Rapport

Baneplanlegging for ringnotfartøy

Sluttrapport fra arbeidspakke 2 i prosjektet *Fangstkontroll i notfiske etter pelagiske arter: Fase 2*

Forfatter(e)
Joakim Haugen
Lars T. Kyllingstad

Rapportnummer
2021:00578– Åpen

Oppdragsgiver(e)
Fiskeri- og havbruksnæringens forskningsfinansiering (FHF)

SINTEF Ocean AS

Postadresse:
Postboks 4762 Torgarden
7465 Trondheim

Sentralbord: 46415000

ocean@sintef.no

Foretaksregister:
NO 937 357 370 MVA

EMNEORD:
Fiskeriteknologi
Ringnot
Baneplanlegging
Fangstkontroll

Rapport

Baneplanlegging for ringnotfartøy

Sluttrapport fra arbeidspakke 2 i prosjektet *Fangstkontroll i notfiske etter pelagiske arter: Fase 2*

VERSJON
1

DATO
2021-06-01

FORFATTER(E)
Joakim Haugen
Lars T. Kyllingstad

OPPDRAGSGIVER(E)
Fiskeri- og havbruksnæringens forskningsfinansiering (FHF)

OPPDRAGSGIVERS REFERANSE
FHF-prosjektnummer 901350

PROSJEKTNUMMER
302002652

ANTALL SIDER OG VEDLEGG
12 + 87

SAMMENDRAG
Hovedresultatet fra arbeidspakke 2 i prosjektet *Fangstkontroll i ringnot etter pelagiske arter: Fase 2* er en baneplanleggingsalgoritme for ringnotfartøy. Gitt informasjon om fartøy, stim og omgivelser kan algoritmen beregne en bane som fartøyet bør følge, et punkt langs denne banen hvor settingen av nota bør starte, samt tidspunktet dette bør skje på. Algoritmen kan benyttes til beslutningsstøtte om bord, før og under kasting av nota. Vi har laget et grafisk beslutningsstøttesystem som demonstrerer dette i praksis.

UTARBEIDET AV
Joakim Haugen

SIGNATUR
Joakim Haugen (Jun 1, 2021 17:42 GMT+2)

KONTROLLERT AV
Harry Westavik

SIGNATUR

GODKJENT AV
Gunvor Øie

SIGNATUR
Gunvor Øie (Jun 1, 2021 17:30 GMT+2)

RAPPORTNUMMER
2021:00578

ISBN
978-82-14-07678-3

GRADERING
Åpen

GRADERING DENNE SIDEN
Åpen

Historikk

VERSJON	DATO	VERSJONSBESKRIVELSE
1	2021-06-01	Første versjon levert til FHF.

Innhold

VEDLEGG

1. *mimir*: Teknisk dokumentasjon av baneplanleggingsalgoritmen, inkludert matematisk formulering
2. *balder*: Teknisk dokumentasjon av beslutningsstøttesystem

1 Introduksjon

Det å sette en ringnot krever at skipperen har god situasjonsforståelse og kontroll på de ulike elementene i prosessen. Fartøyets posisjon, kurs og hastighet må ses i sammenheng med stimens hastighet, retning og størrelse, samt vannstrøm og andre eksterne forhold. Det krever mye trening og erfaring å få til dette på en god måte. Selv for en erfaren ringnotskipper er det mye å holde styr på. Idéen bak arbeidspakke 2 i prosjektet *Fangstkontroll i notfiske etter pelagiske arter: Fase 2* har vært å lage et beslutningsstøttesystem som kan avlaste skipperen og bidra til situasjonsforståelsen ved å automatisere noen av disse elementene. Hovedresultatet fra arbeidspakken er en *baneplanleggingsalgoritme*. Det vil si en algoritme som – gitt tilstrekkelig informasjon om fartøy, stim og omgivelser – kan beregne en bane som fartøyet bør følge og et punkt langs denne banen hvor man bør begynne å sette ut nota for å fange stimen på best mulig vis.

1.1 Digitalisering av fangstprosessen

Et moderne ringnotfartøy er utstyrt med sofistikerte instrumenter som bidrar til situasjonsforståelsen under en fangstoperasjon, slik som sonar, GPS, strømprofilmåler og mye annet. Informasjonen fra disse presenteres via en rekke ulike skjermer på broa. Det er opp til skipperen å integrere informasjonen til et samlet bilde av situasjonen. En dyktig ringnotskipper må derfor både ha god persepsjons- og prediksjonsevne. Det vil si at vedkommende må evne å forutsi sannsynlige utfall av ulike operasjonelle valg basert på den tilgjengelige informasjonen. Valgene som vil påvirke utfallet er i hovedsak knyttet til navigasjon av fartøyet og tidspunktet man begynner å sette ut nota.

Prosedyren har mye til felles med et typisk *optimalreguleringsproblem*, og dette har vært mye av bakgrunnen for fremgangsmåten vi har valgt under arbeidet med å lage algoritmen. På mange måter kan man si at skipperen opptrer som en *modellprediktiv regulator* hvis *målsetning* det er å gjennomføre en ringnotoperasjon på optimalt vis. Litt mer systematisk kan vi identifisere følgende sentrale elementer:

- bruk av tilgjengelig sensorisk informasjon
- forståelse av prosessens hovedkomponenter
- prediksjon av sannsynlige utfall
- planlegging av en rekke målrettede handlinger
- oppnåelse av et veldefinert mål

Disse sammenfaller akkurat med byggesteinene i et optimalreguleringsproblem. Dette er hovedbegrunnelsen for vår beslutning om å bruke optimering som problemløsningsmetode, men det er ikke den eneste. En annen årsak er at metoden gir stor fleksibilitet når det gjelder problemformulering, og dermed også det mulige løsningsrommet. Med dette mener vi at det kan tenkes å finnes brukerspesifikke preferanser som vil få følger for rådene som presenteres til brukeren. Metoden er også svært åpen for senere utvidelser og forbedringer, for eksempel dersom man ønsker å inkludere nye datakilder eller endre på den matematiske formuleringen.

1.2 Beskrivelse av leveransen

Følgende elementer inngår i prosjektleveransen:

- en programvareimplementasjon av baneplanleggingsalgoritmen (kalt *Mimir*)
- et grafisk beslutningsstøttesystem for demonstrasjonsformål (kalt *Balder*)
- teknisk dokumentasjon av programvaren, inkludert en matematisk beskrivelse av algoritmen
- denne rapporten

Denne rapporten er kortfattet og skrevet med tanke på et generelt publikum. Vi gir en populærvitenskapelig beskrivelse av baneplanleggingsmetoden i seksjon 2 og en demonstrasjon i bruk av beslutningsstøttesystemet i seksjon 3. Teknisk dokumentasjon med mer dyptpløyende matematisk beskrivelse er inkludert som vedlegg til rapporten. Denne er først og fremst skrevet med tanke på aktører som ønsker å bygge videre på våre resultater, enten for å utvikle det videre til kommersiell beslutningsstøtteprogramvare eller for videre forskning. Vi diskuterer noen slike muligheter i seksjon 4. Selve programkoden er i skrivende stund publisert som åpen kildekode på *GitHub*, et populært nettsted for deling av kildekode og samarbeid om programvareutvikling. Her vil man i tillegg kunne finne enda mer detaljert dokumentasjon (API-dokumentasjon) av programvaren. Lenker til kildekode og andre ressurser kan finnes via prosjektets hjemmeside, `http://fangstkontroll.no`.

2 Metode

2.1 Algoritmeutvikling

Målet vårt var å lage et beslutningsstøttesystem for bruk ombord under setting av ringnot. Vi begynte med å etablere sentrale byggeklosser for formulering av optimalreguleringsproblemet. Nærmere bestemt måtte vi lage detaljerte matematiske beskrivelser av elementene vi nevnte i seksjon 1.1. Vi utformet formuleringen som et såkalt *baneplanleggingsproblem* hvis løsning består av

- en foreslått bane som fartøyet skal følge
- et punkt i tid og rom der settingen av nota skal starte
- eventuell tilleggsinformasjon som kan bidra til økt situasjonsforståelse

Når systemet har funnet en løsning kan denne presenteres til skipperen på en lettfattelig måte, for eksempel via en grafisk visualisering som inkluderer kartvisning av den foreslåtte banen.

Informasjonen vi valgte å la inngå i løsingen av problemet var:

- brukerens preferanser for hvordan prosessen skal utføres
- stimens dybde, posisjon og hastighet i forhold til fartøyet
- informasjon om overflatestrømhastighet og -retning

Data om de to sistnevnte må være gjort tilgjengelig fra de relevante instrumentene (sonar, GPS osv.) via et passende kommunikasjonsgrensesnitt.[1]

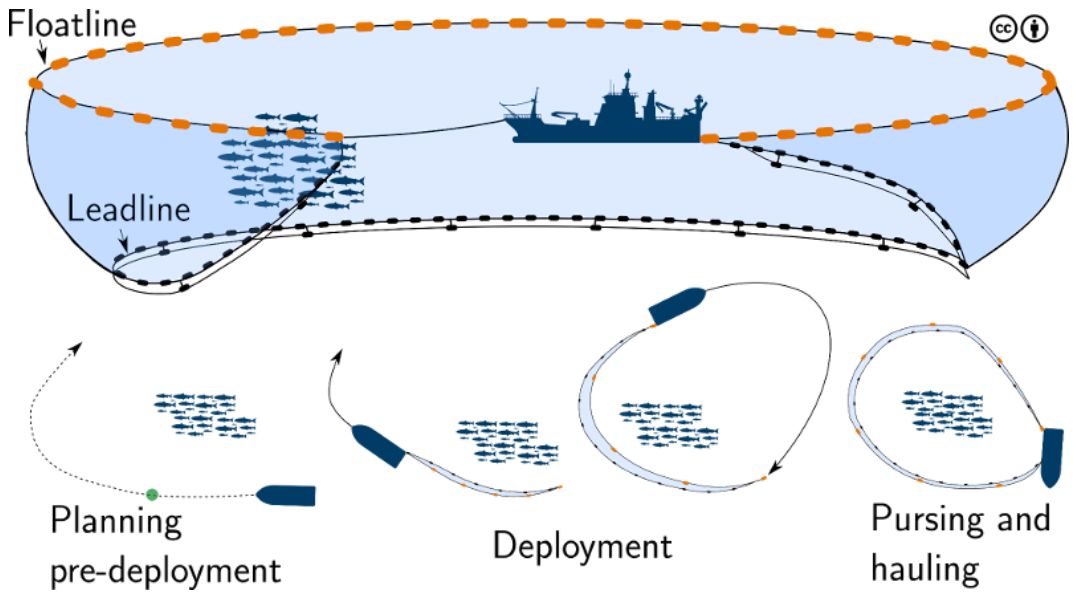
En gyldig løsning av problemet må oppfylle to hovedkrav:

- Nota settes ut i en ellipsebane rundt stimen, sett fra en referanseramme som følger overflatestrømmen.
- Grunntelna skal ha sunket en brukerdefinert margin under stimens dybde innen stimen når frem til nota.

Vi delte fangstprosessen inn i to faser: *før* og *etter* at man har begynt å sette nota. I den første fasen handler det om å styre fartøyet til det punktet der nota skal settes ut og ellipsebanen skal starte. Her må man ta i betraktning fartøyets til enhver tid gjeldende posisjon og forventede hastighet for å foreslå en bane som ender på riktig sted til riktig tid. Etter at setting har startet må fartøyet følge ellipsebanen i riktig hastighet slik at nota rekker å synke til riktig dybde innen fisken har nådd frem til notveggen. En illustrasjon av disse stegene er vist i figur 1.

Vi jobbet ut fra en hypotese om at en optimeringsbasert algoritme kan produsere et sett av foreslåtte handlinger basert på prediksjon. Dette beror på at man har laget en tilstrekkelig detaljert matematisk beskrivelse av de overnevnte prosessene, kravene og ikke minst det fysiske systemet. Sistnevnte består av fartøyet, stimen,

[1] Et slikt kommunikasjonsgrensesnitt var den første leveransen fra arbeidspakken. Se J. Haugen (2018), *Programvare for datainnsamling* (prosjektnotat, SINTEF Ocean) for mer informasjon.



Figur 1: En skisse av fangsprosessen. De nederste bildene viser henholdsvis fasen før setting, under setting og etter setting av nota.

nota og havmiljøets påvirkning på disse. Når det gjelder nota så er det i hovedsak *synkeresponsen*, altså hvor fort grunntelna synker, som er det essensielle.

Et viktig aspekt ved formuleringen var timing av grunntelnas synkedybde i forhold til fiskestimens og fartøyets bevegelsesbaner. Vi identifiserte fire avgjørende tidspunkter knyttet til sentrale hendelser under notutsettingsprosessen:

1. Setting av nota iverksettes ved posisjon p_d (*deployment point*) og tidspunkt t_d.
2. Fartøyet når posisjonen p_c (*collision point*) ved tidspunkt t_s.
3. Fisken er innringet ved tidspunkt t_{pd}, idet fartøyet når posisjon p_{pd}.
4. Fisken treffer notveggen ved p_c på tidspunkt t_c.

På disse tidspunktene vil fartøyet og stimen befinne seg på bestemte posisjoner i forhold til hverandre som illustrert i figur 2. (Starttidspunktet for hele prosessen er angitt som t_0 og utgangsposisjonene til fartøyet og stimen som henholdsvis $p_{v,0}$ og $p_{s,0}$ i figuren.) Baneplanleggerens mål er å bestemme tidspunktet t_d på en måte som oppfyller alle de tidligere nevnte kravene til en vellykket operasjon. Vi gir en detaljert beskrivelse av hvordan dette er løst matematisk i vedlegget.

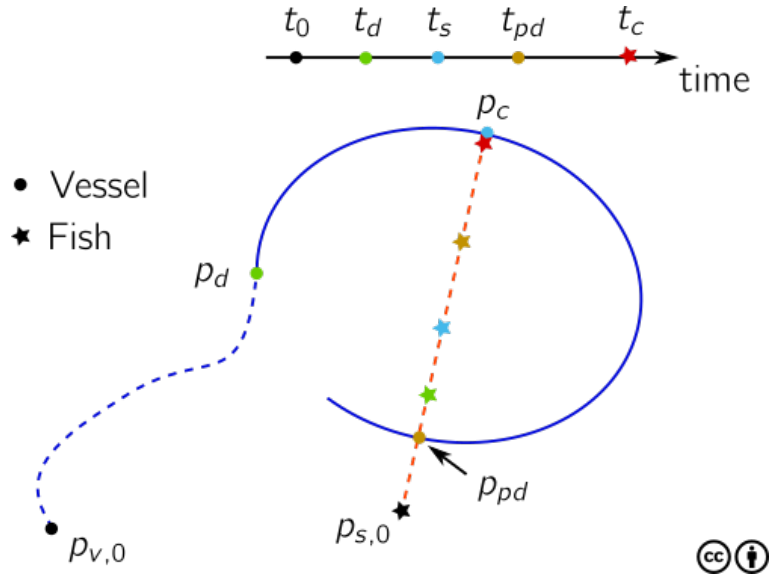
2.2 Testing

Vi har laget en simulator for å enkelt kunne teste og demonstrere baneplanleggeren under ulike forhold og situasjoner. Denne fungerer som en "stand-in" for den fysiske virkeligheten på den måten at den inneholder (sterkt forenklede) matematiske modeller av et fartøy og en fiskestim, og den sender ut signaler som tilsvarer dem algoritmen ville benytte om bord i et virkelig fartøy:

- fartøyets posisjon, kurs og hastighet
- stimens posisjon, kurs, hastighet og dybde

Simulatoren er implementert ved hjelp av en modifisert versjon av simuleringsprogramvaren *cosim* fra prosjektet *Open Simulation Platform*.[2] Modifikasjonene går ut på at programvaren er utvidet med mulighet for å sen-

[2] `https://https://opensimulationplatform.com`



Figur 2: En oversikt over viktige punkter i fangstprosessen.

de ut outputsignaler over kommunikasjonsprotokollen DDS, den samme protokollen som benyttes av SINTEF Oceans datainnsamlingsprogramvare *Ratatosk*. Dermed skal veien fra simulatortesting til fremtidig ombordtesting være relativt smertefri.

3 Programvare

Programvaren består av en beregningsapplikasjon og en grafisk brukerapplikasjon. Her beskriver vi kort den grafiske applikasjonen og hvordan brukeren kan konfigurere brukerinnstillinger som påvirker foreslåtte settebaner. Det er to sentrale visninger, nemlig en dashbordfane og en konfigurasjonsfane. Hovedvisningen er et dashbord med kartplott og nøkkeltall om miljø, fartøy og stim, se figur 3a. Kartet viser foreslått settebane med settepunkt, anbefalt kursrate og nedtelling til anbefalt settetidspunkt. Konfigurasjonsfanen består av innstillinger som påvirker den kalkulerte settebanen, se figur 3b. Dette inkluderer settefart, settebanedimensjoner, settepunkt i forhold til stimen, marginer og koeffisienter for synkerespons.

En typisk brukersesjon består av fire steg som illustrert i figur 4:

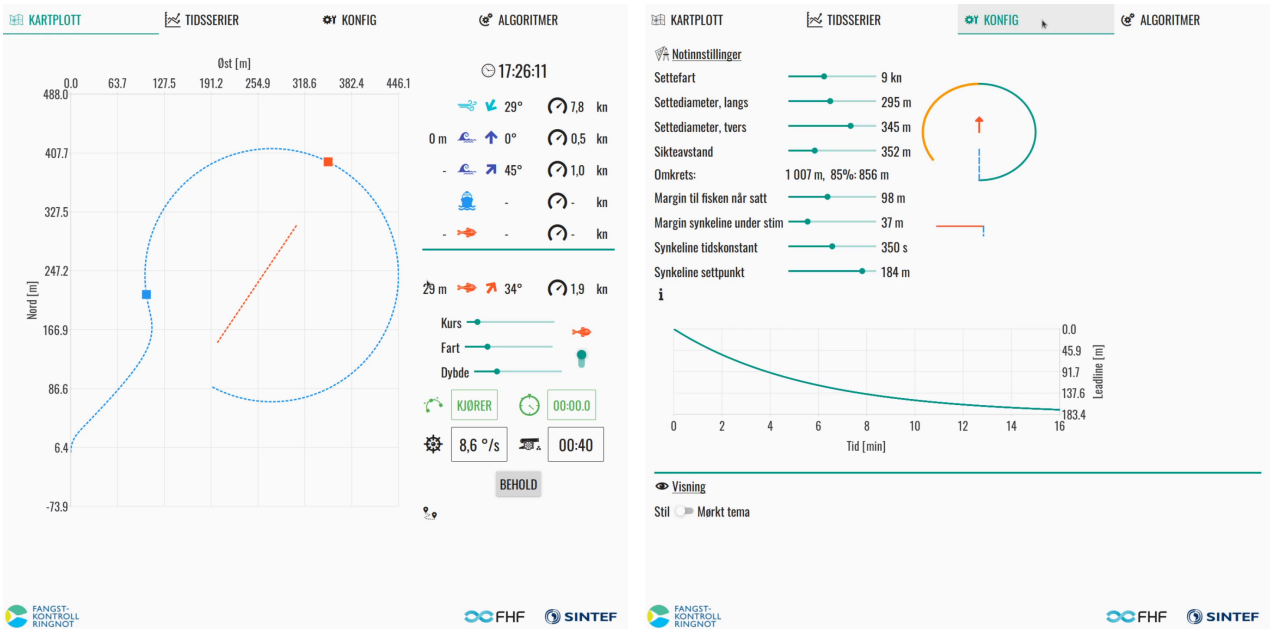
1. Start algoritmene i algoritmefanen, figur 4a. Beregningsapplikasjonene initialiserer og gjør seg klar til å levere anbefalte settebaner.

2. Sett brukerinnstillinger, figur 4b. Brukeren setter innstillinger i henhold til redskapkonfigurasjon, ønsket settebanedimensjoner og marginer som synkeline–stim og stim–fartøy.

3. Brukeren velger å overstyre stimbeskrivelse, figur 4c. Brukeren kan sette retning, fart og dybde på stimbeskrivelsen i stedet for beregninger fra sonaren.

4. Foreslått settebane er akseptabel og fryses i visningen, figur 4d. Den foreslåtte settebanen beholdes i kartplottet. Anbefalt kursrate og nedtelling i henhold til den beholdte settebanen oppdateres etter hvert som tiden går.

Brukerinnstillinger og andre forhold påvirker de foreslåtte settebanene. Vi viser skjermdumper av anbefalte settebaner for ulike scenarier i figur 5.



(a) Dashbord

(b) Konfigurasjon

Figur 3: Hovedvisning og konfigurasjonfane.

4 Fremtidsutsikter

Som nevnt i introduksjonen er alle programvarekomponentene publisert som åpen kildekode på nettjenesten GitHub. Vi har publisert dem under OSI-godkjente[3] lisenser slik at hvem som helst – både bedrifter, andre forskningsinstitusjoner og privatpersoner – kan benytte seg av dem og bygge videre på dem.

I de følgende seksjonene vil vi gi noen betraktninger av hvilke steg vi mener gjenstår for å gjøre dette til et kommersielt verktøy med nytteverdi for fiskerinæringen, samt si noen ord om hvor vi mener det er behov for en ytterligere forskningsinnsats.

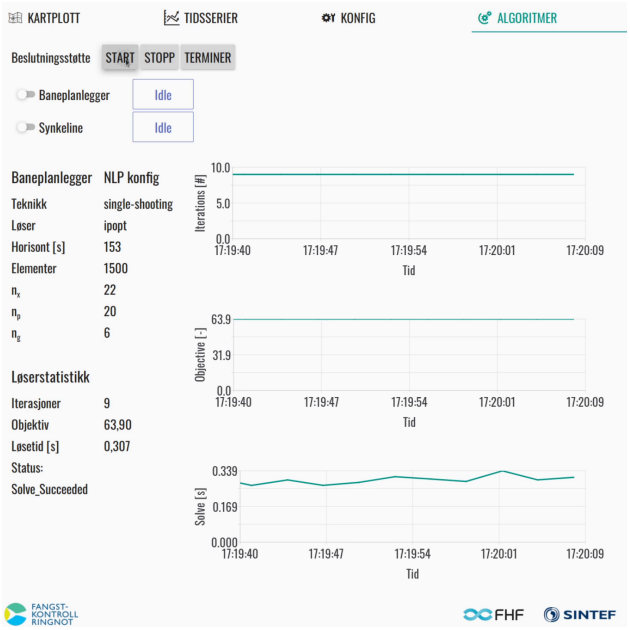
4.1 Industrialisering og kommersialisering

Målet for arbeidet i dette prosjektet var å nå TRL 6, som vil si at teknologien skal være demonstrert i relevant miljø.[4] Av praktiske og økonomiske årsaker har vi ikke kunnet gjennomføre avsluttende tester av programvaren i reelle ringnotoperasjoner. I stedet har vi testet den mot simulerte operasjoner, som vist i seksjon 3. Med dette har vi demonstrert metodens potensielle nytteverdi for praktisk bruk i ringnotfiske. Vi mener også at programvaren i seg selv holder et teknisk kvalitetsnivå som tilsvarer TRL 6. Vi vil allikevel være forsiktige med å påberope oss en slik TRL før programvaren er testet i en reell operasjon. Dette er dermed et naturlig neste steg i en industrialiseringsprosess.

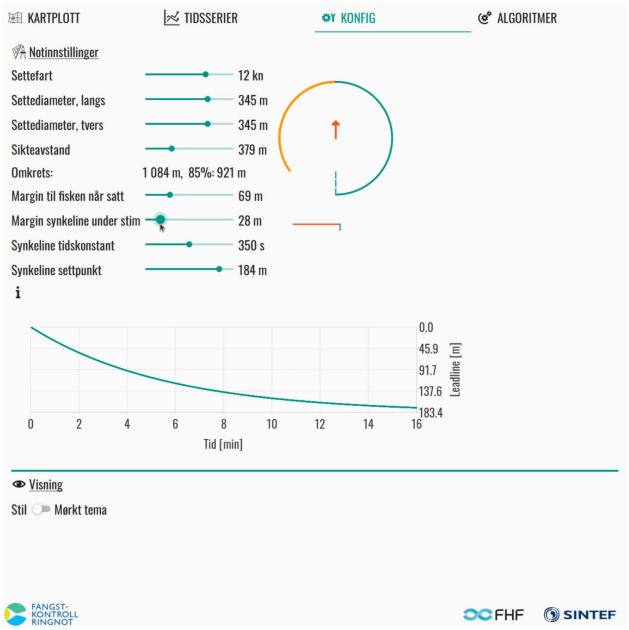
Utvidet testing, både i simulator og i full skala, vil være essensielt under hele prosessen med å utvikle dette til et kommersielt system. Algoritmen må testes mot ulike fartøy, nøter, strømningsbilder og stimoppførsler. Bare slik kan man “tune” parametrene dens slik at systemet kan håndtere de fleste situasjoner det kan tenkes å bli brukt i.

[3] OSI (Open Source Initiative) er en ideell organisasjon som jobber for at programvare skal være åpen og gratis. Se `https://opensource.org` for mer informasjon.

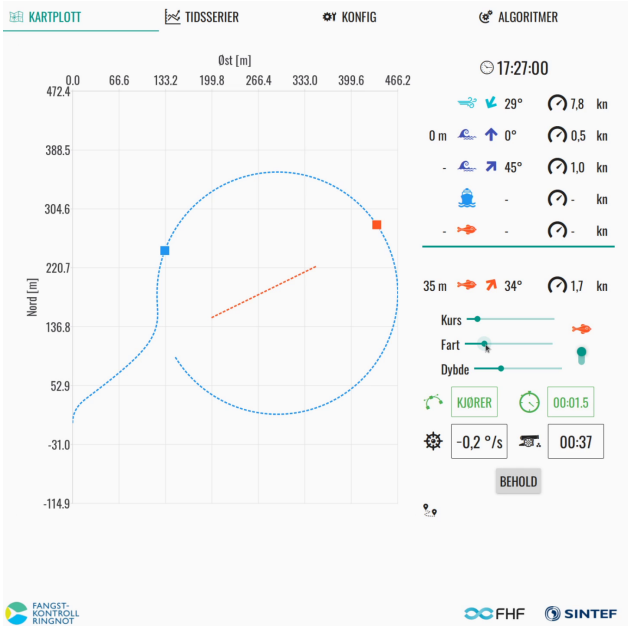
[4] TRL (Technology Readiness Level) er en indikator på modenheten til en teknologi. Vi benytter oss av Europakommisjonens TRL-skala, som er en *de facto* standard innen europeisk FoU. Se f.eks. *Horizon Europe Main Work Programme, Annex B*, for definisjoner av nivåene.



