment for the system to remain stable. The concept of the metacenter originates from maritime engineering and refers to the point at which the vertical line through the shifted center of buoyancy intersects the original vertical line through the center of gravity. This is a key concept for evaluating stability (“Metacenter | Definition & Facts | Britannica”, n.d.).

For wall-sided floating structures, where the vertical sides remain unchanged in shape as the platform heels the geometry of the submerged volume changes symmetrically. When small angle assumptions are applied when heel angles are kept below 10°, the complex Scribanti formulation can be simplified. Under these assumptions, the distance between the metacenter and center of buoyancy BM can be calculated using:

$$BM = \frac{I}{V_{\text{submerged}}} \tag{5.1}$$

where I is the second moment of area of the waterplane (in m^4), and $V_{\text{submerged}}$ is the submerged volume of the platform (in m^3). This relation provides a practical and efficient way to assess the shift in the center of buoyancy and the system’s ability to generate a righting moment.

With the metacenter determined, the distance GM (metacentric height) can be calculated. This value represents the vertical distance between the center of gravity (G) and the metacenter (M), and is a key parameter for assessing stability. The metacentric height can be computed using:

$$GM = BM + KB - KG \tag{5.2}$$

where:

- BM is the distance between the center of buoyancy (B) and the metacenter (M)
- KB is the distance from the keel to the center of buoyancy
- KG is the distance from the keel to the center of gravity

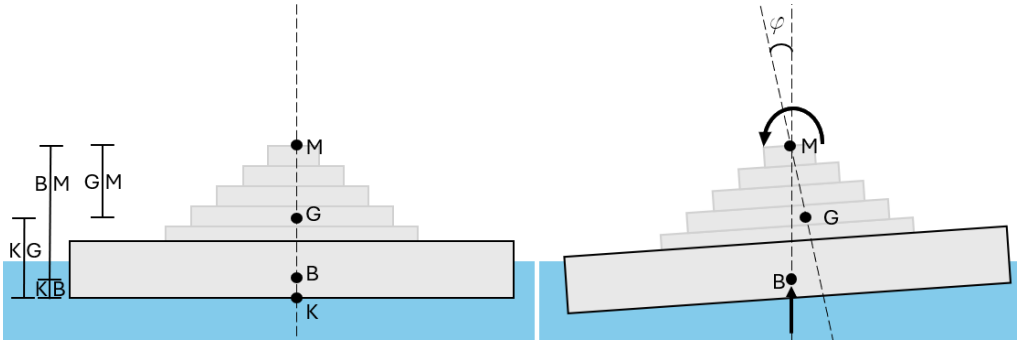


Figure 5.2: Force increase for the square (left) and terraced (right) building

5.4. Data

The following criteria must be satisfied for a system configuration to be considered valid:

- The building dimensions (length and width) may not exceed the platform dimensions.
- Rib–portal alignment is required, with a tolerance of 0.1 m to allow for small deviations.
- The total buoyant force must exceed the total gravitational force of the combined building and platform system.
- The platform must maintain a minimum freeboard of 0.3 m, as required by NTA 8111 (“NTA 8111”, 2011).
- The metacentric height (GM) must be greater than 0.25 m to ensure static stability (“NTA 8111”, 2011).
- The topmost length in terraced buildings must be at least 8 m to ensure functional space.
- Only the outermost columns may shift between floors in terraced configurations, and overlapping columns must be removed to reduce structural weight.
- The building must be centered on the platform.
- Portal spacing is evenly distributed across the building width and varied between 3 and 8 portals.

In 5.1 three of each geometry are shown to give an idea about the information obtained. In total for the square building 1015 comply with all the criteria. For the terraced 2750 comply with the criteria.

Table 5.1: Platform–building configurations and stability parameters

Platform					Building					Stability	
L (m)	W (m)	Subm. D (m)	Real Subm. D (m)	Payload (kN)	Bldg. L (m)	Bldg. W (m)	Terrace Step	# Stories	Weight (kN)	GM_x (m)	GM_y (m)
Square geometry											
35	40	3	2.96	15569	18	15	–	6	15012	33.62	23.21
45	45	6	5.99	63658	27	28	–	10	63395	9.82	9.82
55	45	6	5.89	80296	27	29	–	12	77713	6.77	20.66
Terraced geometry											
35	40	3	2.81	15569	29	18	1	12	12902	23.05	12.63
45	45	6	3.34	63658	29	18	1.5	8	10768	9.99	9.99
55	45	4	2.68	45651	28	27	2	6	13704	28.73	49.56

6

Stability

In the previous chapter, the system was evaluated under hydrostatic conditions to verify its compliance with essential design criteria. Only vertical forces were considered, and the platform–building combination was assumed to be in static equilibrium. This chapter introduces dynamic excitations, wind and wave forces,that influence the stability of the floating system.

Similar to marine structures, the combined platform and building system exhibits six degrees of freedom. These are illustrated in figure 6.1 and described in table 6.1. The translational motions occur along the principal axes, while rotational motions take place around them. Due to the simplification to a 2D model in this chapter, yaw motion (rotation about the vertical z-axis) is excluded from further analysis, as it requires 3D modeling.

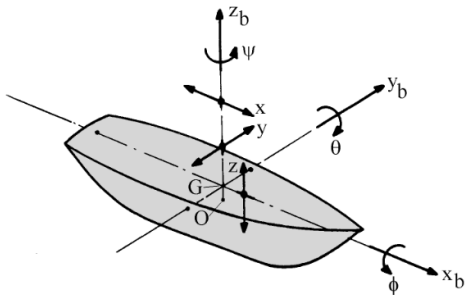


Figure 6.1: Definition of motions (J.M.J. Journée & W.W. Massie, 2001)

Axis	Translation	Rotation
x	Surge	Roll (ϕ)
y	Sway	Pitch (θ)
z	Heave	Yaw (ψ)

Table 6.1: Direction of motions

Figure 6.2: Degrees of freedom in a floating platform system

6.1. Load Combinations

As discussed in Chapter 2, only inner harbor locations are considered suitable for the floating system, with the Maas River being excluded as a viable area. Consequently, collision loads are not included in the load combinations for assessing the stability of the structure. Instead, two environmental conditions are considered: regular (everyday) and storm scenarios, focusing on wind and wave loads.

6.1.1. Wind Load

Unlike common high-rise wind models where the wind pressure is reduced near ground level—resulting in a non-uniform load distribution this study uses a uniform wind load profile across the full height of the building. This simplification is justified by the relatively modest height of the structure (18–36 m), which does not significantly exceed the footprint dimensions (18–35 m). Therefore, a constant wind load across height is considered a conservative yet reasonable approximation for the configurations

studied. To maintain a conservative approach, the wind pressure corresponding to the maximum height of 35 m is used for all configurations.

For the regular condition, a wind speed of 14 m/s results in a wind pressure of approximately 0.46 kN/m^2 (Kroftman, n.d.). In storm conditions, a wind speed of 27 m/s is assumed, yielding a pressure of 1.09 kN/m^2 (F.A.M. Soons et al., n.d.).

6.1.2. Wave Load

Wave data is derived from five years of hourly records near the Maeslantkering, obtained via the “Kust en benedenrivieren - Rijkswaterstaat Waterinfo” (n.d.) database. The significant wave height, defined as the average of the highest 1/3 of waves in a given hour, is used. For everyday conditions, this height is 0.21 m with a corresponding wave period of 4.6 s.

Due to insufficient storm data for the inner harbors of Rotterdam, a factor of 5 is applied to the regular wave height, based on findings from the Jacht- en Handelshaven Breskens case study (Rijkswaterstaat, 2014). This results in a design wave height of 1.05 m. The wavelength is taken equal to either the platform length or width, depending on the direction of loading.

The wave profile is approximated by the following expression:

$$M_{\text{wave}} = \int_0^{\lambda} \left(A \cdot \sin\left(\frac{2\pi}{\lambda} \cdot x \right) + \rho \cdot g \cdot d_{\text{submerged}} \right) \cdot \left(x - \frac{\lambda}{2} \right) dx \tag{6.1}$$

where A is the wave amplitude, λ the wavelength, ρ the fluid density, g the gravitational acceleration, and $d_{\text{submerged}}$ the submerged depth. The first term represents the dynamic wave load, and the second the hydrostatic pressure. When the integral is solved the equation done below is left.

$$M_{\text{wave}} = -\rho * g \frac{A \cdot \lambda^2}{2\pi} \tag{6.2}$$

This gives the moment for a unit length. For the heel angle the full moment is necessary obtained by multiplying with the perpendicular direction the then one used for λ.

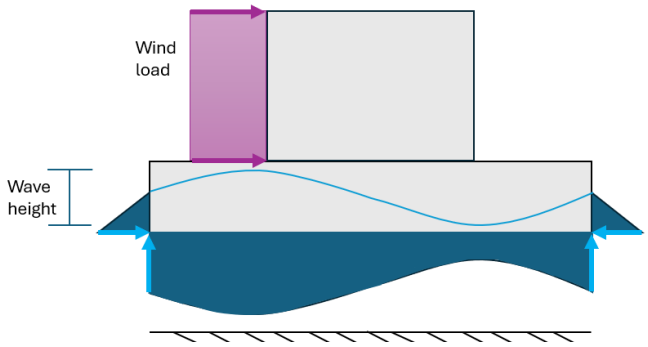


Figure 6.3: Combined wind and wave loading on the floating structure

6.1.3. Combined Effects

No single load direction can be identified as universally governing. For wind loads, the critical moment arises when the largest projected area of the building facade is exposed. For wave loads, the largest moment arises along the longest platform dimension. However, the governing situation always occurs when both wind and wave forces act on the same side, as illustrated in Figure 6.3.

Table 6.2: Summary of environmental load combinations

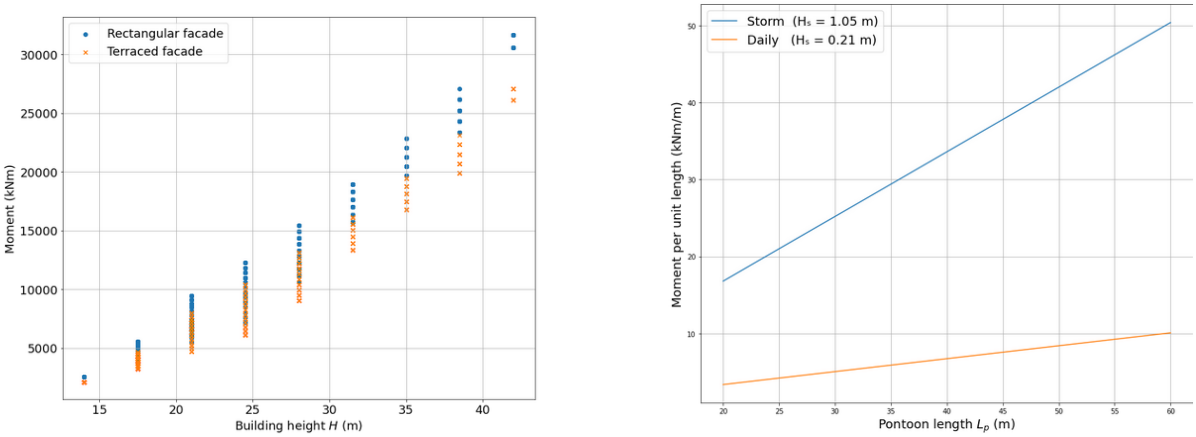
Scenario	Wind Pressure (kN/m^2)	Wave Height (m)	Wave Period (s)	Wave Frequency (Hz)
Regular conditions	0.46	0.21	4.6	0.22
Storm conditions	1.09	1.05	4.6	0.22

6.1.4. Moment and Heel Angle

After computing the external moments caused by wind and wave loading, the resulting heel angle of the system can be determined for both storm and everyday conditions. This angle is used to assess whether the floating structure remains within acceptable stability criteria. The following thresholds are applied:

- The heel angle must not exceed 10° under storm conditions.
- During normal weather conditions, the heel angle should remain below 3° (J.M.J. Journée & W.W. Massie, 2001).

where Δ is the displacement (equal to the total weight of the floating system).



(a) Increase moment with height

(b) Increase of moment wave with length

Figure 6.4: Moment increase with increase of dimensions

The heel angle φ is derived from the following expression:

$$M_{\text{extern}} = \Delta \cdot GM \cdot \sin\varphi \tag{6.3}$$

The resulting heel angles under both load scenarios are visualized in the following graph.

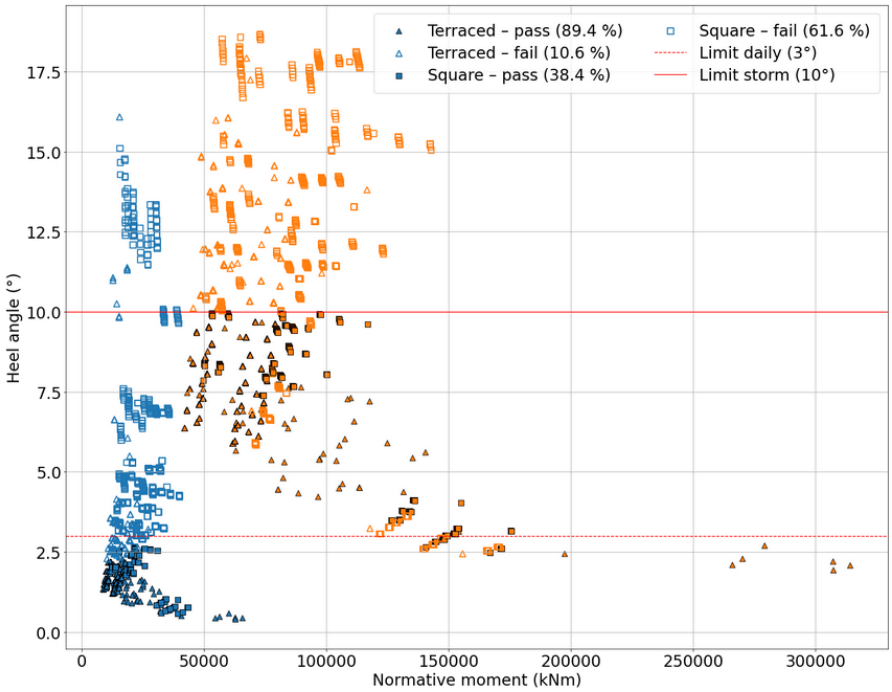


Figure 6.5: Heel angle of all visuable options until now

6.2. Natural Frequency

As discussed in Chapter A, determining the natural frequency of the floating system is essential. This frequency characterizes the rate at which the structure oscillates after a disturbance, assuming no external damping is present. An accurate estimate is crucial to avoid resonance under environmental loading and to prevent occupant discomfort or health risks associated with low-frequency motion. In this study, only configurations with natural frequencies outside the known motion sickness range are considered viable. A full analysis of RMS accelerations lies beyond the scope of this research and is therefore not included.

The starting point is from the single–degree-of-freedom (SDOF) heave model shown in Figure 6.6 and then generalise the result to the other modes of motion.

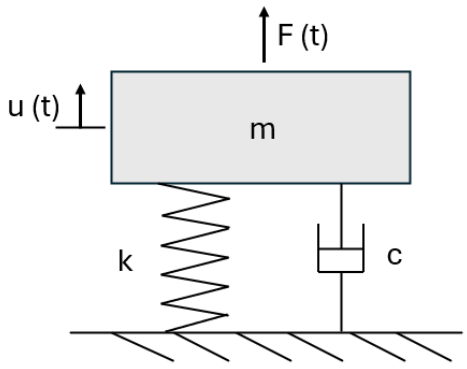


Figure 6.6: Mass–spring–damper model for heave motion

6.2.1. Heave: reference SDOF system

The equation of motion for pure heave is

$$(M + m)\,\ddot{u} + c\,\dot{u} + k\,u = F, \tag{6.4}$$

with

- u vertical displacement (heave) [m],
- M structural mass [kg],
- m added mass (see Appendix B) [kg],
- c damping coefficient [Ns/m],
- k hydrostatic stiffness [N/m],
- F external forcing term [N].

