

Vizualizacija i izračun hidrostatičkih karakteristika triangularizirane forme broda primjenom programa otvorenog koda linaetal-fsb/d3v

Erhardt, Luka Josip

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture / Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje**

Permanent link / Trajna poveznica: <https://urn.nsk.hr/urn:nbn:hr:235:839406>

Rights / Prava: [In copyright](#) / [Zaštićeno autorskim pravom.](#)

Download date / Datum preuzimanja: **2024-04-17**

Repository / Repozitorij:

[Repository of Faculty of Mechanical Engineering and Naval Architecture University of Zagreb](#)



SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Luka Josip Erhardt

Zagreb, 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentori:

Izv. prof. dr. sc. Pero Prebeg

Student:

Luka Josip Erhardt

Zagreb, 2020.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se mentorima dr.sc. Peri Prebegu i mag.ing. Ivanu Muniću na pomoći i strpljenju.

Također se želim zahvaliti svojoj djevojci Nini, roditeljima i prijateljima na motivaciji i podršci.

Luka Josip Erhardt



SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija brodogradnje



Sveučilište u Zagrebu	
Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa:	
Ur.broj:	

ZAVRŠNI ZADATAK

Student: **Luka Josip Erhardt**

Mat. br.: 0035201915

Naslov rada na hrvatskom jeziku: **Vizualizacija i izračun hidrostatičkih karakteristika triangularizirane forme broda primjenom programa otvorenog koda linaetal-fsb/d3v**

Naslov rada na engleskom jeziku: **Use of open source program linaetal-fsb/d3v for ship hull form visualization and hydrostatic characteristics calculation**

Opis zadatka:

Analitičko zadavanje forme broda ima široku primjenu u ranim fazama projektiranja broda jer omogućuje jednostavno modificiranje forme broda putem promjene manjeg broja parametara. Program otvorenog koda linaetal-fsb/d3v temeljen je na *Qt for Python* tehnologiji, koja omogućuje trodimenzijsku (3D) vizualizaciju geometrijskih entiteta primjenom *OpenGL* tehnologije te jednostavno definiranje grafičkog sučelja za pojedine specifične namjene. U radu je potrebno, proširenjem funkcionalnosti programa otvorenog koda linaetal-fsb/d3v u programskom jeziku Python, omogućiti grafičku kontrolu parametara analitički zadane forme broda, vizualizaciju trenutne forme primjenom adekvatne triangularizacije te izračun hidrostatičkih karakteristika triangularizirane forme broda.

Zadatak obuhvaća sljedeće:

- upoznavanje s programom otvorenog koda linaetal-fsb/d3v
- odabir hidrostatičkih karakteristika forme bez privjesaka za koje će se izraditi matematički model, a koje minimalno moraju uključiti karakteristike vidljive na uobičajenom dijagramnom listu
- upoznavanje s parametarskim modeliranjem forme trupa i nadgrađa fregate putem analitički zadanih brodskih linija
- izradu Python modula za analitičko definiranje forme trupa broda i njenu transformaciju putem manjeg broja parametara prema zadanoj literaturi i postojećem, dostupnom programskom kodu slične namjene
- izradu Python modula koji omogućuje triangularizaciju forme broda temeljem analitički zadane forme trupa i nadgrađa
- izradu Python modula za izračun odabranih hidrostatičkih karakteristika triangularizirane forme
- izradu Python modula koji omogućuje 3D vizualizaciju triangularizirane forme broda u programu otvorenog koda linaetal-fsb/d3v
- izradu Python modula koji omogućuje grafičku kontrolu parametara analitički zadane forme broda u programu otvorenog koda linaetal-fsb/d3v te 3D vizualizaciju trenutne forme broda.

U radu treba navesti korištenu literaturu te eventualno dobivenu pomoć.

Zadatak zadan:

28. studenog 2019.

Datum predaje rada:

1. rok: 21. veljače 2020.

2. rok (izvanredni): 1. srpnja 2020.

3. rok: 17. rujna 2020.

Predviđeni datumi obrane:

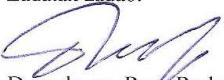
1. rok: 24.2. – 28.2.2020.

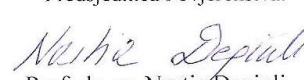
2. rok (izvanredni): 3.7.2020.

3. rok: 21.9. – 25.9.2020.

Predsjednica Povjerenstva:

Zadatak zadao:


Doc. dr. sc. Pero Prebeg


Prof. dr. sc. Nastia Degiuli

SADRŽAJ

SADRŽAJ	I
POPIS SLIKA	II
POPIS TABLICA.....	III
SAŽETAK.....	V
SUMMARY	VI
1. UVOD.....	1
1.1. Parametarski model forme	1
2. TRIANGULARIZACIJA FORME BRODA	3
2.1. Openmesh.....	3
2.2. Princip triangularizacije	3
2.3. Vizualizacija triangularizirane forme primjenom Linaetal-fsb/d3v programa	4
3. HIDROSTATIČKE KARAKTERISTIKE TRIANGULARIZIRANE FORME	6
3.1. Površina vodne linije.....	6
3.2. Volumen istisnine	7
3.3. Težište vodne linije	7
3.4. Težište volumena istisnine	8
3.5. Uzdužni i poprečni moment inercije vodne linije	8
3.6. Ostali hidrostatički podaci	9
4. ODREĐIVANJE HIDROSTATIČKIH KARAKTERISTIKA NA PRIMJERU FREGATE	11
4.1. Usporedba s modelom smanjene točnosti	12
4.2. Usporedba hidrostatičkih karakteristika modela računatih različitim softverima.....	16
5. DIJAGRAMNI LIST	19
5.1. Upotreba i značenje dijagramnog lista	19
5.2. Crtanje dijagramnog lista	20
5.3. Očitavanje vrijednosti u dijagramnom listu	21
5.4. Grafička kontrola parametara analitički zadane forme broda	24
6. ZAKLJUČAK.....	26
LITERATURA	27
PRILOZI.....	28

POPIS SLIKA

Slika 1. Fregata Narvik F304	2
Slika 2. Prikaz triangularizacije forme	3
Slika 3. Vizualizacija brodske forme dobivene triangularizacijom	4
Slika 4. Vizualizacija brodske forme dobivene triangularizacijom	5
Slika 5. Prikaz presjecanja trokuta i vodne linije. Sivom bojom su označene površine dijelova trokuta čije projekcije na vodnu liniju pridodajemo površini.	6
Slika 6. Primjer prizme na koje je podijeljen volumen istisnine.....	7
Slika 7. Prikaz težišta trokuta C	8
Slika 8. Prikaz podjele glavnog rebra na trapeze	9
Slika 9. Dijagramni list.....	22
Slika 10. Prikaz sučelja koji generira dijagramni list.....	23
Slika 11. Prikaz prozora za modifikaciju brodske forme	24
Slika 12. Prikaz modificirane forme	25

POPIS TABLICA

<i>Tablica 1. Osnovne karakteristike broda</i>	<i>2</i>
<i>Tablica 2. Hidrostatički podatci (Nvl=50, broj točaka po VL=50)</i>	<i>11</i>
<i>Tablica 3. Usporedba hidrostatičkih podataka ovisno o broju trokuta</i>	<i>12</i>
<i>Tablica 4. Relativno odstupanje ovisno o broju trokuta</i>	<i>13</i>
<i>Tablica 5. Usporedba hidrostatičkih podataka ovisno o broju trokuta</i>	<i>14</i>
<i>Tablica 6. Relativno odstupanje ovisno o broju trokuta</i>	<i>15</i>
<i>Tablica 7. Usporedba hidrostatičkih podataka</i>	<i>17</i>
<i>Tablica 8. Relativno odstupanje hidrostatskih podataka</i>	<i>18</i>
<i>Tablica 9. Mjerilo</i>	<i>23</i>

POPIS OZNAKA

Oznaka	Jedinica	Opis
LOA	[m]	duljina preko svega (length overall)
BOA	[m]	širina preko svega (breadth overall)
T	[m]	gaz
KM _L	[m]	visina uzdužnog metacentra iznad osnovice
KM ₀	[m]	visina poprečnog metacentra iznad osnovice
X _{CG}	[m]	uzdužni položaj težišta istisnine
A _{WL}	[m ²]	površina vodne linije
X _{WL}	[m]	uzdužni položaj težišta vodne linije
Z _{WL}	[m]	vertikalni položaj težišta vodne linije
I _L	[m ⁴]	uzdužni moment inercije vodne linije
I _B	[m ⁴]	poprečni moment inercije vodne linije
V	[m ³]	volumen istisnine
Δ	[t]	masa istisnine
KB	[m]	vertikalni položaj težišta istisnine
M _{0B}	[m]	poprečni metacentarski radijus
M _L B	[m]	uzdužni metacentarski radijus
L	[m]	duljina broda
L _{WL}	[m]	duljina vodne linije
B	[m]	širina broda
B _{WL}	[m]	Širina vodne linije
T	[m]	gaz broda
M _I	[tm/m]	jedinični moment trima
C _B	[-]	koeficijent punoće
C _{WL}	[-]	koeficijent vodne linije
C _P	[-]	prizmatički koeficijent
C _X	[-]	koeficijent glavnog rebra

SAŽETAK

U ovom radu izrađen je python modul za triangularizaciju forme broda. Zatim je takva triangularizirana forma u programu otvorenog koda `linaetal-fsb/d3v` 3D vizualizirana. Na osnovi triangualizirane forme, izrađene su metode za izračun su hidrostatičkih karakteristike koje su onda prikazane u dijagramnom listu. Na kraju je python modulom omogućena grafička kontrola parametara analitički zadane forme broda. Izrađeni moduli primijenjeni su na parametarskom modelu fregate KNM Narvik F304, iako se modul može primjeniti na bilo koji brod koji čija se forma može opisati istim analitičkim modelom.

Ključne riječi: python modul, triangularizacija forme broda, hidrostatičke karakteristike, dijagramni list

SUMMARY

In this paper, a python module for the triangulation of a ship hull was made. The triangulated hullform was then visualized in 3D in the open source program linaetal-fsb/d3v. From the parametrically modeled shape of the frigate hull and analytically given ship lines, the hydrostatic characteristics were calculated and then shown in a diagram sheet. To conclude, a python module was used to enable graphic control of the parameters of the analytically given hull form. This thesis was done using the analytically given form of the frigate KNM Narvik F304, although the module can be applied to any ship given a similar analytical form.

Key words: python module, triangulation, hydrostatic characteristics, diagram sheet

1. UVOD

Hidrostatske karakteristike uvelike ovise o formi broda. Zbog toga je u ranoj fazi projektiranja broda vrlo bitno odrediti osnovne karakteristike kao što su volumen, površina i težište vodne linije te uzdužni i poprečni moment inercije. Uobičajeno je da se hidrostatske karakteristike unose u dijagramni list kako bi sve hidrostatske karakteristike broda bile vidljive na jednom mjestu. Analitičko zadavanje forme broda ima široku primjenu u ranim fazama projektiranja broda jer omogućuje jednostavno modificiranje forme broda putem promjene manjeg broja parametara. U ovom radu iz analitički zadane forme generirati će se jednostavna triangulizirana forma koja će služiti za vizualizaciju, ali i za izračun hidrostatskih karakteristika.

1.1. Parametarski model forme

Parametarsko modeliranje koristi računalo za konstruiranje objekata ili sustava koji komponentama modela pripisuje realno ponašanje. Parametarski modeli koriste alate za modeliranje površina i 3D tijela temeljene na značajkama koje im omogućuju jednostavnije manipuliranje karakteristikama modela.

Model trupa izgrađen je uzimajući u obzir skup parametara koji povezuju glavne dimenzije s drugim dimenzijama geometrije trupa i neke ovisnosti između različitih parametara. Korištenjem ovog pristupa, lakše je unijeti promjene u oblik trupa. Naime, kada se promijeni jedan parametar geometrije, svi ostali parametri koji ovise o prvom automatski se prilagode [2].

U ovom radu korišten je parametarski model forme fregate (Slika 1, **Tablica 1.**). Parametarski model forme kao i iznosi parametara koji određuju formu ove fregate, preuzeti su s mrežne aplikacije 'DBB(*design building blocks*) method' [3]. Trup je parametarski zadan i može se jednostavno mijenjati promjenom nekoliko parametara koji ga definiraju. Iz mrežne aplikacije DBB uzete su osnovne dimenzije broda i dobivena je analitička forma trupa, što je omogućilo njenu vizualizaciju i diskretizaciju.



Slika 1. Fregata Narvik F304

Tablica 1. Osnovne karakteristike broda

LOA	BOA	T
100 m	15 m	9.4 m

2. TRIANGULARIZACIJA FORME BRODA

2.1. Openmesh

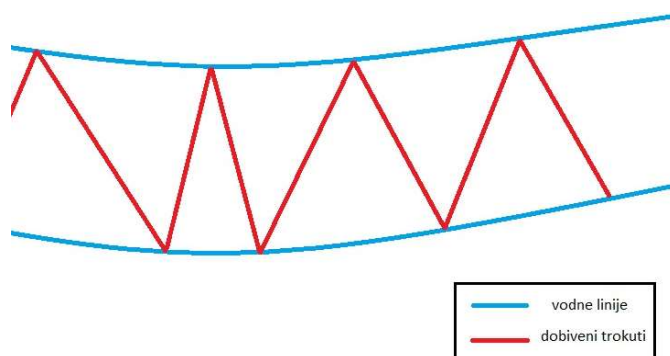
OpenMesh je generička i učinkovita struktura podataka za predstavljanje i manipuliranje poligonalnim mrežama razvijen u programskom jeziku C ++, a dostupan je i u Pythonu. Razvijen je u Computer Graphics Group, RWTH Aachen. [4].

U ovom radu Openmesh je korišten za zapisivanje triangualizirane forme broda odnosno manipuliranje vrhovima (*eng. vertex*) i poligonskim ploham (*eng. facets*) na način da je svaki trokut u triangulariziranoj formi bio sačinjen od tri vrha i jedne poligonske plohe. Te poligonske plohe i vrhovi su pohranjeni u Openmesh jezgri odkud ih zatim možemo po potrebi koristiti.

2.2. Princip triangularizacije

Triangularizacija površine podrazumijeva generiranje mreže trokuta koja ju potpuno ili djelomično prekriva. Trokuti se razlikuju oblikom ovisno o njihovom položaju na zakrivljenoj površini kako bi se ona što bolje opisala.

Triangularizacija zadane forme fregate dobije se poprečnim spajanjem točaka između susjednih vodnih linija čime se dobivaju trokuti (Slika 2). Na svakoj vodnoj liniji je jednak broj točaka. Spajanjem točaka dobivaju se trokuti koji naizmjenično imaju po dvije točke na gornjoj vodnoj liniji ili dvije točke na donjoj vodnoj liniji.



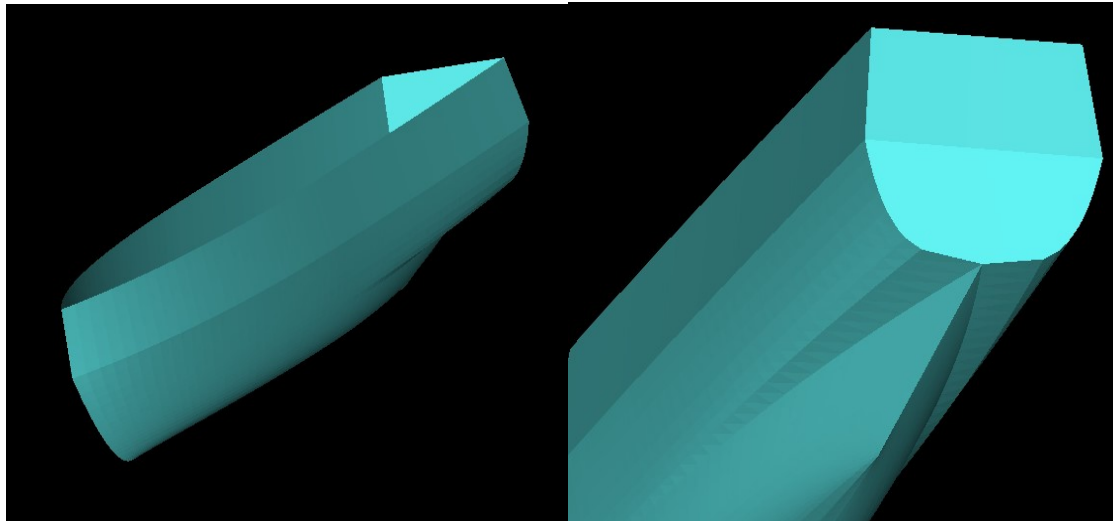
Slika 2. Prikaz triangularizacije forme

U Python programskom sučelju triangularizacija se napravila korištenjem modula 'OpenMesh Python'. U tom modulu se zadavanjem vrhova funkciji jednostavno iscrta ploha koja ih povezuje koja je u ovom slučaju bio trokut.