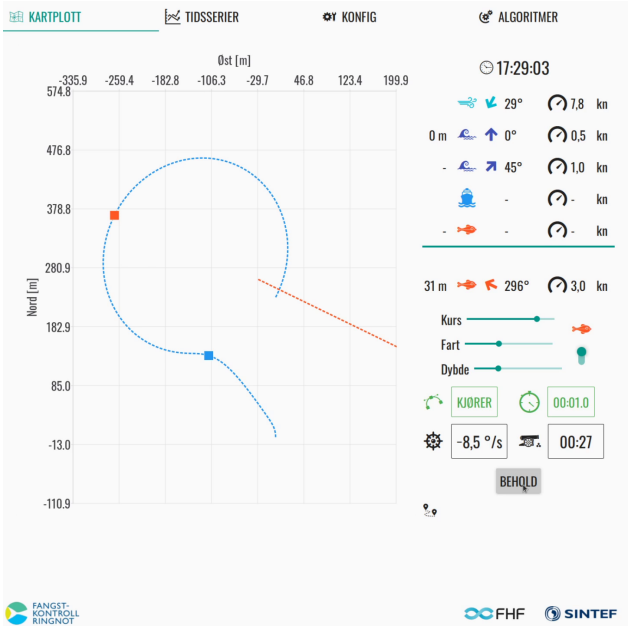
(a) Start algoritmer



(b) Sett brukerinnstillinger

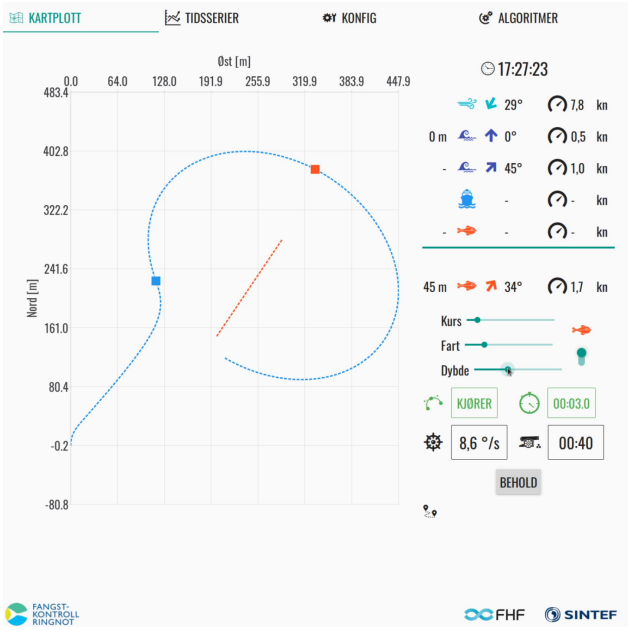


(c) Manuell overstyring av stimhastighet

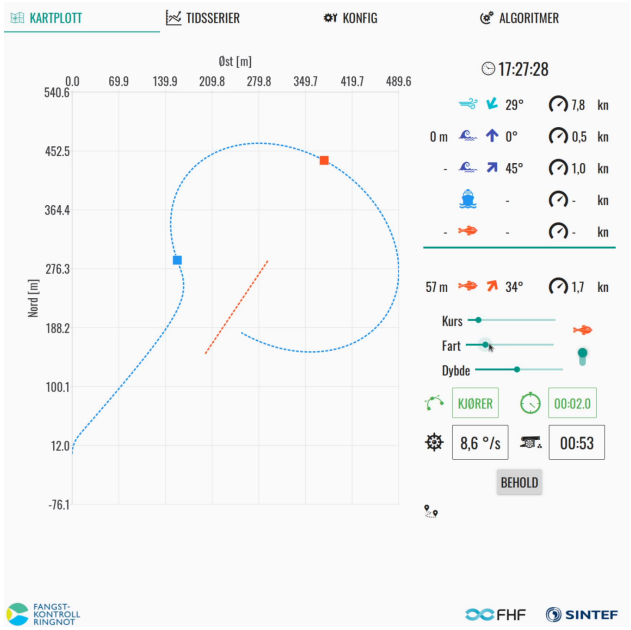


(d) Behold foreslått settebane

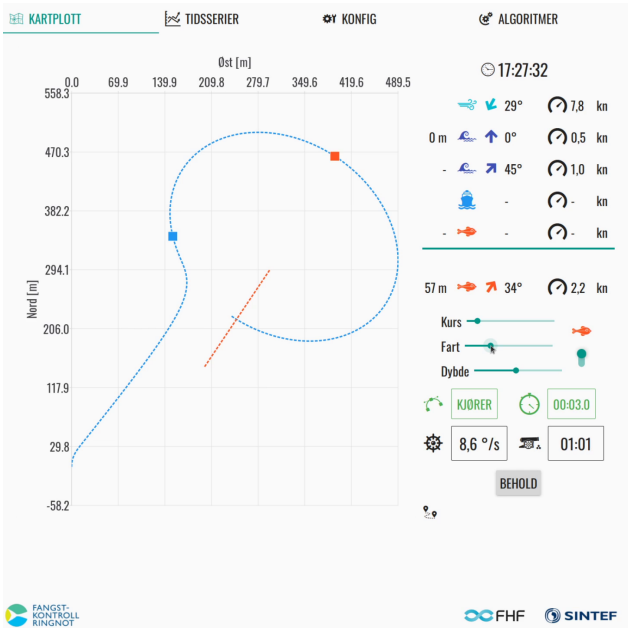
Figur 4: Typiske steg i en brukersesjon.



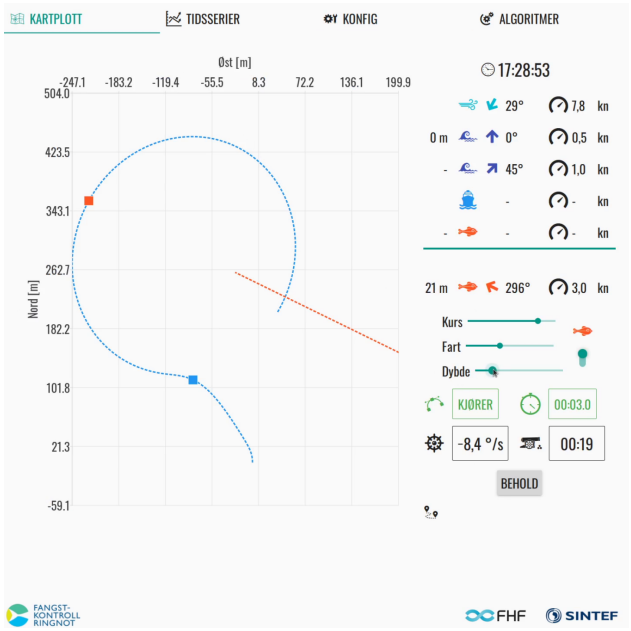
(a) Konfigurert til annen settebanedimensjon



(b) Stimen går dypere



(c) Stimen går raskere



(d) Konfigurert til større synkemargin

Figur 5: Skjermdumper av dashbord for noen scenarier.

Allikevel vil man neppe klare å lage et system som håndterer *alle* situasjoner. Derfor er *feilhåndtering* en annen viktig ting man må fokusere på under videreutviklingen. Det grunnleggende her er å detektere når algoritmen foreslår settebaner som ikke er praktisk gjennomførbare. Håndteringen kan være så enkel som å informere brukeren om feilen, eller man kan gå for en mer sofistikert løsning hvor man fortløpende prøver ut alternative parametersett eller løsningsmetoder.

Inputsignalene vil også være en viktig feilkilde. For eksempel kan nødvendige signaler falle bort på grunn av sensorfeil eller kommunikasjonssvikt. Problemene kan også være mer subtile, for eksempel i form av målestøy, målefeil eller simpelthen unøyaktige sensorer. Det finnes flere velkjente måter å håndtere slikt på, som sensorredundans, filtrering av signalene, tilstandsestimering og mer. Vi forventer at et kommersielt system vil benytte seg av flere slike metoder, men dette er ikke en del av leveransen i dette prosjektet.

Per nå er ikke baneplanleggeren knyttet til noen bestemt instrumentpakke eller -leverandør. Dette er en styrke, da det gjør det mulig å tilpasse den til ulike leverandørers løsninger. Et fremtidig kommersielt system vil derimot dra nytte av en tettere integrasjon med andre systemer ombord, det være seg et mer detaljrikt grensesnitt mot sonar eller mulighet til å vise foreslåtte settebaner i kartplotter. Vi forventer derfor at leverandører som ønsker å utvikle metoden videre vil gjøre det i kontekst av sine eksisterende systemer. Vi har utviklet programvarekoden slik at den er svært modulær, nettopp for å gjøre slik integrasjon lettere. En fremtidig systemleverandør kan plukke ut de delene de ønsker å bruke i sitt system og forkaste resten.

Bedrifter som kan føre dette arbeidet videre vil typisk være leverandører av broløsninger, fiskeletingsutstyr, beslutningsstøttesystemer eller lignende. SINTEF Ocean bidrar gjerne i en prosess med å videreutvikle dette til et produkt, og interesserte leverandører er svært velkomne til å ta kontakt.

4.2 Forskningsbehov

Vi ser behov for videre forskning innenfor tre hovedområder: modellering av notas synkerespons, formulering av optimeringsproblemet og bruk av nye datakilder.

Måten vi har modellert notas synkerespons på er svært enkel. Synkedybden øker simpelthen med en såkalt første ordens respons, se figur 3b. Gjennom prosjektet samlet vi inn data fra ringnotsnurperen M/S *Eros*, deriblant strømningsprofil fra fartøyets strømningsmåler (*acoustic doppler current profiler*) og synkedybde fra dybdemålere montert på nota. Idéen var å identifisere en sammenheng mellom strømningsbildet og synkeresponsen basert på empiriske data og lage en parametrisert modell basert på dette. Dessverre var mye av dataene av svært dårlig kvalitet, og til syvende og sist hadde vi kun gode data fra noen titalls kast – langt fra tilstrekkelig til å kunne si noe om en slik sammenheng. I tillegg hadde vi ikke data fra notvinsjene, og vi har etter hvert fått vite at disse ofte brukes aktivt under setting av nota på en måte som kan ha sterk påvirkning på synkeresponsen. Men i de få tidsseriene vi hadde, observerte vi – heldigvis, kan man kanskje si – at synkehastigheten følgte noenlunde samme trend i alle kast. Det kan dermed tenkes at en første ordens modell som den vi har valgt vil fungere godt i de fleste situasjoner, forutsatt at man har en god metode for å bestemme koeffisientene for en gitt not og situasjon. Dette vil kunne avdekkes med mer omfattende og systematisk datainnsamling fra flere fartøy, med flere datakilder, sterkere fokus på datakvalitet og lengre varighet.

Dersom det viser seg at det er behov for en mer detaljert modell finnes det ulike måter dette kan løses på. Vi er kjent med at det finnes detaljerte metoder for matematisk modellering av notdynamikk i den vitenskapelige litteraturen. Et grundig litteraturstudium vil kunne gi en pekepinn på hvilke, om noen, som kan egne seg til vårt bruk. Det er sannsynlig at en slik modell vil være for beregningstung til å kunne brukes direkte i baneplanleggeren, som må gi resultater i sanntid basert på et stort antall modellkjøringer. Men en detaljert modell kan danne grunnlag for en mer lettbent modell, enten ved at man forenkler ligningene og lager en såkalt *effektiv modell* eller ved at man benytter systemidentifikasjonsmetoder for finne statistiske sammenhenger mellom input- og outputverdier. Dersom man ikke finner matematiske modeller som kan brukes så kan det også være en mulighet å lage en empirisk modell basert på laboratorietester i strømningstank.

Når det gjelder formuleringen av optimeringsproblemet så er det rom for videre utforskning av ulike muligheter. Per nå så søker baneplanleggeren å finne en settebane som er ellipseformet sett fra referanserammen

som følger overflatestrømmen (altså rammen med netto null strøm). Man kan for eksempel vurdere om det finnes andre baneformer som vil være mer hensiktsmessige ut fra fiskeoperasjonsmessige hensyn. Det krever liten innsats å erstatte idealisert settebane med noe annet enn en ellipseform med nåværende formulering. Det er verdt å nevne at formen på banen ikke var innskrenket til en bestemt geometrisk form i den første formuleringen vi brukte. Optimeringsalgoritmen sto fritt til å foreslå enhver løsning som oppfylte realistiske forventninger til fartøyets manøvreringsevne og avstand til stimen. Denne åpne formuleringen viste seg å være *for* fri i den forstand at foreslåtte baner ikke alltid hadde en form man kunne forvente av en settebane. Vi gikk derfor over til å forordne en ellipsebane som i større grad hjelper algoritmen med å finne hensiktsmessige løsninger. En svakhet med den nåværende formuleringen er at i fasen før nota begynner å sette, så benyttes en modell som ikke forsøker å unnvike stimen. Heldigvis er optimeringsrammeverket fleksibelt nok til at det er mulig å erstatte kun denne delen av formuleringen og dermed får stimunnvikelsesatferd.

Merk at det er en tett sammenheng mellom problemformuleringen og hvilke konfigurasjonsmuligheter skipperen vil ha under bruk, så dette er noe som også må tas i betraktning når man utforsker alternativer.

Den siste forskningsmuligheten vi vil peke på er at man kan vurdere ulike nye og alternative datakilder som kan gi systemet en bedre "situasjonsforståelse". For eksempel kan det være svært nyttig å få mer informasjon om fiskestimens oppførsel, utstrekning og biomassetetthet fra sonaren. Dersom systemet kan observere stimens posisjon og hastighet over tid så kan man bruke dette til å gi et mål på dens retningsstabilitet – altså om stimen som et hele ser ut til å gå jevnt i en bestemt retning eller om den beveger seg mer tilfeldig. I sistnevnte tilfelle vil systemet kunne anbefale skipperen å *ikke* sette nota. Dersom man kan få et mål på stimutstrekning og biomassetetthet så vil baneplanleggeren kunne foreslå en bane som gjør at man kun fanger opp en begrenset mengde fisk, ved at den beregner volumet som sirkles inn med ulike settebaner. Dette ville være svært nyttig med tanke på fangstbegrensning.

Avslutningsvis tillater vi oss noen mer visjonære betraktninger om fremtiden. I et beslutningsstøttesystem er det til syvende og sist et menneske som tar avgjørelsen om å "lukke sløyfa" fra anbefaling til faktisk handling – eller å *ikke* gjøre det. Det naturlige neste steget vil være å lukke sløyfa på systemnivå, altså at algoritmens baneforslag automatisk blir omsatt til handling. I første omgang kan dette ta form av en autopilot som styrer fartøyet og gir signal til mannskapet når de skal utføre ulike handlinger, som å begynne å kaste nota. Etter hvert kan man se for seg at også dekksoperasjonene gradvis blir mer automatiserte, og man er godt på vei mot et helautomatisert – og kanskje til slutt autonomt? – ringnotfiske.

Vedlegg 1

mimir

Joakim Haugen

May 31, 2021

OVERVIEW

CHAPTER

ONE

INTRODUCTION

1.1 Overview

mimir serves as a simple reference implementation to achieve an optimization-based decision support in the FHF project: Catch control in purse seining [33]. The program typically runs to provide algorithm outputs to another application, for instance a graphical application that visualizes the results. The algorithms are specifically set up to use the CasADi [4] framework and its interfaces to various algorithm libraries, including optimization problem solvers and numerical integrators.

We provide a few algorithms that follows an interface defined by `mimir::IAlgorithm`. The interface consists of only three functions: `initialize()`, `solve()`, and `timer()`. These functions are controlled by a state machine, `mimir::StateMachine`, using Boost StateChart [24], which strives to ensure that the solve step is executed on a regular interval. The state machine is run by the `mimir` executable and loads one algorithm with settings specified by a YAML configuration file. The inputs to, and outputs from, an algorithm are exchanged using the data distribution service standard OMG DDS [32], which is a communication middleware that takes care of the data sharing with other applications. The Diagram in Fig. 1.1 illustrates the uncomplicated algorithm wrapper implementation in *mimir*.

1.2 Running an algorithm

To load `mimir` with a specific algorithm, you need to provide the executable with a YAML configuration file similar the the one below. Each algorithm has its own expected schema. Available algorithms are found in namespace_mimir__algorithm. Most application settings are contained in the configuration file, but there are a few more command line options available, see *Manpage*.

1.2.1 YAML config file

The `algorithm:command` and `algorithm:notifier` entries are used to configure the DDS topics for user commands and state machine transition notifications. They use the interface definition language types indicated in the comment, which are defined in the `RatatoskIDL` dependency. Each algorithm has its own *algorithm-specific* schema; consult the documentation for the algorithm you intend to use.

```
# Excerpt of /path/to/config.yml, with 'algorithm-specific' map not expanded:
---
dds:
  domain: 0                                # DDS domain to connect to
algorithm:
  name: PursePlanner                       # Name identifier of algorithm
  command:
```

(continues on next page)

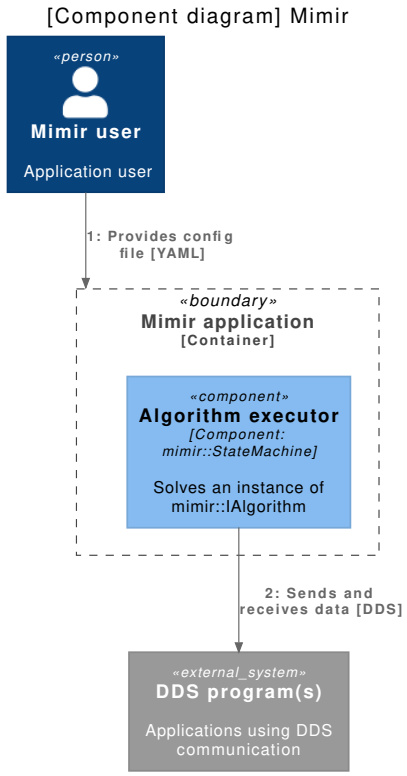


Fig. 1.1: Component diagram for Mimir.

(continued from previous page)

```
    request_topic: fkin_cmd                # IDL type: fkin::Command
    reply_topic: fkin_cmd_resp             # IDL type: fkin::CommandResponse
    recipient: PursePlanner                # Recipient identifier request/reply
  notifier:
    notify_topic: fkin_state_notification  # IDL type: fkin::ProcessStateKind
    identifier: PursePlanner               # Identifier of notification

PursePlanner:
  *algorithm-specific
...
```

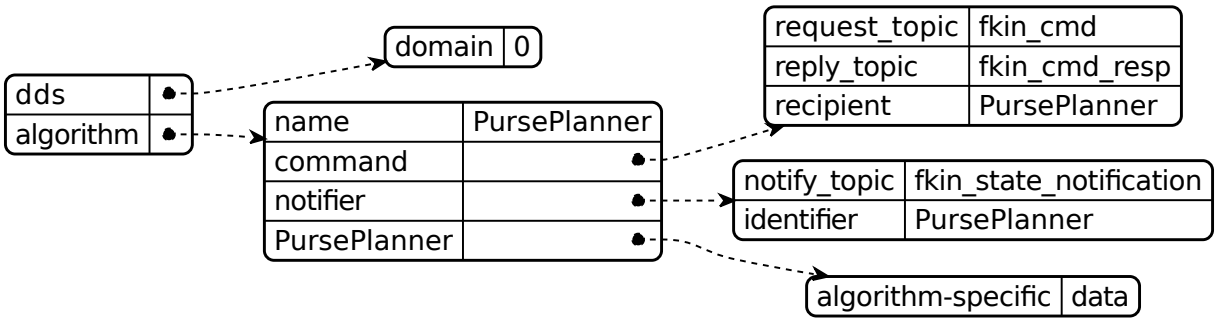


Fig. 1.2: YAML config visualization.

Note: As can be seen in the `mimir::StateMachine`, the algorithm enters a standby state by default. The user needs to issue a `START_PROCESS` to the user specified `request_topic` (DDS) in order for the algorithm to start.

1.3 Interacting with the algorithm

The user can post commands to the `mimir::StateMachine` to control in which state the algorithm should be. This is achieved through DDS signals, which is translated to state machine events.

- The user sends a DDS `fkin::Command` defined in `RatatoskIDL`.
- `mimir::control::CommandResponder` translates between commands and state machine events.
- `mimir::control::StateNotifier` broadcasts changes on DDS using a `fkin::ProcessStateKind`.

The sequence diagram in Fig. 1.3 shows a possible realization.

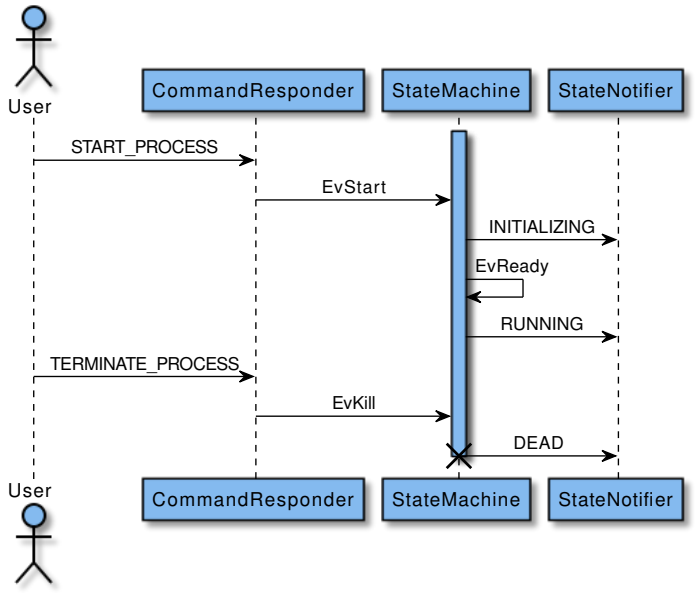


Fig. 1.3: A sequence diagram for a simple start-then-stop of Mimir.

CHAPTER

TWO

BUILD INSTRUCTIONS

You need a compiler that supports *c++17* to build `mimir`. It is known to compile with `gcc 8`, `clang 9`, and `msvc++ 14.2`. You need the CMake build system generator. The application requires RatatoskIDL, opensplice-ce, casadi, `yaml-cpp`, and `boost`.

The **recommended** approach to build the application is with the help of conan. `conan` is a python tool, and once installed, you need to set the following conan remotes.

```
python -m pip install conan
conan remote add sintef https://conan.sintef.io/public
# or
conan remote add sintef https://artifactory.smd.sintef.no/api/conan/conan-local

conan config set general.revisions_enabled=1
```

Building the documentation is *optional*. It is generated using `doxygen`, `sphinx`, `emacs`, and `plantuml`. You also need the following python modules (`docs/requirements.txt`):

```
breathe
Sphinx >= 4.0.0
exhale
sphinx-rtd-theme
sphinxcontrib-plantuml
sphinxcontrib-bibtex
sphinxcontrib-svg2pdfconverter
doc2dash
```

Instructions for each platform is found below.

2.1 Configuration options and targets

The table below specifies available options when building the application.

CMake Option	conan option	Default	Comment
See comment	`with_fPIC`	True	`CMAKE_POSITION_INDEPENDENT_CODE=ON`
`WITH_DOC`	`with_doc`	False	Use `cmake --build . --target doc`
`MIMIR_WITH_GNUPLOT`	`with_gnuplot`	False	Build with `gnuplot` visualization
`MIMIR_SKIP_UNOFFICIAL_DEPS`	N/A	False	Skip non-public dependencies in *.deb* file

The following extra build targets are available:

- `doc` will build documentation html.
- `package_it` will create *.deb* (linux), *.exe* (windows), and *.tar.gz*

If the executable is built without `conan`, the Debian packages will contain dependencies to system packages and non-standard packages: `casadi-dep`, `casadi`, `opensplice-{hde,rts}`, which can be skipped with `MIMIR_SKIP_UNOFFICIAL_DEPS=ON`. This variable has no effect if the package is built with conan.

- An additional option `WITH_API_DOC=OFF` disables API documentation, and enables LaTeX and PDF output of the user documentation. This currently requires linux OS, `inkscape`, `latexmk`, and a LaTeX distribution to be installed. The output will be available in `<build_folder>/docs/sphinx/latex/`.

2.2 Linux

2.2.1 Prerequisites (debian-based)

These instructions assume a `gcc` compiler and using `conan`.

```
apt-get install -y build-essential cmake pkg-config python3-pip readelf
python -m pip install setuptools wheel conan
```

Note: There may be other (unknown) packages needed by transitive dependencies, which are not installed by conan. You may either install them manually when build errors occur, or you can set environment variable CONAN SYSREQUIRES MODE=enabled, which lets conan automatically install them for you.

2.2.2 Documentation prerequisites *(optional)*

```
apt-get install -y doxygen emacs-nox graphviz plantuml wget pandoc
python -m pip install -r docs/requirements.txt --upgrade
emacs -Q --batch -l docs/emacs-install-pkgs.el
```

If your distribution is a bit old, you may have to update `plantuml`.

```
wget https://sourceforge.net/projects/plantuml/files/plantuml.jar
mv plantuml.jar /usr/share/plantuml/
```

2.2.3 Building and running (debian-based)

To install dependencies and build the application you can run the following commands:

```
mkdir build && cd build
conan install .. \
  --options mimir:with_doc=True \
  --build missing \
  --settings compiler.libcxx=libstdc++11
conan build ..
```

You need to activate virtual environments to run the built application.

```
. activate.sh     # Opensplice environment variables
. activate_run.sh # Dynamic libraries added to LD_LIBRARY_PATH
bin/mimir /path/to/config.yml
```

Note that if HSL solvers are to be used, they need to be enabled, for instance as follows, where you provide an absolute path to your HSL source copy:

```
conan install .. \
  -o casadi:with_common=True \
  -o casadi:hsl=True \
  -o coinhsl:hsl_archive=<path to coinhsl.tar.gz> \
  -o ipopt:with_hsl=True
```

2.2.4 Packaging into artifacts

The project builds into various artifacts on Linux.

- *.deb* package, build target `package_it`.
- *.tar.gz* archive, build target `package_it`.
- `conan package ..` will build all supported targets into the `package` subdirectory of your build directory. You must call `conan install ..` as described above first.

Warning: When building with conan, the *.deb* packages will only be built with very limited number of system package dependencies. This is because the dependencies are bundled with the package to ensure that the versions are as the built executable expects. Notable exceptions are OpenMP and Fortran runtimes. They are guessed by the compiler version used, but can be overridden by specifying `MIMIR_OMP` and `MIMIR_FORTRAN_RT`, e.g. `libomp5` and `libgfortran6`.

- By default, the application loads a bundled config file. It can be overridden by setting the environment variable: `export OSPL_URI=file:///path/to/ospl.xml`.
- **Note**, the automatically set `OSPL_URI` and `OSPL_HOME` only works if you install to the default location specified by `MIMIR_INSTALL_PREFIX` in `CMakeLists.txt`, it is `/opt/sintef` with conan and `/usr/local` otherwise.

2.3 Windows

2.3.1 Prerequisites

Prerequisites using `conan`. Most commands expect you to run with elevated privileges. We make use of chocolatey package manager for windows:

```
powershell -Command Set-ExecutionPolicy Bypass -Scope Process -Force; \
  [System.Net.ServicePointManager]::SecurityProtocol = \
  [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; \
  iex ((New-Object System.Net.WebClient).DownloadString( \
  'https://chocolatey.org/install.ps1'))
```

```
choco install -y python3 Wget
choco install -y cmake --installargs '"ADD_CMAKE_TO_PATH=System"'
choco install -y git.install --params "/GitAndUnixToolsOnPath"
python -m pip install setuptools wheel conan win-unicode-console
```

Microsoft Visual Studio build tools if they are not already installed. These commands must to be run with `cmd.exe`. **Note** that the `vs_buildtools.exe` will run in the background. These steps are not necessary if you have Visual Studio with C++ compilers installed.

```
mkdir C:\TEMP && cd C:\TEMP
wget https://aka.ms/vs/16/release/vs_buildtools.exe
vs_buildtools.exe --quiet --norestart --wait --nocache \
  --installPath C:\BuildTools \
  --add Microsoft.VisualStudio.Workload.MSBuildTools \
  --add Microsoft.VisualStudio.Workload.VCTools --includeRecommended
setx path "%path%;C:\BuildTools\Common7\Tools"
```

Tip: You may need to start new command window sessions between commands to load the new PATH variables.

2.3.2 Documentation prerequisites on Windows *(optional)*

```
choco install -y doxygen.install emacs plantuml pandoc
choco install -y graphviz.portable --force # Maybe optional: 2.44 broken, downgrades to␣
↪2.38
python -m pip install -r docs/requirements.txt --upgrade
emacs -Q --batch -l docs/emacs-install-pkgs.el
```

2.3.3 Building and running

To install dependencies and build the application you can run the following commands:

```
mkdir build
cd build
conan install .. \
  --options mimir:with_doc=True \
  --build missing
conan build ..
```

You need to use a virtual environment to run the application.

- `activate.bat` sets `OSPL_URI` and `OSPL_HOME` environment variables.
- `activate_run.bat` sets `PATH` to directories with dynamic libraries.

```
activate.bat
activate_run.bat
cd bin
mimir.exe -v 4 ../path/to/config.yml
```

2.3.4 Packaging into installer and archive

The project is set up with packaging into an executable installer (*.exe.*) and an archive (*.tar.gz*) using the build target named `package_it`. The installer is made with NSIS through CPack. NSIS and can be installed with chocolatey:

```
choco install -y nsis
```

This step assumes that you have called `conan install` as described above.

```
cd build
conan package ..
# or
cmake --build . --config Release --target package_it
```

- By default, the application loads a bundled configuration file `ospl.xml`. It can be overridden by setting the environment variable: `set OSPL_URI=file://C:\\path\\to\\ospl.xml`.
- **Note**, the automatically set `OSPL_URI` and `OSPL_HOME` only works if you install to the default location specified by `MIMIR_INSTALL_PREFIX` in `CMakeLists.txt`, which is `C:/Program Files/mimir-<version>`.

Tip: If the package is installed with docs, `WIN + "Mimir Documentation'"` should link to the bundled html documentation.

CHAPTER

THREE

CREATING A NEW ALGORITHM

It is possible to create a new algorithm to be run with *mimir*. Currently, the system is not pluggable, meaning that when a new algorithm is to be supported, the application needs to be recompiled. Generally, we do not recommend to use *mimir* to develop new algorithms. There is a plethora of choices available. The Functional Mock-up Interface (FMI) [31] is recommended. Converting a mimir algorithm to use FMI is straightforward.

