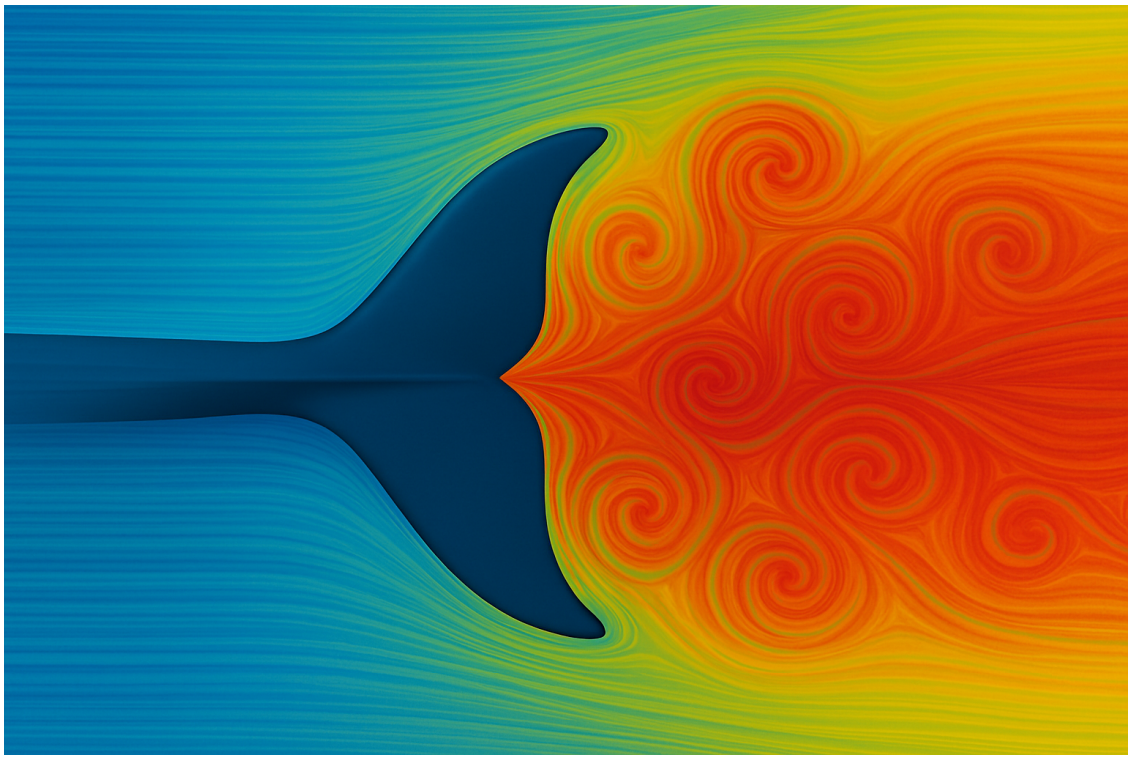
CHALMERS



Strömningssimulering av fendrift

Fluid-struktur-växelverkan med deformerbar fena

Kandidatarbete inom Mekanik och maritima vetenskaper

Dexter Balazs, Gustav Bergman, Max Dahlsten Andius,
Ludvig Karlsson, Albin Larsson

Instutionen för Mekanik och maritima vetenskaper

CHALMERS TEKNISKA HÖGSKOLA
Göteborg 2025
www.chalmers.se

Kandidatarbete 2025

Strömningssimulering av fendrift

Fluid-struktur-växelverkan med deformerbar fena

DEXTER BALAZS, GUSTAV BERGMAN,
MAX DALHSTEN ANDIUS, LUDVIG KARLSSON,
ALBIN LARSSON

CHALMERS

Institutionen för Mekanik och maritima vetenskaper
Chalmers Tekniska Högskola
Göteborg 2025

Strömningssimulering av fendrift
Fluid-struktur-växelverkan med deformerbar fena
DEXTER BALAZS
GUSTAV BERGMAN
MAX DALHSTEN ANDIUS
LUDVIG KARLSSON
ALBIN LARSSON

Handledare: Rickard Everyd Bensow, Institutionen för Mekanik och maritima vetenskaper
Examinator: Huadong Yao, Institutionen för Mekanik och maritima vetenskaper

Kandidatarbete 2025
Institutionen för Mekanik och maritima vetenskaper
Chalmers Tekniska Högskola
SE-412 96 Göteborg
Telefon +46 31 772 1000

Omslagsbild: Flöde över delfinfena, genererad med AI (OpenAI, 2025b).

Skriven i LaTeX
Göteborg 2025

Strömningssimulering av fendrift
Fluid-struktur-växelverkan med deformerbar fena
Dexter Balazs, Gustav Bergman, Max Dahlsten Andius, Ludvig Karlsson, Albin Larsson
Institutionen för Mekanik och maritima vetenskaper
Chalmers Tekniska Högskola

Sammanfattning

Detta kandidatarbete syftar till att undersöka och modellera fendrift, ett biomimetiskt alternativ till traditionella propellrar, genom utveckling av en beräkningseffektiv CFD-simuleringsmodell. Arbetet fokuserar på att efterlikna rörelsemönstret hos en delfinfena, med hjälp av simuleringsverktyget WaterLily och en egenutvecklad FEM-lösare i Julia. Målet var att skapa en parametriserbar modell som kan användas för att analysera och optimera fenans design och rörelse, samt att utvärdera WaterLilys lämplighet som undervisningsverktyg i strömningsmekanik. Projektet resulterade i en simuleringsmodell som möjliggör visualisering av fenans rörelse och vissa grundläggande kraftanalyser, men visade också på begränsningar i både modellens noggrannhet och simuleringsverktygets kapacitet. Slutsatsen är att fendrift har potential som ett mer miljövänligt och effektivt framdrivningssätt, men att mer avancerade och precisa simuleringar krävs för tillförlitliga resultat.

Nyckelord: Delfinfena, fendrift, CFD, FEM, strömningssimulering, WaterLily, bioinspiration, fluid-struktur-interaktion, verkningsgrad, marina tillämpningar.

Abstract

This bachelor's thesis aims to investigate and model fin propulsion, a biomimetic alternative to traditional propellers, through the development of a computationally efficient CFD simulation model. The work focuses on mimicking the motion pattern of a dolphin fin using the simulation tool WaterLily and a custom-developed FEM solver in Julia. The goal was to create a parameterized model that can be used to analyze and optimize the fin's design and motion, as well as to assess the suitability of WaterLily as a teaching tool in fluid mechanics. The project resulted in a simulation model that enables visualization of the fin's movement and basic force analysis, but also revealed limitations in both the model's accuracy and the capabilities of the simulation tool. The conclusion is that fin propulsion has potential as a more environmentally friendly and efficient means of propulsion, but more advanced and precise simulations are required to produce reliable results.

Keywords: dolphin fin, fin propulsion, CFD, FEM, flow simulation, WaterLily, bioinspiration, fluid-structure interaction, hydrodynamic efficiency, marine applications.

Förord

Denna rapport har tagits fram som en del av ett kandidatarbete som genomfördes vid institutionen för Mekanik och maritima vetenskaper, Chalmers Tekniska Högskola. Arbetet omfattar 15 högskolepoäng och genomfördes under våren 2025. Efter att ha genomfört detta spännande och utmanande arbete vill vi tacka de individer och aktörer som har hjälpt oss framåt. Vi riktar vår tacksamhet till Thomas Jemt på Dolprop Industries, Gabriel Weymouth och Marin Lauber, utvecklare av WaterLily, och Jim Brouzoulis från Chalmers Tekniska Högskola. Ett extra stort tack till vår handledare Rickard Bensow och vår examinator Huadong Yao.

Dexter, Gustav, Max, Ludvig, Albin,

Göteborg, Maj 2025

Innehåll

Figurer

1
Inledning

Inom sjöfart används i hög utsträckning propellern till framdrift, en beprövad lösning som utvecklats under lång tid. Propellern har dock nackdelar som är svåra att undkomma; den orsakar mycket buller, når en relativt låg hydrodynamisk verkningsgrad och utgör en skaderisk för marint liv och annat som kan komma nära (FOI, 2024). Att övergå från användningen av propellrar till alternativa framdrivningssystem kan potentiellt förbättra effektiviteten och minimera negativa egenskaper hos den marina trafiken. Ingenjörsvetenskapen har gjort många försök att efterlikna fiskar och marina däggdjurs rörelsemönster vid framdrift. Av dessa har några lyckats väl och resulterat i bioinspirerade undervattensfordon, som överträffat sina motsvarigheter med propellerdrift (Guo m. fl., 2023).

Dagens mest använda kraftkälla till handelsfartyg är tvåtakts dieselmotorer med en termodynamisk verkningsgrad runt 50% (Benvenuto m. fl., 2014). Det är svårt att öka den verkningsgraden mycket mer med befintlig teknik, däremot har framdrivningen stor förbättringspotential. I och med att framdrivningen hanterar effekten direkt från drivkraften, och ansvarar för att omsätta den i nyttig rörelse, är den högst relevant för den totala verkningsgraden (Zhu och Gao, 2021).

Ett alternativ till traditionell framdrift inom sjöfart är den så kallade fendriften. En lösning som biomimetiskt tar inspiration från marint liv med fenor. Genom att en fena pendlar upp och ner eller från sida till sida, drivs farkosten framåt på samma sätt som en delfin. Fenan som framdriftssystem kan möjliggöra en mer energieffektiv gång, mindre buller i marina miljöer samt en minskad skaderisk för marint liv.

1.1 Bakgrund

Företaget Dolprop Industries har, genom inspiration från djurvärlden utvecklat en fena som ska fungera som alternativ till propellern. De har tagit inspiration från delfinfenor i designen och har som mål att deras fenor ska kunna användas för undervattensdrönare, mindre fritidsbåtar och i framtiden fartygstrafik. Företaget hävdar lovande egenskaper som hög verkningsgrad och låg ljudnivå. Grundaren av Dolprop Industries är entreprenören och uppfinnaren Thomas Jemt, som för cirka 20 år sedan blev fascinerad av hur snabbt och kraftfullt delfiner kan röra sig, och ville utnyttja det i en teknisk tillämpning.