2.3. Vizualizacija triangularizirane forme primjenom Linaetal-fsb/d3v programa

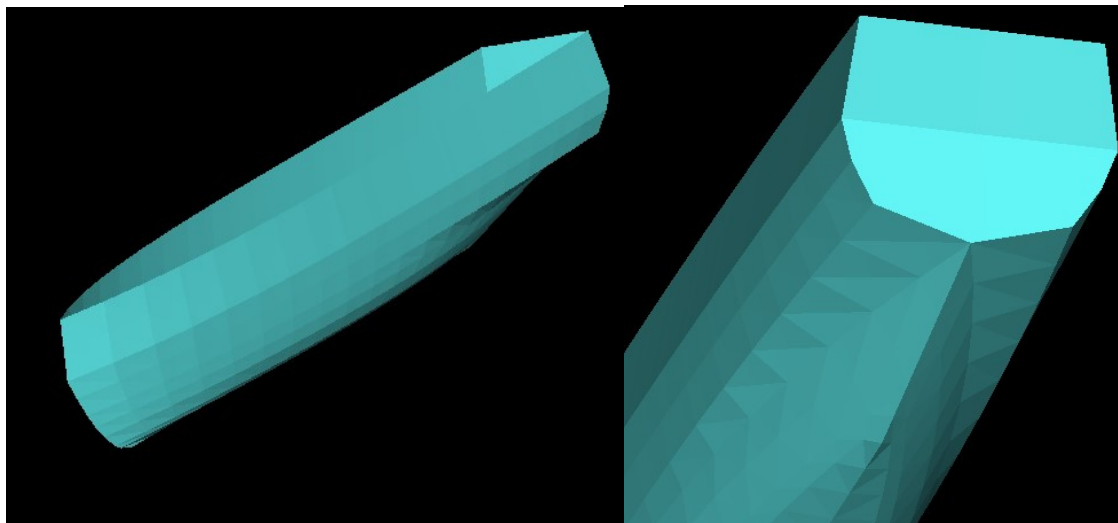
Linaetal-fsb d3v (*design visualizer*) je modularna Python aplikacija otvorenog programskog koda, prvenstveno namijenjena za 3D vizualizaciju inženjerskih modela u fazi projektiranja. Inicijalno je nastala kao rezultat dugogodišnje suradnje Fakulteta strojarstva i brodogradnja s USCS.d.o.o te obrtom Linaetal. Struktura programa temeljena je na dvanaestogodišnjem iskustvu razvoja programa ShipExplorer. Program je u ranoj fazi razvoja no već je moguća vizualizacija s ograničenim brojem funkcionalnosti. Implementacije je bazirana na bibliotekama QtForPython (PySide2) i OpenMesh[5].

Spomenute plohe (trokuti) koje su crtane između zadanih točaka zajedno čine formu broda (slika 3.).



*Slika 3. Vizualizacija brodske forme dobivene triangularizacijom
(30vl i 30 točaka po vl)*

Broj vodnih linija i točaka na vodnim linijama je moguće mijenjati, čime dobivamo glađu ili grublju formu, odnosno kontroliramo točnost prikaza forme. Slika 3. prikazuje formu generiranu pomoću 30 vodnih linija i 30 točaka po vodnoj liniji, a slika 4. prikazuje formu s 5 vodnih linija i 10 točaka po vodnoj liniji.



***Slika 4. Vizualizacija brodske forme dobivene triangularizacijom
(5vl i 10 točaka po vl)***

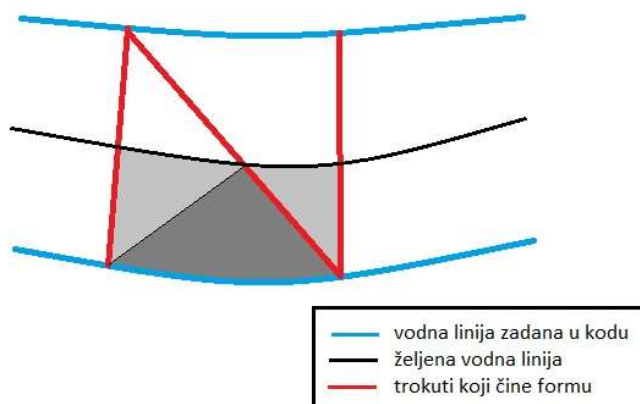
3. HIDROSTATIČKE KARAKTERISTIKE TRIANGULARIZIRANE FORME

Na temelju izrađene triangularizirane forme broda se mogu izračunati hidrostatički podaci. Navedene karakteristike se u ovom slučaju mogu izračunati i iz analitički zadane forme, no računanje na osnovu triangularizirane forme omogućuje da se hidrostatičke karakteristike odrede i za brodske forme koje su npr. izrađene u nekom programskom paketu, pa eksportirane u neki standardni zapis poput .STL datoteka (*STereo Lithography CAD*).

3.1. Površina vodne linije

Vodna linija je presjek teoretske forme vodoravnom linijom na zadanoj visini iznad osnovice, tako da njena površina ima oblik tlocrta broda. Radi oblika fregate u ovom radu smanjenjem visine na kojoj se nalazi vodna linija se smanjuje i njena površina.

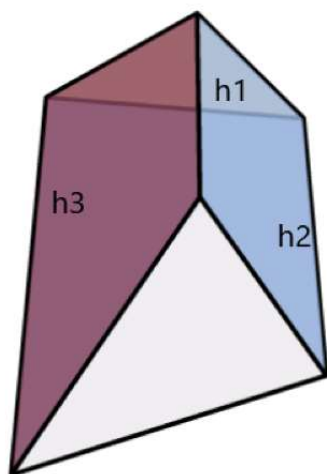
Površina vodne linije u ovom radu je dobivena tako da su se projicirali trokuti koji sačinjavaju formu do određene visine na vodnu liniju koja se nalazi na toj visini. To zapravo znači da se efektivno gleda tlocrt broda sa različitim visinama vodne linije i zbrajaju se površine trokuta koje pritom vidimo. S obzirom da vodna linija čiju površinu želimo se ne poklapa uvijek sa već postojećim vodnim linijama dobivenim u kodu, potrebno je pribrojiti i površine dijelova trokuta koji nastaju presjecanjem trokuta i vodne linije. U kodu je to postignuto uvjetom da ako se sva tri vrha trokuta nalaze ispod vodne linije, program računa površinu trokuta, ako su dva vrha trokuta iznad vodne linije program računa površinu preostalog presječenog trokuta, a ako je jedan vrh trokuta iznad vodne linije ostaje nam trapez kojeg dijelimo na dva trokuta čije površine doprinose površini vodne linije.



Slika 5. Prikaz presjecanja trokuta i vodne linije. Sivom bojom su označene površine dijelova trokuta čije projekcije na vodnu liniju pridodajemo površini.

3.2. Volumen istisnine

Volumen istisnine je volumen uronjenog dijela trupa. Mijenjanjem visine vodne linije mijenja se i volumen istisnine. S obzirom da je forma broda izgrađena od trokuta, volumen se dobije zbrojem volumena krnjih trostranih prizmi čija je baza trokut projiciran na vodnu liniju, a suprotna ploha je trokut koji čini formu broda[6].



$$V = \frac{1}{3}A(h_1 + h_2 + h_3) \quad (1)$$

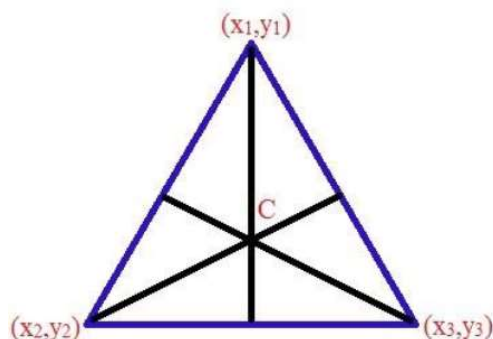
Slika 6. Primjer prizme na koje je podijeljen volumen istisnine

Volumen jedne takve krnje prizme se računa prema formuli (1) gdje A označava površinu baze a h_1, h_2 i h_3 su visine bridova. Površina baze računa se jednako kao što se računala kod površine vodne linije a visine bridova se dobivaju oduzimanjem visine vodne linije od vrhova trokuta koji čine formu. Ovdje isto treba pripaziti kada vodna linija sječe dani trokut da se točno dodaju volumeni tijela koji se dobiju tim presjecanjem.

3.3. Težište vodne linije

Težistu vodne linije su već poznate dvije koordinate. Naime, njegova z koordinata će odgovarati visini vodne linije, a s obzirom da x os prolazi po simetrali broda težište će se nalaziti na $y=0$. Položaj težišta na x osi može se naći nalaženjem težišta projekcija trokuta koji čine vodnu liniju. Težište jednog trokuta dobije se kao srednja vrijednost koordinata vrhova. Dobivena težišta trokuta je zatim potrebno pomnožiti sa površinama trokuta i onda podijeliti sa ukupnom površinom vodne linije. Računanjem težišta svakog pojedinog trokuta i njihovim zbrajanjem

efektivno se računa položaj težišta skupa točaka raspoređenih po vodnoj liniji. Taj položaj se predviđa da će biti otprilike na sredini broda jer je zadana fregata relativno simetrična oko poprečne osi. Položaj težišta se mijenja na svakoj zadanoj visini vodne linije[7].



Slika 7. Prikaz težišta trokuta C

3.4. Težište volumena istisnine

Težište volumena istisnine se računa na sličan način kao kod računanja težišta vodne linije samo što su trostrane krnje prizme zamijenile ulogu trokuta. Naime, računanja težišta svake krnje prizme i njihovo množenje sa pripadnim volumenom u zbroju vode na položaj težišta istisnine. Potrebno je naći težište po x osi (X_{cg}) i po z osi (KB). Težište svake krnje prizme dobiveno je njenom podjelom na prizmu i na trostranu piramidu. Težište prizme je jednostavno naći jer je to zapravo težište trokuta baze translirano po z osi na pola visine prizme. Pomak tog težišta radi doprinosa trostrane piramide je zanemariv jer je ona puno manjeg volumena od same prizme.

3.5. Uzdužni i poprečni moment inercije vodne linije

Moment inercije je fizikalna veličina koja opisuje inerciju čestice ili krutoga tijela pri promjeni brzine ili smjera vrtnje. On je jednak zbroju umnožaka mase i kvadrata udaljenosti od osi rotacije svake čestice koja čini tijelo ili sustav:

$$I = \sum_{i=1}^N m_i r_i^2. \quad (2)$$

Uzdužni moment inercije vodne linije je moment inercije oko osi koja prolazi kroz njeno težište i koja je okomita na simetralu trupa, a poprečni moment inercije mjeri inerciju oko osi uzduž simetrale broda. S obzirom da se vodna linija ovdje promatra kao skup projekcija trokuta s trupa broda koji nemaju masu, težinski faktor mase iz izraza (2) potrebno je zamjeniti pripadnom površinom danog trokuta, dok je udaljenost r_i^2 ovdje udaljenost težišta trokuta od osi. Uzdužni

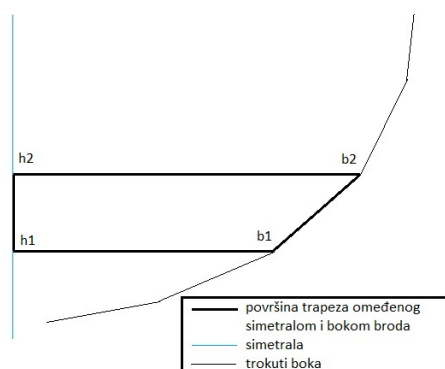
moment inercije je izračunat primjenom Steinerovog teorema. Prvo je izračunat moment inercije oko osi na krmi broda, gdje se nalazi ishodište koordinatnog sustava pa je zatim dodan Steinerov dodatak, čime je moment inercije pomaknut u paralelnu os koja prolazi kroz težiste vodne linije.

3.6. Ostali hidrostatički podaci

Dosad navedeni hidrostatički podaci služe pri definiranju novih koji su prikladniji pri opisu brodske forme i omogućuju lakšu međusobnu usporedbu brodova različitih veličina. Kako bi se definirali, potrebno je još nekoliko parametara broda.

Duljina vodne linije je udaljenost između uzdužno najdaljih točaka na određenoj vodnoj liniji, a širina je udaljenost između poprečno najudaljenijih točaka na istoj visini. Za numerički izračun obje veličine se generiralo polje svih točaka presjecišta vodne linije i trokuta koje su se zatim sortirale od najmanje prema najvećoj. Za dobiti duljinu, točke se sortiraju po rastućoj x koordinati te je od najveće vrijednosti oduzeta najmanja, a za širinu se sortiralo po rastućoj y koordinati.

Površina glavnog rebra je površina omeđena rebrom i određenom vodom linijom na sredini duljine broda. Na sličan način kao kod računanja duljine i širine vodne linije, su se generirale točke, u ovom slučaju, presjecišta rebra i trokuta, a sortiranje se izvodilo po rastućoj z koordinati. Susjedne točke počevši od točke najmanje z koordinate, su se preslikale na simetralu te se površina između njih računala formulom za površinu trapeza (slika 8.).



$$A = \frac{1}{2}(b_2 + b_1)(h_2 - h_1) \quad (3)$$

Slika 8. Prikaz podjele glavnog rebra na trapeze

Sada su definirani svi parametri potrebni za uvođenje novih hidrostatičkih podataka. Osim bezdimenzijskih veličina koriste se poprečni metacentarski radijus (M_0B), visina poprečnog

metacentra iznad osnovice (KM_0), uzdužni metacentarski radijus ($M_L B$), visina uzdužnog metacentra iznad osnovice (KM_L), jedinični zagažaj (JZ) i jedinični moment trima (M_1).

Bezdimenzijske veličine korisne su kod usporedbi brodova različitih dimenzija, zbog čega se uvode parametri volumena, odnosno, koeficijent punoće (C_B), prizmatički koeficijent (C_P), koeficijent vodne linije (C_{WL}) i koeficijent glavnog rebra (C_X). Navedene veličine računaju se prema izrazima [1]:

$$M_0 B = \frac{I_B}{V} \quad (4)$$

$$KM_0 = M_0 B + KB \quad (5)$$

$$M_L B = \frac{I_L}{V} \quad (6)$$

$$KM_L = M_L B + KB \quad (7)$$

$$JZ = 0.01 \cdot A_{WL} \cdot \rho \quad (8)$$

$$M_1 = \frac{I_L}{L_{WL}} \quad (9)$$

$$C_{WL} = \frac{A_{WL}}{L_{WL} \cdot B_{WL}} \quad (10)$$

$$C_B = \frac{V}{L_{WL} \cdot B_{WL} \cdot T} \quad (11)$$

$$C_P = \frac{V}{A_X \cdot L_{WL}} \quad (12)$$

$$C_X = \frac{A_X}{B_{WL} \cdot T} \quad (13)$$

4. ODREĐIVANJE HIDROSTATIČKIH KARAKTERISTIKA NA PRIMJERU FREGATE

Svi spomenuti hidrostatički podaci, numerički su izračunati na opisani način. Njihove vrijednosti nalaze se u Tablici 2., a dobivene su numeričkim proračunom koji koristi podjelu trupa na 50 vodnih linija te na svakoj vodnoj liniji koristi 50 točaka za iscrtavanje trokuta. U tablici 2. su ispisane vrijednosti spomenutih karakteristika na nekoliko različitih visina vodnih linija, kako bi se uočio trend porasta ili opadanja dobivenog rezultata s promjenom visine. Dok, na primjer, volumen istisnine i površina vodne linije rastu s povećanjem visine, težište vodne linije ne mijenja se značajno, već je uvijek pri sredini trupa. Za x koordinatu težišta istisnine vrijedi isto. Promjenom visine, vrijednosti bezdimenzionalnih koeficijenata se minimalno mijenjaju.