To add a new algorithm you need to:

- Create new files `src/mimir/algorithm/<ALGORITHM_NAME>{.cpp,.hpp}`
- Implement the interface `mimir::IAlgorithm`, use other algorithms as examples.
- Add its `.cpp` to `MIMIR_SRC` in `mimir/src/CMakeLists.txt`
- Include header in file_src_mimir_algorithm_AlgorithmFactory.hpp.
- Instantiate it in `mimir::AlgorithmCreator()`.
- Optionally add a startup banner in file_src_mimir_mimir.hpp, `src/mimir/MimirPriv.cpp`, and a call it in the if-else block of `src/programs/mimir.cpp`.

CHAPTER

FOUR

MIMIR

4.1 NAME

`mimir` – an optimization-based purse seine path planner

4.2 SYNOPSIS

```
/opt/sintef/bin/mimir [OPTIONS] CONFIG
```

4.3 DESCRIPTION

A purse seine path planner algorithm that calculates suggested deployment paths based on user preferences and environmental conditions.

4.4 OPTIONS

The command is valid with install prefix set to ``*/opt/sintef*``. If `$PATH` has `/opt/sintef/bin`, the executable name is simply `mimir`.

`CONFIG` Path to YAML config file.

`-h, --help` Display the help message.

`-v number, --severity number` Print severity, with *number* as defined in *Severity* below, default: 3.

`-l number, --log number` Log file severity: with *number* as defined in *Severity* below, default: 3.

Severity is an integer *number* in [0-5], 5: trace, 4: debug, 3: info, 2: warning, 1: error, 0: fatal

4.5 CONFIGURATION

The YAML configuration file follows a specific schema, with two common map attributes: *dds* and *algorithm*. The *algorithm:name* defines the name of the algorithm to be run and imples that a map with the same name is present in the YAML file. The configuration specification for a selected algorithm is specific to that particular algorithm. See *EXAMPLE* below.

4.6 ENVIRONMENT

OSPL_URI If `$OSPL_URI` is not set, it defaults to ``*file:///opt/sintef/etc/config/ospl.xml*``. This is the configuration file for the OpenSplice middleware service used by the application. See **OpenSplice deployment** for details on configuring the middleware service.

OSPL_HOME If `$OSPL_HOME` is not set, it defaults to ``*/opt/sintef/*``. This is the root path used by the OpenSplice middleware.

4.7 FILES

``***/opt/sintef/etc/config/ospl.xml/***`` OpenSplice configuration file as described in **OpenSplice deployment**.

4.8 EXAMPLE

/opt/sintef/bin/mimir -v 0 -l 3 /path/to/config.yml Run algorithm specified in ``*/path/to/config.yml/*`` with fatal severity output to terminal, and info severity output to log file. The YAML config file below shows the common schema that `mimir` expects. Note that the algoritm enters a standby state and will not execute before a `START_PROCESS` command is issued to the `request_topic` specified in the config file.

4.8.1 YAML config file

The `algorithm:command` and `algorithm:notifier` entries are used to configure the DDS topics for user commands and state machine transition notifications. They use the interface definition language types indicated in the comment, which are defined in the `RatatoskIDL` dependency. Each algorithm has its own *algorithm-specific* schema; consult the documentation for the algorithm you intend to use.

```yaml
# Excerpt of /path/to/config.yml, with 'algorithm-specific' map not expanded:
---
dds:
  domain: 0                                 # DDS domain to connect to
algorithm:
  name: PursePlanner                        # Name identifier of algorithm
  command:
    request_topic: fkin_cmd                 # IDL type: fkin::Command
    reply_topic: fkin_cmd_resp              # IDL type: fkin::CommandResponse
    recipient: PursePlanner                 # Recipient identifier request/reply
  notifier:
    notify_topic: fkin_state_notification   # IDL type: fkin::ProcessStateKind
    identifier: PursePlanner                # Identifier of notification
```

(continues on next page)

(continued from previous page)

```
PursePlanner:
  *algorithm-specific
...
```

4.9 NOTES

Currently, the log outputs are only printed to stdout. This means that the ``**-l | –log**`` option has no effect.

4.10 SEE ALSO

- **yamllint**.1 can be useful to validate the YAML config file (not installed by default).
- OpenSplice deployment

4.11 COPYRIGHT

CHAPTER

FIVE

NONLINEAR PROGRAMMING FORMULATION

This chapter describes theory for mathematical optimization used in this project. It includes mathematical constructs that constitute our implementations. You can skip this chapter, but it is advisable to at least read *NLP formulation overview* and *Receding horizon*.

5.1 Brief introduction to nonlinear programming problems

Dynamical systems can often be formulated with a system of differential-algebraic equations (DAE). The DAEs can describe phenomena that one want to interact with to achieve a specific *objective*. A dynamical system usually evolve as time progresses, and therefore depends on a continuous time variable. We define an *objective* as a quantifiable goal, which is related to the dynamical response of the DAE by means of input signals. An optimal control problem (OCP) deals with finding control inputs in such a way that the *objective* is minimized. This formulation rarely has an explicit solution and needs to be solved numerically. One approach for solving OCPs is to implement a "discretize then optimize" numerical algorithm [7], such as single shooting, multiple shooting, direct collocation, or a hybrid technique [1]. These techniques quickly become computationally demanding, so a *receding horizon* approach is usually employed to strive for solutions in real time. In a receding horizon implementation, successive finite time horizon problems are solved. When the optimization objective is to control the process, a reciding horizon OCP is often called *model predictive control* (MPC). The resulting discretized problem formulation takes the form of a so-called nonlinear programming (NLP) problem.

A nonlinear programming (NLP) problem is a subclass of optimization problems, which can be stated in minimization form as in Problem 5.1.

(Nonlinear programming problem)

$$\begin{array}{ll} \text{minimize:} & \\ z(\cdot) \in \mathbb{R}^n & F(z) \end{array} \qquad \text{subject to:} \quad \begin{array}{l} g(z) = 0, \\ h(z) \leq 0, \end{array} \tag{5.1}$$

where $g(z) \in \mathbb{R}^{n_g}$ and $h(z) \in \mathbb{R}^{n_h}$ are equality and inequality constraint functions, respectively. $F(z) : \mathbb{R}^n \to \mathbb{R}$ is defined as the *objective function*, which is to be minimized by determining z. This formulation is very general and it is necessary to impose restrictions on, or assumptions about, the properties of the involved functions. As an example, the function $F(z)$ may only have one global minimum, but many local minima. An iterative numerical solver may converge to a local minimum depending on the starting condition of the solver. Therefore, one usually set necessary and sufficient conditions to define that a local minimum is an acceptable solution to the problem stated above. Resources on numerical optimization, OCP, NLP and discretization techniques can be found in e.g. [6, 7, 11, 21].

5.2 Formulating an NLP from DAE, OCP and discretization

Our chosen procedure of formulating a nonlinear programming problem involves the steps below.

1. Describe the process under scrutiny with a system of a differential-algebraic equations (DAE).
2. Formulate the optimal control problem (OCP) with the desired objective and other constructs.
3. Use a discretization technique to convert the OCP to a nonlinear programming (NLP) problem.
4. Set initial conditions, solve the NLP and extract the solution.

The procedure involves implementation of systematic techniques, but also a fair amount of iterative processes such as tuning, redesign and application of engineering skills. In the following sections we provide mathematical descriptions of the steps outlined above and attempt to closely relate them to the source code implementation in *NLP formulation overview*.

The NLP is solved in a receding horizon fashion. This means that the problem is solved with a limited, but moving time horizon, which is described in *Receding horizon*.

Note: We describe the procedure generally, but the implementation is currently not generalized, that is, it is only available within the path planner formulation source code.

5.2.1 NLP formulation overview

The above outline procedure outlined is implemented in `mimir::algorithm::PursePlannerFormulation`. The sequence of execution is indicated by Table 5.1. Note that the functions with an anonymous namespace reside in the `.cpp` of `PursePlannerFormulation`. We provide details on each of these items in the remainder of this chapter.

Table 5.1: The NLP formulation procedure.

Step	Theory	Implementation
Matematically model system	Problem 5.3	`::formulate_dynamics`
Formulate optimal control problem	Problem 5.6	`::formulate_dynamics`
Transcribe OCP to NLP	Problem 5.10	`::formulate_single_shoot`
	Problem 5.11	`::formulate_multi_shoot`
	Problem 5.19	`::formulate_collocation`
Rearrange NLP on *casadi form*	Problem 5.20	`::formulate_nlp`
• Instantiates `casadi::nlpsol`	*variable_bounds*	`::add_nlp_variable_bounds`
Prepare *Helper functions*		`::create_helper_functions`
	Eq. (5.26)	`::create_solution_timegrid`
	Eq. (5.28)	`::create_solution_unpacker`
	Eq. (5.32)	`::create_horizon_shifter`
	Eq. (5.33)	`::create_trajectory_function`
	Eq. (5.30)	`::create_decision_parameter_extractor`
	Eq. (5.31)	`::create_system_extractor`

5.3 Receding horizon

The NLP formulation procedure of Table 5.1 only defines the necessary functions to be used. To actually use them we need to put them in an event loop that properly prepares and solves the NLP, and then extracts information from the solution. These steps are done within an implementation of the function `mimir::IAlgorithm::solve()`. Below we list the actions performed in such a solve step.

1. Fetch inputs and parameters from DDS; prepare $x(t_0)$ and p;
2. Shift solution from previous step using Eq. (5.32) – for warm starting optimization problem;
3. Set initial condition for $x(t_0)$;
4. Solve Problem 5.20;
5. Verify successful solution and publish statistics on DDS;
6. Extract solution and store in data structures in preparation for next time step;
7. Acquire solution trajectories with the help of Eq. (5.26), Eq. (5.28), Eq. (5.33);
8. Publish solution trajectories on DDS;

The above steps are executed on regular intervals and as time progresses, we achieve a receding horizon NLP problem solving.

5.4 Mathematical notation

This document contains terminology that should be clarified.

- We use over-dot notation for time derivatives, that is $\dot{x} := \frac{dx}{dt}$.
- Denote the *extended real number line* as $\overline{\mathbb{R}} := \mathbb{R} \cup \{-\infty, +\infty\}$. The reason for introducing $\overline{\mathbb{R}}$ is that some numerical algorithms treat $\pm\infty$ distinctly from a "not quite infinity" number.
- The orientation space is defined by $\mathbb{S} \in [-\pi, \pi)$.
- We use subscripts $\geq$ and $>$ to indicate non-negative and positive subsets; so $\mathbb{R}_{\geq} := \{x \in \mathbb{R} : x \geq 0\}$, and $\mathbb{R}_{>} := \{x \in \mathbb{R} : x > 0\}$.
- Similarly, $\mathbb{N}_{\geq}$ and $\mathbb{N}_{>}$ are non-negative and positive integers, or natural numbers.
- Countable finite index sets of non-negative and positive natural numbers are defined as $\mathbb{I}_{\geq,n} := \{i \in \mathbb{N}_{\geq} : i < n\}$ and $\mathbb{I}_{>,n} := \{i \in \mathbb{N}_{>} : i \leq n\}$, both with cardinality n.
- I_n is the $n \times n$ identity matrix.
- $1_{n\times m}$ is an n-by-m matrix of ones.
- $0_{n\times m}$ is an n-by-m matrix of zeros.
- $A_{[i,j]}$ is the matrix element at the i-th row and j-th column.
- Define a tuple selector operator subscript $\langle m \rangle$ as a mapping that extracts the m-th element of an n-tuple, so $(x_1, x_2, \cdots, x_m, \cdots, x_n)_{\langle m \rangle} = x_m$.
- A block diagonal matrix of other matrices $X_{i\in\mathbb{I}_{>,s}} \in \mathbb{R}^{m_i \times n_i}$ is defined as $\mathrm{bdiag}_{i\in\mathbb{I}_{>,s}}(X_i) := \bigoplus_{i\in\mathbb{I}_{>,s}} X_i$, where $\oplus$ is the direct sum.
- The symbol $\otimes$ is the Kronecker product.
- The vertically stacked matrix of other matrices $X_{i\in\mathbb{I}_{>,s}} \in \mathbb{R}^{m_i \times n}$ is denoted $\mathrm{col}_{i\in\mathbb{I}_{>,s}}(X_i) := \mathrm{bdiag}_{i\in\mathbb{I}_{>,s}}(X_i) \cdot (1_{s\times 1} \otimes I_n)$.
- The horizontally stacked matrix of other matrices $X_{i\in\mathbb{I}_{>,s}} \in \mathbb{R}^{m \times n_i}$ is denoted $\mathrm{row}_{i\in\mathbb{I}_{>,s}}(X_i) := \mathrm{col}_{i\in\mathbb{I}_{>,s}}(X_i^T)^T$.
- Let e_i be the i-th canonical basis for an n-dimensional space. The column vectorization of a matrix $X \in \mathbb{R}^{m\times n}$ is denoted $\mathrm{vec}(X) := \sum_{i\in\mathbb{I}_{>,n}} e_i \otimes Xe_i$.
- Vertically stacked identical matrices $X \in \mathbb{R}^{m\times n}$ for an s number of times is denoted $\mathrm{repvert}_s(X) := 1_{s\times 1} \otimes X$.
- Horizontally stacked identical matrices $X \in \mathbb{R}^{m\times n}$ for an s number of times is denoted $\mathrm{rephorz}_s(X) := 1_{1\times s} \otimes X$.
- A band diagonal square matrix of size n with ones at superdiagonal k is denoted $\mathrm{band}(n, k)$.

5.5 Differential-algebraic equations (DAEs)

A system of differential-algebraic equations (DAEs) is a class of equations where some of the differential states cannot be written explicitly, and/or there are algebraic constraints. Here, we consider first order DAEs with explicit, implicit, and algebraic equations. Let $t \in \mathbb{R}_{>}$ be the independent time variable. Define $x_e(t) \in \mathbb{R}^{n_e}$ and $x_i(t) \in \mathbb{R}^{n_i}$ as the explicit and implicit differential state variables, and let $q(t) \in \mathbb{R}^{n_q}$ be quadrature differential states for which do not explicitly appear in the function mappings. Further, let $z(t) \in \mathbb{R}^{n_z}$ be algebraic variables, $u(t) \in \mathbb{R}^{n_u}$ input variables,

and $y(t) \in \mathbb{R}^{n_y}$ output variables. In addition, we define $p \in \mathbb{R}^{n_p}$ as time-independent fixed and tunable parameters[1] , and finally, $d \in \mathbb{R}^{n_d}$, which is a dependent variable vector, but independent of time. A DAE system can be stated as

$$\text{DAE:} \quad \begin{cases} \dot{x}_e &= f_e(x_e, x_i, z, u, p, d, t), \\ 0 &= f_i(x_e, x_i, z, u, p, d, t, \dot{x}_i), \\ 0 &= f_a(x_e, x_i, z, u, p, d, t), \\ \dot{q} &= f_q(x_e, x_i, z, u, p, d, t), \\ d &= f_d(p, d), \\ y &= f_y(x_e, x_i, z, u, p, d, t), \end{cases} \tag{5.2}$$

where $f_e(\cdot)$ is the ordinary differential equations (ODE), and $f_a(\cdot)$ is the algebraic expression function. If $x_i(t) \in \emptyset$, the DAE can be seen as a set of ODEs with constraints, but it is usually called a semi-explicit DAE. Our formulation can be further condensed by combining functions to get a standard form; perhaps even made semi-explicit DAE of index 1, see e.g. [12]. We retain the current form, because it closely resembles the notation used in casadi's DaeBuilder, which is also used in our implementation.

Numerically solving a DAE requires special attention to the properties of the problem at hand, in addition to knowledge about the capabilities and limitations of the applied solvers, see [12] for a detailed account on the matter. A DAE can be stated as a boundary value problem or an initial value problem (IVP). We concern ourselves with the IVP form, where the problem is to solve the system of equations given initial conditions for the differential states. It should be noted that the "discretize then optimize" technique is a powerful approach, since it can deal with DAEs by using direct collocation [12] or a DAE-capable solver, such as IDAS from SUNDIALS [15].

Tip:

- casadi's DaeBuilder simplifies the formulation of a DAE to be used with the casadi framework. We use it in `::formulate_dynamics()` of `src/algorithm/PursePlannerFormulation.cpp`.
- casadi's Integrator supports various solvers, including IDAS and collocation. Note that a DAE needs to be made semi-explicit and be of index-1 in order to be used with this integrator. There are functions in DaeBuilder that may help in achieving this.

5.5.1 The constrained DAE initial value problem (IVP)

We introduce the concept of a constrained DAE initial value problem (cDAE_{IVP}) as a problem where the variables of a DAE may be restricted to a subset of all extended real numbers. These restrictions can either be imposed by the phenomena they are describing, or indirectly through the optimal control problem formulation. Care must be taken when constraining the problem. Adding unreasonably strict constraints may render the IVP inconsistent and unsolvable.

Let $\mathbb{T} := [t_0, t_f]$ be the time interval for the IVP. Suppose $u(t) \in \mathcal{U} \subseteq \overline{\mathbb{R}}^{n_u}$ is known for $t \in \mathbb{T}$, with $x_e(t_0) = x_{e,0}$, $x_i(t_0) = x_{i,0}$, $q(t_0) = 0$, and $p =$ given, and consistent. The constrained DAE IVP is stated in Problem 5.3. The solution to (cDAE_{IVP}) are $\forall t \in \mathbb{T}$ the trajectories $x_e(t)$, $x_i(t)$, $z(t)$, $q(t)$, and $y(t)$. The control input $u(t)$ is usually something to be determined based on a quantifiable goal or objective.

[1] *variability* in the FMI sense. For the initial value problems we consider, parameters are fixed.

(Constrained DAE IVP)

$$
\text{cDAE}_{\text{IVP}}: \quad
\begin{aligned}
&\forall t \in \mathbb{T} \begin{cases} \dot{x}_e &= f_e(x_e, x_i, z, u, p, t), \\ 0 &= f_i(x_e, x_i, z, u, p, t, \dot{x}_i), \\ 0 &= f_a(x_e, x_i, z, u, p, t), \\ \dot{q} &= f_q(x_e, x_i, z, u, p, t), \\ d &= f_d(p, d), \\ y &= f_y(x_e, x_i, z, u, p, t), \end{cases} \\
&x_e \in \mathcal{X}_e \subseteq \overline{\mathbb{R}}^{n_e},\ x_i \in \mathcal{X}_i \subseteq \overline{\mathbb{R}}^{n_i},\ z \in \mathcal{Z} \subseteq \overline{\mathbb{R}}^{n_z}, \\
&u \in \mathcal{U} \subseteq \overline{\mathbb{R}}^{n_u}, \\
&y \in \mathcal{Y} \subseteq \overline{\mathbb{R}}^{n_y}. \\
&x_e(t_0) = x_{e,0},\ x_i(t_0) = x_{i,0},\ q(t_0) = 0, \\
&p = \text{given}, \\
&u(t) = \text{defined}.
\end{aligned}
\tag{5.3}
$$

5.6 Optimal control problem (OCP)

The task of the optimal control problem (OCP) is, by means of determining the control input $u(t)$, to find a solution to the constrained DAE IVP in Problem 5.3, while (locally) minimizing some scalar objective function. Thus, the two main components in an OCP are:

1) the objective function,
2) the constrained initial value problem.

The objective function can take several forms, but herein we use the general Bolza-type optimal control problem [7]. Let us use the same entities as defined earlier, but also let $x(t) := \text{col}(x_i(t), x_e(t))$. The objective function maps to a scalar value and is declared as

$$
J(x, z, u, p, d, t) = \Phi_L(x, z, u, p, d, t) + \Phi_M(p, d, t_f) \in \mathbb{R}, \tag{5.4}
$$

where the terms are

$$
\begin{aligned}
\Phi_L(x, z, u, p, d, t) &= \int_{t_0}^{t_f} \phi_L(x, z, u, p, d, t) dt, && \text{Lagrange integral term} \\
\Phi_M(p, d, t_f) &= \phi_M(x(t_f), z(t_f), p, d, t_f), && \text{Mayer terminal cost term}
\end{aligned}
\tag{5.5}
$$

and should be sufficiently smooth. The structure and properties of the objective function have large impact on the optimal control problem, refer to [7, 21] for details.

Now that the two main components are declared, we present the OCP in Problem 5.6.

(Optimal control problem)

$$
\text{OCP:} \quad \underset{u(\cdot) \in \mathcal{U}}{\text{minimize:}} \ J(x, z, u, p, d, t) \quad \text{subject to:} \quad \text{cDAE}_{\text{IVP}}. \tag{5.6}
$$

In a typical OCP formulation, *path constraints* are stated as equality and inequality constraints $g_I(\cdot) \leq 0$ and $g_E(\cdot) = 0$. These constraints are mostly covered in Problem 5.3 by the algebraic term $f_a(\cdot) = 0$, and $f_y(x_e, x_i, z, u, p, d, t) \in \mathcal{Y}$, which may be written as $y_{\min} \leq f_y(\cdot) \leq y_{\max}$, noting that elements of $y_{\{\min,\max\}}$ can be unbounded. The OCP formulation has the benefit that it often is straightforward to include various types of dynamic systems and restrictions. Finding analytic solutions to these problems are often very difficult, but there exist several approaches to solve the problem numerically.

5.7 Discretization of the continuous time OCP

The OCP in Problem 5.3 is a continuous time problem that needs to be made finite dimensional by *transcribing* it to an NLP. We briefly describe three approaches for achieving the transcription. These three techniques are available as options when solving the *purse planning problem*, but the collocation approach appears to be the preferred choice. For more details on the inner workings of each technique, please refer to for instance [7].

One common trait for all discretization approaches below is the notion of time *elements*. We define an *element* as a time interval in such a way that the time horizon, $\mathbb{T} = [t_0, t_f]$, is divided into N elements of equal length. The time instant t_k is the time point at beginning of the k-th element. The discrete time points are $\forall k \in \{i \in \mathbb{N}_{\geq} : i < N\} = \mathbb{I}_{\geq,N}$ given as $t_k = t_0 + hk$, where $h := \frac{t_f - t_0}{N}$. Notice that $t_f = t_0 + Nh$. We define the sequence of discrete time points as $\mathbb{T}_d := \{t \in \mathbb{T} : \forall k \in \mathbb{I}_{\geq,N},\, t = t_0 + hk\}$.

Our implementation assumes that $u(t)$ is piecewise constant, that is, for all $t \in \mathbb{T}_d$, we have $u(t_k) = u_k$, where u_k is constant. This means that within a single time element, u can be considered a fixed variable, effectively a parameter.

Tip: If an input signal in the DAE needs to be smoother than piecewise constant, a straightforward way of achieving this is to parameterize u and introduce its parameters as the new input variables. For instance, a piecewise linear signal can be described for element k as

$$\begin{aligned} u_k(t) &= a + \tfrac{b}{h}(t - t_k), \\ u_k(t_{k+1}) &= u_{k+1}(t_{k+1}), \\ a &\in [u_{\min}, u_{\max}],\, b \in [\dot{u}_{\min}, \dot{u}_{\max}]\,, \end{aligned} \tag{5.7}$$

where $\tilde{u}_k = \mathrm{col}(a, b)$ is the new input vector, and the boundary condition at time point t_{k+1} acts as an equality constraint that ensures zero order continuity between elements.

(DAE Integral)

An integral map $F_I(x, z, p, t)$ is the solution to the unconstrained, semi-explicit DAE problem at the end of a time interval.

$$\begin{aligned} &F_I : \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_p} \times \mathbb{R} \to \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_q} \\ &(x_0, z_0, p, t_0) \mapsto (x_f, z_f, q_f) := \left(\textstyle\int_{t_0}^{t_f} \dot{x}\, dt, z(t_f), \int_{t_0}^{t_f} \dot{q}\, dt\right) \\ &\quad \text{subject to:} \\ &\quad \forall t \in [t_0, t_f] : \quad \begin{cases} \dot{x} &= f_e(x, z, p, t), \\ 0 &= f_a(x, z, p, t), \\ \dot{q} &= f_q(x, z, p, t), \end{cases} \\ &\quad x(t_0) = x_0,\, q(t_0) = 0,\, z(t_0) \overset{?}{=} z_0,\, t_f = t_0 + h,\, h = \text{given}, \end{aligned} \tag{5.8}$$

where z_0 is the initial guess of the algebraic variable. Note that the codomain is a 3-tuple, so we may use the tuple selector notation to extract an element of the tuple, for instance $F_I(x_0, z_0, p, t_0)_{\langle 1 \rangle} = (x_f, z_f, q_f)_{\langle 1 \rangle} = x_f$.

Note: In our shooting-based implementations, we assume that the constrained DAE IVP may be reduced to a semi-explicit DAE of index-1, that is, x_i may be reduced to be part of x_e, and that $f_a(\cdot)$ is solvable with respect to z.

5.7.1 Shooting approaches

In the **single-shooting approach**, the OCP is formulated by applying the integral map successively in a composition-like manner. Let the $k \in \mathbb{I}_{\geq,N}$ be the indices for the time elements of length $h = t_{k+1} - t_k$, where $u_k = u(t_k)\ \forall t_k \in \mathbb{T}_d$ and $p_k := \mathrm{col}(u_k, d, p)$. Introduce the *Lagrange term* as a quadrature state $\dot{q}_L := \phi_L(x, z, u, p, d, t)$, which becomes part the quadrature state vector. The sum of this quadrature state for all elements is equal to the *Lagrange integral term*,

$$\Phi_L(x, z, u, p, d, t) = \sum_{k \in \mathbb{I}_{>,N}} q_L(t_k). \tag{5.9}$$

State variables are evaluated by repeatedly applying integrals as specified in Definition 5.8, that is, $F_I(x_0, z_0, p_0, t_0) = (x_1, z_1, q_1) =: F_{I,1}$; $F_I(F_{I,1\langle 1\rangle}, F_{I,1\langle 2\rangle}, p_1, t_1) = F_{I,2}$, and so on.

(Single shooting)

$$\begin{aligned}
&\underset{u_{k\in\mathbb{I}_{\geq,N}}}{\text{minimize:}} && \overbrace{\sum_{k\in\mathbb{I}_{>,N}} q_L(t_k)}^{\Phi_L(\cdot)} + \Phi_M(p, d, t_f) \\
\text{NLP}_{\text{SS}}: \quad &\text{subject to:} \\
&\forall k \in \mathbb{I}_{\geq,N}: && \begin{cases} F_I(F_{I,k\langle 1\rangle}, F_{I,k\langle 2\rangle}, p_k, t_k) = F_{I,k+1} \in \mathcal{X} \times \mathcal{Z} \times \overline{\mathbb{R}}^{n_q}, \\ f_y(F_{I,k\langle 1\rangle}, F_{I,k\langle 2\rangle}, u_k, p, d, t_k) \in \mathcal{Y}, \\ u_k \in \mathcal{U}, \end{cases} \\
&p = \text{given},\ F_{I,0} = (x(t_0), z(t_0), q(t_0)) = (x_0, z_0, 0),
\end{aligned} \tag{5.10}$$

Problem 5.10 is equivalent to Problem 5.1 and can be achieved by rearranging terms and expressions. The decision variables of Problem 5.10 are the discretized inputs u_k. Let $u_D := \mathrm{row}_{k\in\mathbb{I}_{\geq,N}}(u_k) \in \mathbb{R}^{n_u \times N}$ be the horizontal concatenation of all the discretized u_k. The argument that minimizes the single shooting problem is indicated with $\star$ and is $\mathrm{vec}(u_D^\star) \in \arg\min \text{NLP}_{\text{SS}}$. The piecewise constant input is $u^\star(t) : \mathbb{T} \to \{u \in \mathbb{R}^{n_u} : u(t) = u_D^\star\, \mathrm{col}_{k\in\mathbb{I}_{\geq,N}}((t_k \leq t < t_{k+1}))\}$, where $(t_k \leq t < t_{k+1}) \in \{0, 1\}$, which ensures to switch on and off the appropriate $u_k^\star$ for its valid time interval. State and output trajectories can be obtained by solving the corresponding Problem 5.3 with $u(t) = u^\star(t)$.

The **multi-shooting approach** also uses integral maps, but not as compositions. Let $u_k = u(t_k)$, $x_k = x(t_k)$, $z_k = z(t_k)\ \forall t_k \in \mathbb{T}_d$ be decisions variables. The state variables at the boundary of time elements are explicitly "seamed" together by defining equality constraints, where the integral map must be equal to decision variables, that is, $(F_{I,k+1\langle 1\rangle}, F_{I,k+1\langle 2\rangle}) = (x_{k+1}, z_{k+1})$ must hold for all k. The multi shooting formulation is stated in Problem 5.11.