Forskare från olika kinesiska universitet har tagit fram en radiostyrd undervattensfarkost i formen av en delfin. Denna robot är ledad i flera sektioner för att kunna replikera rörelsemönstret hos en verklig delfin. Genom tester kom de fram till att den maximala verkningsgraden för den robotiserade delfinen var över 80% (Liu m. fl., 2021). Märk väl att den hydrodynamiska verkningsgrad som hänvisas till i övrigt genom denna rapport endast betraktar fenan eller propellern i fråga. För delfinroboten är hela kroppen inräknad, varför fenan i sig kan tänkas ha en ännu högre verkningsgrad.

För att undersöka hur väl fendrift fungerar som alternativ till propellern kan strömningssimuleringar, CFD - Computational fluid dynamics, utföras. Från dessa kan intressanta värden extraheras, vilka kan användas för att vidareutveckla fenan och dess egenskaper. Ett verktyg som kan utföra strömningssimuleringar är WaterLily. Det är ett paket till programmeringsspråket Julia, utvecklat för att utföra snabba och beräkningseffektiva strömningssimuleringar. WaterLily är skrivet med öppen källkod, och löser Navier-Stokes ekvationer för inkompressibla fluider över ett uniformt rutnät i två eller tre dimensioner.

1.2 Syfte

Detta kandidatarbete har ett primärt syfte och ett sekundärt syfte. Det huvudsakliga är att utveckla en modell för strömningssimulering över en fena från Dolprop Industries, ämnat att hjälpa företaget i vidareutveckling och optimering. För att modellen ska vara användbar behöver möjligheten finnas att kunna förändra storlek och styvhet på fenan, samt rörelsemönstret, frekvensen vilken fenan rör sig med samt hastigheten på vätskan. Det är därför nödvändigt att simuleringsmodellen är parametriserad, med parametrar angivna av användaren före simulering, för att få önskat resultat. Genom förändring av dessa tidigare nämnda parametrar ska modellen kunna köra tester i syfte att förbättra delfindriften. Detta ska fungera som en mer resurseffektiv och lättanvänd metod än fysiska tester, vilka kräver dyr utrustning som testanläggning och testrigg. Med hjälp av det utvecklade programmet ska utvecklingen av delfindrift kunna effektiviseras genom snabbare tester av olika parametrar.

Det sekundära syftet med arbetet är att undersöka om strömningssimulering med paketet WaterLily är användbart i utbildningssyften. Det ska undersökas om WaterLily är en effektiv CFD-lösare som kan användas i kurser för att ge snabb visualisering och ökad förståelse för grundläggande strömningsmekanik.

1.3 Problemformulering

Problemet som ämnas lösas formuleras: *För att underlätta Dolprops utveckling av delfindrift skall en beräkningsmodell utvecklas, som med hjälp av simuleringsprogrammet WaterLily simulerar delfinfenan i aktiv drift. Programmet ska köra simuleringar på en parametriserad och deformerbar fena i sitt verkliga rörelsemönster genom en fluid.*

Modellen skall fungera för följande parametrar givna av användaren:

- Geometri i form av .stl-fil.
- Rörelsemönster i form av tidsberoende geometrisk funktion.
- Materialegenskaper i form av elasticitetsmodul.
- Fluidegenskaper i form av kinematisk viskositet och hastighet.
- Varierbar noggrannhet i form av värden på nätstorlek och längd på tidssteg.

Modellen ska returnera följande parametrar:

- Framdrift uttryckt i effekt.
- Effekt som krävs för fenans rörelse.
- Hydrodynamisk verkningsgrad.
- Visualisering av simuleringen
- Spänningar i fenan för hållfasthet- och utmattningsanalys.

1.4 Avgränsningar

Arbetet kommer fokusera på att skapa en simuleringsmodell för flödet runt fenan. Detta är en avgränsning av problemet då det verkliga fallet består av fler komponenter som verkar i samband med fenan. Detta inkluderar bland annat en konverterare för rörelsen och olika typer av geometrier framför fenan som påverkar vattenflödet. Avgränsningen görs för att minska omfattningen av arbetet i form av kunskap och tid då en mer avancerad modell är för komplex samt kräver mer genomarbetning.

Arbetet kommer att använda mindre etablerad programvara för att göra CFD-simulering. Simuleringar kommer ske i öppen källkod programvaran WaterLily i språket Julia vilket ger snabba och effektiva beräkningar till priset av en något lägre precision. Detta anses passande för projektet då behoven uppfylls på ett effektivare sätt. Med detta är arbetet även begränsat av utvecklingen av strömningssimuleringsverktyget, som ej anses vara färdigt vid projektets start.

2
Teori

2.1 Computational Fluid Dynamics

CFD är en vedertagen metod för numerisk beräkning och analys av strömning i fluider. Det utgår från Navier-Stokes ekvationer om fluiders beteende och beräknar en approximation av dessa för flödet i fråga. Eftersom ekvationerna är ickelinjära och kopplade saknas ofta analytiska lösningar. Därmed behöver problemet diskretiseras och lösas numeriskt av datorn. Med detta möjliggör CFD att mer komplicerade strömningsberäkningar kan utföras (Anderson, 1995).

2.2 Strömningssimulering med WaterLily

För att studera strömningen av delfinfenans rörelser, form och material i vattenmiljö används CFD-paketet WaterLily, vilket är skapat för simulering av parametriserade kroppar i fluider. Det har öppet källkodsformat vilket gör det tillgängligt och möjligt att modifiera. WaterLily är ett paket för programmeringsspråket Julia och skiljer sig på en del sätt från traditionella CFD-program. Den mest betydande skillnaden är att ett helt uniformt rutnät används istället för att ett unikt nät genereras runt objektet som ska undersökas, vanligtvis med finare nät i kritiska områden runt kroppen. Det senare leder till en långsammare simulering jämfört med WaterLily eftersom den nät-genereringen är beräkningsintensiv, och kan behöva göras om när fluiden deformeras runt kroppen. Nackdelen med det fasta nät som WaterLily använder är att den ökade noggrannheten i kritiska områden uteblir, vilket främst leder till sämre simulering av gränsskiktet och skjuvspänningarna som finns där. Utsträckningen av rutnätet väljs i de två eller tre dimensioner som skall simuleras, fördelaktigt för simuleringshastighet är det att välja utsträckningar som är multiplar av tvåpotenser (Weymouth & Font, 2025).

Genom ett gränssnitt mot kopplingsbiblioteket PreCICE kan WaterLilys CFD-lösare kopplas till andra lösare som också har ett gränssnitt mot PreCICE. Under arbetets gång var WaterLilys gränssnitt fortfarande under utveckling och saknade funktioner som skulle möjliggöra simulering av fluid-struktur växelverkan. Vissa funktioner var dock utvecklade och kunde användas. 3-dimensionella nät-kroppar kan fås med i WaterLily-simuleringar genom funktionen `MeshBody` från paketet WaterLilyPreCICE som tar emot antingen sökväg till en stl-fil eller en variabel av datatypen `Mesh` från paketet GeometryBasics som innehåller ett redan inläst nät.

I varje tidssteg kan data för fluiden extraheras från simuleringen, bland andra tryck och hastighet. När sådan data extraheras erhålls en 3-dimensionell matris med värden för mittpunkten av varje cell i rutnätet, samt för ett extra lager av celler som omringar den definierade domänen. Alltså om en storlek på [64 64 64] används fås en matris med dimensionerna [66 66 66].

2.3 Finita Elementmetoden

Finita ElementMetoden - FEM, används bland annat för att beräkna deformationer i kroppar. Metoden går ut på att dela in objektet som ska räknas på i ett nät av tetraedrar, samt noder som utgör hörnen av dessa. Sedan beräknar programmet för varje tidssteg i simuleringen, vilka spänningar som uppkommer i alla noder och deformationerna dessa ger upphov till. Eftersom FEM är den branschledande metoden för att beräkna soliddeformationer antas den vara fullt tillräcklig för projektets behov (Zienkiewicz m. fl., 2021).

2.4 Explicit FEM

När FEM skall beräknas över ett tidsintervall behöver kroppens tröghet tas med i beräkningarna, för vilket det finns olika tidsintegreringsmetoder. För detta projekt kommer explicit FEM att användas, explicit syftar till den tidsintegreringsmetod som används. Nästa tidssteg som ska tas fram beräknas med hjälp av kända värden från föregående tidssteg. Explicit FEM kräver korta tidssteg för att inte bli instabilt och divergera. Anledningen är att deformationer sprider sig som vågor genom nätet, och om en tetraeder deformeras för snabbt och någon nod förflyttas förbi nästa rad av noder blir volymen lokalt negativ och simuleringen kraschar. En konsekvens som dock uppstår av att ha många korta tidssteg, är att simuleringen blir beräkningsintensiv och tidskrävande (Wu & Gu, 2012). I kort går beräkningarna i tidssteg t_{n+1} till så här:

$$\mathbf{a}(t_{n+1}) = \mathbf{f}_{\text{tot}}(t_n) = \mathbf{f}_{\text{yttre}}(t_n) - \mathbf{f}_{\text{inre}}(t_n) \tag{2.1}$$

$$\mathbf{v}(t_{n+1}) = \mathbf{v}(t_n) + \Delta t \cdot \mathbf{a}(t_{n+1}) \tag{2.2}$$

$$\mathbf{x}(t_{n+1}) = \mathbf{x}(t_n) + \Delta t \cdot \mathbf{v}(t_{n+1}) \tag{2.3}$$