Tablica 2. Hidrostatički podatci (Nvl=50, broj točaka po VL=50)

T	m	1.00	3.00	5.00	7.00	9.00
Volumen	m³	302.51	1459.44	3175.08	5334.57	7731.17
Awl	m²	434.47	701.44	1005.88	1144.28	1248.89
Xwl	m	53.04	50.21	44.01	43.87	44.30
KB	m	0.592	1.769	3.011	4.230	5.40
Xcg	m	53.50	51.95	49.04	46.94	46.05
lb	m⁴	1705	5299	11098	15561	19394
ll	m⁴	134004	292803	593999	720463	824893
Lwl	m	78.39	91.03	96.73	98.32	99.73
Bwl	m	8.29	11.48	13.03	14.07	14.86
Ax	m²	6.01	26.31	50.95	78.14	107.11
MoB	m	5.63	3.63	3.49	2.91	2.50
Kmo	m	6.22	5.40	6.50	7.14	7.91
MIb	m	442.97	200.62	187.08	135.05	106.69
KMI	m	443.56	202.39	190.09	139.28	112.10
JZ	t	4.45	7.19	10.31	11.72	12.80
Cwl	/	0.668	0.671	0.798	0.827	0.843
CB	/	0.465	0.465	0.503	0.551	0.579
CP	/	0.642	0.609	0.644	0.694	0.724
CX	/	0.724	0.764	0.782	0.793	0.801

4.1. Usporedba s modelom smanjene točnosti

Točnost modela može se smanjiti smanjenjem broja trokuta korištenih za triangularizaciju forme trupa, čime ona poprima 'grublji' oblik kao što se vidi na slikama 3. i 4. To se postiže smanjenjem broja vodnih linija i točaka na njima koje određuju vrhove trokuta. Tablica 3. prikazuje usporedbu hidrostatskih podataka dobivenih za 50 vodnih linija i 50 točaka po vodnoj liniji s hidrostatskim podacima dobivenim za formu izrađenu s 30 vodnih linija i 30 točaka po vodnoj liniji.

Tablica 3. Usporedba hidrostatskih podataka ovisno o broju trokuta

T		m	1	3	5	7	9
Volumen	50vl,50točaka	m ³	302.51	1459.44	3175.08	5334.57	7731.17
	30vl,30točaka		301.12	1457.22	3165.35	5323.51	7719.31
Awl	50vl,50točaka	m ²	434.47	701.44	1005.88	1144.28	1248.89
	30vl,30točaka		434.39	701.19	1004.91	1144.09	1248.43
Xwl	50vl,50točaka	m	53.04	50.21	44.01	43.87	44.30
	30vl,30točaka		53.04	50.21	44.03	43.87	44.30
KB	50vl,50točaka	m	0.592	1.76	3.01	4.23	5.40
	30vl,30točaka		0.595	1.77	3.01	4.232	5.40
Xcg	50vl,50točaka	m	53.50	51.95	49.04	46.94	46.05
	30vl,30točaka		53.50	51.95	49.11	46.98	46.08
Ib	50vl,50točaka	m ⁴	1704	5299	11097	15560	19394
	30vl,30točaka		1696	528	11044	15528	19351
Il	50vl,50točaka	m ⁴	134004	292803	593999	720463	824893
	30vl,30točaka		133929	292587	592962	720133	824219
Kmo	50vl,50točaka	m	6.22	5.4	6.50	7.14	7.91
	30vl,30točaka		6.23	5.39	6.5	7.14	7.91
KMI	50vl,50točaka	m	443.56	202.39	190.09	139.28	112.1
	30vl,30točaka		445.36	202.55	190.34	139.50	112.18
CB	50vl,50točaka	/	0.465	0.465	0.503	0.551	0.579
	30vl,30točaka		0.463	0.465	0.502	0.549	0.579
CP	50vl,50točaka	/	0.642	0.609	0.644	0.694	0.724
	30vl,30točaka		0.642	0.609	0.643	0.693	0.723
CX	50vl,50točaka	/	0.724	0.764	0.782	0.793	0.801
	30vl,30točaka		0.721	0.763	0.781	0.792	0.801

Kao što se vidi iz tablice 4. unatoč smanjenju broja trokuta koji tvore formu broda, vrijednosti su vrlo bliske vrijednostima forme izrađene s 50 vodnih linija i 50 točaka po vodnoj liniji, dok neke vrijednosti čak ostaju nepromijenjene. Relativno odstupanje navedenih veličina ne doseže vrijednosti iznad 0.5%, zbog čega se može zaključiti da se model može pojednostaviti manjim brojem trokuta, čime se smanjuje vrijeme izvođenja programa uz zanemarivi gubitak na točnosti.

Tablica 4. Relativno odstupanje ovisno o broju trokuta

T		m	1	3	5	7	9
Volumen	50wl,50točaka	m^3	302.51	1459.44	3175.08	5334.57	7731.17
	30wl,30točaka		-0.46%	0.15%	0.31%	0.21%	0.15%
Awl	50wl,50točaka	m^2	434.47	701.44	1005.88	1144.28	1248.89
	30wl,30točaka		-0.02%	0.04%	0.10%	0.02%	0.04%
Xwl	50wl,50točaka	m	53.04	50.21	44.01	43.87	44.30
	30wl,30točaka		0.000%	0.002%	0.036%	0.007%	0.005%
KB	50wl,50točaka	m	0.59	1.76	3.01	4.23	5.40
	30wl,30točaka		0.50%	0.06%	0.00%	0.05%	0.05%
Xcg	50wl,50točaka	m	53.50	51.95	49.04	46.94	46.05
	30wl,30točaka		-0.01%	0.006%	0.15%	0.09%	0.07%
Ib	50wl,50točaka	m^4	1704	5299	11097	15560	19394
	30wl,30točaka		-0.46%	0.23%	0.48%	0.21%	0.22%
Il	50wl,50točaka	m^4	134003	292802	593998	720463	824892
	30wl,30točaka		-0.05%	0.07%	0.17%	0.05%	0.08%
Kmo	50wl,50točaka	m	6.22	5.4	6.50	7.14	7.91
	30wl,30točaka		0.05%	0.02%	0.11%	0.03%	0.01%
KMI	50wl,50točaka	m	443.56	202.39	190.09	139.28	112.1
	30wl,30točaka		0.40%	0.08%	0.13%	0.16%	0.07%
CB	50wl,50točaka	/	0.465	0.465	0.503	0.551	0.579
	30wl,30točaka		-0.43%	0.00%	0.20%	0.36%	0.00%
CP	50wl,50točaka	/	0.642	0.609	0.644	0.694	0.724
	30wl,30točaka		0.00%	0.00%	0.15%	0.14%	0.14%
CX	50wl,50točaka	/	0.724	0.764	0.782	0.793	0.801
	30wl,30točaka		-0.42%	0.13%	0.13%	0.13%	0.00%

Zbog povoljnih karakteristika triangularizacije i precizne izrade koda za presjek forme na vodne linije, smanjenjem broja vodnih linija na 10 i broja točaka po svakoj vodnoj liniji na 10, vrijednosti se minimalno mijenjaju.

Tablica 5. Usporedba hidrostatičkih podataka ovisno o broju trokuta

T		m	1	3	5	7	9
Volumen	50vl,50točaka	m³	302.51	1459.4	3175.1	5334.6	7731.2
	10vl,10točaka		290.21	1436.7	3127.2	5272.2	7659.9
Awl	50vl,50točaka	m²	434.47	701.4	1005.88	1144.28	1248.89
	10vl,10točaka		433.17	698.01	1001.46	1138.98	1244.73
Xwl	50vl,50točaka	m	53.04	50.21	44.02	43.88	44.31
	10vl,10točaka		53.04	50.21	44.03	43.87	44.25
KB	50vl,50točaka	m	0.592	1.769	3.011	4.23	5.403
	10vl,10točaka		0.612	1.786	3.021	4.244	5.421
Xcg	50vl,50točaka	m	53.50	51.95	49.04	46.94	46.05
	10vl,10točaka		53.42	51.90	49.21	47.07	46.12
Ib	50vl,50točaka	m⁴	1704	5299	11097	15560	19394
	10vl,10točaka		1639	5175	10859	15260	19134
II	50vl,50točaka	m⁴	134003	292802	593998	720463	824892
	10vl,10točaka		132983	289792	588289	713301	817779
Kmo	50vl,50točaka	m	6.227	5.4	6.507	7.147	7.912
	10vl,10točaka		6.262	5.38	6.494	7.138	7.919
KMI	50vl,50točaka	m	443.56	202.39	190.09	139.28	112.1
	10vl,10točaka		458.82	203.49	191.14	139.53	112.18
CB	50vl,50točaka	/	0.465	0.465	0.503	0.551	0.579
	10vl,10točaka		0.446	0.459	0.496	0.545	0.575
CP	50vl,50točaka	/	0.642	0.609	0.644	0.694	0.724
	10vl,10točaka		0.678	0.614	0.643	0.692	0.721
CX	50vl,50točaka	/	0.724	0.764	0.782	0.793	0.801
	10vl,10točaka		0.658	0.747	0.772	0.788	0.797

Relativno odstupanje od modela s 50 vodnih linija i 50 točaka po vodnoj liniji ne prelazi 10% na najnižoj vodnoj liniji gdje je greška najveća.

Mjesta gdje su najveća relativna odstupanja su na dnu broda pri nižim visinama, iz toga se zaključuje da bi idealan slučaj triangularizacije uz što veću točnost i što manje elemenata bio da pri nižim visinama izradimo gušću mrežu trokuta nego na većim visinama. Na visini od 1 m je maksimalno odstupanje 10% dok je pri visini od 9 m maksimalno odstupanje 1.4%.

Tablica 6. Relativno odstupanje ovisno o broju trokuta

T		m	1	3	5	7	9
Volumen	50vl,50točaka	m³	302.51	1459.44	3175.08	5334.57	7731.17
	10vl,10točaka		-4.23%	1.58%	1.53%	1.18%	0.93%
Awl	50vl,50točaka	m²	434.47	701.44	1005.88	1144.28	1248.89
	10vl,10točaka		-0.30%	0.49%	0.44%	0.46%	0.33%
Xwl	50vl,50točaka	m	53.04	50.21	44.0	43.87	44.308
	10vl,10točaka		-0.01%	0.02%	0.04%	0.03%	0.12%
KB	50vl,50točaka	m	0.592	1.769	3.011	4.23	5.40
	10vl,10točaka		3.27%	0.95%	0.33%	0.33%	0.33%
Xcg	50vl,50točaka	m	53.50	51.95	49.04	46.94	46.05
	10vl,10točaka		-0.16%	0.09%	0.35%	0.2762%	0.15%
lb	50vl,50točaka	m⁴	1704	5299	11097	15560	19394
	10vl,10točaka		-3.95%	2.4%	2.19%	1.96%	1.36%
ll	50vl,50točaka	m⁴	134003	292802	593998	720463	824892
	10vl,10točaka		-0.77%	1.04%	0.97%	1.00%	0.87%
Kmo	50vl,50točaka	m	6.22	5.4	6.50	7.14	7.91
	10vl,10točaka		0.56%	0.22%	0.20%	0.13%	0.09%
KMI	50vl,50točaka	m	443.56	202.395	190.0	139.28	112.1
	10vl,10točaka		3.33%	0.54%	0.55%	0.18%	0.07%
CB	50vl,50točaka	/	0.465	0.465	0.503	0.551	0.579
	10vl,10točaka		-4.26%	1.31%	1.41%	1.10%	0.69%
CP	50vl,50točaka	/	0.642	0.609	0.644	0.694	0.724
	10vl,10točaka		5.31%	0.81%	0.15%	0.29%	0.42%
CX	50vl,50točaka	/	0.724	0.764	0.782	0.793	0.801
	10vl,10točaka		-10.03%	2.27%	1.29%	0.63%	0.50%

4.2. Usporedba hidrostatičkih karakteristika modela računatih različitim softverima

Tablica 7. prikazuje usporedbu hidrostatičkih podataka dobivenim triangularizacijom s 50 vodnih linija i 50 točaka po vodnoj liniji, s numeričkim proračunom rađenim u Python programskom sučelju uz korištenje podataka dobivenih u programu Rhinoceros[1] i s podacima dobivenim iz Orca3D programskog dodatka aplikaciji Rhinoceros[1]. Potrebno je imati na umu da je forma u aplikaciji Rhinoceros generirana na osnovu desetak vodnih linija.

Tablica 8. prikazuje odstupanje svake vrijednosti od vrijednosti dobivenih iz programa Orca3D. Vrijednosti volumena najviše odstupaju, to odstupanje se može smanjiti povećanjem broja točaka i vodnih linija. Na najnižoj visini vodne linije kod volumena nalazimo odstupanje od 5%, slična greška se zato i očitava kod vrijednosti izvedenih iz volumena. To odstupanje bi se smanjilo kada bi smanjili broj vodnih linija i točaka na tim visinama, kako bi se točnost zapisa forme približili onoj koja je korištena za generiranje forme u aplikaciji Rhinoceros.