The stiffness originates from hydrostatic restoring forces and is given by

$$k = \rho\, g\, A_{\text{platform}} \quad [\mathrm{N/m}], \tag{6.5}$$

where ρ is water density, g gravitational acceleration and A_{platform} the cross sectional area.

Because every point of the rigid body experiences the *same* vertical displacement, only the *total* mass $M + m$ appears in the inertial term; the internal mass distribution is irrelevant. Neglecting damping in the first instance, the undamped natural frequency is

$$\omega_{0,\text{heave}} = \sqrt{\frac{k}{M+m}} \quad [\mathrm{rad/s}], \tag{6.6}$$

and the damped natural frequency becomes

$$\omega_{n,\text{heave}} = \omega_{0,\text{heave}} \sqrt{1-\zeta^2}, \tag{6.7}$$

with a typical marine damping ratio of $\zeta = 0.1$ (Spiering, 2022).

Translation modes Because surge and sway are also pure translations, their natural frequencies follow directly from

$$\omega_{n,\text{trans}} = \sqrt{\frac{k}{M + m_{\text{added}}}}, \tag{6.8}$$

where only the total mass and the appropriate added-mass term (M_x or M_y) enter the denominator.

For roll and pitch, however, the mass distribution *does* matter, and the mass moment of inertia I must be included. These rotational modes are discussed in detail in the next part

6.2.2. Mass distribution for roll and pitch

In contrast to translational modes (heave, surge, sway), where the entire structure moves uniformly in a straight line and only the *total mass* affects the system's response, rotational motions such as roll and pitch are governed by the *rotational moment* generated by mass located at a distance from the axis of rotation.

When a structure rotates about an axis (typically near the center of buoyancy), each mass element contributes to the system's resistance to rotation depending on its distance to that axis. The farther the mass is located from the axis, the greater the *moment* it generates—and the more energy is required to rotate the structure. This resistance is quantified by the *mass moment of inertia* I.

The natural frequency for rotational modes thus includes this effect:

$$\omega_{n,\text{rot}} = \sqrt{\frac{k}{(I + M_\theta)/r_g^2}}, \tag{6.9}$$

where:

- I is the mass moment of inertia of the platform–building system,
- M_θ is the added mass moment from the surrounding water,

- r_g is the radius of gyration,
- k is the hydrostatic restoring stiffness.

The value of I depends on the *shape and distribution* of the structure's mass. A taller structure (larger H) or a wider base (larger L_{down}) moves mass farther from the rotation axis, increasing I and thus *lowering the natural frequency*.

Square geometry For a rectangular building:

$$I_{\text{square}} = m\,\frac{B^2 + H^2}{12}. \tag{6.10}$$

Terraced geometry For the terraced shaped building a simplification is chosen For a terraced shape represented by a trapezoidal profile, the mass moment of inertia is given by:

$$I_{\text{terrace}} = m\,\frac{H(L_{\text{down}} + L_{\text{up}})(L_{\text{up}}^2 + 7L_{\text{down}}^2)}{48}. \tag{6.11}$$

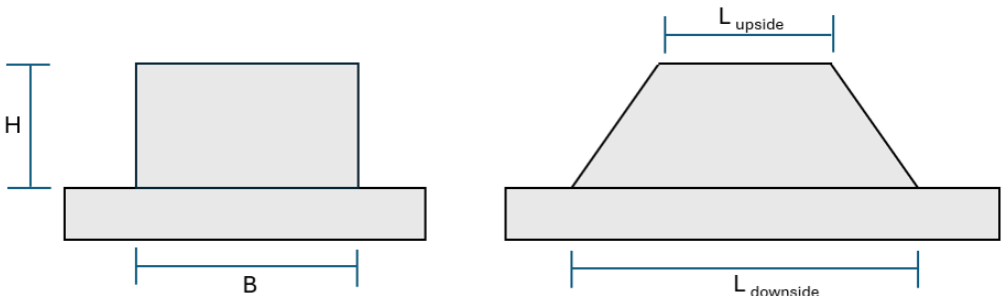


Figure 6.7: Centroidal mass moment of inertia for square (left) and terraced (right) structures.

Figure 6.8 shows the natural frequencies of all previously identified feasible configurations, categorized by motion mode. It is evident that for all configurations, both the surge and sway modes fall within the motion-sickness frequency range (0.1–0.5 Hz). This implies that, under excitation, occupants may be susceptible to motion-induced discomfort.

Additionally, the graph indicates a wave excitation peak around 0.22 Hz. Natural frequencies near this peak are especially vulnerable to resonance, which can amplify motion responses significantly. As a result, configurations whose natural frequencies align closely with this wave peak may experience increased oscillations.

However, it is important to emphasize that the occurrence of motion sickness not only depends on frequency but also on the magnitude of resulting accelerations. Since this study focuses on natural frequencies only, a full evaluation of occupant comfort based on RMS acceleration thresholds remains outside the scope of this research.

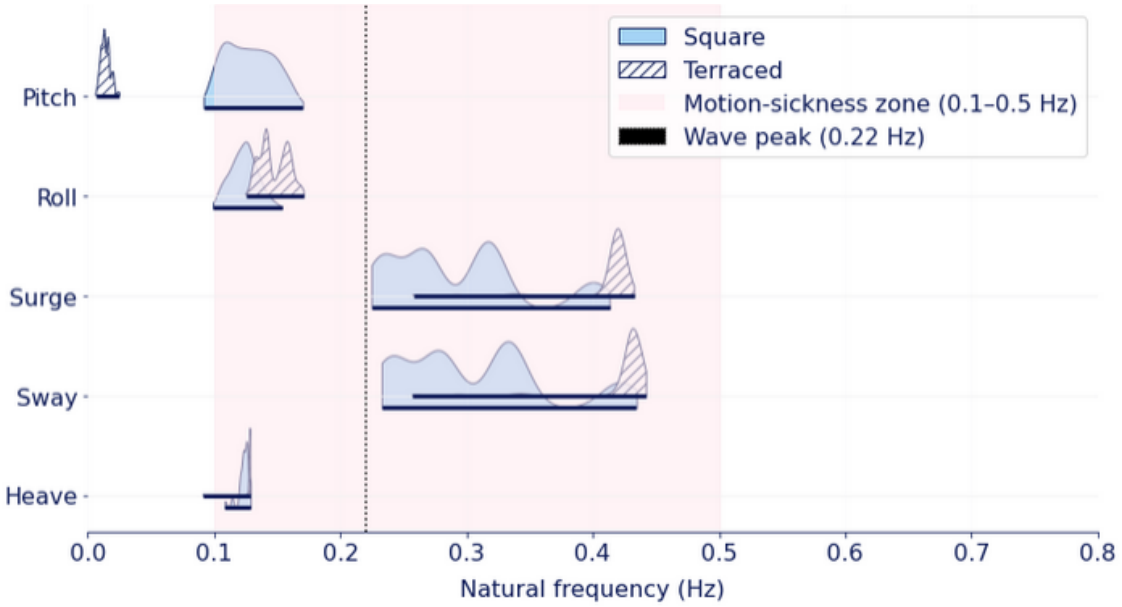


Figure 6.8: Natural frequency per mode

6.2.3. Iterations

An effective strategy to reduce the risk of motion sickness is to shift the natural frequencies of the system outside the critical motion-sickness zone (0.1–0.5Hz). Figure6.9 explores this approach by applying additional ballast, up to a total platform draft of 6 meters, to all feasible configurations. The added ballast, modeled as water-filled compartments, increases the total mass of the system and thus lowers the natural frequencies. This can help prevent resonance-induced motion discomfort, even under significant excitation.

While the added ballast successfully shifts some natural frequencies further from the wave excitation peak (0.22 Hz), none of the configurations fully avoid the motion-sickness zone across all modes. In particular, surge and sway frequencies remain within the sensitive range, indicating that even with increased mass, motion sickness cannot be ruled out solely by tuning frequency.

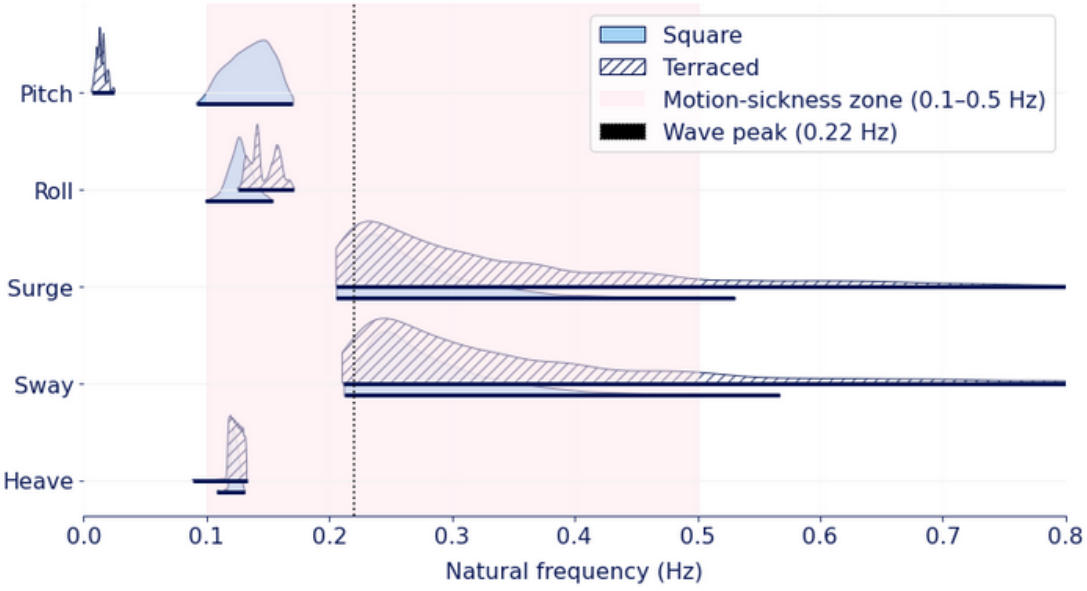


Figure 6.9: Natural frequency per mode with ballast until 6 meter depth

7

Discussion

This chapter reflects on the technical findings presented in the previous chapters and places them in a broader context. The study assessed the feasibility of buoyant apartment buildings in Rotterdam's inner-city harbours by analysing spatial constraints, structural design, platform sizing, and stability performance.

Rijnhaven and Maashaven were identified as suitable locations based on available depth and urban integration. A modular pontoon system (40–75 m wide, up to 6 m deep) was developed within construction constraints. Steel frames in a regular grid proved structurally viable, and terraced geometries offered better stability due to a lower centre of gravity.

While most terraced configurations met heel angle criteria under both daily and storm loads, many square options failed. Comfort performance remains uncertain: surge and sway frequencies fall within the motion sickness range, but no acceleration analysis was conducted. The following sections interpret these findings, identify limitations, and recommend directions for further research.

7.1. Interpretations

The results confirm that large floating apartment buildings are technically feasible within Rotterdam's inner-city harbours, especially when using terraced configurations and a modular steel frame on a buoyant pontoon system. The high GM values (6.1–33.4 m) indicate excellent hydrostatic stability, although they also shorten the natural roll period, which could influence comfort.

The strong performance of terraced geometries, both in terms of hydrostatic GM and storm heel angle, demonstrates the structural and stability benefit of gradually reducing mass at higher levels. The square configurations, while simpler in form, often fail to meet heel limits, suggesting that mass distribution plays a critical role in dynamic stability.

The fact that surge and sway modes lie in the motion sickness frequency range implies that even technically stable structures may still cause occupant discomfort without careful dynamic tuning (e.g. ballast or damping). However, without RMS acceleration data, no conclusive statements can be made about compliance with comfort criteria such as ISO 2631.

7.2. Implications

These findings demonstrate that Rotterdam's inner harbours can be repurposed to accommodate large scale floating housing, expanding the city's residential capacity without additional land take. The modular platform system enables flexible adaptation to varying harbour geometries, and the terraced building form provides a viable balance between usable floor area and stability.

From an engineering standpoint, the study shows that conceptual level modelling using hydrostatics, simplified wave loading, and geometric inertia approximations can already yield valuable insights into platform feasibility and design sensitivity. Importantly, it highlights that stability compliance does not

automatically guarantee comfort compliance, and that early-phase designs should include both.

For urban planning, the results support the integration of floating structures into medium-density urban waterfronts, particularly where land-based development is constrained. They also suggest that zoning plans and harbour design codes should be updated to account for the specific characteristics and needs of floating multi-storey buildings.

7.3. Limitations and Recommendations

Although the study demonstrates the technical and spatial feasibility of buoyant apartment buildings, several simplifying assumptions restrict the reliability and completeness of its conclusions. This section outlines those limitations and provides targeted recommendations to address them in future studies.

7.3.1. Structural refinement and heel-induced moment shifts

The current structural model assumes axial column loading and symmetric beam bending, aligned with design rules such as those in F.A.M. Soons et al. (n.d.). However, when the platform heels due to wind and wave excitation, gravity loads shift laterally, generating significant shifts in bending moments in columns, beams, and the pontoon itself.

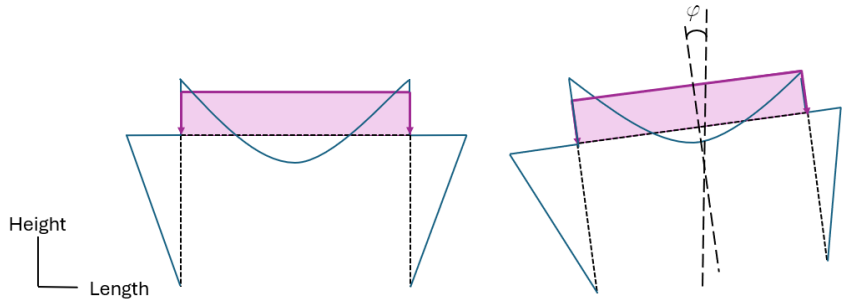


Figure 7.1: Illustration of differing bending moments in the upright (left) and heeled (right) condition.

Figure 7.1 shows how the vertical gravity force no longer acts perpendicular to the structural members when the platform is heeled. This leads to increased bending moments, especially in the left column and adjacent beam, because the effective lever arm increases. Decomposing the gravity load into vertical and horizontal components reveals that more of the vertical force is now transmitted eccentrically relative to the original axis. These additional effects are not accounted for in the current 2D hydrostatic model, resulting in underestimated forces and potentially unsafe member sizing.

Future work should combine hydrostatic righting moments with a 3D structural model to verify sizing under heel induced loading. In addition, alternative frame designs such as core systems, composite beams, or tapered floor plates could be explored to reduce vulnerability to such lateral load redistribution.

7.3.2. Platform utilisation and buoyancy strategy

Beyond structural support, the pontoon volume could be further utilised to optimise buoyancy and stability. For example, integrating technical spaces such as storage units, HVAC systems, or shared infrastructure in the lower levels of the platform would increase mass low in the system, enhancing stability while improving functional space use.

Material choice also plays a critical role. Future concepts could investigate platforms made of lightweight steel, EPS foam cores, or hybrid concrete–EPS systems to tailor buoyancy and stiffness more precisely.

Furthermore, the current design assumes a uniform rib layout across all pontoons. Optimising rib geometry, orientation, and spacing per module may reduce structural weight or improve stiffness. Tailoring pontoon shapes could also improve wave shedding and hydrostatic performance.

subsectionHydroelastic response and platform flexibility Given the platform's large dimensions and modular nature, rigid-body assumptions may no longer be sufficient. Recent studies such as Amouzadrad

et al. (2024) highlight the importance of modelling the hydroelastic response and gap resonance in large multi-body floating structures.