Där a är acceleration, v hastighet, x position, Δt tidssteget, f_{yttre} yttre krafter som verkar på noderna och f_{inre} inre krafter från deformationen. Inre krafterna beräknas med deformationsgradient och en materialmodell, exempelvis Neo Hookes materialmodell för hyperelastiska material. Deformationsgradienten $\mathbf{F}$ i ett tidssteg beräknas enligt följande:

$$\mathbf{Jac} = \begin{bmatrix} x_2 - x_1 & x_3 - x_1 & x_4 - x_1 \\ y_2 - y_1 & y_3 - y_1 & y_4 - y_1 \\ z_2 - z_1 & z_3 - z_1 & z_4 - z_1 \end{bmatrix} \tag{2.4}$$

$$V = \det(\mathbf{Jac}) \tag{2.5}$$

$$\nabla \mathbf{N} = \left(\mathbf{Jac}^{\top}\right)^{-1} \begin{bmatrix} -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

$$\mathbf{F} = \mathbf{Jac} \cdot \mathbf{Jac}(t_0) \tag{2.7}$$

Från deformationsgradienten beräknas Cauchy-spänningen med materialparametrar D_1 och C_1 enligt Neo Hookes materialmodell:

$$J = \det(\mathbf{F}) \tag{2.8}$$

$$\mathbf{B} = \mathbf{F} \cdot \mathbf{F}^{\top} \tag{2.9}$$

$$I_1 = \mathrm{tr}(Be) \tag{2.10}$$

$$\bar{I_1} = I_1 \cdot J^{-\frac{2}{3}} \tag{2.11}$$

$$p = -2 \cdot D1 \cdot J \cdot (J - 1) \tag{2.12}$$

$$\bar{B} = J^{-\frac{2}{3}} \cdot \mathbf{B} \tag{2.13}$$

$$\mathrm{dev}(\bar{B}) = \bar{B} - \frac{1}{3} \cdot \bar{I_1} \cdot I \tag{2.14}$$

$$\sigma = \frac{1}{J} \cdot (-p \cdot I + 2 \cdot C_1 \cdot \mathrm{dev}(\bar{B})) \tag{2.15}$$

Slutligen beräknas f_{inre} enligt:

$$f_{inre} = (\nabla N)^{\top} \cdot \sigma \cdot V \tag{2.16}$$

2.5 Fluid-struktur-växelverkan

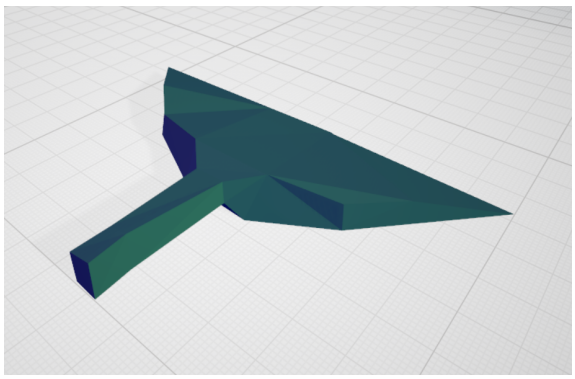
FSI - Fluid struktur interaktion beskriver hur två system påverkar varandra, hur krafter från en vätska påverkar en struktur och hur strukturens deformation påverkar vätskan. FSI delas upp i två olika metoder, svagt formulerad och starkt formulerad. I svag FSI stegar man mellan fluid- och strukturlösaren en gång per tidssteg, fluidens flöde kring nuvarande kropp beräknas och krafterna från fluiden används för beräkning av strukturen, som används till nästa tidssteg o.s.v. I stark FSI stegar lösaren i varje tidssteg mellan fluiden och strukturen till jämvikt uppnås för att försäkra att växelverkan beräknas rätt. I stark formulering tar varje tidssteg längre tid att beräkna men längre tidssteg kan tas (Semenov, 2022).

3
Metod

Arbetet delades upp i två moment. Det ena området handlade om att implementera en FEM-lösare i Julia som tar hänsyn till deformationen som uppstår i geometrin. Det andra området var att skapa strömningsmodellen med WaterLily och att undersöka hur användbara värden exporteras i programmet. Det genomfördes även en intervju med utvecklarna av paketet där användning av WaterLily diskuterades grundläggande.

3.1 Nätgenerering och importering av geometri i WaterLily

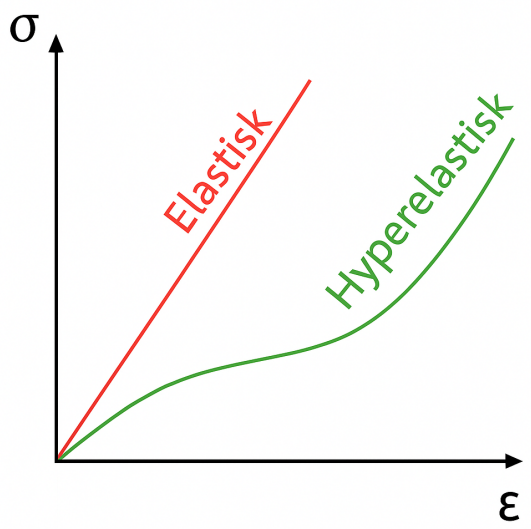
WaterLily kan hantera nät-geometri med hjälp av paketet WaterLilyPreCICE, därför kunde nätet som strukturlösaren använder, användas även till WaterLily. Enda skillnaden är att för WaterLily används bara ytnätet medan i strukturlösaren används även volymelement i form av tetraedrar. För generering av volym-nätet utifrån ytnätet användes programmet Gmsh. Men eftersom elementen inte kan vara för små på grund av begränsningar i metoden explicit FEM, behövde ytnätet först hanteras i Blender, ett CAD-program som enkelt kan manipulera upplösningen i 3D-objekt. Finheten på nätet valdes så att tidssteg kortare än 1×10^{-5} s inte skulle behövas. Elementen måste vara små nog för att kunna beräkna en tillräckligt verklighetstrogen modell, samtidigt så stora att programmet går att köra på en rimlig tid. I Blender roterades även kroppen så att den bakre delen av stjärten skulle peka i x-riktning. Geometrin som den ser ut efter hanteringen i Blender visas i figur 3.1. Ytnätet tillhandahölls av Dolprop i form av en stl-fil.



Figur 3.1: Geometri hanterad i Blender

3.2 Deformation av geometri

För att simulera strukturdeformationerna finns olika materialmodeller i FEM. Med hänsyn till fenans material valdes Neo-Hookes modell för hyperelastiska kompressibla material, se figur 3.2. Elastisk polyuretan, PU, materialet fenan består av, betraktas förvisso som inkompressibelt. Men då simulering av inkompressibel deformation blir betydligt mer komplicerad görs denna avgränsning. Genom att sätta Poissons konstant, förhållandet för hur mycket ett material komprimeras eller deformeras under tryck, till nära 0,5; värdet för ett inkompressibelt material, antas simuleringen kunna ske verklighetstroget nog utan att bli för resurskrävande.



Figur 3.2: Illustration av hyperelasticitet, genererad med AI (OpenAI, 2025a).

En annan materialegenskap som måste tas i beaktning är det faktum att PU deformeras hyperelastiskt och inte linjärt. Töjningen av materialet är således inte rakt proportionellt mot den pålagda spänningen, se figur 3.1, vilket försvårar beräkningarna. Till detta används Neo-Hookes materialmodell för hyperelastiska material.

$$J\boldsymbol{\sigma} = -p\mathbf{I} + 2C_1 \operatorname{dev}(\bar{\mathbf{B}}) = -p\mathbf{I} + \frac{2C_1}{J^{2/3}} \operatorname{dev}(\mathbf{B}) \tag{3.1}$$

J beskriver volymförändringen i materialet och σ den sanna (Cauchy-) spänningen. Högerledet i ekvationen beskriver spänningarna och hur de förändras beroende på formen respektive volymen av materialet.

3.3 CFD - Simuleringsmodell i WaterLily

Eftersom fenan är riktad för att färdas i negativ x-riktning definieras simuleringens fluidhastighet $[U\ 0\ 0]$ där U är hastighetens storlek. strömningssimuleringens domän definierades i storleken $[2 \times 128\ \ 128\ \ 128]$. Den längre utsträckningen i x-riktningen valdes för att få med mer av strömmarna bakom fenan. Eftersom kroppen inte är statisk eller parametriserad i tiden genom simuleringen, behöver den uppdateras och defineras på nytt mellan varje tidssteg. Då måste även argumentet `remeasure` vara sant i anropet av funktionen `sim_step`, som stegar fram funktionen i tiden, se bilaga 1. Detta på grund av att kroppen måste mätas ut på nytt i simuleringen.

3.4 FSI ihopkoppling

De två huvuddelarna som utgör programmet är strömningssimuleringen och beräkningsdelen av fenan. Strömningssimuleringens värden för tryck på fenan extraheras och beräkningsdelen använder det extraherade trycket för att beräkna deformationen av fenan i varje nod av nätet som utgör fenan. Eftersom trycket som extraheras från simuleringen ges i en matris med värden för nodernas mittpunkter måste värdena interpoleras för att få värden i nodernas koordinater. Trycket extraheras från interpoleringen i varje nod och medelvärdet av trycket för de noder som utgör en triangel i nätet multipliceras med triangelns area och normalvektor, vilket resulterar i en kraft. Denna kraft divideras jämnt över noderna. Kraften adderas till de yttre krafterna som verkar på fenan, och tillsammans med kroppens inre krafter ger de upphov till en deformation av nämnd fena. Fenans uppdaterade koordinater förändrar nätets geometri vilket i sin tur ger upphov till en förändrad tryckfördelning på fenans yta i nästa tidssteg av strömningssimuleringen.
För detta projekt tillämpas svag FSI, en fördel med svag jämfört med stark är att den är mindre beräkningsintensiv per tidssteg och lättare att implementera. Även om svag FSI är den mindre exakta utav de två metoderna anses den vara tillräckligt exakt för projektets ändamål.