Tablica 7. Usporedba hidrostatičkih podataka

T		m	1	3	5	7	9
Volumen	Orca 3D	m ³	288.10	1444.51	3161.99	5320.92	7716.79
	Iz rada		302.51	1459.44	3175.08	5334.58	7731.18
	iz [1]		289.74	1447.49	3166.49	5326.65	7723.54
Awl	Orca 3D	m ²	434.32	700.95	1005.91	1144.17	1248.58
	Iz rada		434.47	701.44	1005.88	1144.28	1248.89
	iz [1]		434.54	701.62	1006.36	1144.59	1249.00
Xwl	Orca 3D	m	53.045	50.214	43.999	43.865	44.299
	Iz rada		53.041	50.215	44.018	43.877	44.308
	iz [1]		53.041	50.215	44.007	43.873	44.307
KB	Orca 3D	m	0.605	1.782	3.023	4.24	5.412
	Iz rada		0.592	1.769	3.011	4.23	5.403
	iz [1]		0.603	1.781	3.022	4.238	5.41
Xcg	Orca 3D	m	53.993	52.027	49.019	46.918	46.033
	Iz rada		53.509	51.953	49.041	46.94	46.052
	iz [1]		54.041	52.043	49.025	46.924	46.04
Ib	Orca 3D	m ⁴	1709.8	5296.2	11128.6	15593.6	19426.8
	Iz rada		1704.7	5299.4	11097.6	15560.8	19394.3
	iz [1]		1712.2	5310.4	11142.4	15608.2	19442.0
Il	Orca 3D	m ⁴	134003	292646	594217	720483	824686
	Iz rada		134004	292803	593999	720463	824893
	iz [1]		134099	292990	594542	720848	825119
Kmo	Orca 3D	m	6.539	5.449	6.543	7.17	7.929
	Iz rada		6.227	5.4	6.507	7.147	7.912
	iz [1]		6.512	5.449	6.541	7.168	7.927
KMI	Orca 3D	m	465.72	204.37	190.95	139.64	112.28
	Iz rada		443.56	202.39	190.09	139.28	112.10
	iz [1]		463.42	204.19	190.78	139.57	112.24
JZ	Orca 3D	t	4.452	7.185	10.311	11.728	12.798
	Iz rada		4.453	7.190	10.310	11.729	12.801
	iz [1]		4.454	7.192	10.315	11.732	12.802
CB	Orca 3D	/	0.453	0.472	0.51	0.558	0.578
	Iz rada		0.465	0.465	0.503	0.551	0.579
	iz [1]		0.445	0.462	0.502	0.55	0.579
CP	Orca 3D	/	0.635	0.626	0.66	0.712	0.728
	Iz rada		0.642	0.609	0.644	0.694	0.724
	iz [1]		0.613	0.604	0.642	0.693	0.723
CX	Orca 3D	/	0.712	0.754	0.772	0.784	0.794
	Iz rada		0.724	0.764	0.782	0.793	0.801
	iz [1]		0.726	0.764	0.782	0.793	0.801

Tablica 8. Relativno odstupanje hidrostatskih podataka

T		m	1	3	5	7	9
Volumen	Orca 3D	m ³	288.10	1444.51	3161.99	5320.92	7716.79
	Iz rada		4.76%	1.02%	0.41%	0.25%	0.18%
	iz [1]		0.57%	0.21%	0.14%	0.11%	0.09%
Awl	Orca 3D	m ²	434.32	700.95	1005.90	1144.16	1248.58
	Iz rada		0.04%	0.07%	-0.003%	0.01%	0.02%
	iz [1]		0.05%	0.09%	0.045%	0.04%	0.03%
Xwl	Orca 3D	m	53.04	50.21	43.99	43.86	44.29
	Iz rada		-0.008%	0.002%	0.043%	0.027%	0.020%
	iz [1]		-0.008%	0.002%	0.018%	0.018%	0.018%
KB	Orca 3D	m	0.605	1.782	3.023	4.240	5.412
	Iz rada		-2.19%	-0.73%	-0.39%	-0.23%	-0.17%
	iz [1]		-0.33%	-0.05%	-0.03%	-0.05%	-0.04%
Xcg	Orca 3D	m	53.99	52.03	49.02	46.92	46.03
	Iz rada		-0.90%	-0.14%	0.04%	0.05%	0.04%
	iz [1]		0.09%	0.03%	0.01%	0.01%	0.02%
Ib	Orca 3D	m ⁴	1709.76	5296.20	11128.63	15593.60	19426.77
	Iz rada		-0.30%	0.06%	-0.28%	-0.21%	-0.17%
	iz [1]		0.14%	0.27%	0.12%	0.09%	0.08%
II	Orca 3D	m ⁴	134003	292646	594217	720483	824686
	Iz rada		0.0005%	0.053%	-0.037%	-0.003%	0.025%
	iz [1]		0.072%	0.117%	0.055%	0.051%	0.053%
KMo	Orca 3D	m	6.539	5.449	6.543	7.170	7.929
	Iz rada		-5.01%	-0.91%	-0.55%	-0.32%	-0.21%
	iz [1]		-0.41%	0.00%	-0.03%	-0.03%	-0.02%
KMI	Orca 3D	m	465.72	204.37	190.95	139.64	112.28
	Iz rada		-4.99%	-0.98%	-0.45%	-0.26%	-0.16%
	iz [1]		-0.50%	-0.09%	-0.09%	-0.06%	-0.04%
JZ	Orca 3D	t	4.452	7.185	10.311	11.728	12.798
	Iz rada		0.022%	0.070%	-0.010%	0.009%	0.023%
	iz [1]		0.045%	0.097%	0.039%	0.034%	0.031%
CB	Orca 3D	/	0.453	0.472	0.510	0.558	0.578
	Iz rada		2.58%	-1.50%	-1.39%	-1.27%	0.17%
	iz [1]		-1.79%	-2.16%	-1.59%	-1.45%	0.17%
CP	Orca 3D	/	0.635	0.626	0.660	0.712	0.728
	Iz rada		1.09%	-2.79%	-2.48%	-2.59%	-0.55%
	iz [1]		-3.58%	-3.64%	-2.80%	-2.74%	-0.69%
CX	Orca 3D	/	0.712	0.754	0.772	0.784	0.794
	Iz rada		1.65%	1.31%	1.28%	1.13%	0.87%
	iz [1]		1.93%	1.31%	1.28%	1.13%	0.87%

5. DIJAGRAMNI LIST

5.1. Upotreba i značenje dijagramnog lista

Rezultati proračuna plovnosti broda ucrtavaju se u formi krivulja u jedan dijagram koji se naziva „Dijagramni list“. Kako brod za vrijeme svoje eksploatacije mijenja gaz, već prema količini ukrcanog tereta i potrošku zaliha, potrebno je sve veličine, koje su karakteristične za plovnost broda, dakle istisnina i koordinate njegovog težišta, te položaje poprečnih i uzdužnih metacentara, unijeti u ovisnosti o gasu broda. Osim toga u dijagramni list se obično unose i površine, položaji težišta i momenti tromosti površina vodnih linija, te površine i statički momenti površina rebara, budući da se pomoću njih računaju gore navedene veličine karakteristične za plovnost broda[1].

Dijagramni list služi projektantu kao osnova za daljnje proračune, na primjer, za određivanje volumena skladišta, za proračun plovnosti kod prodora vode, za određivanje razmaka nepropusnih pregrada, za proračun porinuća itd.

Dijagramni list služi i pomorcima za određivanje gazova, odnosno trima broda kod ukrcaja ili iskrcaja tereta, pa se u tu svrhu u njega ucrtavaju i krivulje koje daju ovisnost jediničnog momenta trima, ukupnog trima za moment trima od 100 tm, te tona po centimetru zagažaja, u ovisnosti od gaza broda.

Osim spomenutih krivulja dijagramni list obično sadrži i koeficijente punoće brodske forme, nanesene u ovisnosti od gaza i razne skale iz kojih se može očitati gaz i nosivost broda.

Dijagramni list ne sadrži položaje težišta sustava budući, da oni ovise o razmještanju težina na brodu i ne mogu se u pojedinostima predvidjeti. Dijagramni list sadrži samo veličine koje karakteriziraju brodsku formu, a ne daje nikakve podatke o položajima pojedinih težina, ni o položaju težišta sustava. Za svaki određeni slučaj opterećenja broda, položaj težišta sustava treba posebno proračunati te ucrtati taj položaj u dijagramni list. Tek onda se može iz dijagramnog lista, pomoću krivulja poprečnih odnosno uzdužnih metacentara, odrediti poprečna, odnosno uzdužna početna metacentarska visina za određeni gaz.

5.2. Crtanje dijagramnog lista

Osnovni problem prilikom crtanja dijagramnog lista je taj da nisu sve karakteristike istog reda veličine, nego variraju od reda deset na prvu do deset na petu. Zato je za svaku krivulju potrebno odrediti odgovarajuće mjerilo. Postupak određivanja mjerila[1]:

- 1) odrediti u kojem rasponu se kreće pojedina karakteristika
- 2) odrediti maksimalnu vrijednost svake karakteristike, iznimka su KM_L i KM_0
- 3) odrediti na kojem formatu papira se crta dijagram
- 4) odrediti položaj maksimalne vrijednosti pojedine karakteristike na papiru, za KM_L i KM_0 se određuje minimalna vrijednost
- 5) mjerilo je jednako kvocijentu maksimalne (minimalne kod KM_L i KM_0) vrijednosti svake karakteristike i položaju na papiru
- 6) koordinate uzdužnih težišta volumena i vodnih linija te jedinični moment trima crtaju se od glavnog rebra
- 7) koordinate koeficijenta brodske forme crtaju se od krmene okomice
- 8) ostale hidrostatske karakteristike ucrtavaju se od krmene okomice

Lijeva koordinatna os predstavlja krmenu okomicu, desna pramčanu okomicu, a srednja os predstavlja položaj glavnog rebra. Vrijednosti se očitavaju mjerenjem horizontalne udaljenosti od sjecišta gaza s krivuljom željene karakteristike do odgovarajuće osi, te množenjem tog očitavanja sa pripadajućim mjerilom.

Sve skale u dijagramnom listu crtaju se obično u metričkim i engleskim mjernim jedinicama, te za specifičnu težinu mora i slatke vode (budući da mnoge važne luke leže na ušćima velikih rijeka).

Osnovna skala u dijagramnom listu je skala za gaz broda. Da se za kose vodne linije može očitati gaz na pramcu i na krmi, skale za gaz postavljaju se i na pramcu i na krmi broda. Uz skalu za gaz redovno se još ucrtava i skala za nadvođe broda. Nadvođe broda dobiva se ako se na visinu broda doda debljina opločenja i pokrova palube, pa se od tako dobivene vrijednosti oduzima konstrukcijski gaz broda na dotičnoj plovnoj liniji.

Radi upotrebe dijagramnog lista u eksploataciji broda u njega se unose stvarni, a ne konstrukcijski gazovi. Zbog toga se linija nultog gaza nalazi za visinu kobilice ispod osnovke VL_0 (koja prolazi kroz gornji brid kobilice).

Radi lakše upotrebe dijagramnog lista u eksploataciji broda, u njega se još ucertavaju i skale paralelnog zagažaja u t/cm, te skale jediničnog momenta trima. One se konstruiraju iz odgovarajućih krivulja zagažaja i jediničnog momenta trima u dijagramnom listu.

5.3. Očitavanje vrijednosti u dijagramnom listu

Od krmene okomice očitava se:

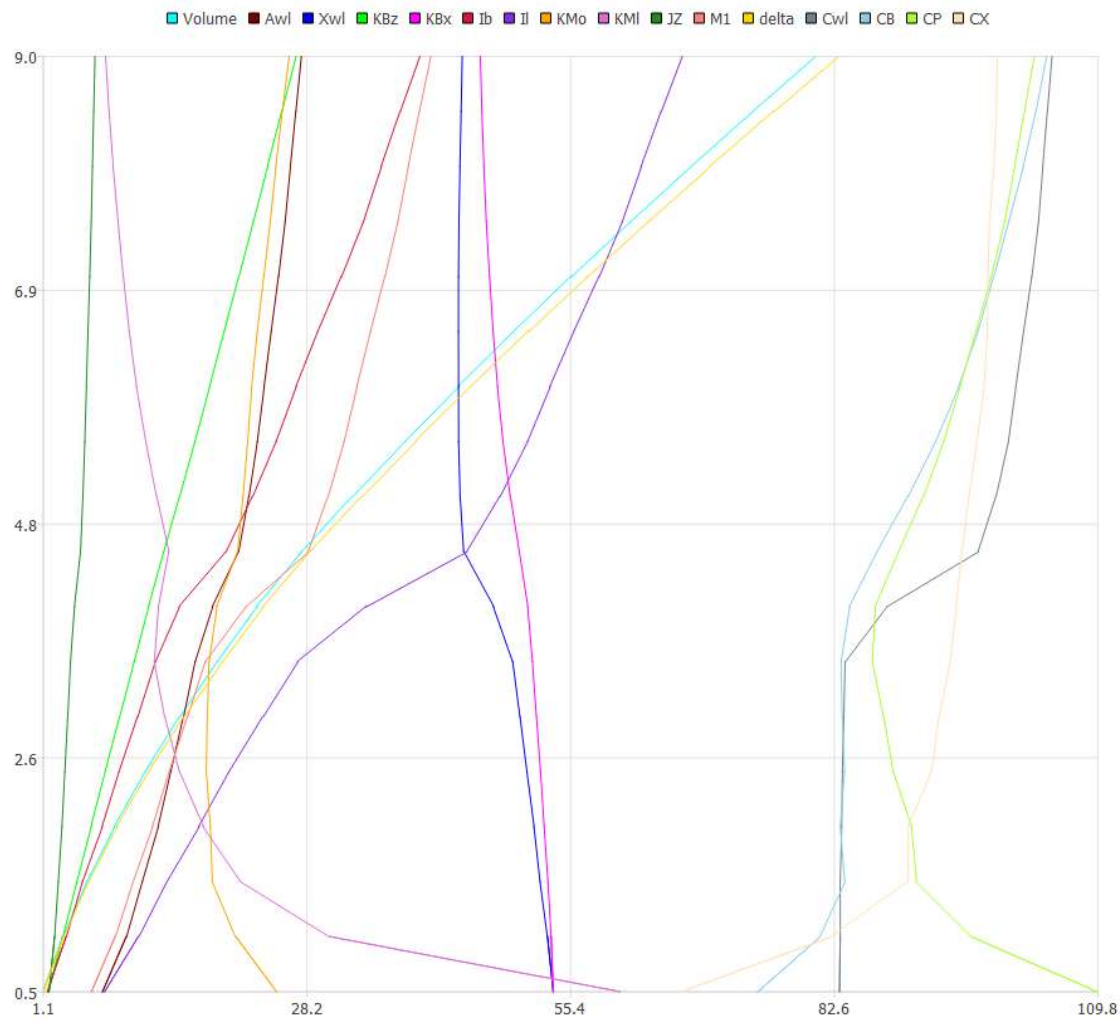
- volumen na rebrima
- volumen s privjescima
- istisnina s privjescima u morskoj vodi
- istisnina broda
- visina uzdužnog metacentra iznad osnovice
- visina poprečnog metacentra iznad osnovice
- poprečni moment inercije vodne linije
- uzdužni moment inercije vodne linije
- površina vodne linije
- jedinični zagažaj
- oplakana površina (rijetko).

Od glavnog rebra očitava se:

- položaj težišta istisnine po duljini
- položaj težišta vodne linije po duljini
- jedinični moment trima.

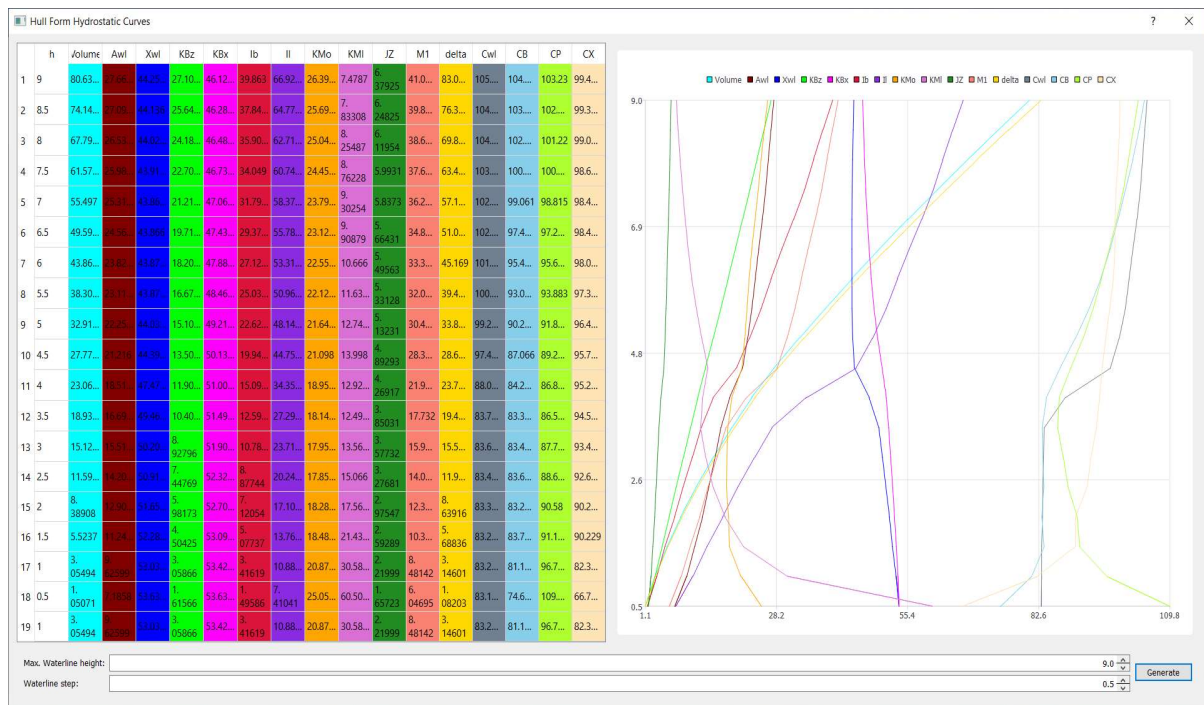
Od pramčane okomice očitava se:

- koeficijent punoće
- uzdužni prizmatički koeficijent
- koeficijent najvećeg rebra
- koeficijent vodne linije.