Flexural deformation of pontoons under wave loading can introduce coupling between structural and hydrodynamic modes, affecting both comfort and fatigue life. Future work should:

- Perform hydroelastic simulations to capture dynamic flexing and its influence on internal stresses.
- Investigate the effect of platform flexibility on resonance amplification, particularly in modular gaps.

7.3.3. Building geometry and optimisation

This study explored square and one-sided terraced forms. A logical extension is the use of pyramid shaped terraced geometries, which could lower the centre of gravity in both axes and improve the system's resistance to heel in multiple directions. These shapes may also provide aesthetic and wind shedding advantages.

Additionally, the design process could be enhanced through parametric optimisation tools that simultaneously evaluate building form, mass distribution, pontoon layout, and rib alignment. This would enable generation of a wider range of feasible configurations with improved performance.

7.3.4. Vibration comfort and RMS accelerations

This study identified that sway and surge modes fall within the 0.1–1 Hz range, which corresponds to the most sensitive band for motion sickness and comfort impairment according to ISO 2631-1:1997. However, comfort evaluation was limited to natural frequency analysis alone. No *root mean square (RMS)* acceleration values were computed, which are necessary for compliance with international vibration comfort standards.

ISO 2631-1 specifies that comfort and motion sickness effects due to whole-body vibration must be assessed using:

- frequency weighting filters W_k (vertical) and W_d (horizontal),
- computation of frequency-weighted RMS accelerations a_w,
- comparison to threshold values (e.g., 0.315 m/s^2 for noticeable discomfort),
- and interpretation over duration and exposure conditions.

While several configurations exhibit natural frequencies within this sensitive zone, human discomfort depends on both frequency and acceleration magnitude. Without a_w-values, no statement can be made on compliance with ISO limits.

Interestingly, a reference case—a two-storey house on a compact pontoon—also exhibits natural frequencies within this critical range, yet likely remains comfortable under real-world conditions due to low RMS accelerations.

Table 7.1: Natural frequencies for a reference two-storey house on a small pontoon

Platform						**Stability & Dynamics**		
L (m)	W (m)	Subm. D (m)	Payload (kN)	Ballast (kg)	Storeys	f_{heave} (Hz)	f_{roll} (Hz)	Freeboard (m)
12.0	9.0	2.0	1 750	37 620	2	0.256	0.278	0.35

This shows that natural frequency alone does not predict discomfort—RMS accelerations must be considered. The house likely remains within ISO 2631-1 limits for comfort and motion sickness because of limited dynamic excitation and high damping from mooring and surroundings.

Recommendation: Future studies should perform time-domain simulations or physical model testing to record acceleration time histories, apply appropriate ISO frequency weightings, and compute RMS accelerations for comparison with ISO 2631-1 and ISO 2631-2 comfort thresholds.

7.3.5. Freeboard and overtopping risk

A fixed freeboard of 0.3 m was used for all configurations, independent of platform size or local wave exposure. In narrow basins or under storm conditions, this may lead to overtopping. Recommendations include:

- Reassessing freeboard based on site specific wave climates.
- Including wave run-up, wind set-up, and dynamic amplification in the analysis.

7.3.6. Inner harbour wave modelling

Storm wave conditions were approximated using a scaling factor on open-sea data. However, real harbour environments involve short-crested, reflected, and ship-generated waves.

Future studies should:

- Develop local wave spectra that reflect these conditions.
- Validate against measured data where available.

7.3.7. Financial feasibility

The study excluded a cost-benefit analysis. Without economic metrics, feasibility remains only technical.

To address this:

- Conduct life-cycle cost analyses including construction, dredging, mooring, and maintenance.
- Assess the financial value of additional waterfront development rights or public-space enhancements.

Addressing these limitations will provide a more robust design framework and improve the viability of floating apartment buildings in Rotterdam's inner-city harbours.

8

Conclusion

Rotterdam's housing shortage requires creative use of underutilised water surfaces to unlock new opportunities for urban densification. This research investigated the spatial and structural requirements for realising large buoyant apartment buildings within the city's inner harbours. Using a combination of bathymetric analysis, structural design modelling, and hydrostatic stability assessment, the study addressed four guiding sub-questions. The main findings are outlined below.

8.1. Area Research question 1

Bathymetric and geospatial analysis confirmed that Rijnhaven and Maashaven offer the most favourable spatial conditions for large floating buildings. Both harbours are surrounded by mid-rise urban fabric averaging 8 storeys. Water depths of up to 6 metres are achievable with minimal dredging, typically only required near quay edges.

Platform length and width are not significantly constrained, particularly when using a modular construction strategy. If mobility or relocation of building units is required, the platform dimensions should not exceed 120 metres.

8.2. Frames Research question 2

A steel frame structure was selected for its low weight and compatibility with floating platforms. Steel columns and beams are arranged in a regular grid, aligned with the pontoon's internal ribs for efficient load transfer. Two building geometries were explored: a square block and a terraced configuration. The terraced form reduces upper-floor loads and lowers the centre of gravity, which improves platform stability. Both geometries are structurally feasible, but the terraced layout offers advantages in terms of weight distribution and integration with buoyant support.

8.3. Platform sizing and configuration Research question 3

Chapter 4 defines a modular pontoon system: individual units are restricted to ≤ 20 m in length and 4–8 m in width (1 m increments) to stay within realistic transport and construction limits . These modules are assembled into square platforms whose overall length and width are varied from 40 m to 75 m in 5 m steps. Platform depth is tied to available water depth and ranges from 3 m to 6 m, after which a fixed 0.3 m freeboard is added to limit overtopping .

Efficient load transfer is ensured by enforcing rib-portal alignment: every building portal must coincide with a platform rib within a ±0.10 m tolerance; the model tests portal counts between three and eight to satisfy this requirement .

Hydrostatic checks in Chapter 5 show that both the square and terraced building options remain stable on these platforms. For a representative 45 × 45 m raft, the square building yields GM ≈ 9.82 m, while the terraced variant—despite its lower centre of gravity—achieves a comparable GM ≈ 9.99 m. At

smaller footprints (35 × 40 m), the square form produces higher GM values (≈ 33 m) than the terraced form (≈ 23 m), but both comfortably exceed the 0.25 m stability criterion. This demonstrates that the modular raft, in depths of 3–6 m, can support either geometry while satisfying buoyancy, freeboard, and stability requirements.

8.4. Stability performance Research question 4

Chapter 6 evaluates the stability of the combined floating system under wind and wave loading for both regular and storm conditions. External moments caused by wind pressure and wave height were calculated and compared against the restoring moment due to buoyancy. The resulting heel angles were used to assess whether the design stays within the 3° (daily) and 10° (storm) limits.

Figure 6.5 shows that a significant number of configurations fail to meet these stability thresholds. In particular, only 38.4% of square configurations passed the heel angle requirements, while the terraced geometry performs significantly better, with an 89.4% pass rate. This highlights the stabilising effect of the terraced mass distribution and lower centre of gravity.

Natural frequencies for heave, sway, surge, roll, and pitch were also evaluated. While surge and sway frequencies fall within the motion sickness range, their impact cannot be fully assessed because acceleration values were not calculated. Without acceleration data, it is not possible to determine whether the structure exceeds comfort thresholds based on standards like ISO 2631.

In summary, the combined system meets the literature-based safety criteria for stability in most terraced cases and some square configurations. However, comfort performance remains uncertain, and a full evaluation would require both frequency and acceleration analysis.

Main research question and objective

This study set out to determine the **technical and spatial feasibility** of large buoyancy-based floating platforms for apartment buildings in Rotterdam's inner city waters and harbors. The main research question focused on identifying the spatial and structural requirements to enable such developments as a way to expand the city's housing capacity.

Based on the findings from the four sub-questions, it is concluded that:

- **Spatial feasibility is confirmed:** locations such as Rijnhaven and Maashaven offer sufficient depth (up to 6 metres) and surface area to accommodate large floating platforms, with surrounding building heights (≈ 8 storeys) that support integration into the existing urban fabric.
- **Structurally feasible solutions exist:** a fully steel structural frame with a regular grid aligns with the modular pontoon system and keeps weight low. Both square and terraced building layouts can be supported, with the terraced form providing improved stability.
- **Platform configurations are viable:** modular pontoons with widths of 4–8 metres and depths up to 6 metres can be assembled into platforms between 40 and 75 metres wide, while satisfying transport, buoyancy, and stability constraints.
- **Stability criteria are largely met:** most terraced configurations and a subset of square layouts remain within accepted heel limits under both regular and storm conditions. Dynamic comfort performance is likely achievable with additional ballast, though further analysis including acceleration is needed.

In summary, the study demonstrates that large buoyant floating apartment buildings are **both spatially and technically feasible** in Rotterdam's inner-city harbors, provided design constraints related to draft, weight distribution, and modularity are respected. These findings support further exploration of floating urban development as a viable strategy to address housing demand in dense delta cities.

References

Amouzadrad, P., Mohapatra, S. C., & Guedes Soares, C. (2024). Review of recent developments on the hydroelastic response and gap resonance of multi-body floating structures. *Ocean Engineering*, *313*, 119398. https://doi.org/10.1016/j.oceaneng.2024.119398

Award winning Nelson Mandela Park masterplan. (2022, January). Retrieved April 30, 2025, from https://www.landscapearchitecture.nz/landscape-architecture-aotearoa/2021/10/13/rotterdams-nelson-mandela-park-design-wins-award

Barltrop, N. (1998). Knovel - Floating Structures - A Guide for Design and Analysis, Volumes 1-2. Retrieved June 15, 2025, from https://app.knovel.com/kn/resources/kpFSAGDAVB/toc

Bikbouw. (n.d.). Retrieved May 19, 2025, from https://www.bikbouw.nl/index.php?option=com_content&view=article&id=40&catid=11

Bouwwereld, R. (2010, May). New Orleans, Rotterdam » Bouwwereld.nl. Retrieved May 7, 2025, from https://www.bouwwereld.nl/bouwtechniek/new-orleans-rotterdam/

Buurt Eemhaven (gemeente Rotterdam) in cijfers en grafieken. (2016, July). Retrieved May 6, 2025, from https://allecijfers.nl/buurt/eemhaven-rotterdam/

Buurt Waalhaven (gemeente Rotterdam) in cijfers en grafieken. (2016, July). Retrieved May 6, 2025, from https://allecijfers.nl/buurt/waalhaven-rotterdam/

CBS. (2023, September). What is working people's income? - The Netherlands in numbers | CBS [Last Modified: 2023-09-14]. Retrieved April 30, 2025, from https://longreads.cbs.nl/the-netherlands-in-numbers-2023/what-is-working-peoples-income

Eurocode. (n.d.). *Table of design properties for flanged steel profiles* (tech. rep. No. EN 1993-1-1:2005+AC2:2009).

F.A.M. Soons, B.P.M. van Raaij, L.A.G. Wagemans, S. Pasterkamp, & S.H.J. van Es. (n.d.). Quick reference.

J.M.J. Journée & W.W. Massie. (2001, January). Https://ocw.tudelft.nl/wp-content/uploads/OffshoreHydromechanics_jou Retrieved June 12, 2025, from https://ocw.tudelft.nl/wp-content/uploads/OffshoreHydromechanics_Journee_Massie.pdf

Keete, M. (2024, February). Woningnood neemt toe: 'Rotterdam heeft veel meer sociale huurwoningen nodig'. Retrieved April 23, 2025, from https://nos.nl/regio/zh-rijnmond/artikel/491226-woningnood-neemt-toe-rotterdam-heeft-veel-meer-sociale-huurwoningen-nodig

Kroftman. (n.d.). Wind speed and resulting horizontal wind load (pressure) on structures.

Kust en benedenrivieren - Rijkswaterstaat Waterinfo. (n.d.). Retrieved June 15, 2025, from https://waterinfo.rws.nl/thema/Kust%20en%20benedenrivieren?view=list

Langen, M. (2025, January). Housing market - building according to need. Retrieved April 29, 2025, from https://www.abnamro.com/research/en/our-research/housing-market-building-according-to-need

Lin, F. Y., Spijkers, O., & van der Plank, P. (2022). Legal Framework for Sustainable Floating City Development: 2nd World Conference on Floating Solutions, WCFS 2020. *Lecture Notes in Civil Engineering*, 433–460. https://doi.org/10.1007/978-981-16-2256-4_27

Mechanische trillingen en schokken Beoordeling van de invloed van trillingen op het menselijk lichaam - Deel 1 : Algemene eisen (tech. rep. No. NEN-ISO 2631-1). (1997).

Metacenter | Definition & Facts | Britannica. (n.d.). Retrieved June 12, 2025, from https://www.britannica.com/science/metacentre

NEN Connect - ISO 2631-2. (2003). Retrieved June 16, 2025, from https://connect.nen.nl/Family/Detail/60254?compId=10037&collectionId=0

NEN Connect - NTA 8111:2011 nl. (2011). Retrieved June 12, 2025, from https://connect.nen.nl/Standard/Detail/162011?compId=10037&collectionId=0

Netherlands, S. (n.d.). Households today [Last Modified: 08-02-2024T14:58:40]. Retrieved April 29, 2025, from https://www.cbs.nl/en-gb/visualisations/dashboard-population/households/households-today

Noorderkwartier, H. H. (2025, April). Buitendijks gebied [Publisher: Hoogheemraadschap Hollands Noorderkwartier]. Retrieved May 2, 2025, from https://www.hhnk.nl/buitendijks-gebied

Peng, L. (2012). New land: A look at very large floating structures.

Port of Rotterdam. (n.d.). Nautical annual figures 2024: Harbour Master focuses on speed limits around the Erasmusbrug in Rotterdam | Port of Rotterdam. Retrieved May 7, 2025, from https://www.portofrotterdam.com/en/news-and-press-releases/nautical-annual-figures-2024-harbour-master-focuses-speed-limits-around

Rijkswaterstaat. (2014, August). *Golfbelastingen in havens en afgeschermde gebieden* (tech. rep. No. RWS.2014.001). Rijkswaterstaat Water, Verkeer en Leefomgeving.