Algorithm 1 FSI-Ihopkoppling av WaterLily och FEM-lösare

1: Initiera parametrar för fluid och struktur (geometri, material, hastighet)
2: Läs in fenans geometri från .stl-fil
3: Generera volymnät och ytnät med Gmsh
4: Definiera tidssteg och initiala randvillkor
5: **while** tidssteget är inom simuleringstiden **do**
6: Extrahera tryckvärden från WaterLily simuleringen
7: Interpolera tryckvärden till nodernas koordinater
8: Beräkna kraft för varje triangel genom medelvärdet av trycket på de närliggande noderna multiplicerat med arean och normalvektorn
9: Fördela kraften jämnt över noderna
10: Beräkna inre och yttre krafter och summan av dessa
11: Beräkna deformationen av fenan i varje nod
12: Uppdatera nodernas positioner
13: Uppdatera nätets geometri baserat på nya positioner
14: **end while**
15: Spara utdata och generera visualisering

3.5 Hantering av utvärden från simulering

Det värde som är av största intresse är verkningsgraden, hur hög andel av den tillförda effekten som bidrar till farkostens framdrift. Från noderna som rör sig i ett föreskrivet mönster extraheras x-komponenten av de inre krafterna och summeras för att få den framåtdrivande kraften. Genom att multiplicera den med hastigheten fås framåtdrivande effekten, som genom integrering över tid ger energi. När energin divideras med totala tiden fås genomsnittliga framåtdrivande effekten.

Eftersom simuleringsmodellen är en approximering, kan upplösningen av fenan ändras för att finna konvergens av utvärdena. Genom att interpolera resultatet fås en uppfattning kring hur mycket precision som försvinner med varje förenkling av modellen.

För att undersöka bland annat inom vilka effekt- och hastighetsintervall fenan fungerar bäst, eller hur materialegenskaperna påverkar framdriften kan iterativa tester användas. Data för varierande ingångsparametrar kan illustreras grafiskt för hjälp vid utformning av fenor för olika tillämpningar.

4
Resultat

Resultatkapitlet presenterar resultatet utifrån metoderna som beskrevs i kapitel 3, Metod. Metoderna som användes resulterade i en simuleringsmodell samt ett utslag gällande tillämpning av WaterLily.

4.1 Simuleringsmodell

En strömningssimuleringsmodell skapades där deformation, rörelse, material och hastighet tas i hänsyn. Modellen består av en beräknings- och en simuleringsdel som kommunicerar med varandra, värden från simuleringen extraheras och används vid deformationsberäkning av fenan.

Simuleringsmodellen läser in fenans geometri och dess nät från en stl-fil, det skapas noder och tetraedrar för att definiera strukturen. Trycket extraheras i geometrins noder och omvandlas till krafter som används för att beräkna deformationen. Parametrar som frekvens, E-modul, flödeshastighet och fenans amplitud kan justeras för att fungera i olika applikationer. Den framåtdrivande kraften extraheras och printas under fenans rörelseförlopp.

Modellen visualiserar simulationen i ett filmformat, och fenan rör sig i ett mönster likt det avsedda, med jämn hastighet och deformationer i ytterkanterna.

Strömningssimuleringen uppvisar ett antal begränsningar och returnerar inte all data som specificerades i problemformuleringen. Det som inte returneras är framdrift uttryckt i effekt, hydrodynamisk verkningsgrad, effekt som krävs för fenans rörelse, spänningar i fenan för hållfasthet- och utmattningsanalys.

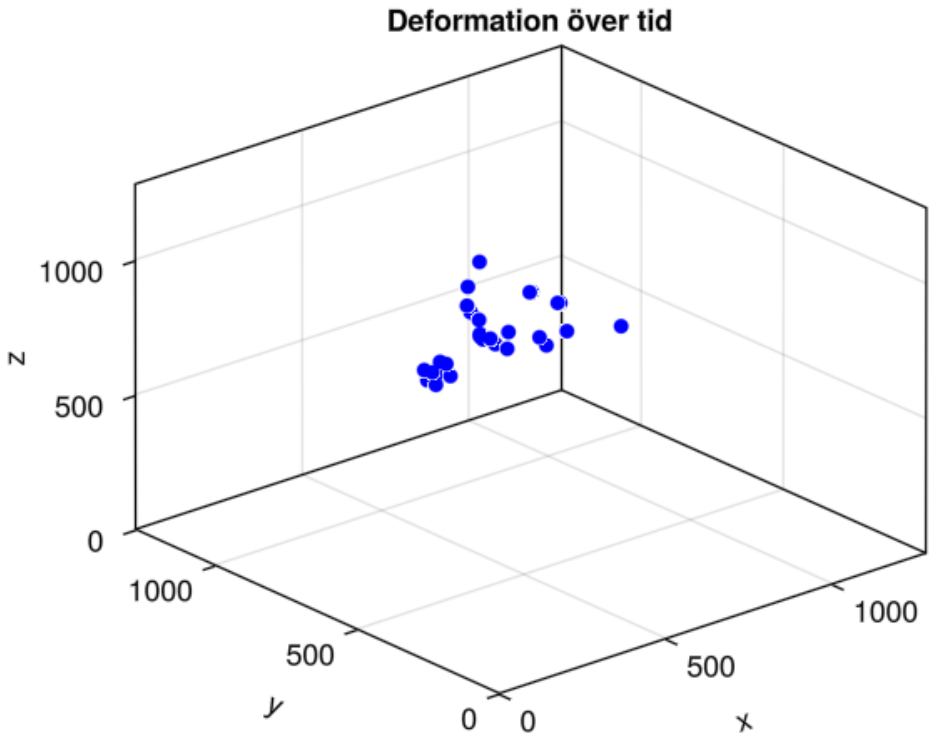
Under rörelseförloppet påverkas fenan inte av hög- och lågtryck på under- respektive ovansidan vilket typiskt förväntas vid liknande rörelser i en fluid.

4.2 WaterLily

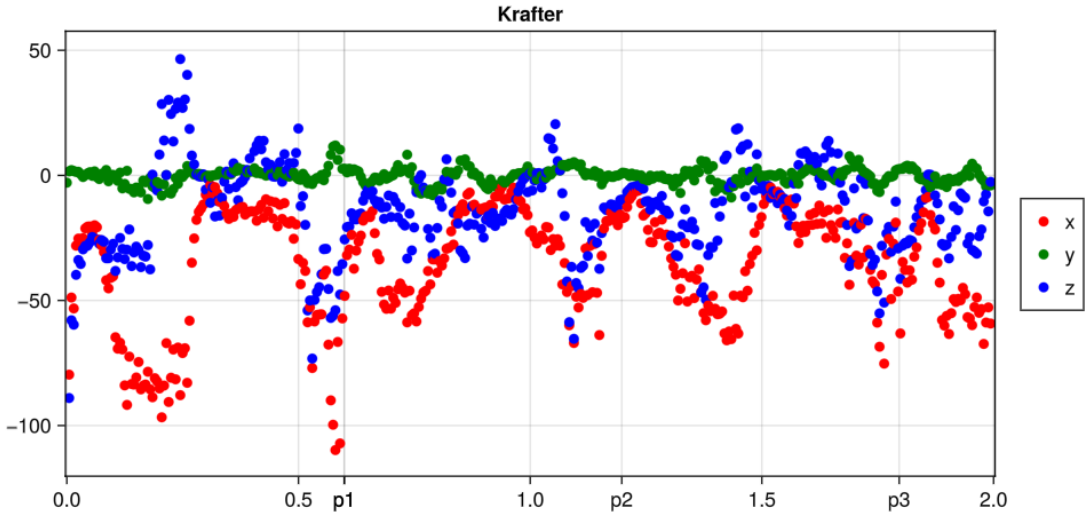
Simpla strömningssimuleringar genomfördes med paketet WaterLily, vilket krävde viss förkunskap, framför allt i programmeringsspråket Julia. För tillämpning inom utbildningssyften identifierades begränsningar i paketet. Det saknades en övergripande struktur och komplett dokumentation av paketet. I den genomförda simuleringen genererade WaterLily krafter och tryck som inte överensstämde med verkliga fysiska förhållanden. De erhållna simuleringsvärdena visade avvikelser från förväntade fysiska principer.

4.3 Simuleringsresultat

I figurer 4.1, 4.2 och 4.3 visas exempel på ut-data som modellen kan generera.

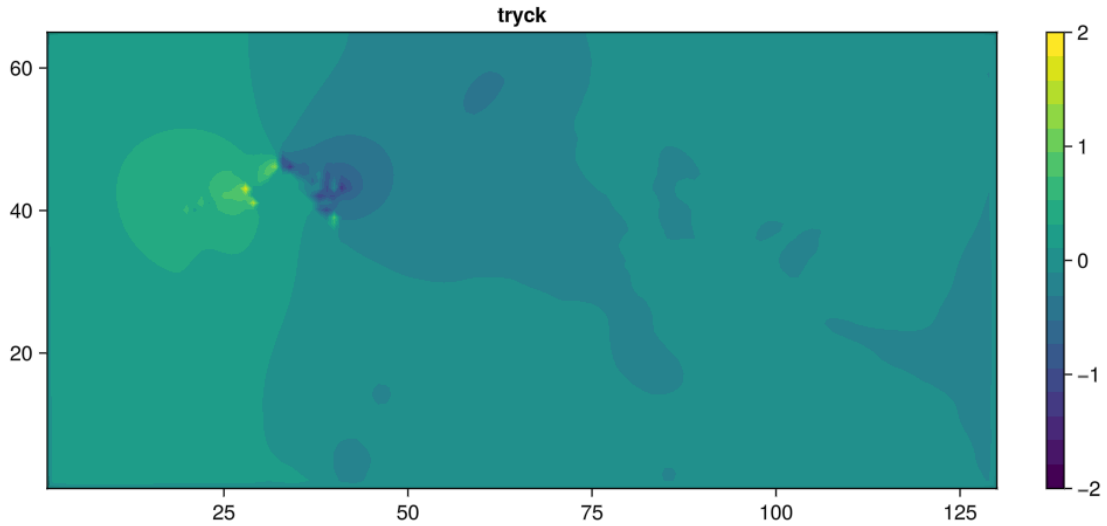


Figur 4.1: Visualisering av fenans noder i godtyckligt tidssteg. [mm]



Figur 4.2: Graf över komposanterna av totala kraften som verkar på fenan över tid. [N, s]

Figur 4.2 illustrerar hur kraften vid fenans rotationspunkt förändras när fenan rör sig i perioder.