Slika 9. Dijagramni list

Slika 9. prikazuje dobiveni dijagramni list. Karakteristike su skalirane odgovarajućim mjerilom, kako se ne bi preklapale i kako bi se njihove vrijednosti mogle lakše očitati. Graf je izrađen u istoj aplikaciji u kojoj se vizualizira i forma, primjenom Python modula PySide2.



Slika 10. Prikaz sučelja koji generira dijagramni list

Tablica na slici 10. prikazuje hidrostatke vrijednosti podijeljene s mjerilom. S različitim bojama povezujemo stupac određene karakteristike sa svojom krivuljom na dijagramnom listu. Na dnu sučelja može se varirati maksimalna visina do koje se računaju hidrostatske karakteristike i korak s kojim se dijeli maksimalna visina.

Tablica 9. Mjerilo

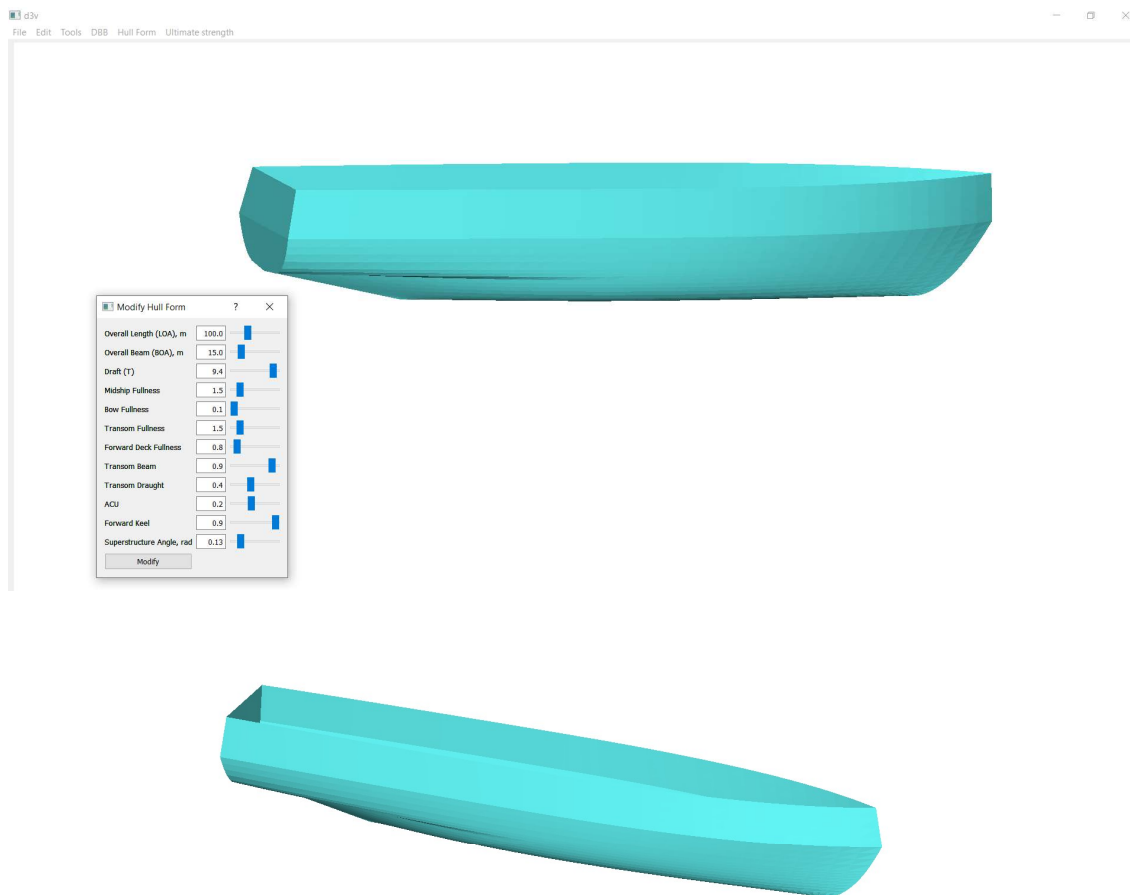
V	A _{wl}	X _{wl}	KB _z	KB _x	I _b	I _l	KM ₀	KM _L	JZ	M1	delta	C _{wl}	CB	CP	CX
95	45	1	0.2	1	480	12220	0.3	15	2	200	90	0.008	0.0055	0.007	0.008

Da bi očitali vrijednosti sa dijagramnog liste potrebno je na željenoj visini očitati vrijednost karakteristike na x osi i zatim taj broj pomnožiti sa mjerilom. X os nema mjernu jedinicu nego svaka karakteristika nosi svoju.

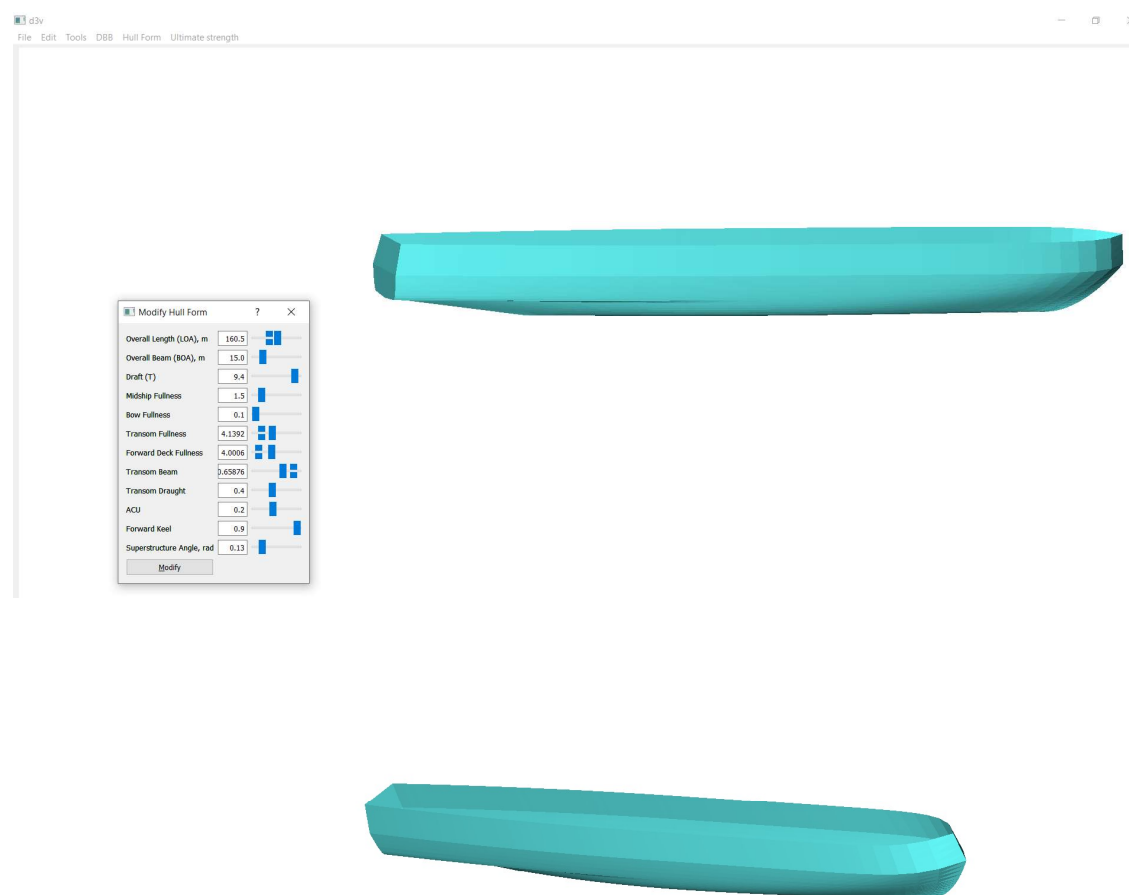
5.4. Grafička kontrola parametara analitički zadane forme broda

Slika 11. prikazuje sučelje za promjenu parametara brodske forme s minimalnim i maksimalnim ograničenjima. Pomicanjem klizača lijevo ili desno, odabrana vrijednost se smanjuje ili povećava, pritiskom na tipku 'modify' nove vrijednosti se unose u kod i promijenjena forma je vizualizirana.

Slika 12. prikazuje formu sa nekoliko izmijenjenih parametara. Brodu je promijenjena duljina, punoća prednje palube, punoća zrcala i širina zrcala.



Slika 11. Prikaz prozora za modifikaciju brodske forme



Slika 12. Prikaz modificirane forme

6. ZAKLJUČAK

U ovom radu je pomoću analitički zadanih parametara napravljena triangularizacija brodske forme, a zatim su provedeni numerički proračuni. Triangularizirana forma se vizualizirala u programu otvorenog koda *linaetal-fsb/d3v*. U Python programskom sučelju su dobiveni hidrostatički podaci kao što su volumen istisnine, težište vodne linije, površina vodne linije te ostali podaci potrebni za izradu dijagramnog lista. Dijagramni list pokazuje ovisnost svih hidrostatičkih karakteristika o visini koja se unutar sučelja jednostavno mijenja.

Analizirana je promjena hidrostatičkih vrijednosti promjenom broja vodnih linija i broja točaka na njima te su uspoređeni svi dobiveni rezultati.

Vidi se da je triangularizacija brodske forme koristan i relativno jednostavan način za izradu modela brodske forme i njenu vizualizaciju. Također je izračun hidrostatičkih karakteristika lako provediv i vidi se iz tih podataka da je rezultat stabilan i točan već za mali broj vodnih linija i točaka na njima.

LITERATURA

- [1] Lovro Ante Rebić: Završni rad, Fakultet strojarstva i brodogradnje, Zagreb, 2020.
- [2] Filipa Marques Sanches: Parametric Modelling of Hull Form for Ship Optimization, Thesis to obtain the Master of Science Degree in Naval Architecture and Marine Engineering, 2016.
- [3] <http://dbb.ucl.im/>
- [4] <https://www.graphics.rwth-aachen.de/software/openmesh/intro/>
- [5] Marko Buconić: Završni rad, Fakultet strojarstva i brodogradnje, Zagreb, 2020.
- [6] <https://math.stackexchange.com/questions/2371139/volume-of-truncated-prism>
- [7] <https://byjus.com/centroid-formula/>
- [8] <https://www.fsb.unizg.hr/geometrija.broda/>

PRILOZI

- I. CD-R disc
- II. Python programski kod za računanje hidrostatičkih karakteristika broda

File - C:\Users\Luka\Desktop\zavrsni rad\3v\src\modules\commands\hullform.py

```
1 from iohandlers import IOHandler
2 from signals import Signals
3 from geometry import Geometry
4 import openmesh as om
5 import os
6 import numpy as np
7 import csv
8 import math as Math
9 import time
10
11 # import matplotlib.pyplot as plt
12
13 class HullFormMeshQuality:
14     def __init__(self):
15         self._numWL = 10
16         self._numPnWLhalf = 10
17         self._distPolyOrder=3
18
19     @property
20     def numPointsWLhalf(self):
21         return self._numPnWLhalf
22
23     def _getDistribution(self, maxv, minv, n, pot):
24         x = [0.0] * n
25         for i in range(n):
26             fi = float(i)
27             fn1 = float(n - 1)
28             x[i] = fi ** pot / fn1 ** pot * (maxv - minv) + minv
29         x.reverse()
30         return x
31
32     def genWLPositions(self,hWL_top, hWL_bottom ):
33         wlPos = self._getDistribution(hWL_top, hWL_bottom, self._numWL, self._distPolyOrder)
34         return wlPos
35     def genWLPositionsUsingObligatory(self,obligatoryLines:list ):
36         testLines = self._getDistribution(obligatoryLines[0], obligatoryLines[-1], self._numWL,
self._distPolyOrder)
37
38         nol=len(obligatoryLines)
39         wlPos=[]
40         i1TL=0
41         for iol in range(1,nol):
42             hmax=obligatoryLines[iol-1]
43             hmin = obligatoryLines[iol]
44             numWL=0
45             for iTL in range(i1TL,self._numWL):
46                 if testLines[iTL] < hmax:
47                     if testLines[iTL] > hmin:
48                         numWL =numWL + 1
49                     else:
50                         i1TL=iTL
51                     break
52             wlPosi = self._getDistribution(hmax, hmin, numWL+2, 1)
53             for wl in wlPosi:
54                 if len(wlPos)==0:
55                     wlPos.append(wl)
56                 elif wl < wlPos[-1]:
57                     wlPos.append(wl)
58         return wlPos
59
60
61 class HullForm(Geometry):
62     def __init__(self, fileName):
63         super().__init__()
64         self.filename = fileName
65         self.shipdata = {}
66         self.pdecks = []
67         self.pbulkheads = []
68         self.hfmq = HullFormMeshQuality()
69         results = self.readShipData()
70         self.shipdata = results[0]
71         self.pdecks = results[1]
72         self.pbulkheads = results[2]
73         self.h = [] # positive y waterlines
74         self.wlinesNeg = [] # negative y waerLines
```

File - C:\Users\Luka\Desktop\zavrsni rad\d3v\src\modules\commands\hullform.py

```
75         self.wlKeel = [] # keel waterline (one waterline)
76         self.generateMesh()
77
78     def generateMesh(self):
79
80         hmax=self.pdecks[0]
81         #hmax=9.4
82         #wlPos=self.hfmq.genWLPositions(hmax, 0)
83         transomTip = self.shipdata["draft_val"] * self.shipdata["td_val"]
84         obligatoryWL= []
85         for dh in self.pdecks:
86             obligatoryWL.append(dh)
87         obligatoryWL.append(transomTip)
88         obligatoryWL.sort(reverse=True)
89
90         wlPos = self.hfmq.genWLPositionsUsingObligatory(obligatoryWL)
91         #results = self.hullGen(self.shipdata, wlPos, self.hfmq.numPointsWLhalf)
92         lines = self.hullGen(self.shipdata, wlPos, self.hfmq.numPointsWLhalf)
93         self.wlinesPos = lines[0] # positive y waterlines
94         self.wlinesNeg = lines[1] # negative y waterlines
95         self.wlKeel = lines[2] # keel waterline (one waterline)
96         self.mesh = self.genHullFormMeshPP(lines)
97         pass
98
99
100    def getResultsOld(self,h,seaDensity):
101        tsStart = time.perf_counter()
102        results = []
103        fvs = self.mesh.fv_indices().tolist()
104        points = self.mesh.points().tolist()
105        xmf = 50
106        # h=9
107        bcwl = self.getBasicDataUsingTrianglesProjectedToWaterline(h,xmf,fvs,points)
108        h = bcwl[0]
109        volume = bcwl[1]
110        area = bcwl[2]
111        Xwl = bcwl[3]
112        KBz = bcwl[4]
113        KBx = bcwl[5]
114        Ib = bcwl[6]
115        Il = bcwl[7]
116        Lwl = bcwl[8]
117        Bwl = bcwl[9]
118        mfarea = bcwl[10]
119
120
121        hsdata = self.getHydrostaticData(seaDensity,h,volume,area,Ib, Il,KBz,Lwl,Bwl,mfarea)
122        dtAll = time.perf_counter() - tsStart
123        print("Hydrostatic results calc time:", dtAll)
124        results = bcwl+hsdata
125        print(results)
126        return results
127
128    def getResults(self,h,seaDensity):
129        tsStart = time.perf_counter()
130        results = []
131        fvs = self.mesh.fv_indices().tolist()
132        points = self.mesh.points().tolist()
133        xmf = 50
134        # h=9
135        bcwl = self.getBasicDataUsingTrianglesProjectedToWaterline(h,xmf,fvs,points)
136        h = bcwl[0]
137        volume = bcwl[1]
138        area = bcwl[2]
139        Xwl = bcwl[3]
140        KBz = bcwl[4]
141        KBx = bcwl[5]
142        Ib = bcwl[6]
143        Il = bcwl[7]
144        Lwl,Bwl = self.getLwlBwl(h,fvs,points)
145        mfarea = self.getMainFrameArea(xmf, h, fvs, points)
146        hsdata = self.getHydrostaticData(seaDensity,h,volume,area,Ib,Il,KBz,Lwl,2*Bwl,mfarea)
147        dtAll = time.perf_counter() - tsStart
148        print("Hydrostatic results calc time:", dtAll)
149        results = bcwl+hsdata
```