Rijnmond, R. (2021, September). Koning en Ban Ki-Moon openen drijvend kantoorgebouw in de Rotterdamse Rijnhaven [Section: nieuws]. Retrieved May 6, 2025, from https://www.rijnmond.nl/nieuws/1372894/koning-en-ban-ki-moon-openen-drijvend-kantoorgebouw-in-de-rotterdamse-rijnhaven

Ritchie, H., Samborska, V., & Roser, M. (2024). Urbanization. *Our World in Data*. Retrieved April 29, 2025, from https://ourworldindata.org/urbanization

Rotterdam. Home port for innovative floating solutions. (n.d.). Retrieved May 19, 2025, from https://rotterdammakeithappen.nl/en/showcases/rotterdam-home-port-for-innovative-floating-solutions/

Rubin, S. (2021, December). In Amsterdam, a community of floating homes shows the world how to live alongside nature. Retrieved May 19, 2025, from https://www.washingtonpost.com/climate-solutions/interactive/2021/amsterdam-floating-houses-schoonschip/

Spiering, B. (2022, April). From the Sea into the Sky. Retrieved June 16, 2025, from https://repository.tudelft.nl/record/uuid:5323efd2-23c3-49b8-9b75-3d707a6fc616

Statistiek, C. B. v. d. (2022, July). Prognose: In 2035 vooral meer inwoners in en om grotere gemeenten [Last Modified: 2022-07-06T00:00:00+02:00]. Retrieved April 29, 2025, from https://www.cbs.nl/nl-nl/nieuws/2022/27/prognose-in-2035-vooral-meer-inwoners-in-en-om-grotere-gemeenten

Uitvoeringsagenda 1.0 NOVEX Rotterdamse haven 2025-2050 (Uitvoeringsagenda). (2025, April). Retrieved May 6, 2025, from https://www.portofrotterdam.com/sites/default/files/2025-04/Uitvoeringsagenda%20NOVEX%20Rotterdamse%20haven.pdf

VROM, M. v. v., ruimtelijke ordening en milieubeheer. (2009, April). Https://open.overheid.nl/documenten/ronl-99bb2a72ea23d8a31889c1d3a8b13f7e8aace52a/pdf. Retrieved May 2, 2025, from https://open.overheid.nl/documenten/ronl-99bb2a72ea23d8a31889c1d3a8b13f7e8aace52a/pdf

Wang, C. M., & Tay, Z. Y. (2011). Very Large Floating Structures: Applications, Research and Development. *Procedia Engineering*, *14*, 62–72. https://doi.org/10.1016/j.proeng.2011.07.007

Waterstaat, M. v. I. e. (n.d.). Ruimte voor de rivieren [Last Modified: 2025-04-10T12:04:56]. Retrieved May 19, 2025, from https://www.rijkswaterstaat.nl/water/waterbeheer/bescherming-tegen-het-water/maatregelen-om-overstromingen-te-voorkomen/ruimte-voor-de-rivieren

A

Background

In this chapter the first section is about the past and present of floating homes and the context in which they are built. The second section is about very large floating structures (VLFS). Then the existing legislation and codes will be discussed and in the last section the allowed movements of the floating system for determining stability is looked upon.

A.1. The Past and Present

Since the beginning of civilization, people have settled along coasts and large bodies of water to secure food and fresh water. An example is Luoyuan Bay in China, where fishermen have lived on floating villages for centuries (Lin et al., 2022). Similarly, in the Netherlands, living on water has a long history. Since the 17th century, people have used water for living, and in the 1960s, houseboats became a way to opt out of conventional societal norms on land (Rubin, 2021).

For much of Dutch history, the relationship with water was about keeping it out. However, since the early 2000s, this strategy has shifted due to climate change (Waterstaat, n.d.). Dutch policy now embraces living with water rather than fighting against it. This change not only influenced river management but also sparked new ways of living on water. Examples include floating neighborhoods and multifunctional facilities, as shown in figures A.1 and A.2.



Figure A.1: Floating homes in Rotterdam Nassauhaven ("Bikbouw", n.d.).

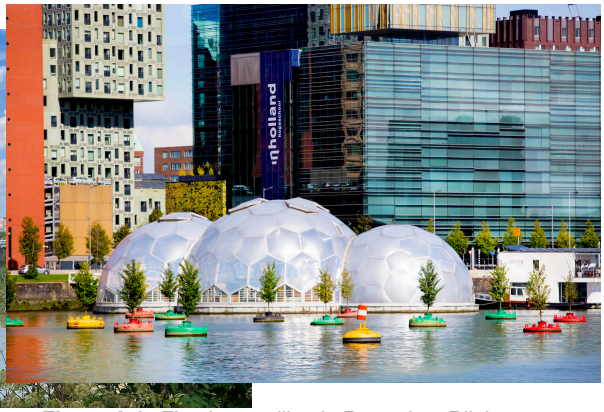


Figure A.2: Floating pavilion in Rotterdam Rijnhaven ("Rotterdam. Home port for innovative floating solutions." n.d.).

A.2. Very large floating structures

According to Wang and Tay (2011), VLFS are generally classified as either semisubmersible-type or pontoon-type. The semisubmersible-type uses columns to lift the platform above the waves, making it suitable for rough sea conditions. The pontoon-type, on the other hand, floats directly on the water and is best suited for sheltered environments like harbors. Due to their large size, these structures face challenges such as bending under loads and the need for proper mooring to prevent platforms from becoming freely drifting structures.

So far, VLFS have mostly been applied in infrastructure projects like floating bridges and airports (Peng, 2012). Their potential for large-scale floating housing, however, remains largely unexplored. This study focuses on the pontoon-type as the most promising concept for developing floating neighborhoods in calm, protected waters.

A.3. Legislation

A key legal consideration is whether a floating home is classified as a *bouwwerk* (building) or a *woonark* (houseboat) (VROM, 2009). When classified as a *bouwwerk*, the construction requires a building permit. In contrast, a *woonark* only requires a mooring permit and is therefore largely exempt from standard building regulations.

However, the status of *bouwwerk* offers several advantages for both developers and owners, including:

- Uniform legislation that supports scalable and repeatable development;
- Greater legal certainty in spatial planning procedures;
- The possibility to own the water plot;
- Potentially lower insurance premiums.

On the other hand, a *woonark* offers more flexibility, as it is not bound by the requirements of the *Bouwbesluit* (Building Decree). In this study, it is assumed that the system of the platform and the building will be classified as a *bouwwerk*. This means that the structure must comply with applicable Eurocodes and the guidelines set out by the Dutch Ministry of Housing, Spatial Planning, and the Environment (VROM) in the document *Drijvende woningen en de bouwregelgeving*.

A.4. Serviceability Limit

To ensure the system meets both static and dynamic stability requirements for safe and habitable floating buildings, relevant guidelines must be considered. This section first addresses static stability, followed by dynamic criteria.

In the Netherlands, the technical agreement for floating structures is described in NTA 8111 (2011). This guideline primarily applies to buildings with up to three storeys above the waterline, corresponding to consequence classes CC1 or CC2 under Eurocode definitions. Since the buildings considered in this research exceed these classes, NTA 8111 is not directly applicable and a safety assessment must be made (NTA 8111, 2011). However, several of its serviceability criteria are still adopted as conservative minimum requirements:

- The platform must maintain a minimum freeboard of 0.3 m.
- The metacentric height (GM) must exceed 0.25 m to ensure static stability.

For an explanation of the metacentric height, see Chapter 5.

Although no specific regulatory limits exist for the dynamic behavior of high-rise floating buildings, performance criteria are adopted from comfort guidelines from ISO 2631. Vibrations in the range of 1–80 Hz are perceptible to occupants and may harm comfort regardless of posture (“NEN Connect - ISO 2631-2”, 2003). In addition, motion sickness may occur in the frequency range of 0.1–0.5 Hz(*Mechanische trillingen en schokken Beoordeling van de invloed van trillingen op het menselijk lichaam - Deel 1 : Algemene eisen*, 1997).

To ensure comfort and safety, the system's natural frequencies should lie outside these ranges and avoid the dominant wave frequency to minimize resonance effects.

B

Added Mass

When a solid body accelerates within a fluid, or when fluid flows around a stationary object, the flow pattern becomes distorted. This leads to a pressure gradient surrounding the object, effectively causing the surrounding fluid to move along with it. This phenomenon can be modeled as an *added mass*, representing the inertia of the fluid that must be accelerated in addition to the solid body itself (Barltrop, 1998).

Exact calculations of added mass are challenging, as the values depend on the direction of motion, oscillation frequency, and the geometry of the submerged body. However, approximate formulas are available in the textbook *Floating Structures* by Barltrop (1998). These expressions give added mass per unit length and must be multiplied by the relevant dimension of the platform. In all equations, T refers to the actual submerged depth, as calculated in Chapter 5.

In the implementation, B is taken as the minimum of the platform's length and width, while C is the maximum, consistent with Equation (B.1).

$$M_z = \frac{1}{2}\,\rho\,\pi\left(\frac{B}{2}\right)^2 \cdot C \tag{B.1}$$

As discussed in Chapter 6, surge occurs along the x-axis. Therefore, the added mass in surge direction is multiplied by the width W, which is perpendicular to the direction of motion:

$$M_x = \frac{1}{2}\,\rho\,\pi\,T^2 \cdot W \tag{B.2}$$

Similarly, sway occurs along the y-axis and is therefore multiplied by the length L:

$$M_y = \frac{1}{2}\,\rho\,\pi\,T^2 \cdot L \tag{B.3}$$

Roll is defined as rotation about the x-axis and thus uses the perpendicular platform dimension W. The center of rotation is assumed at the water surface, and r is defined as the distance from this surface to the center of buoyancy. For these platforms, $r = -T/2$:

$$M_{\text{roll}} = L \cdot \rho \cdot \pi \left[\left(\frac{W}{4}\right)^2 \left(\frac{W}{4}\right)^2 + \left(\frac{T}{2}\right)^2 \left(\frac{T}{2} + r\right)^2\right] \tag{B.4}$$

Pitch, the rotation about the y-axis, uses the perpendicular platform dimension L:

$$M_{\text{pitch}} = B \cdot \rho \cdot \pi \left[\left(\frac{L}{4}\right)^2 \left(\frac{L}{4}\right)^2 + \left(\frac{T}{2}\right)^2 \left(\frac{T}{2} + r\right)^2 \right] \tag{B.5}$$

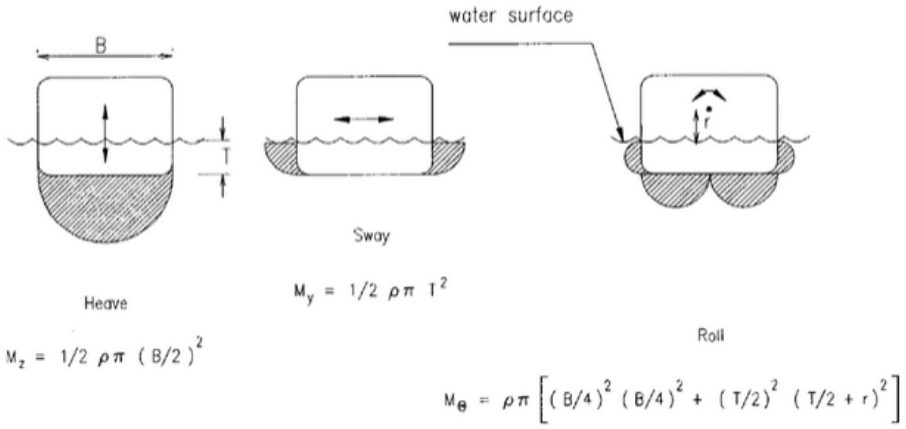


Figure B.1: Mass added by surrounding water accelerating with the platform (**barltrop_ndp_knovel_1998**)

C

IPE profiles

Table C.1: IPE profile dimensions

Profile	Depth (mm)
IPE80	80
IPE100	100
IPE120	120
IPE140	140
IPE160	160
IPE180	180
IPE200	200
IPE220	220
IPE240	240
IPE270	270
IPE300	300
IPE330	330
IPE360	360
IPE400	400
IPE450	450
IPE500	500
IPE550	550
IPE600	600

D

frame code square

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

profiles_df = pd.read_csv('IPE_profiles.csv', delimiter=';', skiprows=2)
profiles_df['Depth'] = pd.to_numeric(profiles_df['Depth'], errors='coerce')
profiles_df['Weight'] = pd.to_numeric(profiles_df['Weight'], errors='coerce')
profiles_df = profiles_df.dropna(subset=['Depth', 'Weight']).reset_index(drop=True)

def select_ipe_profile(required_height_mm):
    candidates = profiles_df[profiles_df['Depth'] >= required_height_mm]
    return candidates.iloc[0] if not candidates.empty else None

class stories_steel_simple:
    def __init__(self, length, width, height_story, number_of_portals, number_of_stories):
        self.length = length
        self.width = width
        self.height_story = height_story
        self.number_of_portals = number_of_portals
        self.number_of_stories = number_of_stories

        self.portal_y_positions = np.linspace(0, width, number_of_portals)
        self.x_positions_full = None  # wordt later gezet door get_lightest_configuration

    @classmethod
    def get_lightest_configuration(cls, length, width, height_story, number_of_portals,
        number_of_stories):
        best = None
        best_weight = float('inf')

        for cols in range(4, 13):
            try:
                inst = cls(length, width, height_story, number_of_portals, number_of_stories)
                inst.x_positions_full = np.linspace(0, length, cols)
                inst.amount_of_columns_row = cols  # optioneel

                frame_weight = inst._frame_weight_for_current_layout()
                studs_weight = inst._calculate_studs_weight_kg()
                total = frame_weight + studs_weight

                if total < best_weight:
                    best = inst
                    best_weight = total
            except Exception as e:
                continue  # deze configuratie is ongeldig

        if best is None:
            raise ValueError("Geen geldige configuratie gevonden")

```

```
        return best

    def _frame_weight_for_current_layout(self):
        beam_weight = 0.0
        column_weight = 0.0

        for y in self.portal_y_positions:
            recs = self.get_profile_recommendations(self.x_positions_full)
            if any(p == 'None' for _, p, _ in recs):
                raise ValueError("Geen geldig profiel voor portaal")

            beam_weight += sum(span * weight for span, _, weight in recs if span is not None)

            col_prof = select_ipe_profile(self.height_story * 1000 / 10)
            if col_prof is None:
                raise ValueError("Geen geldig kolomprofiel")
            col_weight_m = col_prof['Weight']
            col_count = len(self.x_positions_full)
            column_weight += col_weight_m * self.height_story * self.number_of_stories *
                col_count

        return beam_weight * self.number_of_stories + column_weight

    def get_spans(self, x_positions):
        return [x_positions[i + 1] - x_positions[i] for i in range(len(x_positions) - 1)]

    def get_profile_recommendations(self, x_positions):
        spans = self.get_spans(x_positions)
        return [
            (span, prof['Profile'], prof['Weight']) if prof is not None else (span, 'None',
                0)
            for span in spans
            if (prof := select_ipe_profile(span * 1000 / 15)) is not None
        ]

    def _calculate_studs_weight_kg(self):
        total = 0.0
        spacings = np.diff(self.portal_y_positions)
        for i, spacing in enumerate(spacings):
            n_studs = len(self.x_positions_full) * self.number_of_stories
            req_h = spacing * 1000 / 15
            prof = select_ipe_profile(req_h)
            if prof is None:
                raise ValueError("Geen geldig profiel voor studs")
            total += prof['Weight'] * spacing * n_studs
        return total

    def get_area_and_surface_per_storey(self):
        area = self.length * self.width
        surface = 2 * (self.length + self.width) * self.height_story
        return area, surface

    def calculate_total_force_kN(self):
        area, surf = self.get_area_and_surface_per_storey()
        net_area = area * self.number_of_stories
        facade = surf * self.number_of_stories
        roof = self.length * self.width

        floor = net_area * (2.5 + 4.35)
        facade_kN = facade * 1.47
        roof_kN = roof * (1.5 + 4.35)
        applied = floor + facade_kN + roof_kN

        frame = self._frame_weight_for_current_layout() * 9.81 / 1000
        studs = self._calculate_studs_weight_kg() * 9.81 / 1000

        return {
            'floor_force_kN': floor,
            'facade_force_kN': facade_kN,
            'roof_force_kN': roof_kN,
            'applied_force_kN': applied,
```