(Multi shooting)

$$\begin{aligned}
&\underset{\forall k \in \mathbb{I}_{\geq,N},\ u_k, x_k, z_k}{\text{minimize:}} && \overbrace{\sum_{k\in\mathbb{I}_{>,N}} q_L(t_k)}^{\Phi_L(\cdot)} + \Phi_M(p, d, t_f) \\
\text{NLP}_{\text{MS}}: \quad &\text{subject to:} \\
&\forall k \in \mathbb{I}_{\geq,N}: && \begin{cases} F_I(x_k, z_k, p_k, t_k) = F_{I,k+1} \in \mathcal{X} \times \mathcal{Z} \times \mathbb{R}^{n_q}, \\ \left(F_{I,k+1\langle 1\rangle}, F_{I,k+1\langle 2\rangle}\right) = (x_{k+1}, z_{k+1}), \\ f_y(x_k, z_k, u_k, p, d, t_k) \in \mathcal{Y}, \\ u_k \in \mathcal{U}, \end{cases} \\
&p = \text{given},\ x(t_0) = x_0,\ z(t_0) \stackrel{?}{=} z_0,\ q(t_0) = 0,
\end{aligned} \tag{5.11}$$

Since the decision variables of Problem 5.11 are the discretized u_k, x_k, and z_k, we define horizontally stacked discretizations as $u_D = \mathrm{row}_{k\in\mathbb{I}_{\geq,N}}(u_k)$, $x_D = \mathrm{row}_{k\in\mathbb{I}_{\geq,N}}(x_k)$, $z_D = \mathrm{row}_{k\in\mathbb{I}_{\geq,N}}(z_k)$. The argument that minimizes the multi shooting problem is indicated with $\star$ superscript and is $\mathrm{col}(\mathrm{vec}(u_D^\star), \mathrm{vec}(x_D^\star), \mathrm{vec}(z_D^\star)) \in \arg\min \text{NLP}_{\text{MS}}$. The state and output trajectories can be obtained by solving the Problem 5.3 in two different manners:

1. In a similar fashion as can be done with single shooting by using $u^\star(t)\ \forall t \in \mathbb{T}_d$;
2. For each $k \in \mathbb{I}_{\geq,N}$ use $u_k^\star, x_k^\star, z_k^\star$ to solve Problem 5.3 for a total of N number of times.

If the element time range is sufficiently short, one may directly use $u_k^\star, x_k^\star, z_k^\star$ without solving Problem 5.3 at all.

5.7.2 Collocation

The collocation methods are a subclass of implicit Runge-Kutta methods, see [11] and [12]. "For ordinary differential equations it consists in searching for a polynomial of degree d whose derivative coincides ('co-locates') at d given points with the vector field of the differential equation" [11]. In the case of OCP, this amounts to applying the collocation method for each time element. In addition, the state variables are "seamed" together at the boundary of time elements in a similar fashion as was done in multi shooting. In Theorem 7.7 (p. 212) [11] it is shown that the collocation method is equivalent to a d-stage implicit Runge-Kutta method. We will relate the collocation equations to the Butcher tableau Eq. (5.13) and show how the discretization is implemented.

Suppose $\dot{x} = f(x,t) \in \mathbb{R}^n$ is to be integrated from t_n to t_{n+1}, with $h = t_{n+1} - t_n$. The implicit Runge-Kutta setup is

$$\begin{aligned} x_{n+1} &= x_n + h\textstyle\sum_{i=1}^d b_i k_i, \\ k_i &= f\left(x_n + h\textstyle\sum_{j=1}^d a_{i,j}k_j, t_n + c_i h\right)\ \forall i \in \mathbb{I}_{>,d}, \end{aligned} \tag{5.12}$$

where a_{ij}, b_i, c_i are given coefficients from a chosen discretization scheme, e.g. Gauss-Legendre, see Sec. IV.5 [12] and the Butcher tableau in Eq. (5.13).

$$\begin{array}{c|cccc} c_1 & a_{1,1} & a_{1,2} & \cdots & a_{1,d} \\ c_2 & a_{2,1} & a_{2,2} & \cdots & a_{2,d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_d & a_{d,1} & a_{d,2} & \cdots & a_{d,d} \\ \hline & b_1 & b_2 & \cdots & b_d \end{array} = \begin{array}{c|c} \tau_{\text{colloc}} & A \\ \hline & \boldsymbol{b}^T \end{array} \tag{5.13}$$

where τ_{colloc} and $\boldsymbol{b}$ are column vectors.

The equations in Eq. (5.12) can be expressed in matrix form. Let $K = \text{row}_{i\in\mathbb{I}_{>,d}}(k_i)$, so K is an $n \times d$ matrix. The summations can be written

$$\begin{aligned} \textstyle\sum_{i=1}^d b_i k_i &= K\boldsymbol{b}, \\ \textstyle\sum_{j=1}^d A_{[i,j]} k_i &= KA^T, \end{aligned} \tag{5.14}$$

where we use the coefficients A and $\boldsymbol{b}$ from the Butcher tableau. The 'co-locating' derivatives k_i at time instants $t_{n,i} = t_n + c_i h$, have state variables $x(t_n + c_i h) = x_{n,i}$. Define the state collocation matrix $X_c := \text{row}_{i\in\mathbb{I}_{>,d}}(x_{n,i}) \in \mathbb{R}^{n\times d}$; and by including the initial state: $X_e := \text{row}(x_n, X_c)$. If we collect the expressions $x_{n,i} = x_n + h\sum_{j=1}^d a_{i,j}k_j$ in matrix form, we get

$$X_e = \text{rephorz}_d(x_n) + hKA^T. \tag{5.15}$$

By rearranging terms, we see that

$$\begin{aligned} &X_e\,\text{col}(-1_{1\times d}, I_d) = hKA^T, \\ &C := \text{col}(-1_{1\times d}, I_d)A^{-T}, \\ &hK = X_e C, \end{aligned} \tag{5.16}$$

where $K(\cdot)$ are state derivative expressions at collocation time points, arranged column-wise. By using this result, the state $x_{n+1} = x_n + hK\boldsymbol{b}$ can be written as

$$\begin{aligned} x_{n+1} &= x_n + X_e C b, \\ x_{n+1} &= X_e\left[Cb + \text{col}(1, 0_{d\times 1})\right], \\ x_{n+1} &= X_e \boldsymbol{d}. \end{aligned} \tag{5.17}$$

In the special case where the variable to be integrated does not explicitly appear in $f(\cdot)$, it is a quadrature. The quadrature differential $\dot{q} = f_q(x)$ has integral from t_n to t_{n+1} with $q(t_n) = 0$ expressed as

$$q_f = \text{row}_{i\in\mathbb{I}_{>,d}}(\dot{q}(t_{n,i}))\boldsymbol{b}h. \tag{5.18}$$

The results of Eq. (5.16), Eq. (5.17), and Eq. (5.18) are used to establish the necessary expressions to formulate the collocation-based transcriptions of the OCP to NLP. For each element k in $\mathbb{I}_{\geq,N}$ we define state elements $X_{e,k} \in \mathbb{R}^{n_x \times d+1}$. Further let $K_k(X_{e,k}, u_k, p, \tau_{\text{colloc}})$ be the matrix with ODE expressions as columns for all collocation points of element k. We use the same quadrature state q_L as defined in Eq. (5.9). Let $x_{k,0}$ be the state at time point t_k. The decision variables for the collocation problem are u_k and $X_{e,k}$. The collocated OCP is stated in Problem 5.19.

(Collocation)

$$
\begin{aligned}
&\underset{\forall k \in \mathbb{I}_{\geq,N},\ u_k, X_{e,k}}{\text{minimize:}} \quad \overbrace{\sum_{k \in \mathbb{I}_{>,N}} q_L(t_k)}^{\Phi_L(\cdot)} + \Phi_M(p, d, t_f) \\
&\text{subject to:} \\
&\text{NLP}_{\text{colloc}}: \quad \forall k \in \mathbb{I}_{\geq,N}: \begin{cases} hK_k(X_{e,k}, u_k, p, t) = X_{e,k} C, \\ q_L(t_{k+1}) = \text{row}_{i \in \mathbb{I}_{>,d}}(\dot{q}_L(t_{k,i}))\boldsymbol{b}h, \\ X_{e,k}\boldsymbol{d} = x_{k+1,0}, \\ f_y(x_{k,i}, u_k, p, d, t_{k,i}) \in \mathcal{Y} \quad \forall i \in \mathbb{I}_{\geq,d+1}, \\ X_{e,k} \in \mathcal{X}^{d+1}, \\ u_k \in \mathcal{U}, \end{cases} \\
&\qquad p = \text{given},\ x(t_0) = x_{1,0},\ q(t_0) = 0,
\end{aligned}
\tag{5.19}
$$

Tip: casadi provides helper functions to acquire collocation coefficients. The coefficients $\boldsymbol{b}$, C, $\boldsymbol{d}$ can be obtained from `casadi::collocation_coeff`, while τ_{colloc} are provided by `casadi::collocation_points`.

Note: The formulation above requires that A is invertible. This is true for at least *Radau* and *Gauss-Legendre* collocation coefficients, which are the one supported by `casadi::collocation_coeff`.

Note: We only show the theory for explicit ODE, since the algebraic state variables are not available in our collocation implementation.

Further reading on collocation can be found in [11, 12], see especially Eq. 3.1, Def. 4.7, Th. 7.7.

5.8 Nonlinear programming problem (NLP)

Now that we have described discretization approaches for the OCP, we briefly revisit Problem 5.1. There are several ways of instantiating an NLP solver in casadi, see Casadi nlpsol. In any case, `casadi::nlpsol` are formulated with five main constructs:

1. Decision variables χ;
2. Parameters ρ;
3. Objective function $F(\chi, \rho)$;
4. Constraint function $g(\chi, \rho)$;
5. Bounds $\chi_{\{\min,\max\}}$ and $g_{\{\min,\max\}}$;

The NLP formulation on *casadi form* is stated in Problem 5.20.

(Nonlinear programming problem (casadi))

$$\text{NLP:} \quad \begin{aligned} &\text{minimize:} \\ &\chi(\cdot) \in \mathbb{R}^n \quad F(\chi, \rho) \\ &\text{subject to:} \\ &\quad \chi_{\min} \leq \chi \leq \chi_{\max}, \\ &\quad g_{\min} \leq g(\chi, \rho) \leq g_{\max}, \\ &\quad \rho = \text{given } \in \mathbb{R}^{n_p} \end{aligned} \tag{5.20}$$

We assume that the χ is the argument of the NLP problem, even though strictly speaking ρ also is an argument, that is, we have $\chi^\star := \arg\min \text{NLP}(\chi_0; \rho)$. χ generally is a combination of $u_k, x_k, z_k \, \forall k \in \mathbb{I}_{\geq,N}$, so we use helper functions to unpack this vector into a tuple, see further below.

5.8.1 Formulation strategies and extensions

In some cases, the common optimal control problem formulation has limitations on what it can mathematically describe. We extend the NLP formulation to accommodate some additional use cases with the following strategies.

Decision parameters as described in Definition 5.21 are useful for cases where parameter determination is a degree of freedom in the optimization problem. A good example of this is a variable time horizon problem, Definition 5.22. We introduce the notion of *subsystems* in Definition 5.23, so that we can combine different strategies for various parts of a whole problem formulation. In particular, this enables us for instance to use single shooting with a short time horizon for one subsystem and variable time horizon with collocation for another subsystem. One limitation of the subsystem formulation is that we cannot easily support path constraints between the subsystems. Path constraints are valid for all time points in a time interval. Interconnection between subsystems are still possible with the use of *point constraints*, Definition 5.24. The current implementation allows definition of constraints at the time boundaries of the subsystems.

(Decision parameters)

Decision parameters are tunable time-constant parameters

$$v \in \mathcal{V} \subseteq \overline{\mathbb{R}}^{n_v}, \tag{5.21}$$

which are to be determined by the optimization problem. They are added to the decision variables of the NLP.

(Variable time horizon)

In a variable time horizon formulation the final time t_f is to be decided. Suppose $t_f \in [t_{f,\min}, t_{f,\max}]$, with N time elements. If we assume regular elements lengths, so that $h = t_{i+1} - t_i$ is equal for all elements, we can introduce

$$h \in [h_{\min}, h_{\max}], \tag{5.22}$$

with $h_{\{\min,\max\}} = (t_{\{\min,\max\}} - t_0)/N$ as a decision parameter. This new decision parameter can be used by all discretization techniques described herein to achieve a non-fixed time horizon.

(Subsystems)

A subsystem is a dynamic system of the form given in Problem 5.3. Let n_{ss} be the number of subsystems. For each subsystem i we have a distinct time horizon $\mathbb{T}_i$ and discretization technique, so that a subsystem's contribution to the

combined NLP is

$$\text{NLP}_i: \quad \begin{array}{l} \text{minimize:} \\ \chi_i(\cdot) \in \mathbb{R}^{n_i} \qquad F(\chi_i, \rho) \\ \text{subject to:} \\ \quad \chi_{i,\min} \leq \chi_i \leq \chi_{i,\max}, \\ \quad g_{i,\min} \leq g(\chi_i, \rho) \leq g_{i,\max}, \\ \quad \rho = \text{given} \in \mathbb{R}^{n_p} \end{array} \tag{5.23}$$

(Point constraints)

Let $x_0 = \text{col}(x_i(t_0))$ and $x_f = \text{col}(x_i(t_f))$ be the combined initial and terminal state of all n_{ss} subsystems. A point constraint is given as

$$g_\circ(x_0, x_f, v, p) \in [g_{\circ,\min}, g_{\circ,\max}]. \tag{5.24}$$

5.9 Helper functions

The numerical solution to the nonlinear programming problem is on a form that needs processing to be used in a receding horizon fashion. Below we will elaborate on several such helper functions.

Note: The helper functions are currently only implements for cases where there are no algebraic state, that is $z(t) \in \emptyset$. Extending them to also include $z(t)$ is straightforward.

5.9.1 Adding variable bounds `::add_nlp_variable_bounds`

Let $\chi \in \mathcal{Z} \subseteq \overline{\mathbb{R}}^n$ be the decision variable vector of the NLP. The standard form of the NLP solver is to specify $\chi_{\min} \leq \chi \leq \chi_{\max}$. Suppose $\chi = \text{col}(u_1, u_2, \cdots, u_N)$. Then lower and upper bound can be written as

$$\begin{aligned} \chi_{\min} &= 1_{N\times 1} \otimes u_{\min} \\ \chi_{\max} &= 1_{N\times 1} \otimes u_{\max}, \end{aligned} \tag{5.25}$$

where $\mathcal{U} = [u_{\min}, u_{\max}]$. In cases where χ also consists of elements of the state x_k, the `::add_nlp_variable_bounds`, takes care of populating $\chi_{\{\min,\max\}}$ according to the variable ordering and repetitions dictated by the chosen discretization technique.

5.9.2 Time grid function `::create_solution_timegrid`

The time grids of the discretized NLP problem depends on the chosen transcription approach, and for collocation, also the polynomial degree. We declare a function that constructs these time grids for a given method/degree as $T_G : \mathbb{R}_{\geq} \to \mathbb{R}_{\geq}^{1+N(d+1)} \times \mathbb{R}_{\geq}^{N}$, where N is the number of elements and $d \in \{k \in \mathbb{Z} : k \geq -1\}$. Let t_0 be an initial time and $h = t_{k+1} - t_k$ the step size. The time grid function is defined as

$$T_G(t_0) := (T_x(t_0), T_u(t_0)), \tag{5.26}$$

with T_x and T_u defined as

$$\begin{aligned} T_u(t_0) &= \text{col}_{k\in\mathbb{I}_{\geq,N}}(t_0 + hk), \\ T_x(t_0) &= \begin{cases} t_0, & d = -1, \quad \text{Single shooting;} \\ T_u, & d = 0, \quad \text{Multi shooting;} \\ \text{col}(t_0, T_u(t_0) \otimes 1_{d+1\times 1} + \text{repvert}_N(\text{col}(\tau_{\text{colloc,d}}, 1))), & d \geq 1, \quad \text{Collocation;} \end{cases} \end{aligned} \tag{5.27}$$

where $\tau_{\text{colloc},d} \in \mathbb{R}^d_>$ is a column vector of Legendre collocation points of degree d.

5.9.3 NLP tuple unpacker `::create_solution_unpacker`

The argument of Problem 5.1 is $z \in \mathbb{R}^n$, but when a transcription approach has been used, it is necessary to unpack this variable into tuples of discretized control inputs and state variables.

$F_{\text{unpack}} : \mathbb{R}^n \to \mathbb{R}^{n_u} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_x \times N(d+1)}$, where N is the number of elements and $d \in \{k \in \mathbb{Z} : k \geq -1\}$ is the pseudo-degree.

$$F_{\text{unpack}}(z) = (u_D, x_0, X_e), \tag{5.28}$$

where u_D is the inputs u_k arranged in a matrix and the state elements X_e are defined as

$$X_e = \begin{cases} \{\}, & d = -1, \quad \text{Single shooting;} \\ x_D, & d = 0, \quad \text{Multi shooting;} \\ \text{row}(\text{row}_{k \in \mathbb{I}_{\geq,N}}(X_{c,k}), x_{k+1,0}), & d \geq 1, \quad \text{Collocation,} \end{cases} \tag{5.29}$$

where $X_{c,k}$ are the collocated state variables in element k.

5.9.4 NLP decision parameters `::create_decision_parameter_extractor`

In some cases Problem 5.1 consists of decision parameters. Let $v \in \mathbb{R}^{n_v}$ be the decision parameter vector, which is located first in the combined NLP decision variable vector $z = \text{col}(v, s) \in \mathbb{R}^n$, where $s \in \mathbb{R}^{n_s}$ is the remaining decision variables. Extraction of these decision parameters are achieved with

$$F_{\text{v}}(z) = \text{row}(I_{n_v}, 0_{n_v \times n_s}) z = v. \tag{5.30}$$

5.9.5 NLP subsystem extractor `::create_system_extractor`

When the Problem 5.1 consists of subsystems using different discretization schemes, it is practical to first extract the relevant portion of the the whole problem. Let m be the number of subsystems. Suppose $z \in \mathbb{R}^n$ is the whole NLP problem, so that $z = \text{col}(v, s_1, \cdots, s_m)$, where $v \in \mathbb{R}^{n_v}$ is the decision parameters, and s_i is the vectorized discretization for subsystem i.

$F_{\text{extract},i} : \mathbb{R}^n \times \to \mathbb{R}^{n_i}$ is the extractor mapping for subsystem i, so that

$$F_{\text{extract},i}(z) = s_i. \tag{5.31}$$

This function is typically used as a composition with e.g. the solution unpacker and horizon shifter to manipulate a particular subsystem's discretization only. For instance $F_{\text{unpack}}(F_{\text{extract},1}(z)) = (u_{1,D}, X_{1,0}, X_{1,e})$.

5.9.6 NLP shifter `::create_horizon_shifter`

The intent of the horizon shifter is to move variables one time element forward. The goal is to prepare a solution to be used as "warm" start for a problem that has progressed in time. There are slight differences in the operation on each variable type.

- u_k is shifted, so that an old u_{k+1} becomes the new u_k. The exception is at the boundary, where the old u_{N-1} is kept.
- x_0 is unchanged, because it will be overwritten elsewhere.
- X_e is shifted, so that an old x_{k+1} becomes the new x_k, and in the case of collocation: each collocated element matrix is shifted in a similar fashion. The exception is at the boundary, where the old x_{N-1} or $X_{e,N-1}$ is kept.

The above features are achieved by implementing a binary linear mapping for a column vector. Let $\chi \in \mathbb{R}^{n_u N + n_x + n_x N(d+1)}$ be the column vectorized domain for Problem 5.1, given as $\chi = \mathrm{col}(\mathrm{vec}(u_D), x_0, \mathrm{vec}(X_e))$, see Eq. (5.28) and Eq. (5.29). Let $n = n_u N + n_x + n_x N(d+1)$ and $A_{\ll} = \mathrm{band}(N, 1)$ and $A_{\ll[N,N]} = 1$; $A_{\ll}$ implements the shift-keep operation. The NLP shifter function $F_{\ll} : \mathbb{R}^n \to \mathbb{R}^n$ is defined as

$$F_{\ll}(\chi) := \mathrm{bdiag}(A_{\ll} \otimes I_{n_u}, I_{n_x}, A_{\ll} \otimes I_{n_x(d+1)})\chi. \tag{5.32}$$

5.9.7 NLP trajectory `::create_trajectory_function`

The NLP trajectory function solves a series of semi-explicit DAE of index-1 for a given piecewise constant $u(t), \forall t \in \mathbb{T}_d$. The solution is evaluated at regular intervals within each element $k \in \mathbb{I}_{\geq,N}$. Let $\mathbb{T}_{k,D} := \left\{ t \in [t_k, t_{k+1}) : \forall i \in \mathbb{I}_{>,d+1}, t = t_k + i\frac{(t_{k+1}-t_k)}{d+1} \right\}$ be a sequence of time points as a column vector. The trajectory function is

$$\begin{aligned} &F_{\text{traj}} \to \mathbb{R}_{>}^{1+N(d+1)} \times \mathbb{R}^{n_x \times (1+N(d+1))} \\ &(u_D, x_0, p, v, t_0) \mapsto (\mathrm{col}_{k \in \mathbb{I}_{>,N}}(\mathbb{T}_{k,D}), \mathrm{row}(x_0, \mathrm{row}_{k \in \mathbb{I}_{\geq,N}}(\mathrm{row}_{i \in \mathbb{I}_{>,d}}(x(t_{k,i}))))), \end{aligned} \tag{5.33}$$

where $x(t_{k,i}) = F_I(x_{k,i-1}, z_{k,i-1}, p_k, t_{k,i-1}; h = t_{k,i} - t_{k,i-1})_{\langle 1 \rangle}$ for each $i \in \mathbb{I}_{>,d}$, and $p_k = \mathrm{col}(u_k, p, v)$.

CHAPTER

SIX

PATH PLANNER FOR DEPLOYMENT

6.1 Motivation and rationale

During the phase before purse seine deployment the purse seine master ("the purser") needs to acquire and maintain a situational awareness. This is a key success factor in purse seining. Elements of the situational awareness include a comprehension of the main process components in play. A firm comprehension increases the rate of success and the diminishes the probability of disastrous failure due to malpractice. Mastery of purse seining is often correlated with experience. Regardless of the level of experience, there is a fair amount of burden placed on the purser. For this reason, a useful aid would be a tool that could both ease the burden and help acquire the necessary situational awareness. Let us for now denote this tool as an *on-board decision support system* (ODSS) [26].

Modern purse seiners are equipped with sophisticated instruments, which help the captain in executing their job. The wheel house has many monitors with diverse and scattered information, from which some are very important during the pre-deployment phase. A proficient purser possesses both perceptive and predictive abilities. This means that based on the available sensory information about the processes in play, the purser is capable of predicting ahead in time a plausible outcome based on a series of actions. These actions are typically, i) to purposefully maneuver the vessel, and ii) to determine an appropriate time point for purse deployment in such a way that the fish is caught as intended.

The procedure has several similarities with a typical optimal control problem, which help justify our chosen plan of attack in developing a tool in form of an ODSS. This is because, in many aspects, the purser is a *model predictive controller* with the *objective* to optimally execute a purse seine deployment. From the above description it is clear that the following **features** are important:

- Ability to make use of available sensory information;
- Comprehension of the main process components;
- Possess predictive capabilities;
- Devising a set of purposeful actions;
- Fulfill a well-defined purpose, which has a specific objective.

Coincidentally, these features also signify successful building blocks in an optimal control problem [6]. Our rationale for choosing mathematical optimization as the problem solving approach is mainly due to the likeness of features, but it is not the sole reason. One other important justification is the flexibility in problem formulation and, in effect, the possible solution space. With this, we mean that there may be user-specific preferences, which have implications for the decision support presented to the user. Moreover, the flexibility in problem description also facilitates adaptations and improvements with little additional effort. This effort may include better model descriptions based on new data or other tweaks to the mathematical modeling.

6.2 Problem statement of the purse seine path planner

The goal is to create an ODSS for purse seine deployment. We proceed by establishing central building blocks for the optimal control problem formulation. Specifically, we need to describe in detail the **features** as listed above. We pose the formulation as a so-called path planning problem. A solution to this problem contains the following:

- A suggested planar path for the fishing vessel to follow;
- An indicated point in time and space for initiation of purse deployment;
- Auxiliary information about the involved actors, which helps situational awareness.

Typically, such solutions should be presented to the purser in an informative manner, such as a graphical visualization including a map plot[1] . The arrows in Fig. 6.1 indicate information flow and interactions between important elements of the presented decision support. Instruments are capable of acquiring relevant environmental data together with vessel-specific measurements in real time. We assume that there exists a machine-to-machine application programming interface that are capable of sharing this information, for instance "Ratatosk" [23][2] . Herein, we focus on the mathematical modeling of the *Online path planner* indicated in Fig. 6.1.

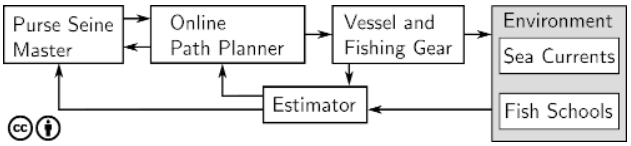


Fig. 6.1: Main elements in a purse seine deployment process.

From a path planning perspective, the relevant phases of a purse seine deployment are the pre-deployment positioning and the purse deployment circling [5]. Questions in need of answers are, where is the school, where will it go, what is the current, how should we maneuver the vessel, where and when should we initiate deployment, will the net sink quick and deep enough. The path planner problem is stated in Problem 6.1.

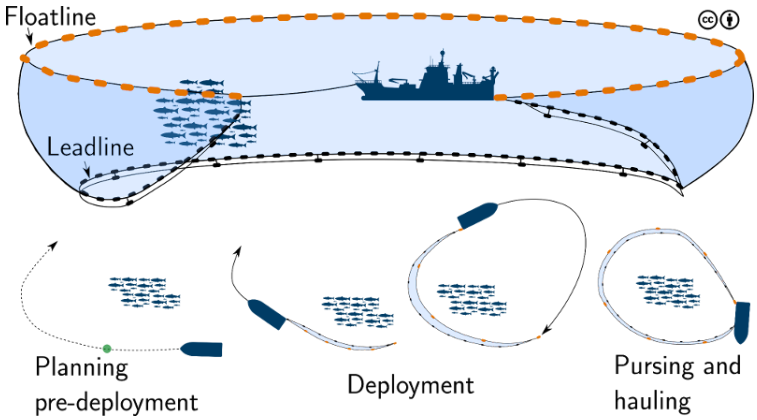


Fig. 6.2: Purse Seine.

(Purse seine path planning)

[1] The sibling project codenamed "Balder" takes care of graphical visualization
[2] Data collection and sharing is achieved with a supporting software named "Ratatosk"

Find an ideal vessel trajectory to surround the targeted fish school, subject to:

- Vessel maneuverability constraints;
- Expected fish school movement;
- Anticipated sinking response; (6.1)
- Environmental conditions;
- Setting orientation and other user preferences.

6.3 Purse planner formulation

Our hypothesis is that an optimization-based algorithm can conceive a sound course of actions based on predictive capabilities. The conception takes form by mathematically describing principal components of the system to a sufficient fidelity. Hence, we formalize Problem 6.1 by utilizing common modeling techniques such as first principles, datasets and engineering cleverness. We should arrive at an optimal control problem formulation, which can be efficiently solved using numerical techniques for nonlinear programming problems [4, 6, 7]. The principal components in need of scrutiny are:

1) Fishing vessel;
2) Fish school;
3) Leadline sinking response;
4) Environment's impact on above components;
5) Criteria for ideal vessel trajectory and deployment initiation.

An important aspect of the formulation is that of timing with respect to the leadline sinking in relation to fish school and vessel trajectories. We identify four time points that signify key events of the purse deployment process, namely:

1) Deployment initiation at deployment point p_d, at time point t_d;
2) Leadline enters waters at collision point p_c, at time point t_s;
3) Fish is surrounded, at time point t_{pd}.
4) Fish arrives at collision point p_c, at time point t_c;

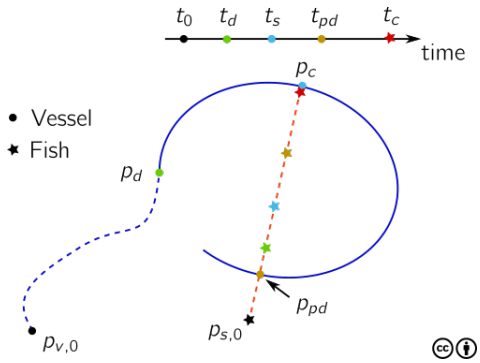


Fig. 6.3: Overview of key time points of the deployment process.