File - C:\Users\Luka\Desktop\zavrsni rad\ld3v\src\modules\commands\hullform.py

```
150     print(results)
151     return results
152
153     def getMainFrameArea(self,x,h,fvs,points):
154         mfpnts = self.getSortedPointsOnAxisAlignedPlane(x, fvs, points, 0)
155         area=0
156         a = len(mfpnts)
157         for i in range(0,a-2,1):
158             h1 = mfpnts[i][2]
159             h2 = mfpnts[i+2][2]
160             b1 = mfpnts[i][1]
161             b2 = mfpnts[i+2][1]
162             if h1<=h and h2<=h:
163                 area = area + 1/2*abs(h2-h1)*(abs(b1)+abs(b2))
164             if h2>h and h1<h:
165                 point = self.getIntersectionPoint(mfpnts[i+2],mfpnts[i],h,2)
166                 H2 = point[2]
167                 area = area + 1/2*abs(H2-h1)*(abs(b1)+abs(b2))
168
169         return area
170
171     def getSortedPointsOnAxisAlignedPlane(self, x, fvs, points, os):
172         mfpnts=[]
173         lpr = []
174         lpl = []
175         for fv in fvs:
176             lpr.clear()
177             lpl.clear()
178             for iv in fv:
179                 p = points[iv]
180                 if p[os] < x:
181                     lpl.append(iv)
182                 elif p[os] > x:
183                     lpr.append(iv)
184                 else:
185                     mfpnts.append(p)
186
187             if len(lpl)>0 and len(lpr) > 0:
188                 if len(lpl) < len(lpr):
189                     mfpnts.append(self.getIntersectionPoint(points[lpl[0]],points[lpr[0]],x,os
190 ))
191                     mfpnts.append(self.getIntersectionPoint(points[lpl[0]], points[lpr[1]], x
192 , os))
193                 elif len(lpl) > len(lpr):
194                     mfpnts.append(self.getIntersectionPoint(points[lpl[0]],points[lpr[0]],x,os
195 ))
196                     mfpnts.append(self.getIntersectionPoint(points[lpl[1]], points[lpr[0]], x
197 , os))
198                 else:
199                     mfpnts.append(self.getIntersectionPoint(points[lpl[0]],points[lpr[0]],x,os
200 ))
201                 pass
202
203         #mfpnts=[[0,1,1],[0,2,21],[1,1,2],[10,0,0]]
204
205         import itertools
206         mfpnts.sort()
207         mftemp = list(mfpnts for mfpnts,_ in itertools.groupby(mfpnts))
208         mfpnts = mftemp
209
210         mfpnts = sorted(mfpnts, key=lambda p: p[2])
211         return mfpnts
212
213     def getHydrostaticData(self,seaDensity,h,volume,area,Ib, I1, KBz,Lw1,Bw1,mfarea):
214
215         MoB = Ib / volume
216         KMo = MoB + KBz
217         M1B = I1 / volume
218         KM1 = M1B + KBz
219         JZ = 0.01 * area * seaDensity
220         M1 = I1 / Lw1
221         delta = volume / seaDensity
222
223         Cw1 = area / (Lw1 * Bw1)
```

File - C:\Users\Luka\Desktop\zavrsni rad\ld3v\src\modules\commands\hullform.py

```
220     CB = volume / (Lwl * Bwl * h)
221
222     CP = volume / (mfarea * Lwl)
223     CX = mfarea / (Bwl * h)
224     # results = [MoB,KMo,MLB,KML,JZ,CwL,CB,CP,CX]
225     return KMo, KML, JZ, M1, delta, Cwl, CB, CP, CX
226
227 def getHydrostaticDataOld(self, seaDensity, h, volume, area, Ib, I1, KBz, Lwl, Bwl, mfarea):
228
229     MoB = Ib / volume
230     KMo = MoB + KBz
231     M1B = I1 / volume
232     KML = M1B + KBz
233     JZ = 0.01 * area * seaDensity
234
235     Cwl = area / (Lwl * Bwl)
236     CB = volume / (Lwl * Bwl * h)
237
238     CP = volume / (mfarea * Lwl)
239     CX = mfarea / (Bwl * h)
240     # results = [MoB,KMo,MLB,KML,JZ,CwL,CB,CP,CX]
241
242     return MoB, KMo, M1B, KML, JZ, Cwl, CB, CP, CX
243
244 def getLwlBwl(self, h, fvs, points):
245     wlpoints = self.getSortedPointsOnAxisAlignedPlane(h, fvs, points, 2)
246     wlpoints = sorted(wlpoints, key=lambda p: p[1])
247     Bwl = wlpoints[-1][1] - wlpoints[0][1]
248
249     wlpoints = sorted(wlpoints, key=lambda p: p[0])
250
251     Lwl = wlpoints[-1][0] - wlpoints[0][0]
252     return Lwl, Bwl
253
254 def getBasicDataUsingTrianglesProjectedToWaterline(self, h, x, fvs, points):
255     mesh = self.mesh
256     lpowl = []
257     lpbwl = []
258     Ib = 0
259     KBz = 0
260     KBx = 0
261     vol = 0
262     Awl = 0
263     Xwl = 0
264     p = []
265     r = []
266     cgTriamid = []
267     a = self.getXwl(h)
268     for fh in fvs: # facet handle
269         p.clear()
270         r.clear()
271         cgTriamid.clear()
272         lpowl.clear()
273         lpbwl.clear()
274         i = 0
275         for vh in fh: # vertex handle
276             p.append(points[vh])
277             if p[i][2] > h:
278                 lpowl.append(i)
279
280             else:
281                 lpbwl.append(i)
282             i = i + 1
283         if len(lpowl) < 1:
284             # A
285             # Ax = points[fvs[fh][0]][0]
286             Ax = p[0][0]
287             Ay = p[0][1]
288             Az = p[0][2]
289             # B
290             Bx = p[1][0]
291             By = p[1][1]
292             Bz = p[1][2]
293             # C
294             Cx = p[2][0]
```

```

295         Cy = p[2][1]
296         Cz = p[2][2]
297         areaXYPlane = self.calcArea2DTria(Ax, Ay, Bx, By, Cx, Cy)
298
299         # Volume
300         hA = h - Az
301         hB = h - Bz
302         hC = h - Cz
303         vol = vol + 1 / 3 * abs(areaXYPlane) * (hA + hB + hC)
304
305         # Ib IL
306         r.append(self.TezisteTrokuta(Ax, Ay, h, Bx, By, h, Cx, Cy, h))
307         Ib = Ib + r[0][1] ** 2 * areaXYPlane
308
309         # Kbz, KBx
310         hsr = (Az + Bz + Cz) / 3
311         cgTriaMid.append(self.TezisteTrokuta(Ax, Ay, (h - hsr), Bx, By, (h - hsr), Cx,
Cy, (h - hsr)))
312         KBz = KBz + abs(1 / 2 * (Ax * (By - Cy) + Bx * (Cy - Ay) + Cx * (Ay - By))) * (h
- hsr) * (
313             hsr + (h - hsr) / 2)
314         KBx = KBx + abs(1 / 2 * (Ax * (By - Cy) + Bx * (Cy - Ay) + Cx * (Ay - By))) * (h
- hsr) * (
315             cgTriaMid[0][0])
316
317         # AwL
318         area = self.calcArea2DTria(Ax, Ay, Bx, By, Cx, Cy)
319
320         # XwL
321         XwL = XwL + (r[0][0] * abs(1 / 2 * (Ax * (By - Cy) + Bx * (Cy - Ay) + Cx * (Ay
- By))))
322
323         pass
324         elif len(lpowl) == 1:
325             # 2 trokuta Ib, IL i XwL
326             lip = self.getIntersectionPoints(p[lpowl[0]], p[lpbwl[0]], p[lpbwl[1]], h, 2)
327             r.append(self.TezisteTrokuta(lip[0][0], lip[0][1], h, lip[1][0], lip[1][1], h, p
[lpbwl[1]][0],
328                 p[lpbwl[1]][1], h))
329             Ib = Ib + r[0][1] ** 2 * self.calcArea2DTria(lip[0][0], lip[0][1], lip[1][0],
lip[1][1], p[lpbwl[1]][0],
330                 p[lpbwl[1]][1])
331
332             XwL = XwL + r[0][0] * self.calcArea2DTria(lip[0][0], lip[0][1], lip[1][0], lip[1
][1], p[lpbwl[1]][0],
333                 p[lpbwl[1]][1])
334
335             r.append(self.TezisteTrokuta(lip[0][0], lip[0][1], h, p[lpbwl[1]][0], p[lpbwl[1
]][1], h, p[lpbwl[0]][0],
336                 p[lpbwl[0]][1], h))
337             Ib = Ib + r[1][1] ** 2 * self.calcArea2DTria(lip[0][0], lip[0][1], p[lpbwl[1]][0]
, p[lpbwl[1]][1],
338                 p[lpbwl[0]][0], p[lpbwl[0]][1])
339
340             XwL = XwL + r[1][0] * self.calcArea2DTria(lip[0][0], lip[0][1], p[lpbwl[1]][0],
p[lpbwl[1]][1],
341                 p[lpbwl[0]][0], p[lpbwl[0]][1])
342
343             # Kbz, KBx
344             hsr = (lip[0][2] + lip[1][2] + p[lpbwl[0]][2]) / 3
345             cgTriaMid.append(self.TezisteTrokuta(lip[0][0], lip[0][1], h, lip[1][0], lip[1][
1], h, p[lpbwl[1]][0],
346                 p[lpbwl[1]][1], h))
347             KBz = KBz + self.calcArea2DTria(lip[0][0], lip[0][1], lip[1][0], lip[1][1], p[
lpbwl[1]][0],
348                 p[lpbwl[1]][1]) * (h - hsr) * (hsr + (h - hsr
) / 2)
349             KBx = KBx + self.calcArea2DTria(lip[0][0], lip[0][1], lip[1][0], lip[1][1], p[
lpbwl[1]][0],
350                 p[lpbwl[1]][1]) * (h - hsr) * (cgTriaMid[0][0])
351
352             hsr = (lip[0][2] + p[lpbwl[1]][2] + p[lpbwl[0]][2]) / 3
353             cgTriaMid.append(
354                 self.TezisteTrokuta(lip[0][0], lip[0][1], h, p[lpbwl[1]][0], p[lpbwl[1]][1
], h, p[lpbwl[0]][0],

```

File - C:\Users\Luka\Desktop\završni rad\ld3v\src\modules\commands\hullform.py

```
355         p[lpbwl[0]][1], h))
356         KBz = KBz + self.calcArea2DTria(lip[0][0], lip[0][1], p[lpbwl[1]][0], p[lpbwl[1]
    ][1], p[lpbwl[0]][0],
357         p[lpbwl[0]][1]) * (h - hsr) * (hsr + (h - hsr
    ) / 2)
358         KBx = KBx + self.calcArea2DTria(lip[0][0], lip[0][1], p[lpbwl[1]][0], p[lpbwl[1]
    ][1], p[lpbwl[0]][0],
359         p[lpbwl[0]][1]) * (h - hsr) * (cgTriaMid[0][0])
360
361         # Volume
362         area1 = self.calcArea2DTria(lip[0][0], lip[0][1], p[lpbwl[0]][0], p[lpbwl[0]][1]
    ], p[lpbwl[1]][0],
363         p[lpbwl[1]][1])
364         area2 = self.calcArea2DTria(lip[0][0], lip[0][1], lip[1][0], lip[1][1], p[lpbwl[1]
    ][0], p[lpbwl[1]][1])
365         areaXYPlane = area1 + area2
366         vol = vol + 1 / 3 * (h - p[lpbwl[0]][2] + h - p[lpbwl[1]][2]) * area1
367         vol = vol + 1 / 3 * (h - p[lpbwl[1]][2]) * area2
368
369         # Awl
370         area = area1 + area2
371
372         pass
373     elif len(lpowl) == 2:
374         # 1 trokut Ib, Il
375         lip = self.getIntersectionPoints(p[lpbwl[0]], p[lpowl[0]], p[lpowl[1]], h, 2)
376         r.append(self.TezisteTrokuta(lip[0][0], lip[0][1], h, lip[1][0], lip[1][1], h, p
    [lpbwl[0]][0],
377         p[lpbwl[0]][1], h))
378         Ib = Ib + r[0][1] ** 2 * self.calcArea2DTria(lip[0][0], lip[0][1], lip[1][0],
    lip[1][1], p[lpbwl[0]][0],
379         p[lpbwl[0]][1])
380
381         # KBz, KBx
382         hsr = (lip[0][2] + lip[1][2] + p[lpbwl[0]][2]) / 3
383         cgTriaMid.append(self.TezisteTrokuta(lip[0][0], lip[0][1], h, lip[1][0], lip[1][1]
    ], h, p[lpbwl[0]][0],
384         p[lpbwl[0]][1], h))
385         KBz = KBz + self.calcArea2DTria(lip[0][0], lip[0][1], lip[1][0], lip[1][1], p[
    lpbwl[0]][0],
386         p[lpbwl[0]][1]) * (h - hsr) * (hsr + (h - hsr
    ) / 2)
387         KBx = KBx + self.calcArea2DTria(lip[0][0], lip[0][1], lip[1][0], lip[1][1], p[
    lpbwl[0]][0],
388         p[lpbwl[0]][1]) * (h - hsr) * (cgTriaMid[0][0])
389
390         # Volume
391         areaXYPlane = self.calcArea2DTria(lip[0][0], lip[0][1], lip[1][0], lip[1][1], p[
    lpbwl[0]][0],
392         p[lpbwl[0]][1])
393         vol = vol + 1 / 3 * (h - p[lpbwl[0]][2]) * areaXYPlane
394
395         # Awl
396         area = self.calcArea2DTria(lip[0][0], lip[0][1], lip[1][0], lip[1][1], p[lpbwl[0]
    ][0], p[lpbwl[0]][1])
397
398         # Xwl
399         Xwl = Xwl + r[0][0] * self.calcArea2DTria(lip[0][0], lip[0][1], lip[1][0], lip[1]
    ][1], p[lpbwl[0]][0],
400         p[lpbwl[0]][1])
401
402         pass
403     else:
404         area = 0
405         Awl = Awl + area
406
407     Xwl = Xwl / Awl
408     Il = float(self.getIl(h, Xwl))
409     KBz = KBz / vol
410     KBx = KBx / vol
411     Lwl, Bwl = self.getLwlBwl(h, fvs, points)
412     mfarea = self.getMainFrameArea(x, h, fvs, points)
413
414     return h, 2 * vol, 2 * Awl, Xwl, KBz, KBx, 2 * Ib, 2 * Il
415
```