```
            'frame_weight_kN': frame,
            'studs_weight_kN': studs,
            'total_force_kN': applied + frame + studs
        }

    def plot_ground_floor_layout(self):
        plt.figure(figsize=(8, 8))
        for y in self.portal_y_positions:
            for x in self.x_positions_full:
                plt.scatter(x, y, color='steelblue')
        plt.title('Ground Floor Layout (No Core)')
        plt.xlabel('X position (m)')
        plt.ylabel('Y position (m)')
        plt.grid(True)
        plt.axis('equal')
        plt.show()

    def plot_column_positions(self):
        xs, zs = zip(*[
            (x, (story + 1) * self.height_story)
            for y in self.portal_y_positions
            for story in range(self.number_of_stories)
            for x in self.x_positions_full
        ])
        plt.figure(figsize=(10, 5))
        plt.scatter(xs, zs, c='steelblue', marker='s', label='Column position')
        plt.title('Side Elevation View of Full Portals (No Core)')
        plt.xlabel('X position (m)')
        plt.ylabel('Story height (m)')
        plt.grid(True)
        plt.show()

    def estimate_center_of_gravity_simple(self):
        return {
            'x': self.length / 2,
            'y': self.width / 2,
            'z': (self.height_story * self.number_of_stories) / 2
        }

    def debug_telling(self):
        total_cols = len(self.x_positions_full) * len(self.portal_y_positions)
        total_spans = (len(self.x_positions_full) - 1) * len(self.portal_y_positions) * self.
            number_of_stories

steel = stories_steel_simple.get_lightest_configuration(
    length=21,
    width=20,
    height_story=3,
    number_of_portals=5,
    number_of_stories=8
)
steel.debug_telling()
steel.plot_column_positions()
steel.plot_ground_floor_layout()
steel.estimate_center_of_gravity_simple()
```

E

Frame code for terrace

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from concurrent.futures import ThreadPoolExecutor

profiles_df = pd.read_csv('IPE_profiles.csv', delimiter=';', skiprows=2)
profiles_df['Depth'] = pd.to_numeric(profiles_df['Depth'], errors='coerce')
profiles_df['Weight'] = pd.to_numeric(profiles_df['Weight'], errors='coerce')
profiles_df = profiles_df.dropna(subset=['Depth', 'Weight']).reset_index(drop=True)

def select_ipe_profile(required_height_mm):
    candidates = profiles_df[profiles_df['Depth'] >= required_height_mm]
    return candidates.iloc[0] if not candidates.empty else None

class stories_steel_terraced_simple:
    def __init__(self, length, width, height_story, number_of_portals, number_of_stories,
        terrace_step=1.0):
        self.length = length
        self.width = width
        self.height_story = height_story
        self.number_of_portals = number_of_portals
        self.terrace_step = terrace_step
        self.minimum_top_length = 8.0

        self.portal_y_positions = np.linspace(0, width, number_of_portals)

        max_valid_stories = 0
        for s in range(1, number_of_stories + 1):
            try:
                _ = self._simulate_story_x_positions(s - 1)
                max_valid_stories = s
            except:
                break

        self.number_of_stories = max_valid_stories
        if self.number_of_stories < 4:
            raise ValueError("Aantal bruikbare verdiepingen is minder dan 4")

        self.column_config_df = self.explore_all_column_configs()
        valid = self.column_config_df.dropna(subset=['total_weight_kg'])
        best_row = valid.loc[valid['total_weight_kg'].idxmin()]
        self.amount_of_columns_row = int(best_row['cols'])
        self.x_positions_base = np.linspace(0, length, self.amount_of_columns_row)

    def _simulate_story_x_positions(self, story_index):
        if story_index == 0:
            return list(np.linspace(0, self.length, 9))
        prev = self._simulate_story_x_positions(story_index - 1)
        if len(prev) <= 2:
```

```
            raise ValueError("Te␣weinig␣kolommen")
        new_positions = []
        for i in range(len(prev)):
            if i == 0:
                shifted = prev[i] + self.terrace_step
                if shifted < prev[i + 1]:
                    new_positions.append(shifted)
            elif i == len(prev) - 1:
                shifted = prev[i] - self.terrace_step
                if shifted > prev[i - 1]:
                    new_positions.append(shifted)
            else:
                new_positions.append(prev[i])
        new_positions = sorted(set(new_positions))
        if len(new_positions) < 2 or (max(new_positions) - min(new_positions)) < self.
            minimum_top_length:
            raise ValueError("Ongeldige␣topverdieping")
        return new_positions

    def get_x_positions_story(self, story_index):
        if story_index == 0:
            return list(self.x_positions_base)
        prev = self.get_x_positions_story(story_index - 1)
        if len(prev) <= 2:
            return []
        new_positions = []
        for i in range(len(prev)):
            if i == 0:
                shifted = prev[i] + self.terrace_step
                if shifted < prev[i + 1]:
                    new_positions.append(shifted)
            elif i == len(prev) - 1:
                shifted = prev[i] - self.terrace_step
                if shifted > prev[i - 1]:
                    new_positions.append(shifted)
            else:
                new_positions.append(prev[i])
        new_positions = sorted(set(new_positions))
        if len(new_positions) < 2 or (max(new_positions) - min(new_positions)) < self.
            minimum_top_length:
            return []
        return new_positions

    def explore_all_column_configs(self, min_cols=4, max_cols=9):
        min_cols = int(min_cols)
        max_cols = int(max_cols)
        def compute_weight_for_cols(cols):
            self.amount_of_columns_row = cols
            self.x_positions_base = np.linspace(0, self.length, cols)
            try:
                total_weight = 0.0
                for story in range(self.number_of_stories):
                    weights = self.calculate_single_portal_weight(story)
                    total_weight += weights['total_story_weight_kg']
                return {'cols': cols, 'total_weight_kg': total_weight}

            except:
                return {'cols': cols, 'total_weight_kg': None}

        with ThreadPoolExecutor() as executor:
            min_cols = int(round(float(min_cols)))
            max_cols = int(round(float(max_cols)))
            results = list(executor.map(compute_weight_for_cols, range(min_cols, max_cols +
                1)))

            df = pd.DataFrame(results)
            df['total_weight_kg'] = pd.to_numeric(df['total_weight_kg'], errors='coerce')
            return df

    def _frame_weight_for_current_layout(self):
        total = 0.0
```

```
        for story in range(self.number_of_stories):
            x_pos = self.get_x_positions_story(story)
            if len(x_pos) < 2:
                continue
            spans = np.diff(x_pos)
            for span in spans:
                key = round(span * 1000 / 15, 1)
                prof = select_ipe_profile(key)
                total += span * prof['Weight'] * len(self.portal_y_positions)
        return total

    def _calculate_studs_weight_kg(self):
        total = 0.0
        spacings = np.diff(self.portal_y_positions)
        for story in range(self.number_of_stories):
            x_pos = self.get_x_positions_story(story)
            for spacing in spacings:
                key = round(spacing * 1000 / 15, 1)
                prof = select_ipe_profile(key)
                total += prof['Weight'] * spacing * len(x_pos)
        return total

    def calculate_total_force_kN(self):
        area_total = self.length * self.width
        surface_total = 2 * (self.length + self.width) * self.height_story * self.
            number_of_stories
        floor = area_total * (2.5 + 4.35)
        facade_kN = surface_total * 1.47
        roof_area = self.length * self.width
        roof_kN = roof_area * (1.5 + 4.35)
        frame = self._frame_weight_for_current_layout() * 9.81 / 1000
        studs = self._calculate_studs_weight_kg() * 9.81 / 1000
        applied = floor + facade_kN + roof_kN
        return {
            'floor_force_kN': floor,
            'facade_force_kN': facade_kN,
            'roof_force_kN': roof_kN,
            'applied_force_kN': applied,
            'frame_weight_kN': frame,
            'studs_weight_kN': studs,
            'total_force_kN': applied + frame + studs
        }

    def calculate_single_portal_weight(self, story):
        x_pos = self.get_x_positions_story(story)

        if len(x_pos) < 2:
            return {
                'story': story,
                'beam_weight_kg': 0,
                'column_weight_kg': 0,
                'stud_weight_kg': 0,
                'column_profile': None,
                'stud_profile': None,
                'profiles': []
            }

        spans = np.diff(x_pos)
        beam_weight = 0.0
        profile_list = []

        for span in spans:
            key = round(span * 1000 / 15, 1)
            prof = select_ipe_profile(key)
            if prof is not None:
                beam_weight += span * prof['Weight']
                profile_list.append((span, prof['Profile']))
            else:
                profile_list.append((span, 'None'))

        col_key = self.height_story * 1000 / 10
```

```
        col_prof = select_ipe_profile(col_key)
        col_weight_m = col_prof['Weight'] if col_prof is not None else 0.0
        column_weight = col_weight_m * self.height_story * len(x_pos)
        column_profile = col_prof['Profile'] if col_prof is not None else 'None'

        spacings = np.diff(self.portal_y_positions)
        stud_weight = 0.0
        stud_profile = None
        for spacing in spacings:
            stud_key = round(spacing * 1000 / 15, 1)
            stud_prof = select_ipe_profile(stud_key)
            if stud_prof is not None:
                stud_weight += stud_prof['Weight'] * spacing * len(x_pos)
                if stud_profile is None:
                    stud_profile = stud_prof['Profile']

        return {
            'story': story,
            'beam_weight_kg': beam_weight,
            'column_weight_kg': column_weight,
            'stud_weight_kg': stud_weight,
            'column_profile': column_profile,
            'stud_profile': stud_profile,
            'profiles': profile_list,
            'total_story_weight_kg': beam_weight + column_weight + stud_weight
        }

    def debug_telling(self):
        col_count = sum(len(self.get_x_positions_story(s)) for s in range(self.
            number_of_stories)) * len(self.portal_y_positions)
        span_count = sum((len(self.get_x_positions_story(s)) - 1) for s in range(self.
            number_of_stories)) * len(self.portal_y_positions)

    def plot_column_positions(self):
        xs, zs = [], []
        for s in range(self.number_of_stories):
            x_pos = self.get_x_positions_story(s)
            z = (s + 1) * self.height_story
            xs.extend(x for x in x_pos for _ in self.portal_y_positions)
            zs.extend(z for _ in x_pos for _ in self.portal_y_positions)
        plt.scatter(xs, zs, c='blue')
        plt.title('Kolompositie␣zijaanzicht')
        plt.xlabel('X␣(m)')
        plt.ylabel('Z␣(m)')
        plt.grid(True)
        plt.show()

    def plot_ground_floor_layout(self):
        for y in self.portal_y_positions:
            for x in self.get_x_positions_story(0):
                plt.scatter(x, y, color='blue')
        plt.title('Begane␣grond␣kolomlay-out')
        plt.xlabel('X␣(m)')
        plt.ylabel('Y␣(m)')
        plt.axis('equal')
        plt.grid(True)
        plt.show()

    def plot_3d_structure(self):
        from mpl_toolkits.mplot3d import Axes3D
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        for s in range(self.number_of_stories):
            x_pos = self.get_x_positions_story(s)
            z = (s + 1) * self.height_story
            for x in x_pos:
                for y in self.portal_y_positions:
                    ax.scatter(x, y, z, c='purple')
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')
```

```
256         plt.show()
257 
258     def estimate_center_of_gravity_weighted(self):
259         z = sum((i + 1) * self.height_story for i in range(self.number_of_stories)) / self.
                number_of_stories
260         return {'x': self.length / 2, 'y': self.width / 2, 'z': z}
261 steel = stories_steel_terraced_simple(length=26, width=20, height_story=3, number_of_portals
        =5, number_of_stories=8, terrace_step=1.0)
262 for s in range(steel.number_of_stories):
263 steel.debug_telling()
264 steel.plot_column_positions()
265 steel.plot_ground_floor_layout()
266 steel.plot_3d_structure()
```

F

Platform code

```python
import numpy as np
import math
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.art3d import Poly3DCollection

class FloatingPlatform:
    def __init__(self, L, W, submerged_depth, rib_width=0.3,
                 platform_density=2400, water_density=1000, force_module_W=None):
        self.L = L
        self.W = W
        self.submerged_depth = submerged_depth
        self.total_height = submerged_depth + 0.3
        self.above_water_height = self.total_height - submerged_depth
        self.platform_density = platform_density
        self.water_density = water_density
        self.rib_width = rib_width

        self.set_modular_dimensions(force_module_W)

        self.num_modules_L = math.ceil(self.L / 20)
        self.module_L = self.L / self.num_modules_L
        self.rib_positions_L = np.linspace(0, self.L, self.num_modules_L + 1)
        self.rib_count_L = len(self.rib_positions_L)

        self.rib_count_W = math.ceil(W / self.rib_spacing) + 2

        self.slab_thickness = math.ceil((self.rib_spacing / 24) * 100) / 100

    def set_modular_dimensions(self, force_module_W=None):
        if force_module_W:
            if self.W % force_module_W != 0:
                raise ValueError(f"Platform width {self.W} not divisible by {force_module_W}.
                    ")
            self.module_W = force_module_W
        else:
            feasible_widths = [w for w in range(4, 9) if self.W % w == 0]
            if not feasible_widths:
                raise ValueError(f"Platform width {self.W} m cannot be divided into 4-8 m
                    modules.")
            self.module_W = feasible_widths[0]

        self.num_modules_W = self.W // self.module_W
        self.rib_spacing = self.module_W

    def top_area(self):
        return self.L * self.W

    def volume(self):
        return self.top_area() * self.total_height
```

```
48
49      def submerged_volume(self):
50          return self.top_area() * self.submerged_depth
51
52      def slab_volume_top_bottom(self):
53          return self.top_area() * self.slab_thickness * 2
54
55      def structural_walls_volume(self):
56          rib_height = self.total_height - 2 * self.slab_thickness
57          ribs_L_vol = self.rib_count_L * self.rib_width * rib_height * self.W
58          ribs_W_vol = self.rib_count_W * self.rib_width * rib_height * self.L
59          perimeter_vol = 2 * (self.L + self.W) * rib_height * self.rib_width
60          return ribs_L_vol + ribs_W_vol + perimeter_vol
61
62      def concrete_volume(self):
63          return self.slab_volume_top_bottom() + self.structural_walls_volume()
64
65      def own_weight(self)-> float:
66          return self.platform_density * self.concrete_volume() * 9.81
67
68      def buoyant_force(self):
69          return self.water_density * 9.81 * self.submerged_volume()
70
71      def net_payload_capacity(self):
72          return self.buoyant_force() - self.own_weight()
73
74      def summary(self):
75
76      def plot_3d(self):
77          fig = plt.figure(figsize=(8, 6))
78          ax = fig.add_subplot(111, projection='3d')
79
80          verts = [
81              [[0, 0, 0], [self.L, 0, 0], [self.L, self.W, 0], [0, self.W, 0]],
82              [[0, 0, self.total_height], [self.L, 0, self.total_height], [self.L, self.W, self
                    .total_height], [0, self.W, self.total_height]],
83              [[0, 0, 0], [0, 0, self.total_height], [self.L, 0, self.total_height], [self.L,
                    0, 0]],
84              [[0, self.W, 0], [0, self.W, self.total_height], [self.L, self.W, self.
                    total_height], [self.L, self.W, 0]],
85              [[0, 0, 0], [0, self.W, 0], [0, self.W, self.total_height], [0, 0, self.
                    total_height]],
86              [[self.L, 0, 0], [self.L, self.W, 0], [self.L, self.W, self.total_height], [self.
                    L, 0, self.total_height]]
87          ]
88          ax.add_collection3d(Poly3DCollection(verts, facecolors='lightgray', linewidths=1,
                edgecolors='black', alpha=0.1))
89
90          for x in self.rib_positions_L:
91              verts_rib_L = [[[x, 0, self.slab_thickness], [x, self.W, self.slab_thickness],
92                              [x, self.W, self.total_height - self.slab_thickness], [x, 0, self
                                    .total_height - self.slab_thickness]]]
93              ax.add_collection3d(Poly3DCollection(verts_rib_L, facecolors='red', linewidths
                    =0.5, edgecolors='black', alpha=0.5))
94
95          rib_positions_W = np.linspace(0, self.W, self.rib_count_W)
96          for y in rib_positions_W:
97              verts_rib_W = [[[0, y, self.slab_thickness], [self.L, y, self.slab_thickness],
98                              [self.L, y, self.total_height - self.slab_thickness], [0, y, self
                                    .total_height - self.slab_thickness]]]
99              ax.add_collection3d(Poly3DCollection(verts_rib_W, facecolors='blue', linewidths
                    =0.5, edgecolors='black', alpha=0.5))
100
101         ax.set_xlabel('Length␣(m)')
102         ax.set_ylabel('Width␣(m)')
103         ax.set_zlabel('Height␣(m)')
104         ax.set_xlim(0, self.L)
105         ax.set_ylim(0, self.W)
106         ax.set_zlim(0, self.total_height)
107         ax.set_title('3D␣Structural␣Visualization')
108         plt.show()
```

```
109
110     def center_of_buoyancy_and_keel(self):
111         z_buoyancy = -(self.total_height - (self.submerged_depth / 2))
112         z_keel = -self.total_height
113         return {
114             'z_buoyancy': z_buoyancy,
115             'z_keel': z_keel
116         }
117
118 platform = FloatingPlatform(L=40, W=60, submerged_depth=6)
119 platform.summary()
120 platform.plot_3d()
121 platform.center_of_buoyancy_and_keel()
```