At these time points, the vessel and the fish school are located at various positions in relation to each other and is indicated in Fig. 6.3. The **main objective** of the planner is to determine the deployment time while fulfilling various criteria for a successful deployment. In the following sections, we will describe the principal components and combine them with additional descriptions to solve the timing and trajectory planning problem. Some descriptions are located in *Appendix*.

6.3.1 Notation

- We denote an inertial reference frame with axes x, y, and z pointing north, east, down as $\{\text{NED}\}$.
- Define a limited time horizon $\mathbb{T} := \{t \in \mathbb{R} : t_0 \leq t \leq t_f\}$, where $t_f > t_0 \geq 0$.
- The partial derivative of a function $f(x, y)$ is written with a superscript $\frac{df(x)}{dx} = f^x(x, y)$.
- If the argument is time, we use the common dot notation, i.e. $\frac{dx(t)}{dt} = \dot{x}(t)$.
- The orientation space is defined by $\mathbb{S} \in [-\pi, \pi)$.
- I_n is the $n \times n$ identity matrix.
- $1_{n\times m}$ is an n-by-m matrix of ones.
- $0_{n\times m}$ is an n-by-m matrix of zeros.
- A block diagonal matrix of other matrices $X_{i\in\mathbb{I}_{>,s}} \in \mathbb{R}^{m_i\times n_i}$ is defined as $\text{bdiag}_{i\in\mathbb{I}_{>,s}}(X_i) := \bigoplus_{i\in\mathbb{I}_{>,s}} X_i$, where $\oplus$ is the direct sum.
- The symbol $\otimes$ is the Kronecker product.
- The vertically stacked matrix of other matrices $X_{i\in\mathbb{I}_{>,s}} \in \mathbb{R}^{m_i\times n}$ is denoted $\text{col}_{i\in\mathbb{I}_{>,s}}(X_i) := \text{bdiag}_{i\in\mathbb{I}_{>,s}}(X_i) \cdot (1_{s\times 1} \otimes I_n)$.
- The horizontally stacked matrix of other matrices $X_{i\in\mathbb{I}_{>,s}} \in \mathbb{R}^{m\times n_i}$ is denoted $\text{row}_{i\in\mathbb{I}_{>,s}}(X_i) := \text{col}_{i\in\mathbb{I}_{>,s}}(X_i^T)^T$.

6.3.2 Sea current and surface current water frame

Suppose the sea current is given by $v_c(p, t) : \mathbb{R}^3 \times \mathbb{R} \to \mathbb{R}^3$, where p is a position in $\{\text{NED}\}$. We assume that the current is slowly-varying and can be approximated in a region of interest by a vertical vector field, which is independent of planar position. The sea current is thus assumed constant in each depth layer for a limited time horizon, and is approximated by $w_c(p_z; t_0) \approx v_c(p, t)\, \forall t \in \mathbb{T}$.

We will make use of a reference frame that moves with the surface current, denoted *water frame*, $\{\text{WF}\}$, with axes aligned with $\{\text{NED}\}$. The constant planar surface current can be written in polar form as

$$w_c(0; t_0) = W \begin{bmatrix} \cos(\alpha) \\ \sin(\alpha) \\ 0 \end{bmatrix}, \tag{6.2}$$

where the origin of the water frame in $\{\text{NED}\}$ is written as an initial value problem

$$\begin{aligned} \dot{p}_w(t) &= W \begin{bmatrix} \cos(\alpha) \\ \sin(\alpha) \end{bmatrix}, \\ p_w(t_0) &= \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \end{aligned} \qquad \forall\, t \in \mathbb{T} \tag{6.3}$$

Notice that the initial value of the water frame is always set to the origin.

6.3.3 Fishing vessel

We need a dynamic model that has enough fidelity to allow us to devise maneuverable paths. The fishing vessel is described as a planar kinematic vehicle under the influence of sea current. The planar position of the vessel on the xy-plane of $\{\text{NED}\}$ is $p_v(t) \in \mathbb{R}^2$. Define $U_v > 0$ as the speed in water. Let $\psi_v(t) \in \mathbb{R}$ be the heading of the vessel, that is, the angle between the vessel's body frame and $\{\text{NED}\}$. Further, let $r_v(t)$ be the rate of turn. Define $x_v(t) = \text{col}(p_v, \psi_v, r_v) \in \mathbb{R}^4$ as the vessel state vector. We state the dynamic model as a constrained initial value problem with ordinary differential equations (ODEs) in Eq. (6.4).

$$\begin{aligned}
\dot{p}_v(t) &= U_v \begin{bmatrix} \cos(\psi_v) \\ \sin(\psi_v) \end{bmatrix} + W \begin{bmatrix} \cos(\alpha) \\ \sin(\alpha) \end{bmatrix}, \\
\dot{\psi}_v(t) &= r_v, \\
\dot{r}_v(t) &= u_r, \qquad\qquad \forall\, t \in \mathbb{T} \\
p_v(t_0) &= p_{v,0}, \quad \psi_v(t_0) = \psi_{v,0}, \quad r_v(t_0) = r_{v,0}, \\
u_r(t) &\in \mathcal{U}_v := [u_{r,\min}, u_{r,\max}], \\
x_v(t) &\in \mathcal{X}_v \subseteq \mathbb{R}^4,
\end{aligned} \tag{6.4}$$

where $u_r(t)$ is a constrained control input, and given initial conditions $p_{v,0}$, $\psi_{v,0}$, $r_{v,0}$. Remark that the state vector is constrained, and especially the rate of turn may be bounded to retain realistic maneuvers.

Note: This dynamic model is currently not used in the optimization problem. We use instead trajectories from a path following algorithm, described in *Path following*.

6.3.4 Fish school

A purse seine master uses a multi-beam sonar to evaluate a fish school size and biomass [27]. Fish schools takes many different shapes and its geometry changes based on complex internal and external factors, such as vessel disturbance, sea currents, and individual-level behavior [10]. However, once a fish school has been singled out for targeting, its extent, center of mass, depth, direction, and speed are the most relevant attributes when devising a deployment path. A sonar can typically provide intermittent estimates of this type of information as a datagram [14]. To secure a computationally efficient model, we deliberately describe the model with limited inter-connectivity with external factors. In particular, we assume that the vessel does not influence the fish school movements in our planning description. Let $p_s(t) \in \mathbb{R}^2$ in the $\{\text{NED}\}$ frame be a point mass representing the fish school. The dynamic fish school model is a constant planar velocity model under influence of sea current. Define $U_s > 0$ as the speed in water and $\psi_s \in \mathbb{S}$ the fish heading. The dynamic fish school model at depth z_s is an ODE with given initial conditions as described in Eq. (6.5).

$$\begin{aligned}
\dot{p}_s(t) &= U_s \begin{bmatrix} \cos(\psi_s) \\ \sin(\psi_s) \end{bmatrix} + \begin{bmatrix} I_2 & 0_{2\times 1} \end{bmatrix} w_c(z_s; t_0), \qquad \forall\, t \in \mathbb{T} \\
p_s(t_0) &= p_{s,0},
\end{aligned} \tag{6.5}$$

where $p_{s,0}$, ψ_s, z_s, U_s are given values. It is straightforward to expand ψ_s to become non-constant, as to allow a turning fish school. Note that V_s and χ_s are the school speed and course over ground, which can be obtained using Eq. (6.48), Eq. (6.49), and Eq. (6.50).

6.3.5 Leadline sinking response

Comprehension of the sinking response of the purse seine is an important part of the deployment planning. There exists high-fidelity models that are capable of modeling the sinking response of a purse seine under influence of sea currents [20]. The sinking dynamics is highly influenced by gear configuration, such as mesh size, lead weights, twine thickness, and more. Unfortunately, such models are too computationally demanding to be used in our setup. A study using multivariate regression models [30] conclude that differences in sinking depths are caused by current speeds, shooting duration, included angle of currents, and shooting angle. We choose a practical approach where a simple, but adaptable model is used to estimate the leadline sinking response. The leadline is the bottom sinking part of a purse seine, see Fig. 6.2. Let $z_l(t) \in \mathbb{R}$ be the depth at a point somewhere along the leadline in $\{\text{NED}\}$ frame. The leadline is modeled as a first-order response, which depends on a time constant $\tau_l > 0$ and set-point depth $z_{l,d} > 0$, as defined in Eq. (6.6).

$$\begin{aligned} \dot{z}_l(t) &= \tfrac{1}{\tau_l}(z_l - z_{l,d}) \\ z_l(t_s) &= 0. \end{aligned} \qquad \text{for } t_s \leq t \leq t_{s,f} \tag{6.6}$$

Note the time $t_s > t_0$, which is the time point at which the leadline point enters the water. This overly simplistic model can be made somewhat dependent on the sea current. For instance, by letting the time constant $\tau_l = \tau_l(w_c(p_z; t_0))$ and set-point $z_{l,d}(w_c(p_z; t_0))$ be evaluated constants for a given sea current profile, the sinking response may display various responses, albeit only with linear dynamics.

Due to the simplicity of the model, we may exploit the explicit solution to Eq. (6.6) instead of numerically integrating. The solution is

$$z_l(t) = \left(1 - e^{-(t-t_s)/\tau_l}\right) z_{l,d}. \tag{6.7}$$

It should be noted that based on our datasets of sinking responses and current profiles, we have not been able to establish a consistent correlation between current speeds and sinking depths. We suspect that there are other unobserved phenomena affecting the sinking response, for instance the operation of the purse winches.

6.3.6 Purse planner criteria

Problem 6.1 states that we need to "find an ideal vessel trajectory to surround the fish school". The definition of an *ideal trajectory* is debatable. In our proposed path planner, we add both constraint expressions and objective function terms, which result in a vessel trajectory that meets certain criteria. We make the criteria adjustable through configurable parameters. The purpose of having tunable parameters is – to some extent – to accommodate individual preferences in what is perceived as an *ideal trajectory*. The configurable parameters are not only for the algorithm implementer, but also for the purse seine master practitioner. Below we define these expressions and terms, and specify which criterion they intend to meet.

(Vessel speed in water is constant)

The vessel speed in water is assumed to be constant

$$U_v(t) = U_v \quad \forall t \in \mathbb{T}. \tag{6.8}$$

The speed is user configurable. Constant speed simplifies timing estimates. Note that this can be relaxed with little additional effort, especially in the pre-deployment phase.

(Ellipse in water frame)

The *ideal deployment trajectory* follows an ellipse as defined by Eq. (6.27). The placement of the ellipse is a key task of the purse planner. The ellipse parameter vector θ_e contains both initial conditions, user parameters, configuration

parameters, and decision parameters. We declare the elements of θ_e as follows

$$\begin{array}{lll} p_{e,0} & = p_{s,0} & \text{School initial position; initial condition} \\ \vartheta & = \psi_{s,w} & \text{School direction in water frame; initial condition} \\ \text{col}(L_x, L_y) & = & \text{Ellipse translation; decision parameters} \\ \text{col}(R_x, R_y) & = & \text{Ellipse radii; user parameters} \\ d_o & = & \text{Setting orientation; configuration parameter.} \end{array} \tag{6.9}$$

Note that $\psi_{s,w}$ is evaluated as $\{NED\}$ velocity minus water surface velocity, using equations defined in *Slip angle*.

(Collision point)

The collision point p_c is the anticipated point at which the purse deployment intersects the fish trajectory. Let $p_p(\varpi_c)$ be a particle on the ellipse defined in Criterion 6.9 and $p_{s,c}(\tau_c)$ a particle along the fish trajectory. The fish trajectory is the solution to Eq. (6.5), which is parameterized as Eq. (6.28), with $\theta_l = \text{col}(p_{s,0}, V_{s,w}, \psi_{s,w})$, where $V_{s,w}$ and $\psi_{s,w}$ is speed and orientation of the constant fish velocity in water frame. The overall objective is $\min_{\varpi_c, \tau_c} \|p_p(\varpi_c) - p_{s,c}(\tau_c)\|$. By letting both particles collaborate to solve the objective, we use mutual particle projections Eq. (6.35), with $\sigma_c(t)$ and $\sigma_\tau(t)$ defined according to Eq. (6.30), so that the closed-loop dynamics of the parametrization variables is an initial value problem

$$\begin{array}{ll} \dot{\varpi}_c & = \frac{\gamma \sigma_c}{\|p_p^{\varpi_c}(\varpi_c)\|} \\ \dot{\tau}_c & = \frac{\gamma \sigma_\tau}{\|p_{s,c}^{\tau_c}(\tau_c)\|} \\ \varpi_c(t_0) & = 0,\ \tau_c(t_0) = 0. \end{array} \tag{6.10}$$

With sufficiently large $t = t_{\text{end}}$, both ϖ_c and τ_c have converged so that the objective is fulfilled. The collision point $p_c = p_p(\varpi_c(t_{\text{end}}))$.

(Deployment point)

The deployment point p_d is point at which the purse deployment is initiated. A user parameter D_s is the arc length along the ellipse from the deployment point to the collision point. We can use the *Arc length control objective* Eq. (6.45), with p_d as particle 1 and p_c as particle 2 to find the deployment point. We substitute our particles in Eq. (6.45) and restate the initial value problem as

$$\begin{array}{ll} \dot{s}_d & = -k_d(s_d - (s_c - D_s)) \\ \dot{\varpi}_d & = \frac{\dot{s}_d}{\|p_p^{\varpi}(\varpi_d)\|} \\ \dot{s}_c & = U_{p,c}(t) = \gamma \sigma_c(t) \\ & \varpi_d(t_0) = \varpi_c(t_0) = 0,\ s_d(t_0) = 0,\ s_c(t_0) = 0, \end{array} \tag{6.11}$$

where $k_d > 0$ and we notice that $\dot{\varpi}_c$ is defined in Eq. (6.10) and therefore skipped in the restatement. The particle speed diminishes when $s_d - s_c \to D_s$ and we obtain the deployment point $p_d = p_p(\varpi_d(t_{\text{end}}))$. Notice that $s_d < s_c$, i.e. $\varpi_d < \varpi_c$, and indicates that the deployment occurs *before* collision, which is what we want.

(Fish surrounded)

The point at which the fish is surrounded is chosen as half a circumference from the collision point. This means that $\varpi_{pd} = \varpi_c + \pi$ because the parametrization is given in radians. The point $p_{pd} = p_p(\varpi_{pd})$ can be evaluated once $\varpi_c(t_{\text{end}})$ is known. We also approximate the arc length from the collision point using arc length dynamics Eq. (6.33)

$$\begin{array}{ll} \dot{s}_{pd} & = k_{pd}(\varpi_c + \pi - \varpi_{pd}) \\ \dot{\varpi}_{pd} & = \frac{\dot{s}_{pd}}{\|p_p^{\varpi}(\varpi_{pd})\|}, \end{array} \tag{6.12}$$

since ellipse circumference does not have an analytic solution and we need it for timing evaluation purposes.

(Timing evaluations)

We need to estimate key time points to determine ellipse placement in relation to the fish trajectory. Since the ellipse is defined in water frame and the vessel speed in water is constant as per Assumption 6.8, it is simple to determine several of the requested time points. The timings are

$$\begin{array}{lll} t_d & & \text{Deployment initiation} \\ t_s & = t_d + D_s/U_v, & \text{Leadline in water at collision point} \\ t_c & = \tau_c, & \text{Fish arrives at collision point} \\ t_{pd} & = t_s + s_{pd}/U_v, & \text{Fish is surrounded} \\ t_{s,\Delta} & = t_c - t_s, & \text{Duration of leadline sinking before fish arrives.} \end{array} \tag{6.13}$$

For given ambient conditions and ellipse parametrization all except t_d are easily evaluated with the help of the previous criteria. t_d is related to the pre-deployment positioning of the vessel and is found based on Criterion 6.15, Criterion 6.14, and Eq. (6.23).

(Pre-deployment positioning)

There is an algorithm that creates a trajectory for pre-deployment positioning. We make use of a path following controller to converge to and follow the ellipse in Criterion 6.9, which is described in *Path following* using Eq. (6.39), Eq. (6.43), Eq. (6.41), Eq. (6.44). The initial value problem for the path following algorithm is

$$\begin{aligned} \dot{p}_v &= V_v(t) \begin{bmatrix} \cos\chi_v \\ \sin\chi_v \end{bmatrix}, \\ \dot{\varpi}_v &= \frac{V_v(t)\cos\chi_r(e_v) + \gamma\sigma_v(t)}{\|p_p^{\varpi}(\varpi_v)\|}, \\ \dot{\chi}_v &= \omega_{\max}\frac{\tilde{\chi}_v}{\sqrt{\tilde{\chi}_v^2 + \Delta_\chi^2}}, \\ p_v(t_0) &= p_{v,0},\ \varpi_v(t_0) = \varpi_0(t_{\text{end}}),\ \chi_v(t_0) = \chi_{v,0}, \end{aligned} \tag{6.14}$$

where $\varpi_0(t_{\text{end}})$ is given by Criterion 6.16, and $\chi_{v,0}$ is calculated with Eq. (6.51), and $V_v(t)$ is evaluated using Eq. (6.49).

(Deployment initiation)

We consider the pre-deployment positioning as finished when the vessel enters within a radius R_{deploy} of the calculated deployment point p_d. This criterion is stated as

$$\|p_v(t_d) - p_w(t_d) - p_d\| \leq R_{\text{deploy}}, \tag{6.15}$$

where $R_{\text{deploy}} > 0$ is the deployment vicinity radius and $p_v(t)$ is the vessel $\{\text{NED}\}$ position.

(Pre-solve vessel particle projection)

It is recommended by [8] to presolve $\varpi(t_0)$ for the path following algorithm of Criterion 6.14. This is achieved with *Particle projection* from Eq. (6.35):

$$\begin{aligned} \dot{\varpi}_0 &= \frac{\gamma\sigma_0(t)}{\|p_p^{\varpi}(\varpi_0)\|}, \\ \varpi_0(t_0) &= 0, \end{aligned} \tag{6.16}$$

where $\sigma_0(t)$ is evaluated by Eq. (6.30) using $p(t) = p_v(t_0)$. The value of ϖ_0 at $t = t_{\text{end}}$ is the point on the ellipse, which is closest to the vessel. It is important to include this in the formulation, because an initial transient of ϖ for the path following algorithm has impact on the pre-deployment closed-loop response. This is especially true if $\varpi(t_0)$ is far from the particle projection and is avoided with this criterion. Note also that this implies that the initial condition

for ϖ in the pre-deployment path following algorithm now becomes "free" in the sense that it is to be found by the optimization problem.

(Sink margin)

Sink margin is the surplus leadline depth beyond the indicated fish school depth. The depth response is calculated at the collision point, where we use Eq. (6.7) with a sinking duration of $t_{s,\Delta} = t_c - t_s$, which is the time difference between fish arrival and vessel crossing at the collision point. Let $z_{\min}$ be the user-specified minimal surplus margin so that the following constraint must be satisfied

$$z_l(t_{s,f}) - z_s \geq z_{\min}, \tag{6.17}$$

where $t_{s,f} = t_s + t_{s,\Delta}$ and z_s is fish depth.

(Fish trapped margin)

At the time point the vessel has completed the deployment ellipse, we need to ensure that the fish is inside the ellipse. We make use of the path-tangential frame at $p_{pd} = p_p(\varpi_{pd}(t_{\text{end}}))$ and calculate the cross track error of the fish position at time point t_{pd}. The fish is inside the ellipse for positive ϵ_{trap}:

$$\begin{bmatrix} \sigma_{\text{trap}} \\ \epsilon_{\text{trap}} \end{bmatrix} = d_o R^T(\chi_{pd})(p_s(t_{pd}) - p_w(t_{pd}) - p_p(\varpi_{pd})), \tag{6.18}$$

where χ_{pd} is the ellipse tangent at $p_p(\varpi_{pd})$ and d_o is setting orientation.

Suppose the user requests a minimal trap margin $\epsilon_{\min}$. We can write this constraint as

$$\begin{aligned} \epsilon_{\text{trap}} - \epsilon_{\min} + s_\epsilon &\geq 0, \\ s_\epsilon &\geq 0, \end{aligned} \tag{6.19}$$

where s_ϵ is a slack variable to avoid infeasible solutions.

Objective function terms

The objective function has the terms listed in Table 6.1, which are to be minimized.

Table 6.1: Objection function terms

Term	Description
$\rho_{\text{deploy}} t_d$	Find a minimal deployment time point that meets our criteria
$\rho_\Delta s_a s_b$	Ensure a non-negative sink duration, $t_{s,\Delta} = \max(0, t_c - t_s)$,
	which is implemented using constraint qualifications as described in [7].
$\rho_{\text{trap}} s_\epsilon$	The fish trap slack is heavily penalized and exists to avoid some infeasible solutions.

Currently, all objective terms are terminal objectives, so the optimization problems is cast as a *Mayer problem*, with

$$\Phi_M(\mathcal{Z}, \theta) = \rho_{\text{deploy}} t_d + \rho_\Delta s_a s_b + \rho_{\text{trap}} s_\epsilon, \tag{6.20}$$

where $\mathcal{Z}$ and θ will be defined further below.

6.3.7 Nonlinear programming problem for purse seine deployment

The criteria we defined in the previous section consists of both initial value problems and constraint expressions, which are valid in various time intervals or at specific time points only. We are motivated to collect the initial value problems into two subsystems, each using their own discretization technique and time horizon. Some of the states are stable and does not have control inputs; they are fully defined by initial conditions and parameters. From these state trajectories we are only interested in their terminal value, which we denote t_{end}. For the pre-deployment response from Criterion 6.14, on the other hand, we are interested in finding the unknown final time t_d.

The *casadi form* of a nonlinear programming formulation defines parameters as entities that are constant for one optimization problem, but may be changed from one solution evaluation to the next. We collect both user parameters and initial conditions into the parameter vector as follows:

$$\theta = \operatorname{col}(U_v, p_{v,0}, V_s, p_{s,0}, \chi_s, \psi_{s,w}, W, \alpha, z_s, \tau_l, z_{l,d}, D_s, R_x, R_y, z_{\min}, \epsilon_{\min}) \tag{6.21}$$

Note that fixed parameters, such d_o are not exposed as part of θ.

We define Subsystem 1 as initial value problems with horizon $t_1 \in [t_0, t_{\text{end}}] =: \mathbb{T}_1$, which are given by Criterion 6.10, Criterion 6.11, Criterion 6.12, and Criterion 6.16. We collect the state variables as $x_1 = \operatorname{col}(\varpi_0, \varpi_c, \tau_c, \varpi_d, \varpi_{pd}, s_c, s_d, s_{pd})$.

Subsystem 2 are the closed-loop vessel trajectory Criterion 6.14, fish trajectory Eq. (6.5), and water frame Eq. (6.3). The state variables are collected as $x_2 = \operatorname{col}(\varpi_v, p_v, \chi_v, p_w, p_s)$. We make use of the non-fixed time horizon formulation, where the discretization is divided into N_2 time elements each with a step size of h_2. The step size is a decision parameter with a step size range

$$h_2 \in [h_{\min}, h_{\max}], \tag{6.22}$$

with $h_{\{\min,\max\}} = t_{d,\{\min,\max\}}/N_2$. The final time of Subsystem 2 is given as

$$t_d = t_0 + N_2 h_2, \tag{6.23}$$

so that the time interval is $t_2 \in [t_0, t_d] =: \mathbb{T}_2$. Criterion 6.15 is the constraint that dictates t_d.

By assembling the equations from the preceding section, we are ready to formalize Problem 6.1 as an optimal control problem in Problem 6.25. We collect decision parameters into the vector $\nu \in \mathcal{V} \subset \mathbb{R}^6$ as

$$\nu = \begin{bmatrix} L_x \\ L_y \\ s_a \\ s_b \\ s_\epsilon \\ h_2 \end{bmatrix} \in \begin{matrix} [0, L_{x,\max}] \\ [-L_{y,\max}, L_{y,\max}] \\ [0, \infty] \\ [0, \infty] \\ [0, \infty] \\ [h_{\min}, h_{\max}] \end{matrix} \tag{6.24}$$

Define $\mathcal{Z}$ as $\operatorname{col}(\nu, x_1, x_2)$. The optimization problem is transcribed into a nonlinear programming problem using hybrid discretization, which implies separate discretization schemes for Subsystems 1 and 2. We can do this because they are loosely coupled through parameters and initial/terminal states only.

(Purse seine OCP)

$$\begin{aligned} &\text{minimize:} && \\ &\nu \in \mathcal{V} && \Phi_M(\mathcal{Z}, \theta) \\ &\text{subject to:} && \forall t_1 \in \mathbb{T}_1, t_2 \in \mathbb{T}_2 \end{aligned} \tag{6.25}$$

Criterion 6.9, Criterion 6.17, Criterion 6.19, Criterion 6.13, Criterion 6.15, Eq. (6.24); Subsystem 1: Criterion 6.10, Criterion 6.11, Criterion 6.12, Criterion 6.16; Subsystem 2: Criterion 6.14, Eq. (6.5), Eq. (6.3).

The argument that minimizes Problem 6.25 is $\nu^\star$. The corresponding vessel and fish trajectories are $p_v^\star(t)$ and $p_s^\star(t)$ and can be found by solving an initial value problem of Subsystem 2 with a desired time horizon and $L_x^\star$ and $L_y^\star$ from $\nu^\star$. These functions can be used as decision support to the purse seine master by visualizing them in a graphical user interface.

6.3.8 Receding horizon

A solution to Problem 6.25 has limited validity because the ambient conditions or user preferences will change. We therefore solve Problem 6.25 on a regular interval with updated initial conditions in a receding horizon fashion. Typically, solutions from overlapping time intervals are used as warm start for the decision variables in the NLP, because it can speed up solution times considerably.

6.3.9 Discussion on solution strategies and improvements

The NLP resulting from discretization of Problem 6.25 has few decision variables, but they are strongly connected to the system dynamics. As a result the computation of the Hessian of the Lagrangian is costly with many numerical evaluations. It is advisable to use hessian approximation, which speeds up solution times by an order of magnitude.

We have had best success in solving the problem using single shooting for both subsystems. There may be opportunities in using multi shooting or collocation with good initial guesses for state trajectories and Lagrange multipliers. We have not investigated this yet.

Unfortunately, the initial guesses of the decision parameters have impact on whether the optimization problem finds a solution or not. Decision parameters L_x and h_2 are especially important, but for both these it is relatively straightforward to provide initial guesses based on some distance and geometry interpretations. Mechanisms for this is not yet implemented.

The optimization problem sometimes arrives at infeasible solutions. Therefore it may be constructive to provide a set of candidate guesses for key decision parameters, so that the algorithm can try various initial guesses.

The pre-deployment positioning currently does respect keeping distance to the fish school as it converges toward the deployment point. This is fine when the vessel is already located on the "right" side in relation to the fish school. A future improvement would be to extend Subsystem 2 with a control scheme that takes into account the distance between the vessel and fish school.

6.4 Appendix

6.4.1 Regularly parameterized paths and path following

Regularly parameterized paths

A planar parametric curve parameterized by a scalar variable $\varpi \in \mathbb{R}$ belongs to a one-dimensional manifold defined as [8]:

$$\mathcal{P} := \{p \in \mathbb{R}^2 : p = p_p(\varpi) \, \forall \, \varpi \in \mathbb{R}\}. \tag{6.26}$$

Below we define some specific curves used in our formulation.

Ellipse

A rotated and translated ellipse can be defined as follows

$$p_e(\varpi; \theta_e) = p_{e,0} + \underbrace{\begin{bmatrix} \cos(\vartheta) & -\sin(\vartheta) \\ \sin(\vartheta) & \cos(\vartheta) \end{bmatrix}}_{R(\vartheta)} \begin{bmatrix} L_x + R_x \cos(d_o \varpi) \\ L_y + R_y \sin(d_o \varpi) \end{bmatrix}, \tag{6.27}$$

where $\theta_e := \text{col}(p_{e,0}, \vartheta, L_x, L_y, R_x, R_y, d_o)$ is a parameter vector that dictates all aspects of the ellipse, see Fig. 6.4. The orientation for increasing ϖ is determined by $d_o \in \{-1, 1\}$ and ϖ is given in radians.