Figur 4.3: Visualisering av trycket i ett tvärsnitt i xz-planet. [Pa]

Kartan över tryckfördelningen av fenan påvisar ett område med grön/gul färg, det vill säga ett högre tryck än resten av omgivningen se figur 4.3. Vid denna punkt möter fenan flödet av vätska.

5
Diskussion

Diskussionskapitlet syftar till att tolka och analysera resultatet som redovisades i föregående kapitel, Resultat. Metod, programvara, resultatens trovärdighet, vidareutveckling samt etiska aspekter diskuteras.

5.1 Parametrar

Den utvecklade modellen är konstruerad för att kunna simulera olika situationer och tillstånd. Genom att ändra olika variabler i programkoden kan programmet skräddarsys för användarens syfte. E-modul, svängningsamplitud och frekvens kan modifieras beroende på fenans storlek och effekten med vilken den ska drivas.

För att simulera olika driftsförhållanden kan även den omgivande vattenhastigheten ställas in, vilket gör att fenans egenskaper både vid start från stillastående och full marschfart kan analyseras.

5.2 Tillämpning av WaterLily i kurser om strömningsmekanik

WaterLily visade goda egenskaper för att kunna vara ett effektivt simuleringsverktyg, men var under tiden för kandidatarbetet i praktiken inte helt enkelt att arbeta med. Fördelarna som WaterLily erbjuder vägs ner av nackdelarna som kommer vid användning av paketet.

WaterLilys Github har ett flertal enklare exempel som illustrerar hur olika simuleringsmodeller kan byggas upp. Det som saknades var hur dessa kan tillämpas på samma problem. Exempelvis var vissa exempel för 2D och andra 3D. Något som saknades var exempel på simuleringar som behandlar geometrier som genomgår deformation under simuleringens gång likt vårt problem. WaterLily visade även brister i att simulera tryckskillnader när fenan rörde på sig. När trycket analyserades runt fenan förväntade vi oss vid pendling nedåt att trycket på undersidan skulle öka och trycket på ovansidan att minska men detta var inget som kunde läsas ut av WaterLily. Anmärkningsvärt högt tryck kunde endast ses vid början av fenan där den möter flödet av fluiden, detta tyder på att WaterLily har svårt att registrera hur fenans rörelse påverkar fluiden.
WaterLily har en relativt lång inlärningskurva och gruppen stötte på en del problem

kring enhets-hantering och -omvandling, detta eftersom paketet inte använder si-enheter utan sitt egna rutnät där en ruta är en längdenhet. Andra mer etablerade simuleringsprogram erbjuder lite mer interaktiv process. Något som är en fördel med programmet och som gör det attraktivt är det faktum att det är en enkel cfd-lösare som inte kostar någonting. Det positiva med att det är en enkel cfd-lösare är att användaren kan skapa simuleringar för sina problem och enkelt kunna köra och visualisera dem på en helt vanlig laptop. Detta innebär dock att man kanske inte kan få riktigt lika exakta simuleringar som med en tyngre cfd-lösare.
Gruppen anser dock att för få instruktioner kring WaterLily existerar och kan därför inte rekommendera att det används vid undervisning inom strömningsmekanik.

5.3 Felkällor

Under projektets gång gjordes ett antal avvägningar som var nödvändiga för att komma vidare, vilket kan ha haft en negativ inverkan på resultatet. För att kunna köra modellen på en laptop under en något sånär rimlig tid, krävdes en väldigt grov-huggen geometri av endast 28 noder. Denna förenkling kommer naturligtvis påverka resultatet, i en utsträckning som inte är helt fastslagen. Men man kan spekulera i att geometrin kanske inte deformeras helt naturligt framförallt ute i kanterna. Mycket av beräkningskraften äts upp av den explicita FEM-lösaren och dess behov av väldigt små tidssteg, som dessutom blir mindre i takt med att finheten på fenans rutnät ökar. Skulle FEM-lösaren modifieras till en stabilare och mindre beräkningsintensiv typ, hade en större upplösning varit möjlig till ett potentiellt mer verklighetstroget resultat.

En annan möjlig felkälla är att FEM-lösaren använder E-modul som främsta materialparameter, medan den närmaste specifikationen för fenan är dess Shore-värde, ett mått på motståndskraften mot intryckning. Tyvärr är det inte möjligt att exakt omvandla mellan dessa två liknande storheter. Istället får approximativa formler användas, vilka varierar ganska kraftigt i resultat. I fall utförligare materialspecifikationer funnits tillgängliga, skulle ingångsvärdena bli mer verklighetstrogna, och simuleringen ha en större chans att beräkna rimliga värden.

När trycket över fenan plottas under simulationen, blir det tydligt att fenans interaktion med det omgivande vattnet ej fungerar korrekt. Det resulterande trycket över fenan förefaller bara komma av massflödet i vattnet, som rör sig med en given hastighet, och inte från fenans egen rörelse. I verkligheten, för att lyckas driva en båt, måste fenans egna rörelse ge upphov till ett tryck som är långt mycket större än vattnets tryck över både fenan och båten.

För varje tidssteg i simulationen deformeras fenan, deformationen ges till WaterLily genom att fenans nät omdefinieras. Någonstans i den processen försvinner information från det föregående tidssteget, och fenan modellen beter sig som att fenan teleporteras genom vattnet. Antingen är modellen felaktigt konstruerad, eller så saknar WaterLily förmågan att hantera en geometri som deformeras utan ett fördefinierat mönster.

5.4 Reflektion på metod

Vid val av metod balanserades noggrannhet och genomförbarhet för arbetet. I detta arbete var en uppdelning i två huvuddelar naturlig, en strukturanalys via en egenutvecklad FEM-lösare i Julia, samt en CFD-modell med WaterLily.

Uppdelningen möjliggjorde parallellt arbete vilket var gynnsamt för arbetsgruppen. Detta innebar också en ökad svårighet när de två delarna skulle kopplas samman. Interpolering, datakonvertering och synkronisering behövde göras mellan simuleringarna. En mer enhetlig metod hade möjligtvis kunnat sköta kopplingen bättre.

Implementationen av den egna FEM-lösaren möjliggjorde ökad frihet och den kunde enkelt justeras efter arbetets behov. Med detta behövdes dock felsökning, verifiering och optimering för att fungera på rätt sätt. Valet att använda sig av explicit tidsintegration ökade genomförbarheten då andra metoder hade varit mer krävande. Den explicita tidintegration satte krav på tidsstegen vilket i praktiken begränsar simuleringshastigheten men i och med att simuleringsmodellen fortfarande är relativt snabbt var det ingen markant begränsning.

Valet att använda WaterLily som CFD-lösare var mer eller mindre del av arbetets syfte. WaterLily var snabbt och flexibelt men skulle behöva mer dokumentation. Det blev tydligt när mycket tid åtgick till felsökning av och förståelse av enkla problem. Även fast WaterLily stundvis inte fungerade på önskat sätt fick arbetsgruppen fram en fungerande strömningsmodell som delvis uppfyller syftet. För att underlätta arbetet hade gruppen kunnat fokuserat mer på WaterLily haft mer kontakt med skaparna för att få mer konsultation direkt från källan.

Den svaga FSI-koppling var ett välgrundat beslut för arbetet. Beräkningsbördan samt implementationsbördan var gynnades då andra mer komplexa metoder innebar stora ändringar i hela metoden.

Sammantaget var valet av metod rimligt, förhållande till arbetets resurser och förkunskaper. Simuleringsmodellen var tillräckligt robust för att modellera fenans grundläggande dynamik och ge inblick i hur parameterförändringar påverkar dess funktion. Med detta fanns det fortfarande flera förbättringsområden som diskuteras i senare kapitel.

5.5 Etik

En betydande del av syftet med delfinfenan som framdrivningssystem är etiskt driven. Dolprops delfinfena kan röra sig sömlöst, tyst och kavitationsfritt genom vattnet, i stark kontrast mot traditionella propellrar. Delfinfenan saknar vassa blad och hög rotationshastighet vilket innebär minskad skaderisk för både människor och andra djur som befinner sig i närheten.

Buller är idag ett stort hot mot känsliga ekosystem (EMSA, 2024), och att sänka magnituden av dessa skulle gynna det marina livet, havet och i förlängningen hela det globala ekosystemet.

De mest effektiva fartygspropellrarna som finns att tillgå har en hydrodynamisk verkningsgrad på ca 70%. Enligt Thomas Jemt kan delfinfenan nå en verkningsgrad på 92%, vilket innebär en bränsle- och utsläppsbesparing på ca. 24%. Dolprop har som mål att tillverka fenan för fraktfartyg, och i en värld där utsläppen från shippingbranschen enligt Europeiska kommissionen uppgår till 1076 miljoner ton (Commission, 2024), skulle en effektivisering på 24% i ett idealfall utgöra en besparing motsvarande hela Storbritanniens årliga CO2-utsläpp.