G

System evaluator

The first evaluator is for the rectangular building and the second has some slight adjustments for the terraced one.

```
class FloatingSystemEvaluator:
    def __init__(self, platform_class, building_class):
        self.platform_class = platform_class
        self.building_class = building_class

    def evaluate(self, platform_L, platform_W, submerged_depth,
                 building_length, building_width, number_of_stories, number_of_portals):

        base_result = {
            'platform_L': platform_L,
            'platform_W': platform_W,
            'submerged_depth': submerged_depth,
            'building_length': building_length,
            'building_width': building_width,
            'number_of_stories': number_of_stories,
            'number_of_portals': number_of_portals,
            'building_weight_N': None,
            'net_payload_capacity_N': None,
            'own_weight_pontoon': None,
            'real_submerged_depth': None,
            'rib_spacing_used_L': None,
            'rib_spacing_used_W': None,
            'status': 'Unknown',
            'pontoon_width': None,
            'num_pontoons': None,
            'floats': False,
            'KB': None,
            'KG': None,
            'BM': None,
            'GM': None,
            'stable': None
        }

        try:
            if building_length > platform_L or building_width > platform_W:
                base_result['status'] = 'Building␣does␣not␣fit␣on␣platform'
                return False, base_result

            if platform_W < building_width:
                base_result['status'] = 'Platform␣is␣too␣narrow␣to␣support␣portal␣columns'
                return False, base_result

            rib_spacing_L = platform_L / (number_of_portals - 1)
            base_result['rib_spacing_used_L'] = rib_spacing_L

```

```
            platform = self.platform_class(L=platform_L, W=platform_W, submerged_depth=
                submerged_depth)
            rib_spacing_W = platform.rib_spacing

            base_result['rib_spacing_used_W'] = rib_spacing_W
            base_result['pontoon_width'] = platform.module_W
            base_result['num_pontoons'] = platform.num_modules_W
            base_result['own_weight_pontoon'] = platform.own_weight()

            aligned = check_portal_rib_alignment(
                W_platform=platform.W,
                W_building=building_width,
                rib_count=platform.rib_count_W,
                number_of_portals=number_of_portals,
                tolerance=0.1
            )

            building = self.building_class.get_lightest_configuration(
                length=building_length,
                width=building_width,
                height_story=3,
                number_of_portals=number_of_portals,
                number_of_stories=number_of_stories
            )

            forces = building.calculate_total_force_kN()
            base_result.update(forces)

            building_force_N = forces['total_force_kN'] * 1000
            base_result['building_weight_N'] = building_force_N

            net_payload = platform.net_payload_capacity()
            real_sub_d = (building_force_N + platform.own_weight()) / (platform.water_density
                 * 9.81 * platform_L * platform_W)

            base_result['net_payload_capacity_N'] = net_payload
            base_result['real_submerged_depth'] = real_sub_d

            floats = net_payload >= building_force_N and real_sub_d <= submerged_depth
            base_result['floats'] = floats
            base_result['status'] = 'Floats' if floats else 'Sinks'

            cg = building.estimate_center_of_gravity_simple()
            KB = real_sub_d / 2
            KG = cg['z'] + abs(platform.total_height)
            I_x = (platform_W ** 3) * platform_L / 12
            I_y = (platform_L ** 3) * platform_W / 12
            V = platform.submerged_volume()
            BM_x = I_x / V
            BM_y = I_y / V
            GM_x = BM_x + KB - KG
            GM_y = BM_y + KB - KG

            base_result['KB'] = KB
            base_result['KG'] = KG
            base_result['BM_x'] = BM_x
            base_result['BM_y'] = BM_y
            base_result['GM_x'] = GM_x
            base_result['GM_y'] = GM_y
            base_result['stable'] = GM_x > 0.25 and GM_y > 0.25

            return floats, base_result

        except Exception as e:
            base_result['status'] = f'Exception: {str(e)}'
            return False, base_result
import numpy as np
import pandas as pd

def check_portal_rib_alignment(W_platform, W_building, rib_count, number_of_portals,
    tolerance=0.1):
```

```
    rib_positions = np.linspace(0, W_platform, rib_count)
    portal_positions = np.linspace(0, W_building, number_of_portals)
    offset = (W_platform - W_building) / 2
    shifted = portal_positions + offset

    return all(
        any(abs(p - r) < tolerance for r in rib_positions)
        for p in shifted
    )

def generate_valid_alignment_configs(platform_W_values, building_W_values,
    number_of_portals_values,
                                     L_fixed=40, submerged_depth_fixed=2.5):
    valid_rows = []
    for W in platform_W_values:
        for b_len in building_W_values:
            for n_portals in number_of_portals_values:
                for module_width in range(4, 9):
                    if W % module_width != 0:
                        continue
                    try:
                        platform = FloatingPlatform(L=L_fixed, W=W, submerged_depth=
                            submerged_depth_fixed,
                                                    force_module_W=module_width)
                        aligned = check_portal_rib_alignment(
                            W_platform=W,
                            W_building=b_len,
                            rib_count=platform.rib_count_W,
                            number_of_portals=n_portals,
                            tolerance=0.1
                        )
                        if aligned:
                            valid_rows.append({
                                'platform_W': W,
                                'building_on_platform_width': b_len,
                                'number_of_portals': n_portals,
                                'module_width_used': module_width,
                                'rib_count': platform.rib_count_W,
                                'pontoon_width': platform.module_W,
                                'num_pontoons': platform.num_modules_W
                            })
                    except Exception:
                        continue
    return pd.DataFrame(valid_rows)

import time
def optimized_batch_evaluate(evaluator,
                             alignment_df,
                             platform_L_values,
                             submerged_depth_values,
                             building_length_values,
                             story_numbers,
                             batch_size=10):
    floating_results = []
    rejected_results = []

    total_batches = int(np.ceil(len(alignment_df) / batch_size))
    successful_configs = set()

    for batch_index in range(total_batches):
        start_time = time.time()

        batch_df = alignment_df.iloc[batch_index * batch_size : (batch_index + 1) *
            batch_size]

        for _, row in batch_df.iterrows():
            W = row['platform_W']
            b_len = row['building_on_platform_width']
            n_portals = row['number_of_portals']
            module_W = row['module_width_used']

```

```
            for b_width in building_length_values:
                for n_stories in story_numbers:
                    config_key = (b_len, b_width, n_stories, n_portals)

                    if config_key in successful_configs:
                        continue

                    found = False

                    for L in platform_L_values:
                        for sub_d in submerged_depth_values:
                            try:
                                floats, result = evaluator.evaluate(
                                    platform_L=L,
                                    platform_W=W,
                                    submerged_depth=sub_d,
                                    building_length=b_len,
                                    building_width=b_width,
                                    number_of_stories=n_stories,
                                    number_of_portals=n_portals
                                )

                                result['floats'] = floats
                                result['pontoon_width'] = module_W
                                result['num_pontoons'] = W // module_W

                                for key in [
                                    'platform_L', 'platform_W', 'submerged_depth',
                                    'building_length', 'building_width', 'number_of_stories',
                                        'number_of_portals',
                                    'building_weight_N', 'own_weight_pontoon', '
                                        net_payload_capacity_N', 'real_submerged_depth',
                                    'rib_spacing_used_L', 'rib_spacing_used_W', 'status',
                                    'pontoon_width', 'num_pontoons', 'floats',
                                    'KB', 'KG', 'BM_x', 'BM_x', 'GM_x', 'GM_y', 'stable'
                                ]:
                                    if key not in result:
                                        result[key] = None

                                if floats:
                                    floating_results.append(result)
                                    successful_configs.add(config_key)
                                    found = True
                                    break
                                else:
                                    rejected_results.append(result)

                            except Exception as e:
                                rejected_results.append({
                                    'platform_L': L,
                                    'platform_W': W,
                                    'submerged_depth': sub_d,
                                    'building_length': b_len,
                                    'building_width': b_width,
                                    'number_of_stories': n_stories,
                                    'number_of_portals': n_portals,
                                    'status': f'Exception: {str(e)}',
                                    'floats': False
                                })

                        if found:
                            break

        end_time = time.time()
        duration = end_time - start_time

    return pd.DataFrame(floating_results), pd.DataFrame(rejected_results)

alignment_df = generate_valid_alignment_configs(
    platform_W_values=np.arange(35, 76, 5),
    building_W_values=np.arange(15, 30),
```

```
    number_of_portals_values=np.arange(3, 9)
)

evaluator = FloatingSystemEvaluator(FloatingPlatform, stories_steel_simple)

df_floating, df_rejected = optimized_batch_evaluate(
    evaluator=evaluator,
    alignment_df=alignment_df,
    platform_L_values=np.arange(35, 76, 5),
    submerged_depth_values=[3, 4, 5, 6],
    building_length_values=np.arange(15, 30),
    story_numbers=np.arange(6, 13)
)

if not df_floating.empty:
else:

reasons = df_rejected['status'].value_counts(normalize=True) * 100

df_floating.to_excel("Excel_sheets/Square_solution.xlsx", index=False)
df_rejected.to_excel("Excel_sheets/Square_rejected.xlsx", index=False)

class FloatingSystemEvaluator:
    def __init__(self, platform_class, building_class):
        self.platform_class = platform_class
        self.building_class = building_class

    def evaluate(self, platform_L, platform_W, submerged_depth,
                 building_length, building_width, number_of_stories, number_of_portals,
                     terrace_step):

        base_result = {
            'platform_L': platform_L,
            'platform_W': platform_W,
            'submerged_depth': submerged_depth,
            'building_length': building_length,
            'building_width': building_width,
            'number_of_stories': number_of_stories,
            'number_of_portals': number_of_portals,
            'building_weight_N': None,
            'own_weight_pontoon': None,
            'net_payload_capacity_N': None,
            'real_submerged_depth': None,
            'rib_spacing_used_L': None,
            'rib_spacing_used_W': None,
            'status': 'Unknown',
            'pontoon_width': None,
            'num_pontoons': None,
            'floats': False,
            'KB': None,
            'KG': None,
            'BM': None,
            'GM': None,
            'stable': None,
            'terrace_step': None
        }

        try:
            if building_length > platform_L or building_width > platform_W:
                base_result['status'] = 'Building␣does␣not␣fit␣on␣platform'
                return False, base_result

            if platform_W < building_width:
                base_result['status'] = 'Platform␣is␣too␣narrow␣to␣support␣portal␣columns'
                return False, base_result

            platform = self.platform_class(L=platform_L, W=platform_W, submerged_depth=
                submerged_depth)
            base_result['rib_spacing_used_W'] = platform.rib_spacing
            base_result['pontoon_width'] = platform.module_W
            base_result['num_pontoons'] = platform.num_modules_W
```

```
            base_result['own_weight_pontoon'] = platform.own_weight()

            try:
                building = self.building_class(
                    length=building_length,
                    width=building_width,
                    height_story=3,
                    number_of_portals=number_of_portals,
                    number_of_stories=number_of_stories,
                    terrace_step=terrace_step
                )
            except ValueError as ve:
                if "bruikbare verdiepingen" in str(ve):
                    base_result['status'] = f"Ongeldige verdiepingconfiguratie ({ve})"
                    return False, base_result
                else:
                    raise ve  # andere fouten gooi je gewoon door

            forces = building.calculate_total_force_kN()
            base_result.update(forces)

            base_result['terrace_step'] = building.terrace_step
            building_force_N = forces['total_force_kN'] * 1000
            base_result['building_weight_N'] = building_force_N

            net_payload = platform.net_payload_capacity()
            real_sub_d = (building_force_N + platform.own_weight()) / (platform.water_density
                 * 9.81 * platform_L * platform_W)

            base_result['net_payload_capacity_N'] = net_payload
            base_result['real_submerged_depth'] = real_sub_d

            floats = net_payload >= building_force_N and real_sub_d <= submerged_depth
            base_result['floats'] = floats
            base_result['status'] = 'Floats' if floats else 'Sinks'
            base_result['number_of_stories'] = building.number_of_stories

            cg = building.estimate_center_of_gravity_weighted()
            base_result['terrace_step'] = terrace_step

            KB = real_sub_d / 2
            KG = cg['z'] + abs(platform.total_height)
            I_x = (platform_W ** 3) * platform_L / 12
            I_y = (platform_L ** 3) * platform_W / 12
            V = platform.submerged_volume()
            BM_x = I_x / V
            BM_y = I_y / V
            GM_x = BM_x + KB - KG
            GM_y = BM_y + KB - KG

            base_result['KB'] = KB
            base_result['KG'] = KG
            base_result['BM_x'] = BM_x
            base_result['BM_y'] = BM_y
            base_result['GM_x'] = GM_x
            base_result['GM_y'] = GM_y
            base_result['stable'] = GM_x > 0.25 and GM_y > 0.25

            return floats, base_result

        except Exception as e:
            base_result['status'] = f'Exception: {str(e)}'
            return False, base_result

import numpy as np
import pandas as pd
import time
from tqdm import tqdm  # voortgangsbalk

def check_portal_rib_alignment(W_platform, W_building, rib_count, number_of_portals,
    tolerance=0.1):
```

```
    rib_positions = np.linspace(0, W_platform, rib_count)
    portal_positions = np.linspace(0, W_building, number_of_portals)
    offset = (W_platform - W_building) / 2
    shifted = portal_positions + offset

    return all(
        any(abs(p - r) < tolerance for r in rib_positions)
        for p in shifted
    )

def generate_valid_alignment_configs(platform_W_values, building_W_values,
    number_of_portals_values,
                                     L_fixed=40, submerged_depth_fixed=2.5):
    valid_rows = []
    for W in platform_W_values:
        for b_width in building_W_values:
            for n_portals in number_of_portals_values:
                for module_width in range(4, 9):
                    if W % module_width != 0:
                        continue

                    try:
                        platform = FloatingPlatform(L=L_fixed, W=W, submerged_depth=
                            submerged_depth_fixed,
                                                    force_module_W=module_width)
                        aligned = check_portal_rib_alignment(
                            W_platform=W,
                            W_building=b_width,
                            rib_count=platform.rib_count_W,
                            number_of_portals=n_portals,
                            tolerance=0.1
                        )
                        if aligned:
                            valid_rows.append({
                                'platform_W': W,
                                'building_on_platform_width': b_width,
                                'number_of_portals': n_portals,
                                'module_width_used': module_width,
                                'rib_count': platform.rib_count_W,
                                'pontoon_width': platform.module_W,
                                'num_pontoons': platform.num_modules_W
                            })
                    except Exception:
                        continue
    return pd.DataFrame(valid_rows)

def optimized_batch_evaluate(evaluator,
                             alignment_df,
                             platform_L_values,
                             submerged_depth_values,
                             building_length_values,
                             terrace_step_values,
                             number_of_stories_values):
    floating_results = []
    rejected_results = []
    successful_configs = set()

    for _, row in alignment_df.iterrows():
        W = row['platform_W']
        b_width = row['building_on_platform_width']
        n_portals = row['number_of_portals']
        module_W = row['module_width_used']

        for story_count in number_of_stories_values:
            for b_len in building_length_values:
                for terrace in terrace_step_values:
                    config_key = (b_len, b_width, story_count, n_portals, terrace)
                    if config_key in successful_configs:
                        continue
                    found_floating = False