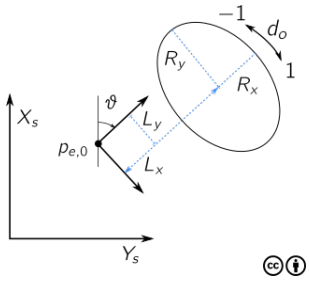


Fig. 6.4: Ellipse path with translation and rotation.

Straight line

The solution to an initial value problem of a constant velocity particle, such as Eq. (6.5) is

$$p_l(\varpi; \theta_l) = p_{l,0} + \varpi V \begin{bmatrix} \cos(\chi) \\ \sin(\chi) \end{bmatrix}, \tag{6.28}$$

where $\theta_l := \mathrm{col}(p_{l,0}, V, \chi)$. Notice that for this parametrization, ϖ is a time variable in seconds, with $\varpi = 0$ being the initial condition.

Path-tangential frame and arc length

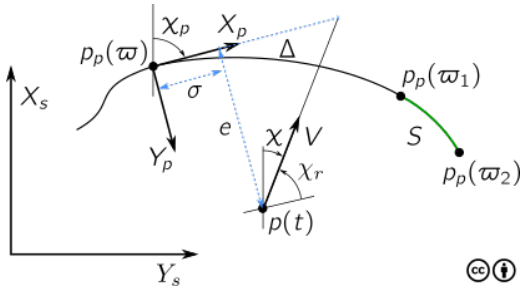


Fig. 6.5: Path-tangential frame and symbols relating to steering on or to regularly parameterized paths.

In the following, it is useful to confer Fig. 6.5. The material is taken from [13]. Consider a planar particle $p_p(\varpi) = \mathrm{col}(x_p(\varpi), y_p(\varpi))$, which is constrained to move along a curve. We define a *path-tangential frame* as a reference frame with origin $p_p(\varpi)$ and x-axis aligned with the tangent of the curve. This frame is rotated an angle $\chi_p(\varpi)$ relative some stationary reference frame using the right-hand screw rule, given by

$$\chi_p(\varpi) = \arctan 2(y_p^{\varpi}(\varpi), x_p^{\varpi}(\varpi)). \tag{6.29}$$

Consider a particle $p(t) \in \mathbb{R}^2$, which may change with time t. The error between p and p_p can be expressed in the path-tangential frame as

$$\begin{bmatrix} \sigma(t) \\ e(t) \end{bmatrix} = \underbrace{\begin{bmatrix} \cos\chi_p & -\sin\chi_p \\ \sin\chi_p & \cos\chi_p \end{bmatrix}^T}_{R^T(\chi_p)} (p(t) - p_p(\varpi)), \tag{6.30}$$

where $\sigma(t)$ is denoted the *along-track error* and $e(t)$ as the *cross-track error*.

The time rate of change of $p_p(\varpi)$ is

$$\begin{aligned} \dot{x}_p(\varpi) &= \tfrac{dx_p}{d\varpi}\tfrac{d\varpi}{dt} = x_p^{\varpi}\dot{\varpi}, \\ \dot{y}_p(\varpi) &= \tfrac{dy_p}{d\varpi}\tfrac{d\varpi}{dt} = y_p^{\varpi}\dot{\varpi}, \\ \dot{p}_p(\varpi) &:= \mathrm{col}(\dot{x}_p, \dot{y}_p), \end{aligned} \tag{6.31}$$

where we have used the chain rule. The speed of a particle is defined as $U_p(t) := \|\dot{p}_p(\varpi)\|$, which also can be written as

$$U_p(t) = \|\operatorname{col}(\dot{x}_p, \dot{y}_p)\| = \|\operatorname{col}(x_p^{\varpi}\dot{\varpi}, y_p^{\varpi}\dot{\varpi})\| = \underbrace{\|\operatorname{col}(x_p^{\varpi}, y_p^{\varpi})\|}_{\|p_p^{\varpi}(\varpi)\|}\dot{\varpi}. \tag{6.32}$$

The arc length between two points is the distance a particle needs to travel when moving from one point to another. If the particle is constrained to move along a curve, we can show that the time derivative of the arc length is, see e.g. [13]

$$\begin{aligned} \dot{s}(\varpi) &= \tfrac{d}{dt}\left(\textstyle\int_a^{\varpi} \|p_p^{\theta}(\theta)\| d\theta\right) \\ &= \|p_p^{\varpi}(\varpi)\|\dot{\varpi} \\ &= U_p(t), \end{aligned} \tag{6.33}$$

which confirms that the arc length is the integral of the particle speed. Notice that the arc length increases for increasing parametrization variable ϖ.

Particle feedback laws

We introduce virtual particles, which we can manipulate to achieve various objectives. Usually, the dynamic response of the parametrization variable ϖ is designed to solve an objective using simple feedback laws. We will cover several such cases in the following.

Particle projection

It is possible to find the point on a curve, which locally minimizes the distance to a point not necessarily located on the curve. The minimization problem

$$\min_{\varpi} \|p(t) - p_p(\varpi)\|, \tag{6.34}$$

has a local minimum when $\sigma(t) = 0$. A simple proportional feedback law achieves this:

$$\dot{\varpi} = \frac{\gamma\sigma(t)}{\|p_p^{\varpi}(\varpi)\|}, \tag{6.35}$$

where $\gamma > 0$ is the feedback gain. With sufficiently large γ, the particle will quickly converge to and be the particle projection of $p(t)$ as it changes with time.

Path following

The path following algorithm for regularly parameterized paths as described in [8] uses particle projection and look-ahead distance to steer a particle $p(t)$ onto a one-dimensional manifold. We restate the algorithm here, because we will use it in the path planner formulation.

Let $p(t) \in \mathbb{R}^2$ represent the planar position of a particle in a stationary reference frame. Let $V(t)$ be the velocity magnitude of the particle, so that

$$\dot{p}(t) = V(t)\begin{bmatrix}\cos\chi(t)\\ \sin\chi(t)\end{bmatrix}, \tag{6.36}$$

where $\chi(t)$ is the particle course – the angle between the x-axis and the particle velocity vector.

Define $p_p(\varpi) \in \mathbb{R}^2$ as a virtual path-constrained particle for which we want $p(t)$ to converge to an move along. The path following objective is therefore

$$\lim_{t\to\infty} \|p(t) - p_p(\varpi(t))\| = 0, \tag{6.37}$$

where we are free to define the dynamics of $\varpi(t)$. Notice that this objective also can be stated using Eq. (6.30) as

$$\lim_{t\to\infty}\begin{bmatrix}\sigma(t)\\ e(t)\end{bmatrix} = 0. \tag{6.38}$$

The dynamics of ϖ is designed as a combination of particle projection feedback Eq. (6.35) and a steering law for the orientation of $p(t)$, given as

$$\dot{\varpi} = \frac{V(t)\cos\chi_r(e) + \gamma\sigma(t)}{\|p_p^{\varpi}(\varpi)\|}, \tag{6.39}$$

where $\chi_r(e)$ is a path-relative steering angle defined as

$$\chi_r(e) = \arctan\left(\frac{-e(t)}{\Delta}\right) \in (-\pi/2, \pi/2), \tag{6.40}$$

with $\Delta > 0$, see Fig. 6.5.

The desired orientation of the particle velocity vector is

$$\chi_d(e, \varpi) = \chi_p(\varpi) + \chi_r(e; \Delta). \tag{6.41}$$

The control objective of the particle course is to drive $\lim_{t\to\infty} \chi(t) - \chi_d(e, \varpi) = 0$. The course error is defined as

$$\tilde{\chi} = \chi_d - \chi, \in (\pi, \pi], \tag{6.42}$$

where $\tilde{\chi}$ is to be within the indicated domain, and can be ensured using [8, 13]:

$$\begin{aligned} \tilde{\chi} &= \arctan 2(\sin(\tilde{\chi}), \cos(\tilde{\chi})) \\ \sin\tilde{\chi} &= \sin(\chi_d - \chi) = \sin\chi_d\cos\chi - \cos\chi_d\sin\chi, \\ \cos\tilde{\chi} &= \cos(\chi_d - \chi) = \cos\chi_d\cos\chi + \sin\chi_d\sin\chi. \end{aligned} \tag{6.43}$$

The course rate controller is given as

$$\dot{\chi} = \omega_{\max}\frac{\tilde{\chi}}{\sqrt{\tilde{\chi}^2 + \Delta_{\dot{\chi}}^2}}, \tag{6.44}$$

where $\omega_{\max} > 0$ is maximal rate of turn and $\Delta_{\dot{\chi}} > 0$ is a rendezvous tuning parameter. This control scheme guarantees that the particle converges to the curve and is proven in [8, 25].

Arc length control objective

Suppose $p_p(\varpi_1)$ and $p_p(\varpi_2)$ are path-constrained particles, denoted 1 and 2. Let the control objective be to obtain a desired arc length between the particles, by manipulating the speed of particle 1. Let S be the desired delta arc length and $s_{\{1,2\}}$ the arc lengths of the particles. The proportional feedback law for s_1 together with the dynamics for particle parametrizations gives us the following initial value problem

$$\begin{aligned} \dot{s}_1 &= -k_d(s_1 - (s_2 - S)) \\ \dot{\varpi}_1 &= \frac{\dot{s}_1}{\|p_p^{\varpi}(\varpi_1)\|} \\ \dot{s}_2 &= U_{p,2}(t) \\ \dot{\varpi}_2 &= \frac{U_{p,2}(t)}{\|p_p^{\varpi}(\varpi_2)\|} \\ & \varpi_1(t_0) = \varpi_2(t_0) = \varpi_0,\ s_1(t_0) = 0,\ s_2(t_0) = 0, \end{aligned} \tag{6.45}$$

where k_d is the feedback gain and $U_{p,2}(t)$ is the speed of particle 2. Note that for $k_d > 0$, we obtain $s_1 < s_2$, and vice versa.

Slip angle

We make a clear distinction between course and heading, where course is the planar orientation of the velocity vector in {NED}, whereas heading is the planar orientation of the body frame. We need expressions to relate them to each other and achieve this using geometric relations.

We consider a particle $p(t) \in \mathbb{R}^2$, which is written in polar form as

$$\dot{p} = V(t) \begin{bmatrix} \cos\chi(t) \\ \sin\chi(t) \end{bmatrix}, \tag{6.46}$$

where $V(t)$ and $\chi(t)$ are the magnitude and orientation of the velocity vector of the particle. Suppose the vector is given as a linear combination of two velocity vectors

$$\dot{p} = U_v \begin{bmatrix} \cos\psi(t) \\ \sin\psi(t) \end{bmatrix} + W \begin{bmatrix} \cos\alpha \\ \sin\alpha \end{bmatrix}, \tag{6.47}$$

where ψ is heading. Let β denote the *slip angle* between the heading and course, that is

$$\chi = \psi + \beta. \tag{6.48}$$

We can find formulas for $V(t)$ and $\chi(t)$, which are expressed in terms of the above variables.

The formula for $V(t)$ is

$$\begin{aligned} V(t) &= \sqrt{(U_v\cos(\psi) + W\cos(\alpha))^2 + (U_v\sin(\psi) + W\sin(\alpha))^2} \\ &= \sqrt{U_v^2 + W^2 + 2U_vW(\cos(\psi(t) - \alpha))} \\ &= \sqrt{U_v^2 + W^2 + 2U_vW(\cos(\chi(t) - \beta(t) - \alpha))} \end{aligned} \tag{6.49}$$

We use the geometric relation in Fig. 6.6 to establish an expression for β. It can be seen that $W\sin(\alpha - \chi) = \ell$, $U_v\sin(\beta) = \ell$, which gives $\sin(\beta) = \frac{W}{U_v}\sin(\alpha - \chi)$. The expression for $\beta(t)$ is thus

$$\beta(t) = \arcsin\left(\frac{W}{U_v}\sin(\alpha - \chi(t))\right). \tag{6.50}$$

When we are given $\psi(t_0)$, we can calculate $V(t_0)$ and

$$\chi(t_0) = \arctan 2(U_v\sin\psi(t_0) + W\sin\alpha, U_v\cos\psi(t_0) + W\sin\alpha)). \tag{6.51}$$

With these variables given, we can evaluate $\beta(t)$ for all t and also calculate $\psi(t)$ using the solution to the differential equations of $\dot{p}(t), \dot{\chi}(t)$.

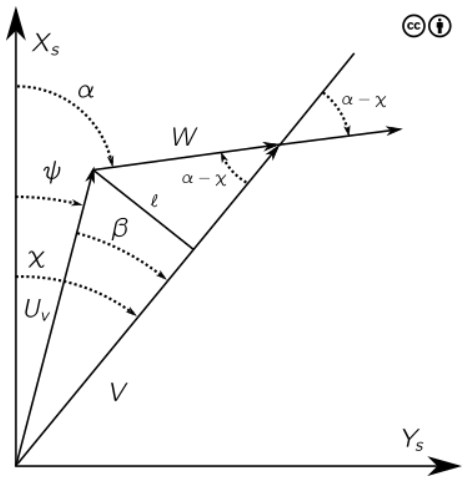


Fig. 6.6: Geometric relations between two velocity vectors.

CHAPTER

SEVEN

A NOTE ON LIBRARY DEPENDENCIES

The nonlinear programming (NLP) formulations in *mimir* uses Casadi's `nlpsol` interface. Casadi implements interfaces to various third-party solvers, but several of them require a commercial license to use. Casadi also provides textbook or experimental solvers to solve such problems, but it is generally recommended to use a third-party solver for best performance and reliability, see [4] for general recommendations. Solvers usually depend on low level routines such as LAPACK/BLAS or METIS. In Table 7.1 we provide an overview of various libraries with their license and availability on Linux and Windows. There are alternatives to these libraries, but these are the options *mimir* can support without extra effort.

The choice of solver can have large impact on solve time of an optimization problem, especially for large problems and by exploiting knowledge of problem structure. There are also other factors that can improve the solution times, including solver settings, using OpenMP, multi-threading, c-code generation, and discretization approach. We have not exhausted the performance improvement potential in our implementation efforts, so there is very likely performance gains available.

Table 7.1: Library dependencies for *mimir*; a summary.

Name	Purpose	License	Conan recipe	Recipe Platforms[1]	Requires
CasADi [4]	Computer algebra system	LGPL-3.0	conan-casadi	Linux, Window	
sqpmethod	NLP solver	LGPL-3.0	with casadi recipe	Linux, Windows	A QO solver (e.g. qrqp)
blockSQP [17]	NLP solver	zlib	with casadi recipe	Linux, Windows	qpOASES
Ipopt [28]	NLP solver	EPL-2.0	conan-ipopt	Linux	MUMPS/HSL or other solver
qpOASES [9]	QP solver	LGPL-2.1	with casadi recipe	Linux, Windows	LAPACK, BLAS
qrqp	QP solver	LGPL-3.0	with casadi recipe	Linux, Windows	
MUMPS [2, 3]	Linear solver	CeCILL-C	conan-coinmumps	Linux	LAPACK, BLAS, (METIS)
HSL [16]	Linear solvers	Various	conan-coinhsl	Linux	LAPACK, BLAS, (METIS)
METIS [18, 19]	Matrix ordering	Custom	conan-coinmetis	Linux, Windows	
OpenBLAS [22, 29]	BLAS, LAPACK	BSD 3-Clause	conan center, conan-openblas	Linux, Windows	
SUNDIALS [15]	Numerical integrators	BSD 3-Clause	with casadi recipe	Linux, Windows	

[1] Platform support only reflects what the referenced recipes can build.

CHAPTER

EIGHT

YAML CONFIGURATION FILES AND SCHEMAS

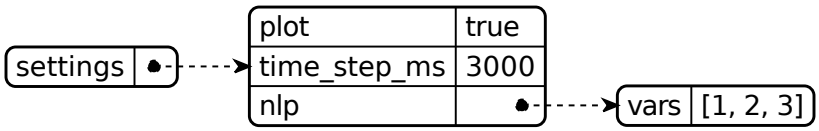
YAML Ain't Markup Language (YAML) is a human-readable data serialization language, which we use to configure the algorithms in this project, yaml-cpp. The application expects the user to adhere to a very specific layout of the configuration file. For some advanced algorithms like `mimir::algorithm::PursePlanner`, care must be taken when specifying the configuration. To help the practitioner, we make use of *schemas* to dynamically validate the configuration files, forked from yavl-cpp-cmake. The YAML validation ensures that the provided YAML structure contains the expected entries, as well as data types. A data type can be one of: `string`, `double`, `float`, `bool`, `uint64`, `int64`, `int`, `uint`, `int16`, `uint16`, `int8`, `uint8`, `enum`.

Under normal circumstances the schema is internal to the program, but for `mimir::algorithm::PursePlanner` it is dynamic in nature, because the settings are propagated to transitive dependencies and changes depending on chosen low-level algorithms (e.g. linear solver). The user needs to expand the schema, which currently resides within the same YAML document as the configuration itself. Below, we present a simple YAML file and its schema, so that a practitioner can grasp the concept when populating the required schema according settings of transitive dependencies.

The YAML in Listing 8.1 has schema Listing 8.2.

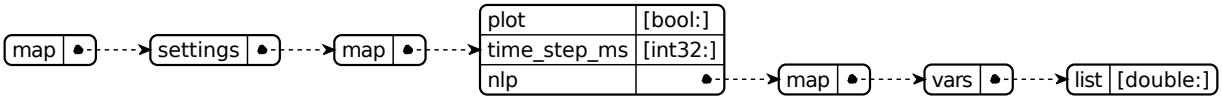
Listing 8.1: (YAML snippet)

```yaml
settings:
  plot: true
  time_step_ms: 3000
  nlp:
    vars: [1, 2, 3]
```



Listing 8.2: (Schema for YAML snippet)

```yaml
map:
  settings:
    map:
      plot: [bool: ]
      time_step_ms: [int32: ]
      nlp:
    map:
      vars:
        list: [double: ]
```



8.1 Purse planner configuration schema

The schema for the `mimir::algorithm::PursePlanner` is given in Listing 8.3. We split each root map of Fig. 8.1 into separate diagrams, as shown in Fig. 8.2, Fig. 8.3, Fig. 8.4, Fig. 8.5, Fig. 8.6, including also Fig. 8.7 and Fig. 8.8.

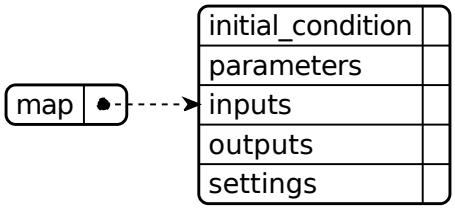


Fig. 8.1: Schema root entries.

Fig. 8.2: Schema initial condition.

Listing 8.3: (Schema for Purse planner configuration)

```
map:
  initial_condition:       { map: { x0: { list: [double: ] }}}
  parameters:
    map:
      setting_speed_U_v:   { map: { topic: [string: ], id: [string: ], default: [double:␣
↪] }}
      setting_radii:       { map: { topic: [string: ], id: [string: ], default: { list:␣
↪[double: ] }}}
      aim_distance_D_s:    { map: { topic: [string: ], id: [string: ], default: [double:␣
↪] }}
      fish_margin_d_f:     { map: { topic: [string: ], id: [string: ], default: [double:␣
↪] }}
      leadline_tau_ll_z_d: { map: { topic: [string: ], id: [string: ], default: { list:␣
↪[double: ] }}}
      sink_margin_z_min:   { map: { topic: [string: ], id: [string: ], default: [double:␣
↪] }}
      current_surface:     { map: { topic: [string: ], default: { list: [double: ] }}}
      current_fish:        { map: { topic: [string: ], default: { list: [double: ] }}}
      fish_velocity_over_ground: { map: { topic: [string: ], default: { list: [double: ]␣
↪}}}
      fish_depth_z_s:      { map: { topic: [string: ], id: [string: ], default: [double:␣
↪] }}
  inputs:
    map:
      GPS_origin:          { map: { topic: [string: ], default: { list: [double: ] }}}
      vessel_pos_info:     { map: { topic: [string: ] }}
```

(continues on next page)

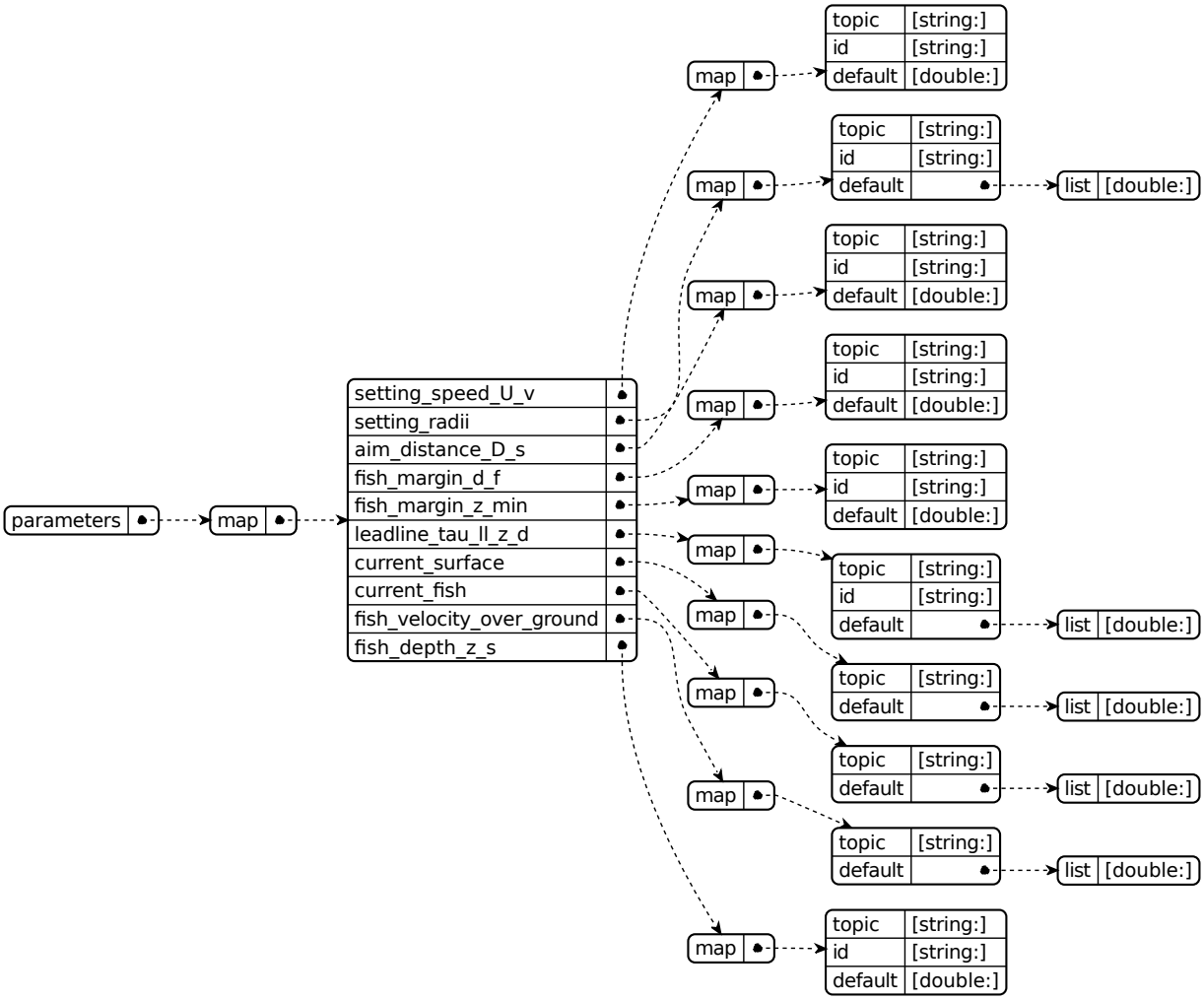


Fig. 8.3: Schema parameters.

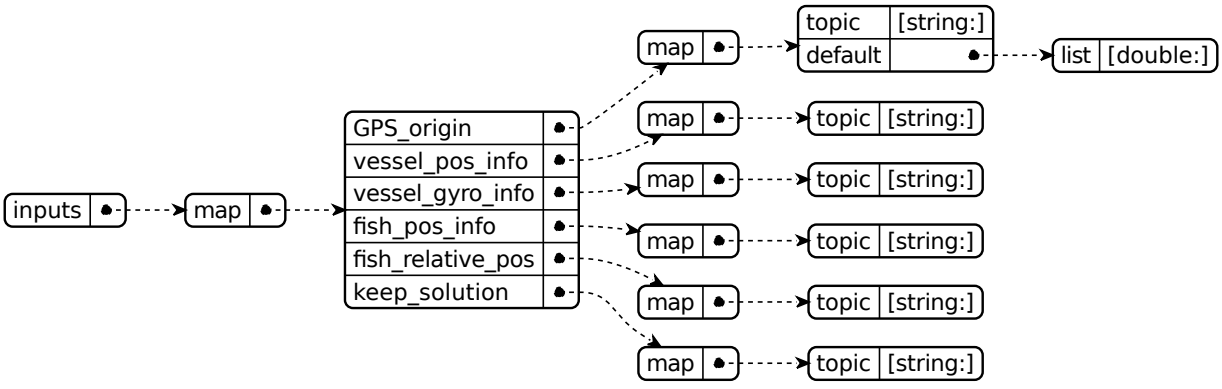


Fig. 8.4: Schema inputs.

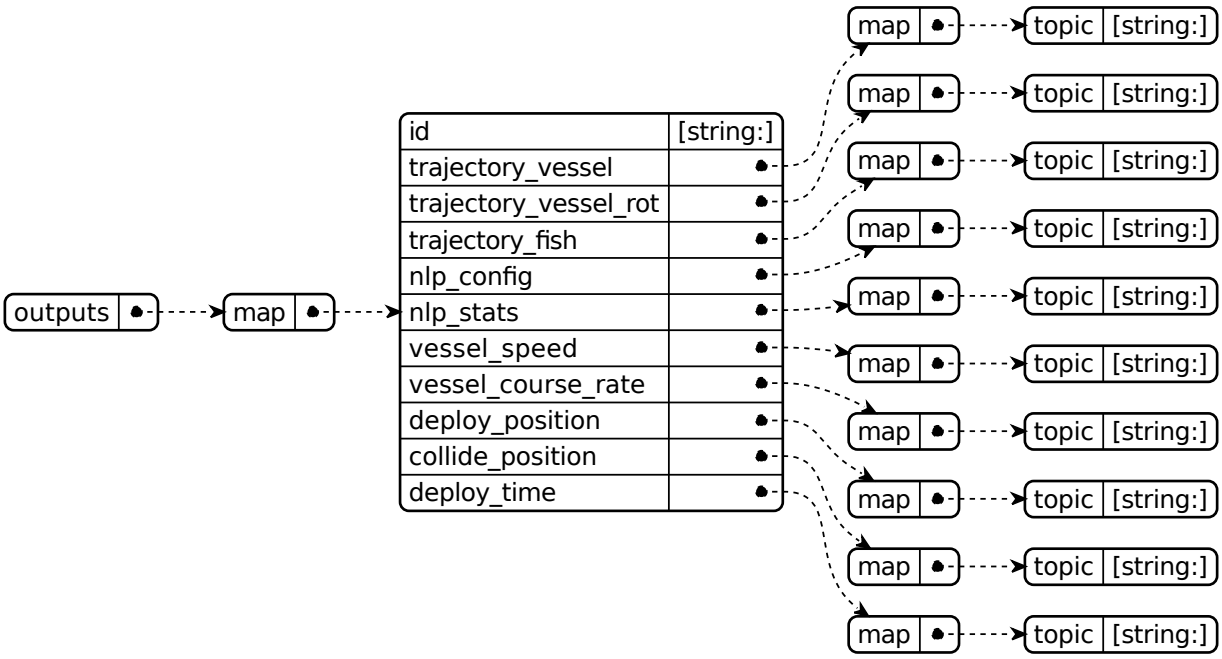


Fig. 8.5: Schema outputs.

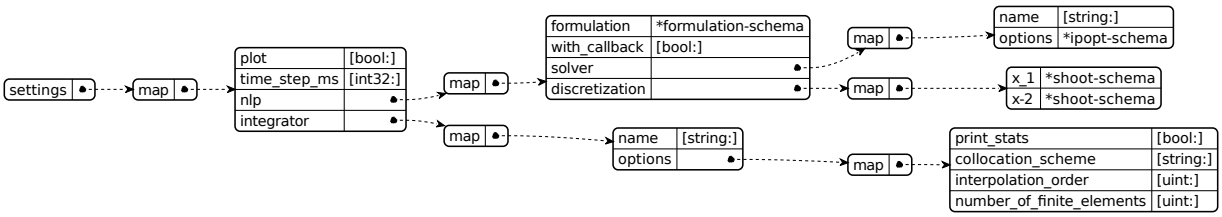


Fig. 8.6: Schema settings.