5.6 Vidareutveckling

På grund av begränsade resurser, i synnerhet tid, behövde kompromisser ske under arbetets gång. Detta ledde till att resultatet inte blev helt till gruppens tillfredsställelse och att vissa val av metod blev annorlunda än hur de hade sett ut i annat fall. Därför lämnas följande rekommendationer för framtida vidareutveckling av arbetet.

Geometri
Användningen av geometri i själva simuleringens två delar, FEM & CFD fungerar väl, men förarbetet innan simuleringarna och kopplingen mellan de två delarna hade med fördel fungerat annorlunda. Istället för att behöva hantera geometrin i Blender för att få ett grövre ytnät borde programmet kunna hantera geometrin som skickas in i form av exempelvis en stl-fil som innehåller ett ytnät eller en ren geometrifil i format .obj eller .step. Gmsh, som redan används i programmet för generering av volymnätet, kan automatiskt skapa nät från .step-filer där användaren kan kontrollera olika parametrar, men under arbetet lyckades inte ett tillräckligt grovt nät genereras för att det skulle fungera med den valda tidsintegreringen av FEM. Genom att antingen använda ett annat program än Gmsh för nätgenerering, eller använda en annan tidsintegrering av FEM som inte strikt kräver lika små element, skulle processen kunna ske automatiskt. På så sätt skulle programmet lättare kunna användas för iterativa tester av olika geometrier.

FEM, oavsett tidsintegrering, inte skall användas med alldeles för små element i nätet på grund av att det blir onödigt beräkningsintensivt. WaterLily å andra sidan kan hantera fina nät på geometrin utan att beräkningsintensiteten påverkas

nämnvärt, samtidigt som det leder till en mer verklighetstrogen simulering om nätet har finare nät. Istället för att samma ytnät används i de två delarna av simuleringen skulle det alltså vara fördelaktigt att använda ett grövre till FEM, och ett finare till WaterLily. Men då måste deformationerna överföras från FEM-nätet till WaterLily-nätet på något sätt, till exempel genom interpolering. Genom användning av exempelvis Julia-paketet ScatteredInterpolation kan en interpoleringsmodell skapas på det ursprungliga FEM-nätets koordinater med nodernas förflyttningar som värden, och WaterLily-nätets noders förflyttningar beräknas genom att i modellen, hitta värdena i WaterLily-nätets ursprungliga koordinater.

Geometrin i simuleringsmodellen kan kompletteras med resterande komponenter som påverkar flödet. Detta inkluderar komponenten som delfinfenan är fäst på och båtskrov samt andra relevanta geometrier som kan störa flödet. Eftersom dessa objekt inte är menade att deformeras under användning, till skillnad från fenan, kan de implementeras simplare enbart i WaterLily utan att några strukturberäkningar behöver utföras på dem. Därför kan de implementeras utan mycket extra beräkningsintensitet.

FEM
Istället för explicit tidsintegrering som kräver mycket korta tidssteg och begränsar vilken storlek elementen i nätet har, skulle en annan implicit tidsintegrering kunna användas. Det skulle möjliggöra att ha grövre nät i de delar av fenan som inte kräver så noggrann modellering samtidigt som det är finare nät i de delar som är bra att modellera noggrannare, bland annat bakkanten av fenan, där den är tunnare. Nackdelen med implicit tidsintegrering är svårare implementering och att varje tidssteg är mer beräkningsintensivt.

WaterLily
För att försäkra att modellen ger ett verklighetstroget resultat behöver WaterLily och hur det skall användas på rätt sätt undersökas mer. Eftersom de exempel på användning som finns på WaterLilys GitHub inte var tillräckliga för att få förståelse för hur programmet är sammanlänkat med fysiska parametrar, skulle det vara fördelaktigt att kontakta och söka information från skaparna av det. Undersökning av vilken storlek på domänen och vilken storlek på tidssteg som krävs för att ge tillräckligt noggrant resultat skulle även behöva genomföras.

FSI
Istället för svag formulering av FSI skulle stark formulering kunna användas. Det försäkrar jämvikt i varje tidssteg, vilket ger en mer pålitlig simuleringsmodell. Huruvida det skulle ge ett tillräckligt mycket mer verklighetstroget resultat skulle dock behöva undersökas, eftersom det innebär förutom mer komplicerad implementering, att varje tidssteg är mer beräkningsintensivt.

Resultatvärden
Eftersom hydrodynamisk verkningsgrad är viktig, skulle den behöva implementeras. För att beräkna den behövs även drivande effekt beräknas. Det kan utföras genom

att för varje noderna som rör sig i ett föreskrivet mönster, ta skalärprodukten av dess momentana hastighet och kraft som verkar på den, och summera alla noders bidrag. Även fler grafer och figurer skulle kunna vara användbara att få ut från modellen, samt en film som illustrerar förloppet. Att utföra en eller flera simuleringar på en stel propeller skulle kunna ge värden för jämförelse för en studie av framdrivningssätt.

6
Slutsats

Detta projekt har gått ut på att modellera en strömningssimulation av en delfinfena med Julia-paketet WaterLily. Simuleringen är uppbyggd av en FEM-lösare i kombination med CFD-lösaren i WaterLily, detta har resulterat i simuleringar av en delfinfena med verklighetstrogen deformation. Simuleringsmodellen visar fenans grundläggande dynamik och kan modifieras vid ändring av ingående parametrar.

Den största utmaningen med simuleringen som fortfarande kvarstår är att få modellen att simulera hur fenans rörelse förflyttar vätskan i sin närhet, vid skrivande stund rör sig fenan och flödet runt den men fenan flyttar ingen vätska när den rör sig. Detta i kombination med att simuleringar gjorts i en ganska grov upplösning gör att gruppen inte kan dra några större slutsatser kring hur väl fungerande fendriften är. Även om det på teoretisk nivå tyder på att fendriften skulle vara effektivare än traditionell framdrift hade mer avancerade simuleringar behövt göras för att kunna dra slutsatser kring detta.

Paketet WaterLily har god potential att vara användbart med sin relativt låga beräkningsintensitet men saknar dokumentation. Med en tydlig användarguide och fler exempel kan CFD-lösaren med stor sannolikhet vara användbar inom undervisning i strömningsmekanik. Metodvalet för projektet innebar några kompromisser för att främja genomförbarheten och hålla en rimlig detaljnivå.

Med vidareutveckling av arbetet kan modellen förbättras genom att justera hur geometrin hanteras, ändra FSI-metod samt hantering av värden in och ur modellen. Sammanfattningsvis har arbetets syfte delvis uppfyllts med en simuleringsmodell som utgör viss nytta för intressenter. En mer utvecklad modell krävs för att tillhandahålla data som kan användas vid optimering av delfinfenan.

Litteratur

Anderson, J. D. (1995). *Computational Fluid Dynamics: The Basics with Applications*. McGraw-Hill.

Benvenuto, G., Campora, U., & Trucco, A. (2014). Comparison of ship plant layouts forpower and propulsion systems withenergy recovery. *Journal of Marine Engineering and Technolog*, *13*(3), 3. https://doi.org/10.1080/20464177.2014.11658117

Commission, E. (2024). *Reducing emissions from the shipping sector*. Hämtad 3 februari 2025, från https://climate.ec.europa.eu/eu-action/transport/reducing-emissions-shipping-sector_en

EMSA. (2024). *Underwater noise*. Hämtad 6 februari 2025, från https://www.emsa.europa.eu/protecting-the-marine-environment/underwater-noise.html

FOI. (2024). *Ljudkartor över Nordsjön visar hur sjöfart påverkar marint djurliv*. Hämtad 21 mars 2024, från https://www.foi.se/nyheter-och-press/nyheter/2024-03-21-ljudkartor-over-nordsjon-visar-hur-sjofart-paverkar-marint-djurliv.html

Guo, J., Zhang, W., Han, P., Fish, F. E., & Dong, H. (2023). Thrust generation and propulsive efficiency in dolphin-like swimming propulsion. *Mechanical and Aerospace Engineering*, *18*(5), 1. https://doi.org/10.1088/1748-3190/ace50b

Liu, J., Zhang, C., Liu, Z., Zhao, R., An, D., Wei, Y., Wu, Z., & Yu, J. (2021). Design and analysis of a novel tendon-driven continuum robotic dolphin. *Bioinspiration & Biomimetics*, *16*(6), 065002. https://doi.org/10.1088/1748-3190/ac2126

OpenAI. (2025a). Illustration av hyperelasticitet ChatGPT [AI-genererad bild].

OpenAI. (2025b). Omslagsbild genererad av ChatGPT [AI-genererad bild].

Semenov, Y. (Red.). (2022). *Fluid/Structure Interactions*. MDPI. https://doi.org/10.3390/books978-3-0365-3251-6

Weymouth, G. D., & Font, B. (2025). WaterLily.jl: Fast and simple fluid simulator in Julia [Accessed: 2025-04-28].

Wu, S. R., & Gu, L. (2012). *Introduction to the Explicit Finite Element Method for Nonlinear Transient Dynamics*. John Wiley & Sons.

Zhu, W., & Gao, H. (2021). Hydrodynamic characteristics of bio-inspired marine propeller with various blade sections. *Ships and Offshore Structures*, *16*(2), 157. https://doi.org/10.1080/17445302.2020.1713039

Zienkiewicz, O. C., Taylor, R. L., & Zhu, J. (2021). *The Finite Element Method: Its Basis and Fundamentals* (7. utg.). Elsevier.