```

```
                    for L in platform_L_values:
                        for sub_d in submerged_depth_values:
                            try:
                                floats, result = evaluator.evaluate(
                                    platform_L=L,
                                    platform_W=W,
                                    submerged_depth=sub_d,
                                    building_length=b_len,
                                    building_width=b_width,
                                    number_of_stories=story_count,
                                    number_of_portals=n_portals,
                                    terrace_step=terrace
                                )

                                result['floats'] = floats
                                result['pontoon_width'] = module_W
                                result['num_pontoons'] = int(W // module_W)

                                for key in ['platform_L', 'platform_W', 'building_length', '
                                    building_width',
                                            'submerged_depth', 'number_of_stories', '
                                                number_of_portals',
                                            'building_weight_N', 'own_weight_pontoon', '
                                                net_payload_capacity_N', '
                                                real_submerged_depth',
                                            'rib_spacing_used_L', 'rib_spacing_used_W', '
                                                status',
                                            'GM_x', 'GM_y', 'KB', 'KG', 'BM_x', 'BM_y', '
                                                stable', 'terrace_step']:
                                    if key not in result:
                                        result[key] = None

                                if floats:
                                    floating_results.append(result)
                                    successful_configs.add(config_key)
                                    found_floating = True
                                    break
                                else:
                                    rejected_results.append(result)

                            except Exception as e:
                                rejected_results.append({
                                    'platform_L': L,
                                    'platform_W': W,
                                    'submerged_depth': sub_d,
                                    'building_length': b_len,
                                    'building_width': b_width,
                                    'number_of_stories': story_count,
                                    'number_of_portals': n_portals,
                                    'status': f'Exception: {str(e)}',
                                    'floats': False,
                                    'terrace_step': terrace
                                })

                        if found_floating:
                            break

    return pd.DataFrame(floating_results), pd.DataFrame(rejected_results)

alignment_df = generate_valid_alignment_configs(
    platform_W_values=np.arange(35, 76, 5),
    building_W_values=np.arange(15, 30),
    number_of_portals_values=np.arange(3, 9)
)

evaluator = FloatingSystemEvaluator(FloatingPlatform, stories_steel_terraced_simple)

chunk_size = 10
results_floating = []
results_rejected = []

```

```
total_start = time.time()

for i in tqdm(range(0, len(alignment_df), chunk_size), desc=" Batches"):
    chunk_start = time.time()

    chunk = alignment_df.iloc[i:i+chunk_size]

    df_f, df_r = optimized_batch_evaluate(
        evaluator=evaluator,
        alignment_df=chunk,
        platform_L_values=np.arange(35, 76, 5),
        submerged_depth_values=[3, 4, 5, 6],
        building_length_values=np.arange(15, 30),
        terrace_step_values=[1, 1.5, 2],
        number_of_stories_values=np.arange(6, 13)
    )

    results_floating.append(df_f)
    results_rejected.append(df_r)

    elapsed = time.time() - chunk_start

total_elapsed = time.time() - total_start

df_floating = pd.concat(results_floating, ignore_index=True)
df_rejected = pd.concat(results_rejected, ignore_index=True)

if not df_floating.empty:
else:

reasons = df_rejected['status'].value_counts(normalize=True) * 100

import os
base_dir = os.path.abspath(os.path.join(os.getcwd(), ".."))  # ga één niveau omhoog
excel_path = os.path.join(base_dir, "Excel_sheets", "terraced_solution.xlsx")

df_floating.to_excel(excel_path, index=False)
excel_path = os.path.join(base_dir, "Excel_sheets", "terraced_rejected.xlsx")
df_rejected.to_excel(excel_path, index=False)
```

H

Moment and heel angle code

```python
import numpy as np
import pandas as pd

rho = 1_000.0
g   = 9.81

scenarios = {
    "storm": {"Hs": 1.05, "q_wind": 1.09e3},   # kN/m² → N/m²
    "daily": {"Hs": 0.21, "q_wind": 0.46e3},
}
#Square
def _stability_for_row(row: pd.Series, Hs: float, q_wind: float):

    L, W   = row["platform_L"], row["platform_W"]
    d      = row.get("real_submerged_depth", row["submerged_depth"])
    H_bld  = row["number_of_stories"] * 3.0
    GM_x   = row["GM_x"]
    GM_y   = row["GM_y"]

    m_ballast = row["mass_ballast_kg"]
    m_struct  = row["total_force_kN"] * 1_000 / g
    Delta     = (m_ballast + m_struct) * g

    A_wave   = 0.5 * rho * g * Hs
    M_wave_x = (A_wave * W**2 / (2*np.pi)) * L
    M_wave_y = (A_wave * L**2 / (2*np.pi)) * W

    lever    = 0.5 * H_bld + d
    F_wind_x = q_wind * H_bld * L
    F_wind_y = q_wind * H_bld * W
    M_wind_x = F_wind_x * lever
    M_wind_y = F_wind_y * lever

    Mx = M_wave_x + M_wind_x
    My = M_wave_y + M_wind_y

    def angle_and_disp(M, GM):
        if (GM <= 0) or (Delta == 0):
            return np.nan, np.nan
        phi = M / (Delta * GM)
        deg = np.degrees(phi)
        if (deg > 20) or (deg <= 0):
            return np.nan, np.nan
        disp = H_bld * np.tan(phi)
        return deg, disp

    ang_x, disp_x = angle_and_disp(Mx, GM_x)
    ang_y, disp_y = angle_and_disp(My, GM_y)

```

```
    if np.isfinite(ang_x) and (not np.isfinite(ang_y) or ang_x >= ang_y):
        axis_n, M_n, ang_n, disp_n = "x", Mx, ang_x, disp_x
    elif np.isfinite(ang_y):
        axis_n, M_n, ang_n, disp_n = "y", My, ang_y, disp_y
    else:
        axis_n, M_n, ang_n, disp_n = np.nan, np.nan, np.nan, np.nan

    return pd.Series({
        "M_x_kNm"        : Mx / 1_000,
        "M_y_kNm"        : My / 1_000,
        "angle_x_deg"    : ang_x,
        "angle_y_deg"    : ang_y,
        "disp_x_m"       : disp_x,
        "disp_y_m"       : disp_y,
        "M_norm_kNm"     : M_n / 1_000 if np.isfinite(M_n) else np.nan,
        "axis_norm"      : axis_n,
        "angle_norm_deg" : ang_n,
        "disp_norm_m"    : disp_n,
    })

def add_stability_cols(df: pd.DataFrame) -> pd.DataFrame:

    df = df.copy()
    for tag, p in scenarios.items():
        res = df.apply(_stability_for_row,
                       axis=1,
                       Hs=p["Hs"],
                       q_wind=p["q_wind"])
        df = df.join(res.add_suffix(f"_{tag}"))
    return df

df_terraced_stab = add_stability_cols(df_terraced)

df_square_stab.to_excel("Square_solution_heel_angle.xlsx", index=False)


#Terraced
def _stability_for_row(row: pd.Series, Hs: float, q_wind: float):

    L_p, W_p = row["platform_L"], row["platform_W"]
    d        = row.get("real_submerged_depth", row["submerged_depth"])

    n_storey   = int(row["number_of_stories"])
    H_bld      = n_storey * 3.0                                  # 3 m/laag
    L_bld      = row.get("building_length", L_p)                 # onderste gevel
    terrace    = row.get("terrace_step", 0.0)                    # m
    L_top      = max(L_bld - n_storey * terrace, 8.0)            #   8 m regel
    W_bld      = row.get("building_width", W_p)                  # voor Y-gevel

    GM_x, GM_y = row["GM_x"], row["GM_y"]
    m_ballast  = row["mass_ballast_kg"]
    m_struct   = row["total_force_kN"] * 1_000 / g               # kg
    Delta      = (m_ballast + m_struct) * g                      # N

    A_wave   = 0.5 * rho * g * Hs
    M_wave_x = (A_wave * W_p**2 / (2*np.pi)) * L_p
    M_wave_y = (A_wave * L_p**2 / (2*np.pi)) * W_p

    A_front_x = 0.5 * (L_bld + L_top) * H_bld                    # m²
    F_wind_x  = q_wind * A_front_x                               # N

    h_c_x = H_bld * (2 * L_bld + L_top) / (3 * (L_bld + L_top))  # m
    lever_x = h_c_x + d                                          # m
    M_wind_x = F_wind_x * lever_x                                # N·m

    F_wind_y = q_wind * W_bld * H_bld                            # N
    lever_y  = 0.5 * H_bld + d
    M_wind_y = F_wind_y * lever_y                                # N·m

    Mx = M_wave_x + M_wind_x
    My = M_wave_y + M_wind_y
```

```
121
122     def angle_and_disp(M, GM):
123         if (GM <= 0) or (Delta == 0):
124             return np.nan, np.nan
125         phi = M / (Delta * GM)          # rad
126         deg = np.degrees(phi)
127         if (deg > 20) or (deg <= 0):    # buiten scope
128             return np.nan, np.nan
129         disp = H_bld * np.tan(phi)
130         return deg, disp
131
132     ang_x, disp_x = angle_and_disp(Mx, GM_x)
133     ang_y, disp_y = angle_and_disp(My, GM_y)
134
135     if np.isfinite(ang_x) and (not np.isfinite(ang_y) or ang_x >= ang_y):
136         axis_n, M_n, ang_n, disp_n = "x", Mx, ang_x, disp_x
137     elif np.isfinite(ang_y):
138         axis_n, M_n, ang_n, disp_n = "y", My, ang_y, disp_y
139     else:
140         axis_n, M_n, ang_n, disp_n = np.nan, np.nan, np.nan, np.nan
141
142     return pd.Series({
143         "M_x_kNm"        : Mx / 1_000,
144         "M_y_kNm"        : My / 1_000,
145         "angle_x_deg"    : ang_x,
146         "angle_y_deg"    : ang_y,
147         "disp_x_m"       : disp_x,
148         "disp_y_m"       : disp_y,
149         "M_norm_kNm"     : M_n / 1_000 if np.isfinite(M_n) else np.nan,
150         "axis_norm"      : axis_n,
151         "angle_norm_deg" : ang_n,
152         "disp_norm_m"    : disp_n,
153     })
154
155 df_terraced_stab = add_stability_cols(df_terraced)
156
157 df_terraced_stab.to_excel("terraced_solution_heel_angle.xlsx", index=False)
158
159 col_daily = "angle_norm_deg_daily"
160 col_storm = "angle_norm_deg_storm"
161
162 n_sq_before = len(df_square_stab)
163 cond_sq = (df_square_stab[col_storm] < 10) & (df_square_stab[col_daily] < 3)
164 df_square_ok = df_square_stab.loc[cond_sq].copy()
165 n_sq_after = len(df_square_ok)
166
167 n_te_before = len(df_terraced_stab)
168 cond_te = (df_terraced_stab[col_storm] < 10) & (df_terraced_stab[col_daily] < 3)
169 df_terraced_ok = df_terraced_stab.loc[cond_te].copy()
170 n_te_after = len(df_terraced_ok)
171
172 df_square_ok.to_excel("Square_solution_filtered.xlsx", index=False)
173 df_terraced_ok.to_excel("Terraced_solution_filtered.xlsx", index=False)
```

I

Natural frequencies code

```
import pandas as pd
import os

os.chdir(r"C:\Users\arend\Documents\Civiel 2024-2025\BEP\Excel_sheets")

df_square = pd.read_excel("Square_solution_filtered.xlsx")
df_terraced = pd.read_excel("Terraced_solution_filtered.xlsx")

def compute_ballast_mass(df, *, rho: float = 1000.0):
    m_added = []
    for _, row in df.iterrows():
        d_unused = max(0.0, row.get('submerged_depth', row['real_submerged_depth']) - row['
            real_submerged_depth'])
        wp = row['platform_L'] * row['platform_W']
        m_added.append(rho * wp * d_unused)
    df_out = df.copy()
    df_out['mass_ballast_kg'] = m_added
    return df_out

df_square_ballast = compute_ballast_mass(df_square.copy())
df_terraced_ballast = compute_ballast_mass(df_terraced.copy())

import numpy as np

def compute_natural_frequencies(df):
    rho = 1000  # kg/m^3,
    g = 9.81    # m/s^2,
    zeta = 0.1  # d

    f_heave = []
    f_surge = []
    f_sway = []
    f_roll = []
    f_pitch = []

    for _, row in df.iterrows():
        L = row['platform_L']
        W = row['platform_W']
        T = row['real_submerged_depth']
        m_ballast = row['mass_ballast_kg']
        m_struct = row['total_force_kN'] * 1000 / g
        m_total = m_struct + m_ballast
        H = row['number_of_stories'] * 3
        wp = L * W
        k = rho * g * wp

        B = min(L, W)
        C = max(L, W)