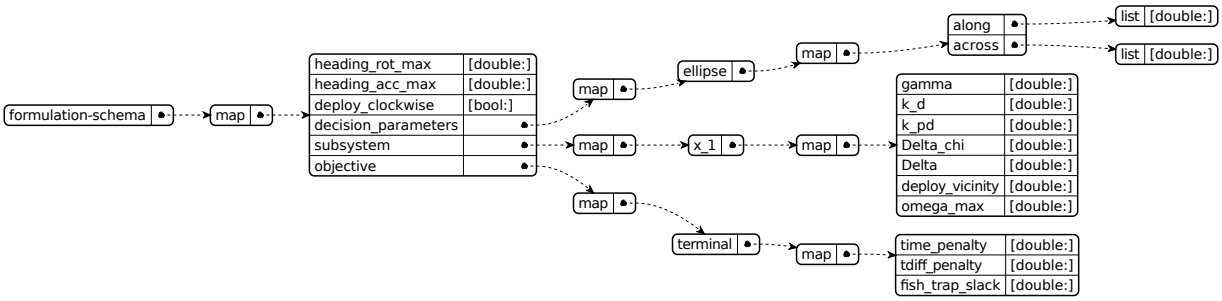


Fig. 8.7: Schema for formulation settings.

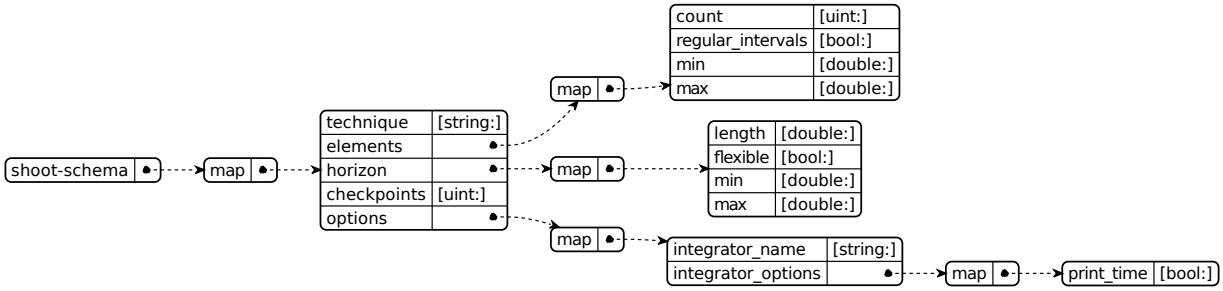


Fig. 8.8: Schema for discretization settings, shooting.

(continued from previous page)

```
    vessel_gyro_info:     { map: { topic: [string: ] }}
    fish_pos_info:        { map: { topic: [string: ] }}
    fish_relative_pos:    { map: { topic: [string: ] }}
    keep_solution:        { map: { topic: [string: ] }}
outputs:
  map:
    id: [string: ]
    trajectory_vessel:    { map: { topic: [string: ] }}
    trajectory_vessel_rot:    { map: { topic: [string: ] }}
    trajectory_fish:      { map: { topic: [string: ] }}
    nlp_config:           { map: { topic: [string: ] }}
    nlp_stats:            { map: { topic: [string: ] }}
    vessel_speed:         { map: { topic: [string: ] }}
    vessel_course_rate:   { map: { topic: [string: ] }}
    deploy_position:      { map: { topic: [string: ] }}
    collide_position:     { map: { topic: [string: ] }}
    collide_time:         { map: { topic: [string: ] }}
settings:
  map:
    plot: [bool: ]
    time_step_ms: [int32: ]
    nlp:
  map:
    formulation:
      map:
        heading_rot_max: [double: ]  # Rate of turn vessel heading [rad/s]
        heading_acc_max: [double: ]  # Heading acceleration [rad/s^2]
        deploy_clockwise: [bool: ]   # Clockwise: true, Counter-clockwise: false
        decision_parameters:
          map:
            ellipse:
              map:
                along: { list: [double: ] } # min, max along-track placement
                across: { list: [double: ] } # min, max cross-track placement
        subsystem:
          map:
            x_1:
              map:
                gamma: [double: ]           # Gain for path constrained particle
```

(continues on next page)

(continued from previous page)

```
                k_d: [double: ]              # Prop. feedback deployment arc length
                k_pd: [double: ]             # Prop. feedback post deploy arc length >>
↪k_d
                Delta_chi: [double: ]        # Rendezvous for tilde chi (should be < 1)
                Delta: [double: ]            # Lookahead distance, large is conservative
                deploy_vicinity: [double: ] # Max. dist. between vehicle and deploy␣
↪point
                omega_max: [double: ]        # Max-ish rate of turn (might be violated)
          objective:
            map:
              terminal:
                map:
                  time_penalty: [double: ]    # Minimize time to arrive at deploy
                  tdiff_penalty: [double: ]   # MPCC dual variable to enforce max(t_diff,
↪0)
                  fish_trap_slack: [double: ] # Penalize slack for fish dist. at deployed
      with_callback: [bool: ]
      solver:
        map:
          name: [string: ]
          options:
            *ipopt-schema
      discretization:
        x_1:
          *shoot-schema
        x_2:
          *shoot-schema
      integrator:
    map:
      name: [string: ]
      options:
        map:
          print_stats: [bool: ]
          #abstol: [double: ]
          collocation_scheme: [string: ]      # collocation
          interpolation_order: [uint: ]       # collocation
          number_of_finite_elements: [uint: ] # collocation
```

BIBLIOGRAPHY

[1] Anders Albert, Lars Imsland, and Joakim Haugen. Numerical optimal control mixing collocation with single shooting: a case study. *IFAC-PapersOnLine*, 49(7):290–295, 2016. 11th IFAC Symposium on Dynamics and Control of Process SystemsIncluding Biosystems DYCOPS-CAB 2016. doi:10.1016/j.ifacol.2016.07.307.

[2] Patrick R. Amestoy, Alfredo Buttari, Jean-Yves L'Excellent, and Theo Mary. Performance and scalability of the block low-rank multifrontal factorization on multicore architectures. *ACM Transactions on Mathematical Software*, 45(1):1–26, February 2019. doi:10.1145/3242094.

[3] Patrick R. Amestoy, Iain S. Duff, Jean-Yves L'Excellent, and Jacko Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, January 2001. doi:10.1137/S0895479899358194.

[4] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019. doi:10.1007/s12532-018-0139-4.

[5] Menakhem Ben-Yami. *Purse seining manual*. Oxford Univ. Press, 1994. ISBN 9780852381939.

[6] John T. Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2 edition, 2010. ISBN 978-0-898-71688-7. doi:10.1137/1.9780898718577.

[7] Lorenz T. Biegler. *Nonlinear Programming: Concepts, Algorithms & Applications to Chemical Processes*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2010. ISBN 978-0-898-71702-0. doi:10.1137/1.9780898719383.

[8] M. Breivik and T. I. Fossen. Guidance laws for autonomous underwater vehicles. In *A. V. Inzartsev (Ed.), Underwater Vehicles*, pages 51–76. IN-TECH Education and Publishing, 2009.

[9] Hans Joachim Ferreau, Christian Kirches, Andreas Potschka, Hans Georg Bock, and Moritz Diehl. qpOASES: a parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6:327–363, January 2014. doi:10.1007/s12532-014-0071-1.

[10] François Gerlotto and Jorge Paramo. The three-dimensional morphology and internal structure of clupeid schools as observed using vertical scanning multibeam sonar. *Aquatic Living Resources*, 16(3):113–122, 2003. Acoustics in Fisheries and Aquatic Ecology. Part 2. doi:https://doi.org/10.1016/S0990-7440(03)00027-5.

[11] Ernst Hairer, Syvert P. Nørsett, and Gerhard Wanner. *Solving Ordinary Differential Equations I*. Springer-Verlag, 2 edition, 1993. ISBN 978-3-540-78862-1. doi:10.1007/978-3-540-78862-1.

[12] Ernst Hairer and Gerhard Wanner. *Solving Ordinary Differential Equations II*. Springer-Verlag, 2 edition, 2010. ISBN 978-3-642-05221-7. doi:10.1007/978-3-642-05221-7.

[13] Joakim Haugen. Guidance algorithms for planar path-based motion control scenarios. Master's thesis, Norwegian University of Science and Technology, Trondheim, Norway, June 2010.

[14] Joakim Haugen and Lars Imsland. Optimization-based motion planning for trawling. *Journal of Marine Science and Technology*, 24(3):984–995, September 2019. doi:10.1007/s00773-018-0600-0.

[15] Alan C Hindmarsh, Peter N Brown, Keith E Grant, Steven L Lee, Radu Serban, Dan E Shumaker, and Carol S Woodward. SUNDIALS: suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):363–396, September 2005. doi:10.1145/1089014.1089020.

[16] HSL. A collection of Fortran codes for large scale scientific computation. 2021. URL: http://www.hsl.rl.ac.uk.

[17] Dennis Janka, Christian Kirches, Sebastian Sager, and Andreas Wächter. An SR1/BFGS SQP algorithm for nonconvex nonlinear programs with block-diagonal Hessian matrix. *Mathematical Programming Computation*, 8:435–459, January 2016. doi:10.1007/s12532-016-0101-2.

[18] George Karypis. *METIS and ParMETIS*, pages 1117–1124. Springer US, Boston, MA, 2011. doi:10.1007/978-0-387-09766-4_500.

[19] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998. doi:10.1137/S1064827595287997.

[20] Hyun Young Kim, Chun Woo Lee, Jong Keun Shin, Hyung Seok Kim, Bong Jin Cha, and Gun Ho Lee. Dynamic simulation of the behavior of purse seine gear and sea-trial verification. *Fisheries Research*, 88(1–3):109–119, 2007. doi:10.1016/j.fishres.2007.08.007.

[21] Jorge Nocedal and Stephen J Wright. *Numerical Optimization*. Springer-Verlag, Berlin Heidelberg, Germany, 2 edition, 2006. ISBN 978-0387-30303-1.

[22] Wang Qian, Zhang Xianyi, Zhang Yunquan, and Qing Yi. AUGEM: Automatically Generate High Performance Dense Linear Algebra Kernels on x86 CPUs. In *Proceeding of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'13)*, 1–12. Denver, USA, November 2013. doi:10.1145/2503210.2503219.

[23] Karl-Johan Reite, Jarle Ladstein, and Joakim Haugen. Data-driven real-time decision support and its application to hybrid propulsion systems. In *ASME 2017 36th International Conference on Ocean, Offshore and Arctic Engineering*, V07BT06A024. American Society of Mechanical Engineers, June 2017. doi:10.1115/OMAE2017-61031.

[24] Boris Schling. *The Boost C++ Libraries*. XML Press, 2011. ISBN 0982219199.

[25] Roger Skjetne, Ulrik Jørgensen, and Andrew R. Teel. Line-of-sight path-following along regularly parametrized curves solved as a generic maneuvering problem. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, volume, 2467–2474. 2011. doi:10.1109/CDC.2011.6161364.

[26] Stian Skjong, Lars T. Kyllingstad, Karl-Johan Reite, Joakim Haugen, Jarle Ladstein, and Karl Gunnar Aarsæther. Generic on-board decision support system framework for marine operations. In *ASME 2019 38th International Conference on Ocean, Offshore and Arctic Engineering*, V07AT06A032. American Society of Mechanical Engineers, June 2019. doi:10.1115/OMAE2019-95146.

[27] Sindre Vatnehol, Hector Pe na, and Egil Ona. Estimating the volumes of fish schools from observations with multi-beam sonars. *ICES Journal of Marine Science*, 74(3):813–821, 12 2016. doi:10.1093/icesjms/fsw186.

[28] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Programming*, 106:25–57, May 2006. doi:10.1007/s10107-004-0559-y.

[29] Zhang Xianyi, Wang Qian, and Zhang Yunquan. Model-driven level 3 BLAS performance optimization on Loongson 3A processor. In *Proceedings of 2012 IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS)*, 684–691. Singapore, December 2012. doi:10.1109/ICPADS.2012.97.

[30] Cheng Zhou, Liuxiong Xu, Hao Tang, and Xuefang Wang. In-situ dynamics of tuna purse seine deployment in various operations and currents. *Fisheries Science*, 81(3):1003–1011, November 2015. doi:10.1007/s12562-015-0918-z.

[31] Modelica Association. Functional Mockup Interface for Model Exchange and Co-Simulation. December 2020. Version 2.0.2. URL: https://fmi-standard.org.

[32] Object Management Group. Data Distribution Service (DDS). Specification formal/2015-04-10, Object Management Group, April 2015. URL: http://www.omg.org/spec/DDS/1.4.

[33] SINTEF Ocean. Catch control in purse seining for pelagic fish: Phase II. 2017–2021. Project supported by the Norwegian Seafood Research Fund, 901350. Partners: SINTEF Ocean, Institute of Marine Research, and Nofima. URL: https://www.fhf.no/prosjekter/prosjektbasen/901350/.

Vedlegg 2

balder

Joakim Haugen

May 31, 2021

OVERVIEW

CHAPTER

ONE

INTRODUCTION

Balder is a graphical user application that provides decision support for the captain in purse seining before gear deployment.

Balder makes use of real-time data and model predictive control algorithms to predict a deployment trajectory for the purse seine. The deployment trajectory is devised by taking into account expected sink depth/speed of the gear and use a prescribed trajectory of the fish school. Environment conditions such as sea currents are also taken into consideration. Figure 1.1 shows a context diagram for Balder.

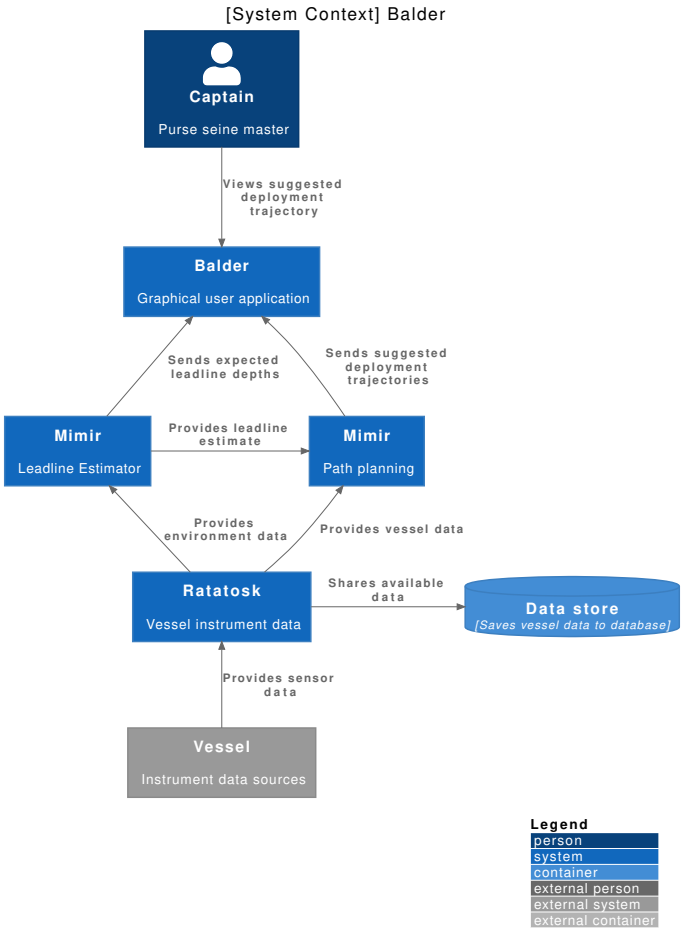


Figure 1.1: System context for Balder.

1.1 Component overview

There are several visual components that display key information to the user, see overview in Figure 1.2. The application is written with Qt modeling language (QML) and the main entrypoint is `balder`. Each component are written using QML classes to declaratively describe the graphical user interface. The main QML classes are indicated within square brackets under each component in Figure 1.2. Details for these classes can be found in API Reference and within links to the corresponding `.qml` files.

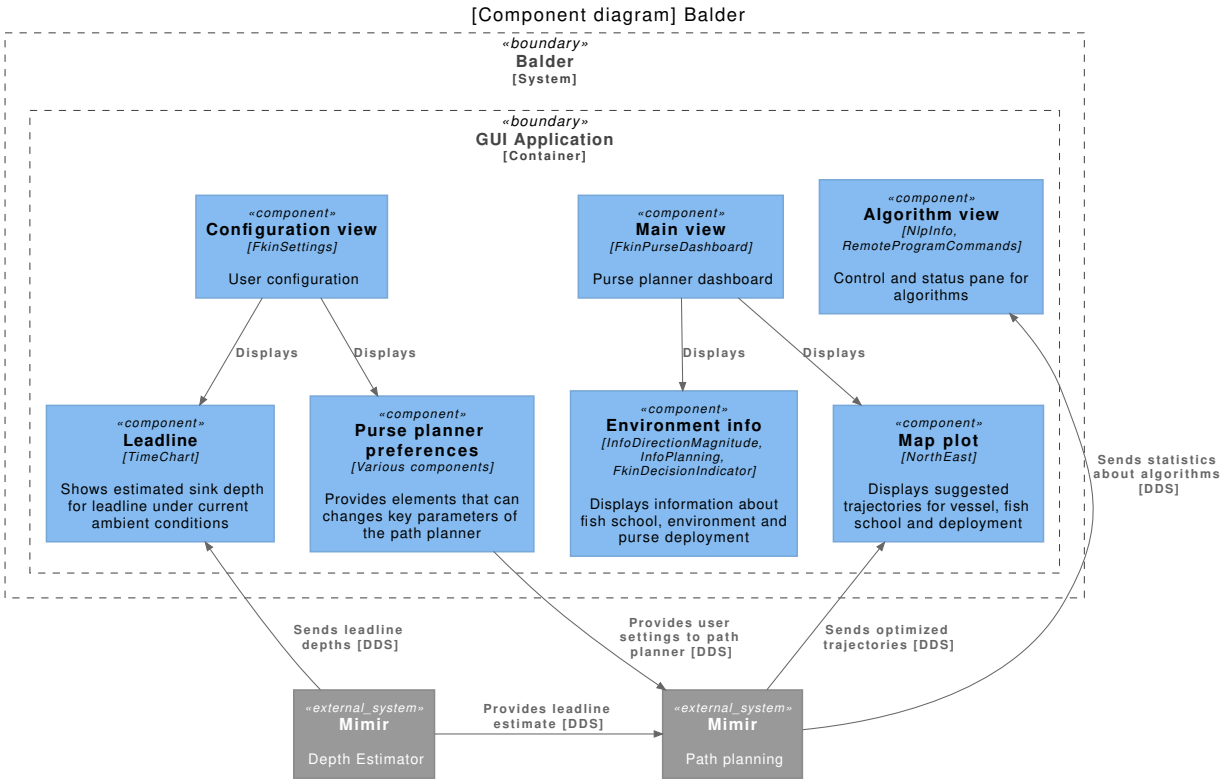


Figure 1.2: Component diagram for Balder.

1.2 Screendumps of application

Below we provide screendumps of three central panel views listed in Figure 1.2:

- Figure 1.3 shows the purse planner dashboard,
- Figure 1.4 shows the user configuration,
- Figure 1.5 show the control and status pane for algorithm.

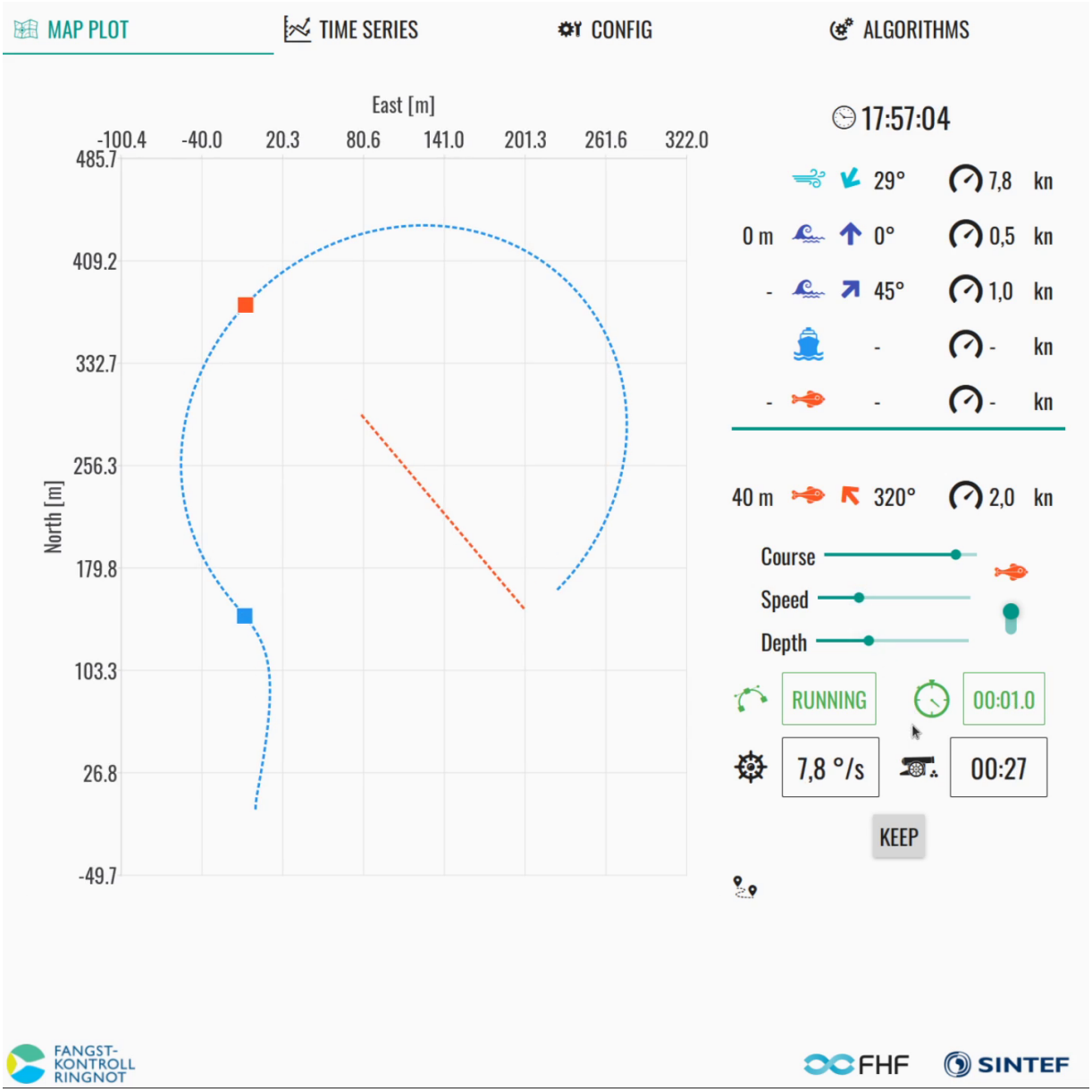


Figure 1.3: Screenshot of main view.

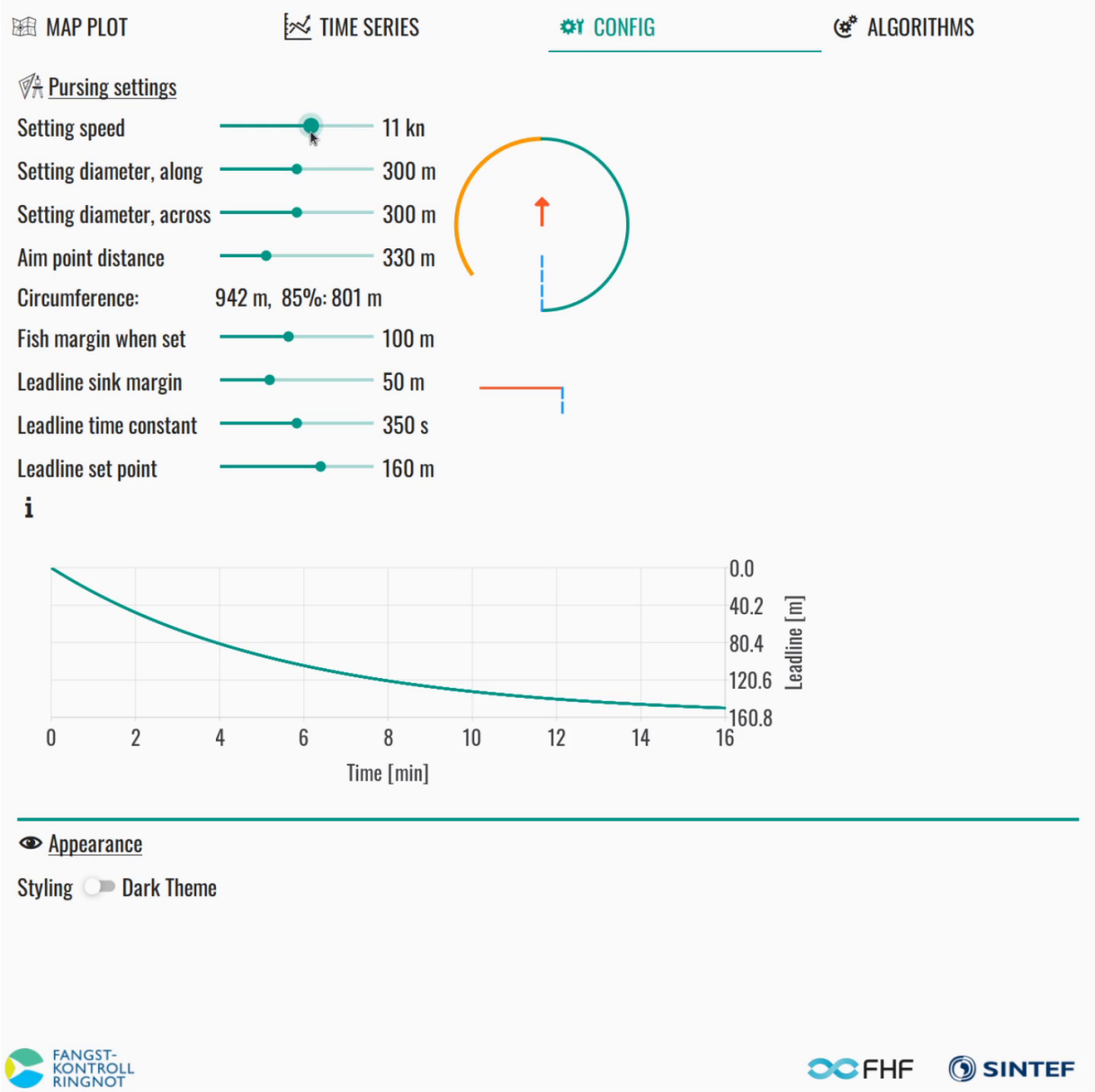


Figure 1.4: Screenshot of configuration view.