A

Bilagor

A.1 Simuleringsmodell

```
include("functions.jl")
using LinearAlgebra, GeometryBasics, WaterLily,
   WaterLilyPreCICE, Makie, StaticArrays, Interpolations,
   Makie


# Geometri, generera eller l s in mesh
newgeometry=false
if newgeometry
    gen_mesh()
end

nodes, triangles, tetrahedras = read_mesh()
num_nodes = size(nodes)[1]
num_elements = size(tetrahedras)[1]

# Material
E_module = 1000  # [N/mm2 = MPa]
   = 0.49
   = 1.2e-6  # [kg/mm^3]
  ,    = lame_parameters(E_module,   )
C1 =   /2
   = E_module/(3*(1-2*  ))
D1 =   /2

resolution = 64
cfd_scale = 5e-2  # G ra om mm till WaterLilys rutor,  ndra
    f r att fenan skall f  plats inom WaterLily-dom nen

characteristic_length = (maximum(nodes[:, 1])-minimum(nodes
   [:, 1]))  # Just nu satt till totala l ngden p  fenan

dt = 1e-5  # Tidssteg [s]
cfd_time_step = 5e-3  # Tidssteg f r CFD [s]
t_stop = 2
speed = 2572/5 # Vattnets hastighet [mm/s]
```

```
FE_length = resolution/cfd_scale  # WaterLily-dom nens
    l ngd i mm
rotation_point = [0 0 FE_length/2]  # Punkt som fenan roterar
    kring
rotation_amp = 30  # [grader]
frequency = 1.67  # [Hz]

cutInd = round(Int,resolution/3)

   = 1.01140  # [mm^2/2]
Re = speed*characteristic_length/    # Reynolds tal

# Ursprungligt Tidssteg
x_0 = copy(nodes) .+[FE_length/2 FE_length/2 FE_length/2] #
    Ursprungliga koordinater
u = zeros(num_nodes, 3)  # F rskjutningar
x = copy(x_0) + u  # Deformerade koordinater
v = zeros(num_nodes, 3)  # Hastigheter
a = zeros(num_nodes, 3)  # Accelerationer

# Initiella v rden f r fenan
invDx_0 = []
tetrahedras_mass = []
m = zeros(num_nodes)
for i in 1:num_elements
    x_01 = x_0[tetrahedras[i,1], :]
    x_02 = x_0[tetrahedras[i,2], :]
    x_03 = x_0[tetrahedras[i,3], :]
    x_04 = x_0[tetrahedras[i,4], :]
    Dxi = [x_02[1]-x_01[1] x_03[1]-x_01[1] x_04[1]-x_01[1];
       x_02[2]-x_01[2] x_03[2]-x_01[2] x_04[2]-x_01[2]; x_02
       [3]-x_01[3] x_03[3]-x_01[3] x_04[3]-x_01[3]]
    invDxi = inv(Dxi)
    push!(invDx_0, invDxi)

    V0e = tetrahedron_volume(x_01, x_02, x_03, x_04)
    massi =    * V0e
    push!(tetrahedras_mass, massi)

    mi = mass_vector(massi, x_01, x_02, x_03, x_04)  # Bidrag
        till nodernas massor
    nodeindicesi = tetrahedras[i, :]
    for (localindexi, nodeindexi) in enumerate(nodeindicesi)
        m[nodeindexi] += mi[localindexi]  # L gg till massan
            f r varje nod
    end
end
```

```
# Skapa figur f r visualiseringar
fig = Figure()
ax1 = Axis3(fig[1, 1], title="Position")
# Skapa observables
xvals = Observable(Float32[])
xvals[] = x[:,1]
yvals = Observable(Float32[])
yvals[] = x[:,2]
zvals = Observable(Float32[])
zvals[] = x[:,3]
scatter!(ax1, xvals, yvals, zvals, color=:blue, markersize=5)
linesegs = Observable(Vector{Tuple{Point3f, Point3f}}())

# Visa n tet f r fenan
function update_linesegs!()
    segs = Tuple{Point3f, Point3f}[]
    for triangle in eachrow(triangles)
        idx = triangle
        p1 = Point3f(xvals[][idx[1]], yvals[][idx[1]], zvals
           [][idx[1]])
        p2 = Point3f(xvals[][idx[2]], yvals[][idx[2]], zvals
           [][idx[2]])
        p3 = Point3f(xvals[][idx[3]], yvals[][idx[3]], zvals
           [][idx[3]])
        push!(segs, (p1, p2))
        push!(segs, (p2, p3))
        push!(segs, (p3, p1))
    end
    linesegs[] = segs
end

# Uppdatera linjesegmenten f rsta g ngen
update_linesegs!()

# Rita linjesegmenten
linesegplot = linesegments!(ax1, linesegs, color=:black)

limits!(ax1, 0, FE_length,
            0, FE_length,
            0, FE_length)


# Tidssteg i explicit FE-l saren
function explicit_time_step( t , f_ext; update_figure=false)
    global x, v, u, time
    # Ber kna interna krafter f r varje element
    f_int = zeros(num_nodes, 3)  # Interna krafter

    for e in 1:num_elements
```

```
        nodeindicese = tetrahedras[e, :]
        xe = x[nodeindicese, :]

        Jac = Jacobian(xe)

        Ve = tetrahedron_volume_new(Jac)

         N   = shape_function_gradients_new(Jac)

        Fe = Jac*invDx_0[e]
        Je = det(Fe)

        #= # Piola Kirschoff, linj r materialmodell
        Ce = Fe'*Fe
        invCe = inv(Ce)
        Se =   *(I-invCe)+  *log(Je)*invCe  # Piola Kirschoff
           -sp nning, f rsta eller andra??
         e  = 1/Je*Fe*Se*Fe'  # Cauchy-sp nning =#

        # Neo Hooke, hyperelastisk materialmodell
        Be = Fe*Fe'
        I1 = tr(Be)
        I1_ = I1*Je^(-2/3)
        p = -2*D1*Je*(Je-1)
        B_ = Je^(-2/3)*Be
        devB_ = B_-1/3*I1_*I
         e  = 1/Je*(-p*I+2*C1*devB_)

        f_inte =  N  '* e *Ve

        f_int[nodeindicese, :] .+= f_inte
    end
    f_tot = f_ext - f_int  # Totala krafter
    a[free_nodes,:] = f_tot[free_nodes,:] ./ m[free_nodes,:]
        # Acceleration
    v[free_nodes,:] +=  t  * a[free_nodes,:]
    x[free_nodes,:] +=  t  * v[free_nodes,:]
    #u[free_nodes,:] +=  t  * v[free_nodes,:]
    #x = x_0 + u

    if update_figure
        xvals[] = x[:,1]  # Uppdatera x-koordinater
        yvals[] = x[:,2]  # Uppdatera y-koordinater
        zvals[] = x[:,3]  # Uppdatera z-koordinater
        update_linesegs!()
    end
end
```

```
glmesh = GeometryBasics.Mesh(map(row -> Point{3, Float64}(row
    ), eachrow(x)), GLTriangleFace.(map(row -> Tuple(row),
    eachrow(triangles))))

# Skapa WaterLily-simulering
function gensim(;L=64,U=1, char_len, Reynolds)
    body = MeshBody(glmesh)
    Simulation((2L,L,L), (U,0,0), char_len;   =U*char_len/
       Reynolds, body=body)
end

# Anrop p  WaterLily-sim, os kert om korrekt v rden
    anv nds
#simul = gensim(L=resolution, U=speed*cfd_scale, char_len=
    characteristic_length*cfd_scale, Reynolds=Re)
simul = gensim(L=resolution, U=speed/1000, char_len=
    characteristic_length/1000, Reynolds=Re)

#Visa geometri
ax2 = Axis3(fig[2, 1], title="Hastighet")
# Skapa observables f r  _array
u_array_obs = Observable(simul.flow.u[:,:,:,1] |> Array)

Makie.contour!(ax2, u_array_obs, levels=[0], color=:black)

# Plotta krafter i x, y och z-riktning
ax3 = Axis(fig[1, 2], title="Krafter")
xlims!(ax3, 0, t_stop)
scatter!(ax3, [-10], [0], color=:red, markersize=10, label="x
    ")
scatter!(ax3, [-10], [0], color=:green, markersize=10, label=
    "y")
scatter!(ax3, [-10], [0], color=:blue, markersize=10, label="
    z")
Legend(fig[1,3], ax3)

# Inkludera ticks f r varje period
ticks = Float32[]
labels = String[]
tt = 0.0
while tt<=t_stop
    global tt
    push!(ticks, tt)
    push!(labels, string(tt))
    tt += .5
end
pp = 1
while pp/frequency<=t_stop
    global pp
```

```
    push!(ticks, pp/frequency)
    push!(labels, "p$pp")
    pp += 1
end
append!(ticks, [1/frequency])
append!(labels, ["p1"])
ax3.xticks = (ticks, labels)



f_ext = zeros(num_nodes, 3)

prescribed_nodes = [17, 18, 21, 22, 25, 26, 27, 28]  # Noder
   som r r sig i fixerade m nstret
free_nodes = setdiff(collect(1:num_nodes), prescribed_nodes)



x, prescribed_x = prescribed_cosinus(x, prescribed_nodes,
   rotation_point, rotation_amplitude=rotation_amp,
   periods_per_second=frequency)  # Skapar en funktion som
   roterar fenan kring rotation_point och uppdaterar x s
   att fenan b rjar i ett av    ndlgena
xvals[] = x[:,1]
yvals[] = x[:,2]
zvals[] = x[:,3]
update_linesegs!()
glmesh = GeometryBasics.Mesh(map(row -> Point{3, Float64}(row
   ), eachrow(x*cfd_scale)), GLTriangleFace.(map(row -> Tuple
   (row), eachrow(triangles))))  # Mesh som MeshBody kan ta
   emot
simul.body = MeshBody(glmesh)
sim_step!(simul, 0.00; remeasure=true)

# Visa tryck i ett tv rsnitt i xz-planet
ax4 = Axis(fig[2, 2], title="Tryck")
limits!(ax4, 1, 2resolution+2,
            1, resolution+1)
tryck = Observable(simul.flow.p[:,32,:] |> Array)
pressure_plot = contourf!(ax4, tryck, levels=range(-2, 2,
   length=20))
colorbar = Colorbar(fig[2, 3], pressure_plot)

display(fig)

# Loopa  ver  tiden
time = 0
for i in 0:t_stop/dt
    global glmesh,  _array  , f_ext, time
```

```
    prescribed_x(i*dt)  # FLytta p  prescribed_nodes
    if i%(round(cfd_time_step/dt)) == false  # K r WaterLily
        varje cfd_time_step
        time = i*dt
        println(time, " s")
        # K r WaterLily