```

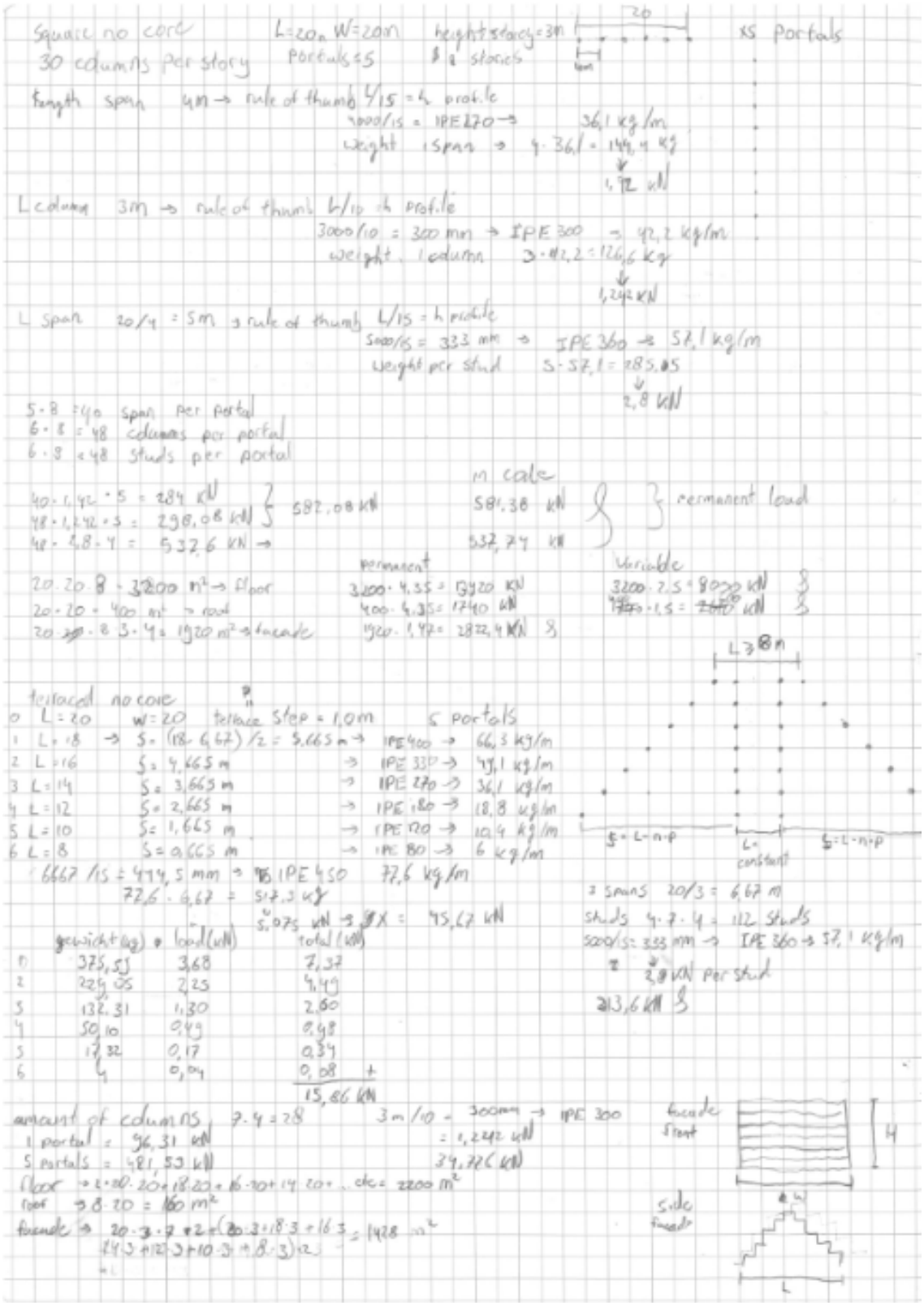
```
        Mz = 0.5 * rho * np.pi * (B / 2) ** 2 * C
        k_heave = rho * g * wp
        omega0_heave = np.sqrt(k / (m_total + Mz))
        omegae_heave = omega0_heave * np.sqrt(1 - zeta ** 2)
        f_heave.append(omegae_heave / (2 * np.pi))

        Mx = 0.5 * rho * np.pi * T**2 * W
        omega0_surge = np.sqrt(k / (m_total + Mx))
        omegae_surge = omega0_surge * np.sqrt(1 - zeta**2)
        f_surge.append(omegae_surge / (2 * np.pi))

        My = 0.5 * rho * np.pi * T**2 * L
        omega0_sway = np.sqrt(k / (m_total + My))
        omegae_sway = omega0_sway * np.sqrt(1 - zeta**2)
        f_sway.append(omegae_sway / (2 * np.pi))

        r_roll = T / 2
        r_gyr_roll   = W / 4
        m_bld  = m_total
        z_c    = H / 2
        I_roll = m_bld * 1/12 * (W ** 2 + H ** 2) + m_bld * z_c ** 2
        Mtheta = rho * np.pi * ((W/4)**2 * (W/4)**2 + (T/2)**2 * (T/2 + r_roll)**2) * L
        denom  = (I_roll + Mtheta) / r_gyr_roll**2
        omega_roll = np.sqrt(k / denom)
        f_roll.append(omega_roll / (2 * np.pi))

        r_pitch = T / 2
        r_gyr_pitch  = L / 4
        I_pitch = m_bld * 1/12 * (L ** 2 + H ** 2) + m_bld * z_c ** 2
        Mtheta = rho * np.pi * ((L/4)**2 * (L/4)**2 + (T/2)**2 * (T/2 + r_pitch)**2) * W
        denom   = (I_pitch + Mtheta) / r_gyr_pitch**2
        omega_pitch = np.sqrt(k / denom)
        f_pitch.append(omega_pitch / (2 * np.pi))

    df['freq_heave_Hz'] = f_heave
    df['freq_sway_Hz'] = f_sway
    df['freq_surge_Hz'] = f_surge
    df['freq_roll_Hz'] = f_roll
    df['freq_pitch_Hz'] = f_pitch

    return df


def compute_natural_frequencies_terraced(df):
    rho = 1000  # kg/m^3
    g = 9.81    # m/s^2
    zeta = 0.1  # demping

    f_heave = []
    f_surge = []
    f_sway = []
    f_roll = []
    f_pitch = []

    for _, row in df.iterrows():
        L_b = row['platform_L']
        W = row['platform_W']
        T = row['real_submerged_depth']
        m_ballast = row['mass_ballast_kg']
        m_struct = row['total_force_kN'] * 1000 / g
        m_tot = m_struct + m_ballast
        n = row['number_of_stories']
        H_b = n * 3
        step = row['terrace_step']
        L_t = max(8.0, L_b - n * step)

        wp = L_b * W
        B = min(L_b, W)
        C = max(L_b, W)
        k = rho * g * wp

```

```
        Mz = 0.5 * rho * np.pi * (B / 2)**2 * C
        omega0_heave = np.sqrt(k / (m_tot + Mz))
        omegae_heave = omega0_heave * np.sqrt(1 - zeta**2)
        f_heave.append(omegae_heave / (2 * np.pi))

        Mx = 0.5 * rho * np.pi * T**2 * W
        omega0_surge = np.sqrt(k / (m_tot + Mx))
        omegae_surge = omega0_surge * np.sqrt(1 - zeta**2)
        f_surge.append(omegae_surge / (2 * np.pi))

        My = 0.5 * rho * np.pi * T**2 * L_b
        omega0_sway = np.sqrt(k / (m_tot + My))
        omegae_sway = omega0_sway * np.sqrt(1 - zeta**2)
        f_sway.append(omegae_sway / (2 * np.pi))

        r_roll = T / 2
        r_gyr_roll = W / 4
        z_c_roll = H_b / 2
        I_roll = m_tot * (1/12 * (W**2 + H_b**2) + z_c_roll**2)
        Mtheta = rho * np.pi * ((W/4)**2 * (W/4)**2 + (T/2)**2 * (T/2 + r_roll)**2) * L_b
        denom = (I_roll + Mtheta) / r_gyr_roll**2
        omega_roll = np.sqrt(k / denom)
        f_roll.append(omega_roll / (2 * np.pi))

        r_pitch = T / 2
        r_gyr_pitch = L_b / 4
        I_pitch_cent = m_tot * H_b * (L_b + L_t) * (L_t**2 + 7 * L_b**2) / 48
        z_c_pitch = H_b * (2 * L_b + L_t) / (3 * (L_b + L_t))
        I_pitch = I_pitch_cent + m_tot * z_c_pitch**2
        Mtheta = rho * np.pi * ((L_b/4)**2 * (L_b/4)**2 + (T/2)**2 * (T/2 + r_pitch)**2) * W
        denom = (I_pitch + Mtheta) / r_gyr_pitch**2
        omega_pitch = np.sqrt(k / denom)
        f_pitch.append(omega_pitch / (2 * np.pi))

    df['freq_heave_Hz'] = f_heave
    df['freq_sway_Hz'] = f_sway
    df['freq_surge_Hz'] = f_surge
    df['freq_roll_Hz'] = f_roll
    df['freq_pitch_Hz'] = f_pitch

    return df


def expand_depth_and_ballast(
        df: pd.DataFrame,
        *,
        max_depth: float = 6.0,    # [m]
        step: float = 0.5,         # [m]
        rho: float = 1000.0      # [kg/m³] dichtheid water
    ) -> pd.DataFrame:
    records = []

    for _, row in df.iterrows():
        orig_depth = float(row['real_submerged_depth'])

        if orig_depth >= max_depth:
            new_row = row.copy()
            new_row['ballast_added_m'] = 0.0
            wp = new_row['platform_L'] * new_row['platform_W']
            new_row['mass_ballast_kg'] = rho * wp * 0.0
            records.append(new_row)
            continue

        target_depths = np.arange(orig_depth, max_depth + 1e-9, step)

        for depth in target_depths:
            new_row = row.copy()

            delta_d = depth - orig_depth
            wp = new_row['platform_L'] * new_row['platform_W']
            m_ballast = rho * wp * delta_d
```

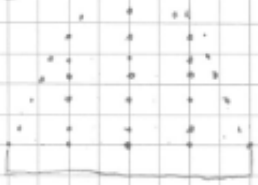
```
191
192             new_row['real_submerged_depth'] = depth
193             new_row['ballast_added_m'] = delta_d
194             new_row['mass_ballast_kg'] = m_ballast
195
196             records.append(new_row)
197
198     return pd.DataFrame(records)
```

J

Code verification



	Permanent	variable
Facade	1428 · 1,47 = 2099,16 kN	0
roof	160 · 4,35 = 696	1,5 · 160 = 936
floor	2200 · 4,35 = 9570	2,5 · 2200 = 5500



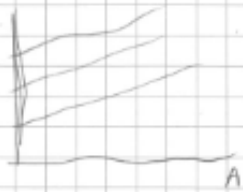
next improved code frame
5 · 5 + 3 · 2 = 31 per portal
5 · 31 = 155 columns
155 · 1,2 = 186 kN

4 m ⇒ IPE 270 → 1,42 kN
14 · 1,42 = 19,88
19,88 · 5 = 99,4 kN

z center of gravity

$$\frac{3 \cdot 20 \cdot 20 \cdot (2,5+4,35) + 6 \cdot 18 \cdot 20 \cdot (2,5 \cdot 4,35)}{2200 \cdot (2,5+4,35)}$$

stories ← N

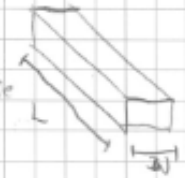


V_1, V_2, V_3 } Increased + square

Visualization platform + building

A (platform)

L = 35 m W = 35 m d = 3 m
configuratie = 2 x 7 → 17,5 X 5 m
$\frac{5000}{24} \approx 210$ mm
0,210 · 2 · 17,5 · 35 = 36,75 m³ concrete
(3 − 2 · 0,21) + 0,3 = 2,88 m
2 · (2,88 · 17,5 · 0,3 + 2,88 · 5 · 0,3) = 38,88 m³ concrete
75,63 m³ per pontoon
75,63 · 14 = 1058,82 m³ code 998,43 m³
5% difference okay
buoyant force = 35 · 35 · 3 · 9,8 · 1000 = 36051750



$$M = \underbrace{\int_0^{\lambda} A \sin\left(\frac{2\pi}{\lambda} \cdot x\right)\left(x - \frac{\lambda}{2}\right) dx}_{M_1} + \underbrace{P_{static} \int_0^{\lambda} \left(x - \frac{\lambda}{2}\right) dx}_{M_2}$$

$$M_2 = P_{static} \int_0^{\lambda} \left(x - \frac{\lambda}{2}\right) dx$$

$$M_2 = P_{static} \left[\frac{x^2}{2} - \frac{\lambda}{2}x\right]_0^{\lambda} = 0$$

$$M_1 = \int_0^{\lambda} A \cdot \sin\left(\frac{2\pi}{\lambda} \cdot x\right)\left(x - \frac{\lambda}{2}\right) dx \qquad \frac{2\pi}{\lambda} = k$$

$$A\left[\int_0^{\lambda} x \sin(kx)\, dx - \frac{\lambda}{2}\int_0^{\lambda} \sin(kx)\, dx\right]$$

$$A\left(\left[-\frac{x\cos(kx)}{k} + \frac{\sin(kx)}{k^2}\right]_0^{\lambda} + \frac{\lambda}{2}\left[-\frac{\cos(kx)}{k}\right]_0^{\lambda}\right.$$

$$A\left(-\frac{\lambda}{k} + 0\right) =$$

$$M = -\frac{A\lambda^2}{2\pi}$$

Moment wave

$$M = -\frac{A\lambda^2}{2\pi} \rho g$$

moment wind

$$F_{wind} = q_{wind} \cdot h_{bld} \cdot L$$

$$0{,}46 \cdot (6-3) \cdot 18 = 149{,}04$$

$$M_{wave} = \int_0^{\lambda} \left(A \sin\left(\frac{2\pi}{\lambda} \cdot x\right) + static\right) \cdot \left(x - \frac{\lambda}{2}\right)$$



$\lambda = 35$ m $A = 0{,}21 + 0{,}5 = 0{,}105$ m $W = 40$

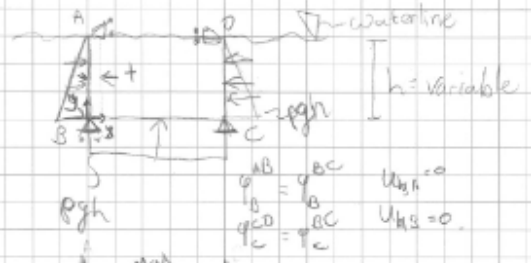
$$\frac{0{,}105 \cdot 10^2}{\pi} \cdot g \cdot 1000 = 200{,}8 \text{ kN/m}$$

$L_{bld} = 18$ $W_{bld} = 15$

storics < 6 Submerged $= 3$ m

arm $= 0{,}5 \cdot H + \text{depth} = 12$ m

$n = 7$
$bx = 3$
$rx = 4$



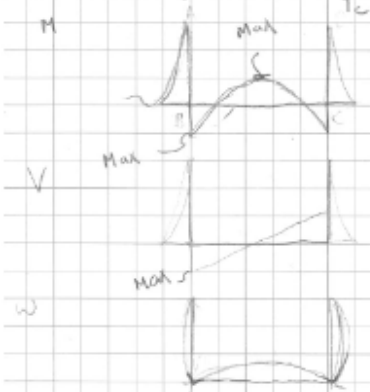
$q_B^{AB} = q_B^{BC}$ $\quad u_{BA} = 0$

$q_C^{CD} = q_C^{BC}$ $\quad u_{BB} = 0$

t = wall thickness dependend on depth

$N = 0$
$M = ?$
$V = ?$
determine t

want to have it from 1 - 9 m every 1 m using that t in pontoon



$q_{AB}(y) = \rho g y$

$q_{BC}(x) = -\rho g h$ $\quad \rho g$

$q_{CD}(y) = -\rho g y$

$V_{AB} = \int q_{AB} = \frac{1}{2}\rho g y^2 + C_1$

$M_{AB} = \int V_{AB} = \frac{1}{6}\rho g y^3 + C_1 x + C_2$

$\theta_{AB} = \int M_{AB} = \left(\frac{1}{24}\rho g y^4 + \frac{1}{2}C_1 y^2 + C_2 y + C_3\right)/EI$

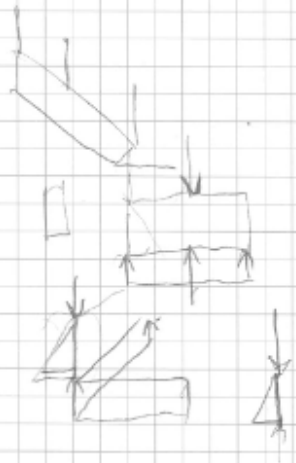
$u_{AB} = \int \theta_{AB} = \left(\frac{9}{120}\rho g y^5 + \frac{1}{6}C_1 y^3 + \frac{1}{2}C_2 y^2 + C_3 y + C_4\right)/EI$

$V_{CD} = \int q_{CD} = -\frac{1}{2}\rho g y^2 + C_5$

$M_{CD} = \int V_{CD} = -\frac{1}{6}\rho g y^3 + C_5 y + C_6$

$\theta_{CD} = \int M_{CD} = \left(-\frac{1}{24}\rho g y^4 + \frac{1}{2}C_5 y^2 + C_6 y + C_7\right)/EI$

$u_{CD} = \int \theta_{AB} = \left(-\frac{\rho g}{120}\rho g y^5 + \frac{1}{6}C_5 y^3 + \frac{1}{2}C_6^2 + C_7 y + C_8\right)/EI$



$V_{BC} = \int q_{BC} = -qx + C_9$

$M_{BC} = \int V_{BC} = -\frac{1}{2}qx^2 + C_9 x + C_{10}$

$\theta_{BC} = \int M_{BC} = \left(-\frac{1}{6}qx^3 + \frac{1}{2}C_9 x^2 + C_{10}x + C_{11}\right)/EI$

$u_{BC} = \int \theta_{BC} = \left(-\frac{1}{24}qx^4 + \frac{1}{6}C_9 x^3 + \frac{1}{2}C_{10}x^2 + C_{11}x + C_{12}\right)/EI$