File - C:\Users\Luka\Desktop\zavrsni rad\3v\src\modules\commands\hullform.py

```
416     def getBasicDataUsingTrianglesProjectedToWaterlineOld(self, h, x, fvs, points):
417         mesh = self.mesh
418         lpowl = []
419         lpbwl = []
420         Ib = 0
421         KBz = 0
422         KBx = 0
423         vol = 0
424         Awl = 0
425         Xwl = 0
426         p = []
427         r = []
428         cgTriaMid = []
429         a = self.getXwl(h)
430         for fh in fvs: # facet handle
431             p.clear()
432             r.clear()
433             cgTriaMid.clear()
434             lpowl.clear()
435             lpbwl.clear()
436             i = 0
437             for vh in fh: # vertex handle
438                 p.append(points[vh])
439                 if p[i][2] > h:
440                     lpowl.append(i)
441
442             else:
443                 lpbwl.append(i)
444                 i = i + 1
445             if len(lpowl) < 1:
446                 # A
447                 #Ax = points[fvs[fh][0]][0]
448                 Ax = p[0][0]
449                 Ay = p[0][1]
450                 Az = p[0][2]
451                 # B
452                 Bx = p[1][0]
453                 By = p[1][1]
454                 Bz = p[1][2]
455                 # C
456                 Cx = p[2][0]
457                 Cy = p[2][1]
458                 Cz = p[2][2]
459                 areaXYPlane = self.calcArea2DTria(Ax, Ay, Bx, By, Cx, Cy)
460
461                 # Volume
462                 hA = h - Az
463                 hB = h - Bz
464                 hC = h - Cz
465                 vol = vol + 1 / 3 * abs(areaXYPlane) * (hA + hB + hC)
466
467                 # Ib IL
468                 r.append(self.TezisteTrokuta(Ax, Ay, h, Bx, By, h, Cx, Cy, h))
469                 Ib = Ib + r[0][1] ** 2 * areaXYPlane
470
471                 #Kbz, KBx
472                 hsr = (Az + Bz + Cz) / 3
473                 cgTriaMid.append(self.TezisteTrokuta(Ax, Ay, (h - hsr), Bx, By, (h - hsr), Cx,
474 Cy, (h - hsr)))
475                 KBz = KBz + abs(1 / 2 * (Ax * (By - Cy) + Bx * (Cy - Ay) + Cx * (Ay - By))) * (h
476 - hsr) * (
477                     hsr + (h - hsr) / 2)
478                 KBx = KBx + abs(1 / 2 * (Ax * (By - Cy) + Bx * (Cy - Ay) + Cx * (Ay - By))) * (h
479 - hsr) * (
480                     cgTriaMid[0][0])
481
482                 #Awl
483                 area = self.calcArea2DTria(Ax, Ay, Bx, By, Cx, Cy)
484
485                 #Xwl
486                 Xwl = Xwl + (r[0][0] * abs(1 / 2 * (Ax * (By - Cy) + Bx * (Cy - Ay) + Cx * (Ay
487 - By))))
488
489             pass
490         elif len(lpowl) == 1:
```

```

487         # 2 trokuta Ib, Il i Xwl
488         lip = self.getIntersectionPoints(p[lpowl[0]], p[lpbwl[0]], p[lpbwl[1]], h, 2)
489         r.append(self.TezisteTrokuta(lip[0][0], lip[0][1], h, lip[1][0], lip[1][1], h, p[
lpbwl[1]][0], p[lpbwl[1]][1], h))
490         Ib = Ib + r[0][1] ** 2 * self.calcArea2DTria(lip[0][0], lip[0][1], lip[1][0], lip
[1][1], p[lpbwl[1]][0], p[lpbwl[1]][1])
491
492         Xwl = Xwl + r[0][0] * self.calcArea2DTria(lip[0][0], lip[0][1], lip[1][0], lip[1
][1], p[lpbwl[1]][0], p[lpbwl[1]][1])
493
494         r.append(self.TezisteTrokuta(lip[0][0], lip[0][1], h, p[lpbwl[1]][0], p[lpbwl[1
]][1], h, p[lpbwl[0]][0], p[lpbwl[0]][1], h))
495         Ib = Ib + r[1][1] ** 2 * self.calcArea2DTria(lip[0][0], lip[0][1], p[lpbwl[1]][0
], p[lpbwl[1]][1], p[lpbwl[0]][0], p[lpbwl[0]][1])
496
497         Xwl = Xwl + r[1][0] * self.calcArea2DTria(lip[0][0], lip[0][1], p[lpbwl[1]][0],
p[lpbwl[1]][1], p[lpbwl[0]][0], p[lpbwl[0]][1])
498
499         #Kbz, KBx
500         hsr = (lip[0][2] + lip[1][2] + p[lpbwl[0]][2]) / 3
501         cgTriaMid.append(self.TezisteTrokuta(lip[0][0], lip[0][1], h, lip[1][0], lip[1][
1], h, p[lpbwl[1]][0], p[lpbwl[1]][1], h))
502         KBz = KBz + self.calcArea2DTria(lip[0][0], lip[0][1], lip[1][0], lip[1][1], p[
lpbwl[1]][0], p[lpbwl[1]][1]) * (h - hsr) * (hsr + (h - hsr) / 2)
503         KBx = KBx + self.calcArea2DTria(lip[0][0], lip[0][1], lip[1][0], lip[1][1], p[
lpbwl[1]][0], p[lpbwl[1]][1]) * (h - hsr) * (cgTriaMid[0][0])
504
505         hsr = (lip[0][2] + p[lpbwl[1]][2] + p[lpbwl[0]][2]) / 3
506         cgTriaMid.append(self.TezisteTrokuta(lip[0][0], lip[0][1], h, p[lpbwl[1]][0], p[
lpbwl[1]][1], h, p[lpbwl[0]][0], p[lpbwl[0]][1], h))
507         KBz = KBz + self.calcArea2DTria(lip[0][0], lip[0][1], p[lpbwl[1]][0], p[lpbwl[1
]][1], p[lpbwl[0]][0], p[lpbwl[0]][1]) * (h - hsr) * (hsr + (h - hsr) / 2)
508         KBx = KBx + self.calcArea2DTria(lip[0][0], lip[0][1], p[lpbwl[1]][0], p[lpbwl[1
]][1], p[lpbwl[0]][0], p[lpbwl[0]][1]) * (h - hsr) * (cgTriaMid[0][0])
509
510         #Volume
511         area1 = self.calcArea2DTria(lip[0][0], lip[0][1], p[lpbwl[0]][0], p[lpbwl[0]][1
], p[lpbwl[1]][0], p[lpbwl[1]][1])
512         area2 = self.calcArea2DTria(lip[0][0], lip[0][1], lip[1][0], lip[1][1], p[lpbwl[
1]][0], p[lpbwl[1]][1])
513         areaXYPlane = area1 + area2
514         vol = vol + 1 / 3 * (h - p[lpbwl[0]][2] + h - p[lpbwl[1]][2]) * area1
515         vol = vol + 1 / 3 * (h - p[lpbwl[1]][2]) * area2
516
517         # Awl
518         area = area1 + area2
519
520         pass
521     elif len(lpowl) == 2:
522         # 1 trokut Ib, Il
523         lip = self.getIntersectionPoints(p[lpbwl[0]], p[lpowl[0]], p[lpowl[1]], h, 2)
524         r.append(self.TezisteTrokuta(lip[0][0], lip[0][1], h, lip[1][0], lip[1][1], h, p
[lpbwl[0]][0], p[lpbwl[0]][1], h))
525         Ib = Ib + r[0][1] ** 2 * self.calcArea2DTria(lip[0][0], lip[0][1], lip[1][0],
lip[1][1], p[lpbwl[0]][0], p[lpbwl[0]][1])
526
527         #KBz, KBx
528         hsr = (lip[0][2] + lip[1][2] + p[lpbwl[0]][2]) / 3
529         cgTriaMid.append(self.TezisteTrokuta(lip[0][0], lip[0][1], h, lip[1][0], lip[1][
1], h, p[lpbwl[0]][0], p[lpbwl[0]][1], h))
530         KBz = KBz + self.calcArea2DTria(lip[0][0], lip[0][1], lip[1][0], lip[1][1], p[
lpbwl[0]][0], p[lpbwl[0]][1]) * (h - hsr) * (hsr + (h - hsr) / 2)
531         KBx = KBx + self.calcArea2DTria(lip[0][0], lip[0][1], lip[1][0], lip[1][1], p[
lpbwl[0]][0], p[lpbwl[0]][1]) * (h - hsr) * (cgTriaMid[0][0])
532
533         #Volume
534         areaXYPlane = self.calcArea2DTria(lip[0][0], lip[0][1], lip[1][0], lip[1][1], p[
lpbwl[0]][0], p[lpbwl[0]][1])
535         vol = vol + 1 / 3 * (h - p[lpbwl[0]][2]) * areaXYPlane
536
537         #Awl
538         area = self.calcArea2DTria(lip[0][0], lip[0][1], lip[1][0], lip[1][1], p[lpbwl[0
]][0], p[lpbwl[0]][1])
539
540         #Xwl

```

```

541         Xw1 = Xw1 + r[0][0] * self.calcArea2DTria(lip[0][0], lip[0][1], lip[1][0], lip[1]
    ][1], p[lpbwl[0]][0], p[lpbwl[0]][1])
542
543         pass
544     else:
545         area = 0
546         Awl = Awl + area
547
548         Xw1 = Xw1 / Awl
549         Il = self.getIl(h, Xw1)
550         KBz = KBz / vol
551         KBx = KBx / vol
552         Lwl,Bwl = self.getLwlBwl(h,fvs,points)
553         mfarea = self.getMainFrameArea(x,h,fvs,points)
554
555     return h, vol, Awl, Xw1, KBz, KBx, Ib, Il, Lwl, Bwl, mfarea
556
557 def getIl(self,h, Xw1):
558     mesh = self.mesh
559     lpowl = []
560     lpbwl = []
561     Il = 0
562     p = []
563     r = []
564     a = Xw1
565     for fh in mesh.faces(): # facet handle
566         p.clear()
567         r.clear()
568         lpowl.clear()
569         lpbwl.clear()
570         i = 0
571         for vh in mesh.fv(fh): # vertex handle
572             p.append(mesh.point(vh))
573             if p[i][2] > h:
574                 lpowl.append(i)
575
576             else:
577                 lpbwl.append(i)
578             i = i + 1
579         if len(lpowl) < 1:
580             # A
581             Ax = p[0][0]
582             Ay = p[0][1]
583             # B
584             Bx = p[1][0]
585             By = p[1][1]
586             # C
587             Cx = p[2][0]
588             Cy = p[2][1]
589             r.append(self.TezisteTrokuta(Ax, Ay, h, Bx, By, h, Cx, Cy, h))
590             Il = Il + (r[0][0] - a) ** 2 * self.calcArea2DTria(Ax, Ay, Bx, By, Cx, Cy)
591             pass
592         elif len(lpowl) == 1:
593             # 2 trokuta
594             lip = self.getIntersectionPoints(p[lpowl[0]], p[lpbwl[0]], p[lpbwl[1]], h, 2)
595             r.append(self.TezisteTrokuta(lip[0][0],lip[0][1], h, lip[1][0], lip[1][1], h, p[
    lpbwl[1]][0], p[lpbwl[1]][1], h))
596             Il = Il + (r[0][0] - a) ** 2 * self.calcArea2DTria(lip[0][0],lip[0][1], lip[1][0]
    ], lip[1][1], p[lpbwl[1]][0], p[lpbwl[1]][1])
597
598             r.append(self.TezisteTrokuta(lip[0][0], lip[0][1], h, p[lpbwl[1]][0], p[lpbwl[1]
    ]][1], h, p[lpbwl[0]][0], p[lpbwl[0]][1], h))
599             Il = Il + (r[1][0] - a) ** 2 * self.calcArea2DTria(lip[0][0], lip[0][1],p[lpbwl[
    1]][0], p[lpbwl[1]][1], p[lpbwl[0]][0], p[lpbwl[0]][1])
600
601             pass
602         elif len(lpowl) == 2:
603             # 1 trokut
604             lip = self.getIntersectionPoints(p[lpbwl[0]], p[lpowl[0]], p[lpowl[1]], h, 2)
605             r.append(self.TezisteTrokuta(lip[0][0], lip[0][1], h, lip[1][0], lip[1][1], h, p
    [lpowl[0]][0], p[lpbwl[0]][1], h))
606             Il = Il + (r[0][0] - a) ** 2 * self.calcArea2DTria(lip[0][0], lip[0][1], lip[1][
    0], lip[1][1], p[lpbwl[0]][0], p[lpbwl[0]][1])
607             pass
608

```

```

609         return l1
610
611     def TezisteTrokuta(self, Ax, Ay, Az, Bx, By, Bz, Cx, Cy, Cz):
612
613         Xcm = (Ax + Bx + Cx) / 3
614         Ycm = (Ay + By + Cy) / 3
615         Zcm = (Az + Bz + Cz) / 3
616
617         return Xcm, Ycm, Zcm
618
619     def getIntersectionPoints(self, p1,p2,p3,h, os):
620         ip1 = self.getIntersectionPoint(p1,p2,h, os)
621         ip2 = self.getIntersectionPoint(p1, p3, h, os)
622         ips = [ip1, ip2]
623         return ips
624
625     def getIntersectionPoint(self, p1,p2,h, os):
626         ip1=0
627         if os == 2:
628             ip1 = [(h-p2[2])/(p1[2]-p2[2])*(p1[0]-p2[0])+p2[0], (h-p2[2])/(p1[2]-p2[2])*(p1[1]-
p2[1])+p2[1] ,h]
629         if os == 0:
630             ip1 = [h, (h-p1[0])/(p2[0]-p1[0])*(p2[1]-p1[1])+p1[1], (h-p1[0])/(p2[0]-p1[0])*(p2[2]
]-p1[2])+p1[2]]
631
632         return ip1
633
634     def calcArea2DTria(self,Ax,Ay,Bx,By,Cx,Cy):
635         area = 1 / 2 * (Ax * (By - Cy) + Bx * (Cy - Ay) + Cx * (Ay - By))
636         return abs(area)
637
638     # kod za formu
639     def genHullFormMesh(self, lines: list):
640         mesh = om.TriMesh()
641         wlinesPos = lines[0] # positive y waterlines
642         wlinesNeg = lines[1] # negative y waerlines
643         wlKeel = lines[2] # keel waterline (one waterline)
644         n1 = np.array([0,0,0])
645         m1 = np.array([0,0,0])
646         m2 = np.array([0,0,0])
647         pt1= np.array(3)
648         pt2= np.array(3)
649         n2 = np.array([0,0,0])
650         n3 = np.array([0,0,0])
651         n4 = np.array([0,0,0])
652
653         for i in range(len(lines) - 2): # Lijevo desno kobilica
654             for j in range(len(lines[i])): # broji vodne Linije
655                 for k in range(len(lines[i][j]) - 2): # broj tocaka na vodnoj Liniji
656                     if j == len(lines[i]) - 1: #kobilica
657                         mpt1 = lines[i][j-1][k]
658                         mpt2 = lines[i][j-1][k + 1]
659                         mpt3 = lines[i][len(lines[i])-1][k]
660                         mpt4 = lines[i][len(lines[i])-1][k + 1]
661
662                         # volumenA = volumenA + Voltot(mpt1,mpt2,mpt3,mpt4)
663                         mesh.add_face(mesh.add_vertex(mpt1), mesh.add_vertex(mpt2), mesh.
add_vertex(mpt3))
664                         mesh.add_face(mesh.add_vertex(mpt2), mesh.add_vertex(mpt4), mesh.
add_vertex(mpt3))
665
666                     if j != len(lines[i])-1: #Sve ostale vodne Linije
667                         mpt1 = lines[i][j][k]
668                         mpt2 = lines[i][j][k + 1]
669                         mpt3 = lines[i][j+1][k]
670                         mpt4 = lines[i][j+1][k + 1]
671                         mesh.add_face(mesh.add_vertex(mpt1), mesh.add_vertex(mpt2), mesh.
add_vertex(mpt3))
672                         mesh.add_face(mesh.add_vertex(mpt2), mesh.add_vertex(mpt4), mesh.
add_vertex(mpt3))
673
674
675         for i in range(len(lines) - 2): # Lijevo desno kobilica
676             for k in range(len(lines[i][0]) - 2): # broj tocaka na vodnoj Liniji
677                 pt1 = lines[i][0][k]