        glmesh = GeometryBasics.Mesh(map(row -> Point{3,
           Float64}(row), eachrow(x*cfd_scale)),
           GLTriangleFace.(map(row -> Tuple(row), eachrow(
           triangles))))
        simul.body = MeshBody(glmesh)
        sim_step!(simul, sim_time(simul)+cfd_time_step;
           remeasure=true)
        #sim_step!(simul, time; remeasure=true)
        #println(sim_time(simul))  # Fels kning, WaterLilys
           tid verkar inte uppdateras enligt tidsstegen den
           borde ta
        scatter!(ax3, [time], [WaterLily.pressure_force(simul
           )[1]], color=:red, markersize=10)
        scatter!(ax3, [time], [WaterLily.pressure_force(simul
           )[2]], color=:green, markersize=10)
        scatter!(ax3, [time], [WaterLily.pressure_force(simul
           )[3]], color=:blue, markersize=10)

        tryck[] = simul.flow.p[:,32,:] |> Array  # Uppdatera
           tryck
        u_array_obs[] = simul.flow.u[:,:,:,1] |> Array  #
           Uppdatera u_array

        p_itp = interpolate(simul.flow.p, BSpline(Linear()),
           OnGrid())

        f_ext = zeros(num_nodes, 3)

        # Fr n trycket, r'kna ut yttre kraften p  varje nod
        pressures = [p_itp((point+[1.5, 1.5, 1.5])...) for
           point in eachrow(x*cfd_scale)]  # trycket i inre
           noder beh ver egentligen inte r knas ut
        for triangle in eachrow(triangles)
            point1 = x[triangle[1], :]
            point2 = x[triangle[2], :]
            point3 = x[triangle[3], :]
            edge1 = point2 - point1
            edge2 = point3 - point1
            n = cross(edge2, edge1)
            areaa = 0.5 * norm(n)
            normal = n / norm(n)
            force = mean(pressures[triangle]) * areaa *
               normal / 3    # Medelv rde av trycket * arean
```

```
                *normalen / 3 f r att f   kraften som verkar
                 p   en nod
            f_ext[triangle, :] .+= force'
        end
        explicit_time_step(dt, f_ext, update_figure=true)
    else
        explicit_time_step(dt, f_ext)
    end
end


println("Done")
```

```
using LinearAlgebra, GLMakie, Statistics, Rotations

function gen_mesh()
    # K r terminalkommandot (t.ex. skapa en mesh-fil med
       Gmsh)
    run(`$(joinpath(pwd(), "gmsh.exe")) $(joinpath(pwd(), "
       mesh.geo")) -3 -format msh`)

    # V nta p  att filen finns innan vi forts tter (extra
       s kerhet)
    while !isfile("mesh.msh")
        sleep(0.1)  # V nta 100 ms och kolla igen
    end
end

function read_mesh()
    lines = readlines("mesh.msh")

    # L s genom alla noder
    node_start = findfirst(l -> l == "\$Nodes", lines)
    node_end = findfirst(l -> l == "\$EndNodes", lines)

    node_lines = lines[node_start+1:node_end-1]  # Extrahera
       noddata

    nodes = Float64[]
    for row in node_lines[3:end]
        rowlist = split(row)
        if length(rowlist) > 1 && length(rowlist) != 4
            rowlist = parse.(Float64, rowlist)
            nodes = isempty(nodes) ? rowlist' : vcat(nodes,
               rowlist')
        end
    end

    # L s genom alla element
```

```
    elem_start = findfirst(l -> l == "\$Elements", lines)
    elem_end = findfirst(l -> l == "\$EndElements", lines)

    elem_lines = lines[elem_start+1:elem_end-1]  # Extrahera
       elementdata

    triangles_until = parse(Int, split(elem_lines[2])[4])

    triangles = Float64[]
    for row in elem_lines[3:triangles_until+2]
        rowlist = parse.(Int, split(row))
        triangles = isempty(triangles) ? rowlist[2:end]' :
           vcat(triangles, rowlist[2:end]')
    end

    tetrahedras = Float64[]
    for row in elem_lines[triangles_until+4:end]
        rowlist = parse.(Int, split(row))
        tetrahedras = isempty(tetrahedras) ? rowlist[2:end]'
           : vcat(tetrahedras, rowlist[2:end]')
    end
    return nodes, triangles, tetrahedras
end

function ShoreA_to_E(shore)
    E = .0981*(56+7.66*shore)/(.137505*(254-2.54*shore))
       *1e6
    return E
end

function ShoreD_to_E(shore)
    E = 2.5*10^(0.0235*shore) * 1e6   #exp((shore+50)
       *0.0235-0.6403) * 1e6
    return E
end

lame_parameters(E_module,   ) = (    = E_module *     / ((1 +
     ) * (1 - 2  )),     = E_module / (2 * (1 +   )))

function tetrahedron_volume(corner1, corner2, corner3,
   corner4)
    x1, y1, z1 = corner1
    x2, y2, z2 = corner2
    x3, y3, z3 = corner3
    x4, y4, z4 = corner4

    J = [x2-x1 x3-x1 x4-x1;
         y2-y1 y3-y1 y4-y1;
         z2-z1 z3-z1 z4-z1]
```

```
    Vl = det(J) / 6  # Volym
    if Vl <= 0
        error("volume wrong")
    end
    return Vl
end
function tetrahedron_volume_new(Jl)  # Snabbare
    # Ber kna volymen
    Vl = det(Jl) / 6  # Volym
    if Vl <= 0
        error("volume wrong")
    end
    return Vl
end

function shape_function_gradients(corner1, corner2, corner3,
   corner4)
    x1, y1, z1 = corner1
    x2, y2, z2 = corner2
    x3, y3, z3 = corner3
    x4, y4, z4 = corner4

    J = [x2-x1 x3-x1 x4-x1;
         y2-y1 y3-y1 y4-y1;
         z2-z1 z3-z1 z4-z1]

     N   = inv(J') * [-1 1 0 0;
                      -1 0 1 0;
                      -1 0 0 1]

    return  N    # Returnera gradienterna f r varje
       formfunktion
end
function shape_function_gradients_new(Jl)  # Snabbare
    # Ber kna gradienterna f r formfunktionerna
     N   = inv(Jl') * [-1 1 0 0;
                       -1 0 1 0;
                       -1 0 0 1]

    return  N    # Returnera gradienterna f r varje
       formfunktion
end

function B_matrix(grad_N)
    B = zeros(6, 12)
    for i in 1:4
        B[1, 3i-2] = grad_N[1, i]
        B[2, 3i-1] = grad_N[2, i]
        B[3, 3i] = grad_N[3, i]
```

```
        B[4, 3i-2] = grad_N[2, i]
        B[4, 3i-1] = grad_N[1, i]
        B[5, 3i-1] = grad_N[3, i]
        B[5, 3i] = grad_N[2, i]
        B[6, 3i-2] = grad_N[3, i]
        B[6, 3i] = grad_N[1, i]
    end
    return B
end

function mass_vector(mass, corner1, corner2, corner3, corner4
   )
    center_of_mass = [mean((corner1[1], corner2[1], corner3
       [1], corner4[1])), mean((corner1[2], corner2[2],
       corner3[2], corner4[2])), mean((corner1[3], corner2
       [3], corner3[3], corner4[3]))]
    w_nodes = []
    for node in (corner1, corner2, corner3, corner4)
        w_node = 1/(norm(node - vec(center_of_mass)))
        append!(w_nodes, w_node)
    end
    sum_w = sum(w_nodes)
    m = w_nodes/sum_w*mass  # Vektor, en vikt f r varje nod
    return m
end
function mass_vector_new(mass, corners)  # Snabbare
    # Ber kna masscentrum
    center_of_mass = mean(corners, dims=1)  # Medelv rdet av
        varje kolumn (x, y, z)

    # Ber kna vikter f r varje nod
    w_nodes = 1 ./ [norm(corners[i, :] .- center_of_mass) for
        i in 1:4]

    # Normalisera vikterna och multiplicera med massan
    w_nodes /= sum(w_nodes)
    m = w_nodes * mass  # Vektor med massandelar f r varje
       nod

    return m
end

function Jacobian(corners::Matrix{Float64})
    J = [corners[2, :]-corners[1, :] corners[3, :]-corners[1,
        :] corners[4, :]-corners[1, :]]
    return J
end

function prescribed_cosinus(xloc, prescribed_nodes,
```

```
    rotation_point; rotation_axis=2, rotation_amplitude=45,
    periods_per_second=1)
     # S tt alla noder i  vre  rotation_angle
     axis = zeros(3)
     axis[rotation_axis] = 1
     angle = deg2rad(rotation_amplitude)
     rotation_matrix = RotMatrix(AngleAxis(angle, axis...))
     translated_point = xloc .- rotation_point
     x_new = (rotation_matrix * translated_point')' .+
        rotation_point

     # Skapa en funktion som kan rotera prescribed_nodes
     x_prescribed = xloc[prescribed_nodes, :]
     translated_pointint = x_prescribed .- rotation_point
     function rot(t)
         current_angle = angle*cos(2*  *periods_per_second*t)
         rotation_matrixint = RotMatrix(AngleAxis(
            current_angle, axis...))
         x[prescribed_nodes, :] .= (rotation_matrixint *
            translated_pointint')' .+ rotation_point
     end
     return x_new, rot
end
```

INSTITUTIONEN FÖR MEKANIK OCH MARITIMA VETENSKAPER
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige
www.chalmers.se

CHALMERS