```

File - C:\Users\Luka\Desktop\zavrsni rad\3v\src\modules\commands\hullform.py

```
678             pt2 = lines[i][0][k + 1]
679             m1[0] = pt1[0]
680             m1[2] = pt1[2]
681             m2[0] = pt2[0]
682
683             m2[2] = pt2[2]
684             mesh.add_face(mesh.add_vertex(pt1), mesh.add_vertex(m1), mesh.add_vertex
(pt2))
685             mesh.add_face(mesh.add_vertex(pt2), mesh.add_vertex(m1), mesh.add_vertex
(m2))
686
687         return mesh
688
689     def _genFaces(self, mesh: om.TriMesh, whs: list, doReverse: bool):
690         n1 = len(whs)
691         npt = len(whs[0])
692         for iL in range(1, n1):
693             npt_iL = len(whs[iL])
694             npt_iL_1 = len(whs[iL-1])
695             dip = 0
696             if npt_iL > npt_iL_1:
697                 if doReverse:
698                     mesh.add_face(whs[iL][0], whs[iL][1], whs[iL-1][0])
699                 else:
700                     mesh.add_face(whs[iL][1], whs[iL][0], whs[iL-1][0])
701             dip = 1
702             for iPL_1 in range(1, npt_iL_1):
703                 ip = iPL_1 + dip
704                 if doReverse:
705                     mesh.add_face(whs[iL-1][iPL_1-1], whs[iL][ip], whs[iL-1][iPL_1])
706                     mesh.add_face(whs[iL-1][iPL_1-1], whs[iL][ip-1], whs[iL][ip])
707                 else:
708                     mesh.add_face(whs[iL-1][iPL_1-1], whs[iL-1][iPL_1], whs[iL][ip])
709                     mesh.add_face(whs[iL-1][iPL_1-1], whs[iL][ip], whs[iL][ip-1])
710
711     def genHullFormMeshPP(self, lines: list):
712         mesh = om.TriMesh()
713         wlinesPos = lines[0] # positive y waterlines
714         wlinesNeg = lines[1] # negative y waterlines
715         wlKeel = lines[2] # keel waterline (one waterline)
716         wlinesPos.reverse()
717         wlinesNeg.reverse()
718
719         whsPos = []
720         whsNeg = []
721         whsi = []
722         whsPos.append(whsi)
723         whsNeg.append(whsi)
724         for p in wlKeel:
725             whsi.append(mesh.add_vertex(p))
726
727         for wl in wlinesPos:
728             whsi = []
729             whsPos.append(whsi)
730             for p in wl:
731                 whsi.append(mesh.add_vertex(p))
732         for wl in wlinesNeg:
733             whsi = []
734             whsNeg.append(whsi)
735             for p in wl:
736                 whsi.append(mesh.add_vertex(p))
737
738         self._genFaces(mesh, whsPos, True)
739         #self._genFaces(mesh, whsNeg, False)
740
741         return mesh
742
743
744
745
746     def hullGen(self, shipdata: dict, pdecks: list, nump):
747         # gs is the grid size of a cell, in pixels
748         # Reminder to make gridsize scaled to the screen width
749         # Sets hullform data to slider values
750         shipdata["loa_val"] = shipdata["loa_val"]
```

```

751     shipdata["boa_val"] = shipdata["boa_val"]
752
753     #
754     midshipsM = shipdata["ms_val"] # Constant m in JC equation
755     bowRakeM = shipdata["bow_val"] # Constant m in JC equation
756     transomM = shipdata["tr_val"] # Constant m in JC equation
757     fwdDeckM = shipdata["deck_val"] # Constant m in JC equation
758
759     transomBeamMax = (shipdata["boa_val"] * shipdata["tb_val"]) / 2 # Transom half beam
760     transomTip = shipdata["draft_val"] * shipdata["td_val"]
761     ACU = shipdata["loa_val"] * shipdata["acu_val"]
762     keelFwd = shipdata["loa_val"] * shipdata["kf_val"]
763     slope = shipdata["sa_val"]
764
765     midBeam = [] # Array with midships half beam per deck
766     bowRake = [] # Array with bow rake per deck
767     bowRakeS = [] # Array with bow rake per deck in superstructure
768     TB = 0 # Transom half beam of a deck
769     transomBeam = [] # Array with transom half beam per deck
770     fwdDeckMArray = [] # Array with constants m in JC equation for deck outlines
771     AE = 0 # Aft end of a deck
772     aftEnd = [] # Array with aft end of each deck
773     aftEndS = [] # Array with aft end of each deck in superstructure
774     noseConeBaseRadius = [] # See excel tool
775     ogiveRadius = [] # See excel tool
776     pdecks2 = [] # Array with deck positions of hull decks
777     pdecks3 = [] # Array with deck positions of superstructure decks
778
779     for i in range(len(pdecks)): # Assign values to variables above
780         if pdecks[i] <= shipdata["draft_val"]: # For each deck that is in the hull
781             midBeam.append((Math.acosh(
782                 (pdecks[i] / shipdata["draft_val"]) * (Math.cosh(midshipsM * Math.pi) - 1
783             ) + 1) / (
784                 midshipsM * Math.pi)) * (shipdata["boa_val"] / 2))
785             bowRake.append((Math.acosh(
786                 (pdecks[i] / shipdata["draft_val"]) * (Math.cosh(bowRakeM * Math.pi) - 1) +
787             1) / (
788                 bowRakeM * Math.pi)) * (shipdata["loa_val"] -
789             keelFwd))
790             if pdecks[i] > transomTip:
791                 TB = ((Math.acosh(((pdecks[i] - transomTip) / (shipdata["draft_val"] -
792                 transomTip))) * (
793                     Math.cosh(transomM * Math.pi) - 1) + 1) / (transomM * Math.pi
794                 )) * (transomBeamMax))
795             else:
796                 TB = 0
797             transomBeam.append(TB)
798             fwdDeckMArray.append(fwdDeckM * (pdecks[i] / (shipdata[
799                 "draft_val"]))) + 0.001) # Changes constant m in JC equation to make deck
800             outlines becomes slimmer with decreasing z position (see below)
801             if (pdecks[i] >= transomTip):
802                 AE = (shipdata["draft_val"] - pdecks[i]) * Math.tan(slope)
803             else:
804                 AE = (shipdata["draft_val"] - transomTip) * Math.tan(slope) + (transomTip -
805                 pdecks[i]) * (
806                     (ACU - (shipdata["draft_val"] - transomTip) * Math.tan(slope
807                 )) / transomTip)
808             aftEnd.append(AE)
809             pdecks2.append(pdecks[i])
810             else: # For each deck that is in the superstructure
811                 aftEndS.append((pdecks[i] - shipdata["draft_val"]) * Math.tan(slope))
812                 bowRakeS.append(shipdata["loa_val"] - ((pdecks[i] - shipdata["draft_val"]) *
813                 Math.tan(slope)) - keelFwd)
814                 pdecks3.append(pdecks[i])
815
816     for i in range(len(midBeam)): # Assign values to variables above cont.
817         noseConeBaseRadius.append(midBeam[i] - transomBeam[i])
818         if noseConeBaseRadius[i] > 0:
819             ogiveRadius.append(
820                 (Math.pow(noseConeBaseRadius[i], 2) + Math.pow((shipdata["loa_val"] / 2) -

```

```

816 aftEnd[i], 2)) / (
817         2 * noseConeBaseRadius[i]))
818
819     else:
820         ogiveRadius.append(0)
821
822     deckOutlinesHull = [] # Array with hull deck outline x, y coordinates
823     # Get y points for every x
824     for idk in range(len(midBeam)): # For each deck in hull
825         deckOutlinesHull.append([]) # For each deck create array
826         if pdecks2[idk] != 0: # If not keel
827             if transomBeam[idk] > 0: # Add vertical hull line at transom
828                 deckOutlinesHull[idk].append([aftEnd[idk], 0])
829                 kmin = aftEnd[idk]
830                 kmax = shipdata["loa_val"] / 2
831                 klist = np.linspace(kmin, kmax, nump)
832                 for xpt in klist:
833                     deckOutlinesHull[idk].append([xpt, (
834                         shipdata["loa_val"] / 2, 2)) +
835                         noseConeBaseRadius[idk] - ogiveRadius[idk] + transomBeam[idk]])
836
837                 kmin = shipdata["loa_val"] / 2
838                 kmax = keelFwd + bowRake[idk]
839                 klist = np.linspace(kmin, kmax, nump)
840                 for xpt in klist:
841                     eqX = (xpt - shipdata["loa_val"] / 2) / (
842                         keelFwd + bowRake[idk] - (shipdata["loa_val"] / 2)) # Value of
843                     # x in JC equation
844                     deckOutlinesHull[idk].append([xpt, (1 - ((Math.cosh(eqX * fwdDeckMArray[idk]
845                         ] * Math.pi) - 1) / (
846                             Math.cosh(fwdDeckMArray[idk] * Math.pi) - 1))) * midBeam[idk]])
847
848             else: # If keel draw top
849                 kmin = aftEnd[idk]
850                 kmax = (keelFwd + bowRake[idk])
851                 klist = np.linspace(kmin, kmax, nump * 2)
852                 for xpt in klist:
853                     deckOutlinesHull[idk].append([xpt, 0]) # Straight Line
854
855     deckOutlinesS = [] # Array with superstructure deck outline x, y coordinates
856     tumblehome = [] # Superstructure tumblehome
857     for n in range(len(aftEndS)): # For each deck in superstructure
858         deckOutlinesS.append([]) # For each deck create array
859         tumblehome = (pdecks3[n] - shipdata["draft_val"]) * Math.tan(
860             slope) # Calculate tumblehome y offset to subtract below
861         deckOutlinesS[n].append([aftEndS[n], 0]) # Add vertical hull line at transom
862
863         kmin = aftEndS[n]
864         kmax = shipdata["loa_val"] / 2
865         klist = np.linspace(kmin, kmax, nump)
866         for xpt in klist:
867             deckOutlinesS[n].append([xpt, (
868                 Math.sqrt(Math.pow(ogiveRadius[0], 2) - Math.pow(xpt - shipdata["
869                 loa_val"] / 2, 2)) +
870                 noseConeBaseRadius[0] - ogiveRadius[0] + transomBeam[0] - tumblehome
871             )])
872
873         kmin = shipdata["loa_val"] / 2
874         kmax = (keelFwd + bowRakeS[n])
875         klist = np.linspace(kmin, kmax, nump)
876         for xpt in klist:
877             eqX = (xpt - shipdata["loa_val"] / 2) / (
878                 keelFwd + bowRakeS[n] - (shipdata["loa_val"] / 2)) # Value of x in
879             # JC equation
880             deckOutlinesS[n].append([xpt, (1 - ((Math.cosh(eqX * fwdDeckMArray[0] * Math.pi
881                 ) - 1) / (
882                     Math.cosh(fwdDeckMArray[0] * Math.pi) - 1))) * (midBeam[0] -
883                     tumblehome)])
884
885     wlinesPos = []
886     wlinesNeg = []
887     wlKeel = []
888

```

```

883     for ii in range(len(deckOutlinesS)):
884         wlineP = list()
885         wlineN = list()
886         for item in deckOutlinesS[ii]:
887             p = np.array([item[0], item[1], pdecks3[ii]])
888             wlineP.append(p)
889             p = np.array([item[0], -item[1], pdecks3[ii]])
890             wlineN.append(p)
891         wlinesPos.append(wlineP)
892         wlinesNeg.append(wlineN)
893
894     for ii in range(len(deckOutlinesHull)):
895
896         if pdecks2[ii] != 0:
897             wlineP = list()
898             wlineN = list()
899             for item in deckOutlinesHull[ii]:
900                 p = np.array([item[0], item[1], pdecks2[ii]])
901                 wlineP.append(p)
902                 p = np.array([item[0], -item[1], pdecks2[ii]])
903                 wlineN.append(p)
904             wlinesPos.append(wlineP)
905             wlinesNeg.append(wlineN)
906         else:
907             for item in deckOutlinesHull[ii]:
908                 p = np.array([item[0], item[1], pdecks2[ii]])
909                 wKeel.append(p)
910
911     return [wlinesPos, wlinesNeg, wKeel]
912
913 def readShipData(self):
914     shipdata = {}
915     pdecks = []
916     pbulkheads = []
917     with open(self.filename, newline='') as csvfile:
918         f = csv.DictReader(csvfile)
919         shipset = 0
920         for row in f: # there is only one row after header row!!!!
921             shipset = row
922
923     shipdata["loa_val"] = float(shipset['LOA']) # treba li učitavanje vrijednosti biti
924     u petlji? shipdata["boa_val"] = float(shipset['BOA']) # treba li učitavanje vrijednosti biti
925     u petlji? shipdata['draft_val'] = float(shipset['D'])
926     shipdata['shipname'] = shipset['Name']
927
928     splitdata = str(shipset['HullData']).split(" ")
929     shipdata["ms_val"] = float(splitdata[0])
930     shipdata["bow_val"] = float(splitdata[1])
931     shipdata["tr_val"] = float(splitdata[2])
932     shipdata["deck_val"] = float(splitdata[3])
933     shipdata["tb_val"] = float(splitdata[4])
934     shipdata["td_val"] = float(splitdata[5])
935     shipdata["acu_val"] = float(splitdata[6])
936     shipdata["kf_val"] = float(splitdata[7])
937     shipdata["sa_val"] = float(splitdata[8])
938
939     shipdata["sp_val"] = float(splitdata[9])
940     shipdata["cwl_val"] = float(splitdata[10])
941     shipdata["lcb_val"] = float(splitdata[11])
942     shipdata["cb_val"] = float(splitdata[12])
943     shipdata["mc_val"] = float(splitdata[13])
944     shipdata["bb_val"] = float(splitdata[14])
945     shipdata["tran_val"] = float(splitdata[15])
946     shipdata["ab_val"] = float(splitdata[16])
947
948     shipdata["lw1_val"] = float(splitdata[17])
949     shipdata["bw1_val"] = float(splitdata[18])
950     shipdata["tf_val"] = float(splitdata[19])
951     shipdata["ta_val"] = float(splitdata[20])
952
953     shipdata["app1"] = float(splitdata[21])
954     shipdata["area_app1"] = float(splitdata[22])
955     shipdata["app2"] = float(splitdata[23])

```

File - C:\Users\Luka\Desktop\zavrsni rad\d3v\src\modules\commands\hullform.py

```
956         shipdata["area_app2"] = float(splitdata[24])
957         shipdata["area_app3"] = float(splitdata[25])
958         shipdata["area_app4"] = float(splitdata[26])
959         shipdata["app5"] = float(splitdata[27])
960         shipdata["area_app5"] = float(splitdata[28])
961         shipdata["area_app6"] = float(splitdata[29])
962         shipdata["area_app7"] = float(splitdata[30])
963         shipdata["app8"] = float(splitdata[31])
964         shipdata["area_app8"] = float(splitdata[32])
965         shipdata["area_app9"] = float(splitdata[33])
966         shipdata["area_app10"] = float(splitdata[34])
967         shipdata["area_app11"] = float(splitdata[35])
968
969         shipdata["cg_val"] = float(splitdata[36])
970         shipdata["heading_val"] = float(splitdata[37])
971         shipdata["amplitude_val"] = float(splitdata[38])
972         shipdata["roll_val"] = float(splitdata[39])
973         shipdata["damping_val"] = float(splitdata[40])
974         shipdata["plr_val"] = float(splitdata[41])
975         shipdata["gmt_val"] = float(splitdata[42])
976
977         draft = shipdata["draft_val"]
978         splitdata = str(shipset['DeckPos']).split(" ")
979         for dp in splitdata:
980             pdecks.append(float(dp))
981         splitdata = str(shipset['BHPos']).split(" ")
982         for dp in splitdata:
983             pbulkheads.append(float(dp))
984     results = [shipdata,pdecks,pbulkheads]
985     return results
986
```