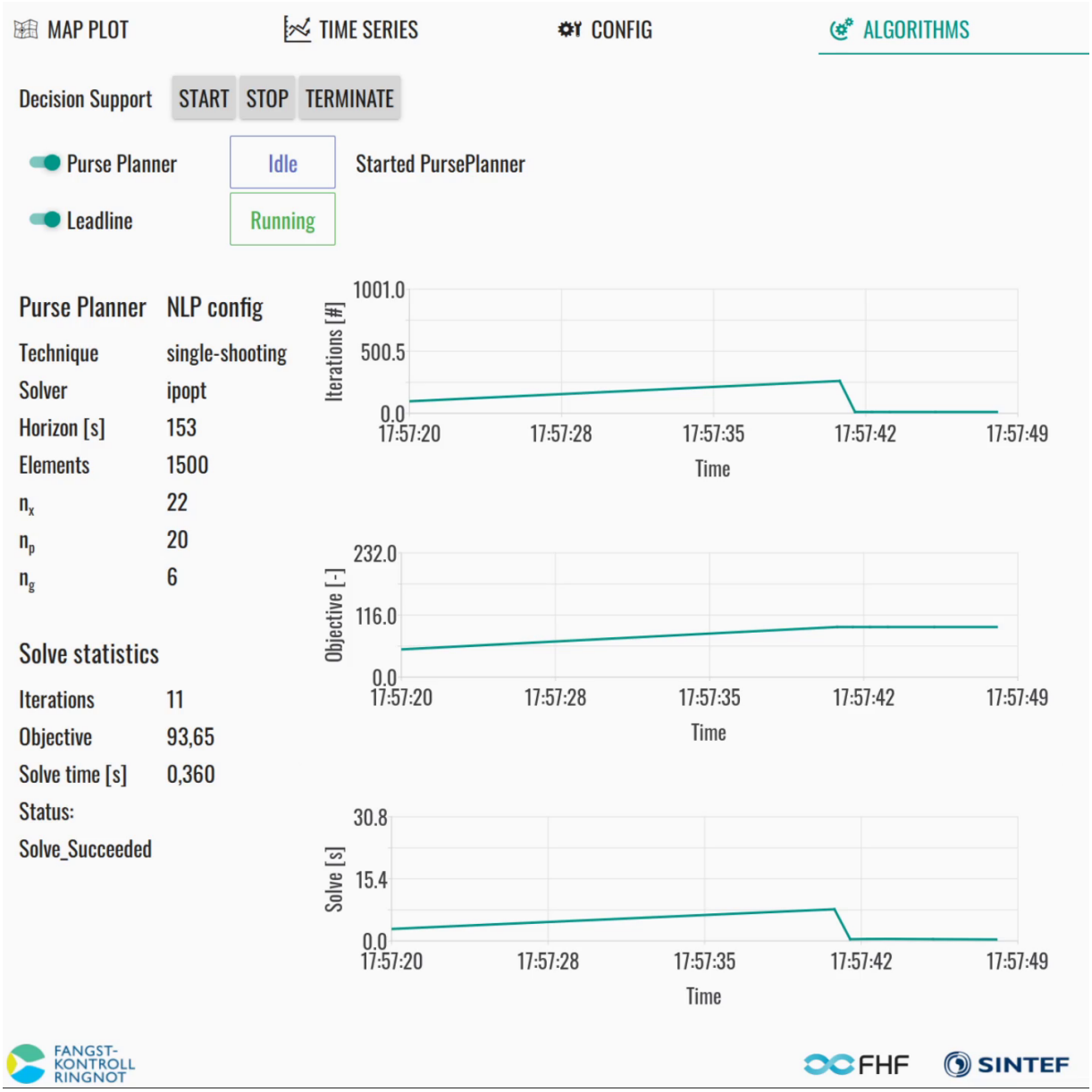


Figure 1.5: Screenshot of algorithm view.

CHAPTER

TWO

USAGE

2.1 Running the application

Once installed, the application should be available as *balder* on the system path or on the start menu.

Balder has support for English (EN) and Norwegian (NO). By default it should load the language specified by the operating system, but can be set explicitly by passing `EN` or `NO` to the command line option as follows:

```
balder --language NO
balder --language EN
```

2.2 Typical usage session

A typical user session can consist of the following steps:

1. Start algorithms; status is indicated as in Figure 2.1.
   - If all input signals are available, the algorithm should provide suggested trajectories regularly.
   - Note that the first iteration may take some extra time (cold start)
2. User preferences are configured.
   - Possible settings are as indicated in Figure 2.2.
   - The user can change setting speed, ellipse shape, aim point, fish margin, leadline sink margin, and leadline sinking coefficients.
   - Please refer to Mimir algorithm for details on user preferences
3. The user navigates to the main dashboard to view suggested setting trajectory, see Figure 2.3.
   - The user has the opportunity to override fish movement and depth
4. The user indicates when he/she wants to keep a suggested trajectory, see Figure 2.4.
   - The planning algorithm will keep the displayed trajectory and only update course rate and countdown deployment
   - When the user rejects the trajectory a new one is provided regularly

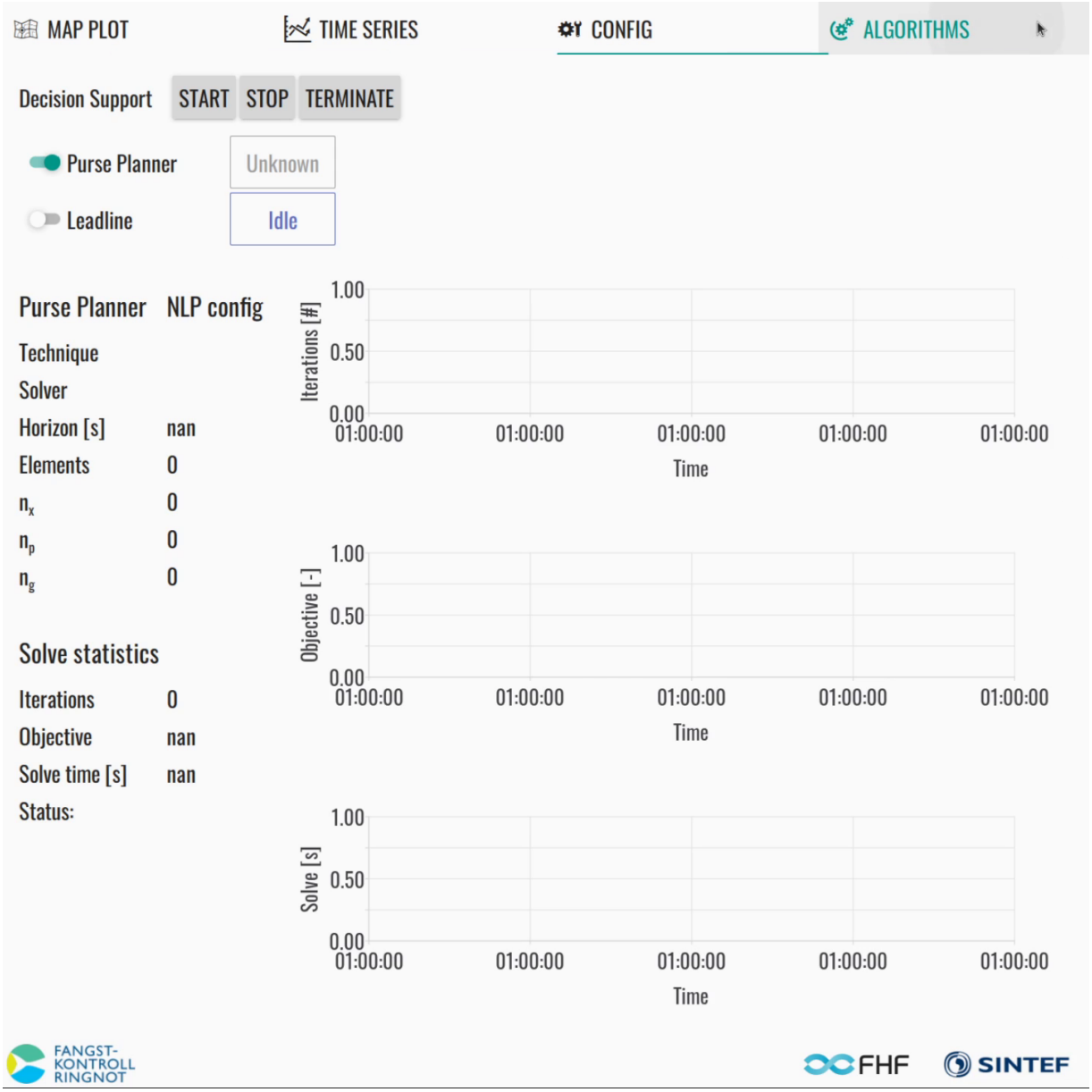


Figure 2.1: Step 1: Start algorithms.

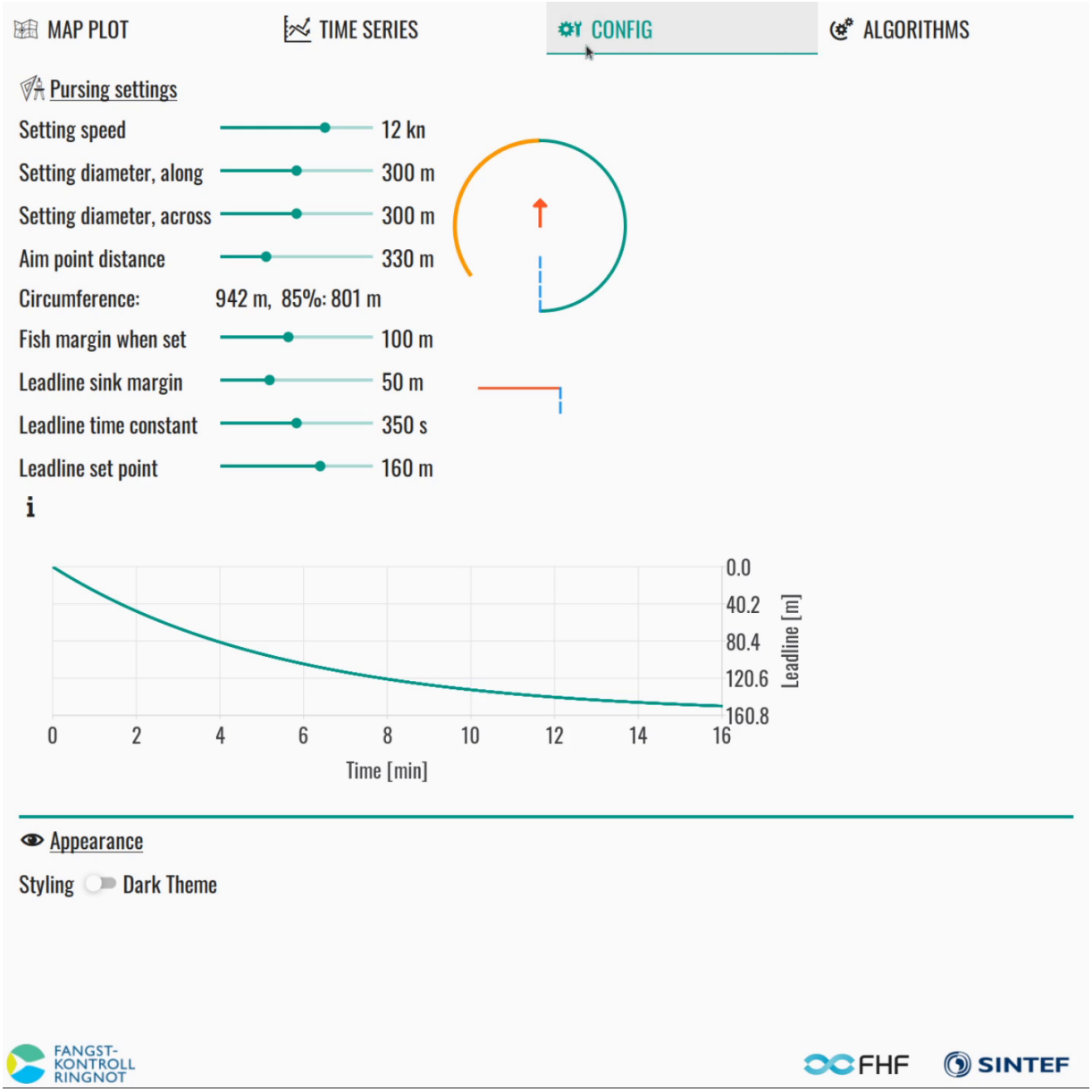


Figure 2.2: Step 2: Configure preferences.

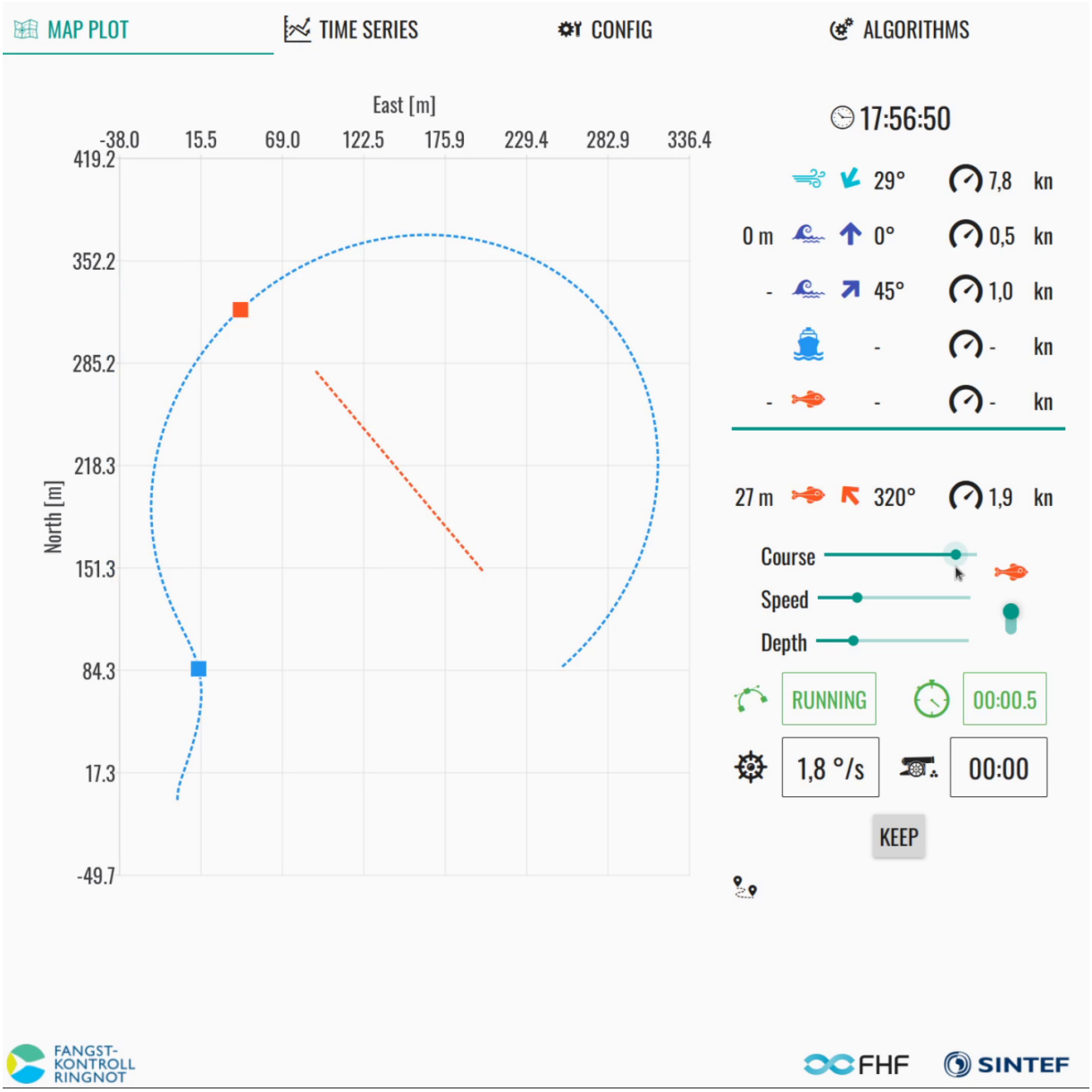


Figure 2.3: Step 3: Dashboard with suggested setting trajectory.

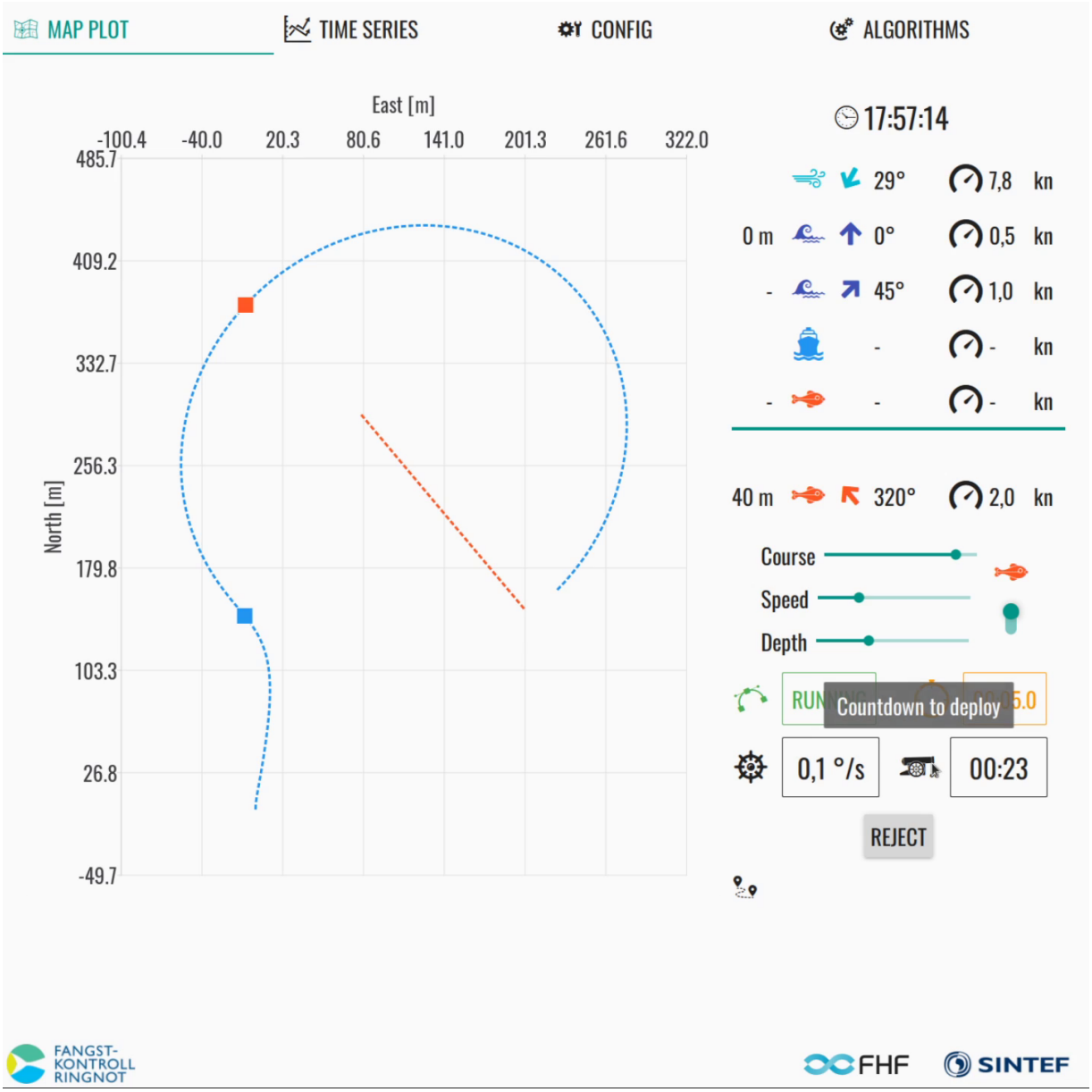


Figure 2.4: Step 4: Dashboard with kept setting trajectory.

CHAPTER

THREE

DATA SHARING

Communication of signals and data between software systems, which is indicated in Figure 3.1, is achieved using a data distribution service standard OMG DDS. In the following, we provide an overview of the various signals. This overview also includes the interface definition language (IDL) types used for the different signals. The IDL types are defined in the companion project `RatatoskIDL`.

3.1 Signal overview

The diagram in Figure 3.1 only give a high-level view of the data sharing between the different systems. With focus on the two algorithm systems *Mimir: Leadline* and *Mimir: Path planner*, we provide a more complete signal list in Table 3.1. *Source* is a signal originator, it is the system that sends out the signal. *Target* is an intended signal receiver. The DCPS DDS interface (data-centric publish-subscribe) is designed to allow multiple recipients – one-to-many. *Causality* is one of *input*, *output*, *parameter* and describes the variable type. In our case, the *variability* for *parameter* in the table is always tunable, that is, they can be changed while algorithms are running. Please consult the documentation for the respective system/algorithm on how to specify the DDS topic and other settings in their configuration files. In the case of *Balder*, the DDS topic names are specified in `FkinDdsTopics`. Note that *Name* in the table below is not the topic name of a signal. It is the variable named used in the *Mimir* configuration file.

Table 3.1: Overview of signal flow between systems.

Name	Causality	IDL type	(S)ource/(T)arget
Mimir: Leadline			
`command`	input	`fkin::Command`	S: Balder
`command`	output	`fkin::CommandResponse`	T: Balder
`notifier`	output	`fkin::ProcessStateAutomaton`	T: Balder
`parameters`	parameter	`fkin::IdVec2d`	S: Balder,
			Mimir: Path planner
`depth`	output	`fkin::BatchIdVec1d`	T: Balder
Mimir: Path planner			
`command`	input	`fkin::Command`	S: Balder
`command`	output	`fkin::CommandResponse`	T: Balder
`notifier`	output	`fkin::ProcessStateAutomaton`	T: Balder
`setting_speed_U_v`	parameter	`fkin::IdVec1d`	S: Balder
`setting_radii`	parameter	`fkin::IdVec2d`	S: Balder
`aim_distance_D_s`	parameter	`fkin::IdVec1d`	S: Balder
`fish_margin_d_f`	parameter	`fkin::IdVec1d`	S: Balder

continues on next page

Table 3.1 – continued from previous page

Name	Causality	IDL type	(S)ource/(T)arget
sink_margin_z_min	parameter	fkin::IdVec1d	S: Balder
leadline_tau_ll_z_d	parameter	fkin::IdVec2d	S: Balder
current_surface	parameter	ratatosk::types::Double2	S: Ratatosk
current_fish	parameter	ratatosk::types::Double2	S: Ratatosk
fish_velocity_over_ground	parameter	ratatosk::types::Double2	S: Ratatosk, Balder[1]
fish_depth_z_s	parameter	fkin::IdVec1d	S: Ratatosk, Balder
GPS_origin	input	ratatosk::types::Double2	S: Balder[2]
vessel_pos_info	input	ratatosk::types::PosInfo	S: Ratatosk
vessel_gyro_info	input	ratatosk::types::PosInfo	S: Ratatosk
fish_pos_info	input	ratatosk::types::PosInfo	S: Ratatosk
fish_relative_pos	input	ratatosk::types::Double3	S: Ratatosk
keep_solution	input	fkin::Bit	S: Balder
trajectory_vessel	output	fkin::BatchKinematics2D	T: Balder
trajectory_vessel_rot	output	fkin::BatchIdVec1D	T: Balder
trajectory_fish	output	fkin::BatchKinematics2D	T: Balder
nlp_config	output	fkin::NlpConfig	T: Balder
nlp_stats	output	fkin::OptiStats	T: Balder
vessel_speed	output	ratatosk::types::DoubleVal	T: Balder
vessel_course_rate	output	ratatosk::types::DoubleVal	T: Balder
deploy_position	output	ratatosk::types::Double2	T: Balder
collide_position	output	ratatosk::types::Double2	T: Balder
deploy_time	output	ratatosk::types::DoubleVal	T: Balder

Note: Balder currently does not support user configuration of topic names. Future improvements include moving `FkinDdsTopics` outside the compiled program, so that it can be loaded at runtime. This would allow configuration of topics without recompilation.

[1] The user can set manually.
[2] Calculated from `vessel_pos_info`.

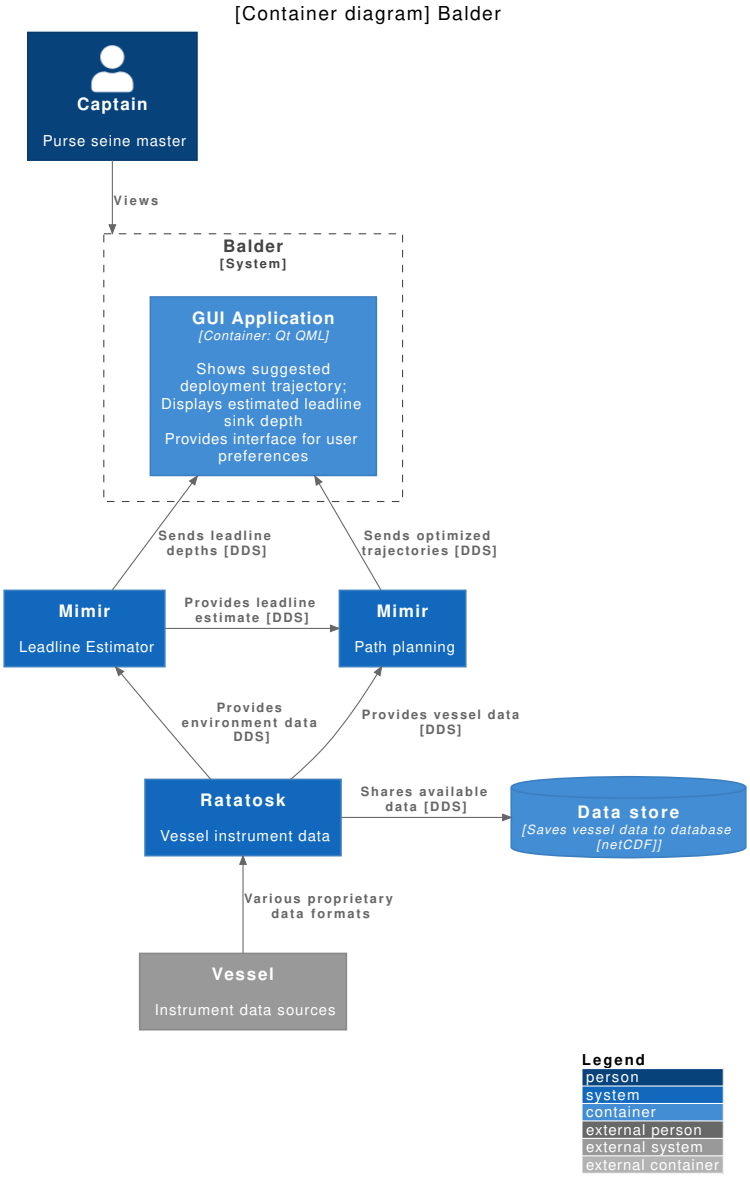


Figure 3.1: Container diagram for Balder.

CHAPTER

FOUR

BUILD INSTRUCTIONS

You need a compiler that supports *c++17* to build `balder`. It is known to compile with `gcc 8`, `clang 9`, and `msvc++ 14.2`. You need the CMake build system generator. The application requires `sinspekto` and Qt.

The **recommended** approach to build the application is with the help of conan. `conan` is a python tool, and once installed, you need to set the following conan remotes.

```
python -m pip install conan
conan remote add sintef https://conan.sintef.io/public
# or
conan remote add sintef https://artifactory.smd.sintef.no/api/conan/conan-local

conan remote add bincrafters https://api.bintray.com/conan/bincrafters/public-conan
conan config set general.revisions_enabled=1
```

Building the documentation is *optional*. It is generated using `doxygen`, `sphinx`, `emacs`, and `plantuml`. You also need the following python modules (`docs/requirements.txt`):

```
breathe
Sphinx >= 4.0.0
doxyqml
exhale
sphinx-rtd-theme
sphinxcontrib-plantuml
sphinxcontrib-svg2pdfconverter
doc2dash
```

Instructions for each platform is found below.

4.1 Configuration options and targets

The table below specifies available options when building the application.

CMake Option	conan option	Default	Comment
See comment	`with_fPIC`	True	`CMAKE_POSITION_INDEPENDENT_CODE=ON`
`WITH_DOC`	`with_doc`	False	Use `cmake --build . --target doc`
`WITH_INSTALL`	`with_install`	False	Deploy APK to connected device (android)
`WITH_CONSOLE`	`with_console`	False	Terminal window side-by-side GUI (windows)

The following build targets are available:

- `doc` will build documentation html.
- `package_it` will create *.deb* (linux), *.exe* (windows), and *.tar.gz* (not android)
- `appimage` (linux) will create an *.AppImage* executable.

If the executable is built without `conan`, the Debian packages will contain dependencies to system packages (and `sinspekto-runtime`, which is non-standard).

- An additional option `WITH_API_DOC=OFF` disables API documentation, and enables LaTeX and PDF output of the user documentation. This currently requires linux OS, `inkscape`, `latexmk`, and a LaTeX distribution to be installed. The output will be available in `<build_folder>/docs/sphinx/latex/`.

4.2 Linux

4.2.1 Prerequisites (debian-based)

These instructions assume a `gcc` compiler and using `conan`.

```
apt-get install -y build-essential cmake pkg-config python3-pip
python -m pip install setuptools wheel conan
```

Note: There may be other (unknown) packages needed by transitive dependencies, which are not installed by conan. You may either install them manually when build errors occur, or you can set environment variable CONAN SYSREQUIRES MODE=enabled, which lets conan automatically install them for you.

4.2.2 Documentation prerequisites *(optional)*

```
apt-get install -y doxygen emacs-nox graphviz plantuml wget
python -m pip install -r docs/requirements.txt --upgrade
emacs -Q --batch -l docs/emacs-install-pkgs.el
```

If your distribution is a bit old, you may have to update `plantuml`.

```
wget https://sourceforge.net/projects/plantuml/files/plantuml.jar
mv plantuml.jar /usr/share/plantuml/
```

4.2.3 Building and running (debian-based)

To install dependencies and build the application you can run the following commands:

```
mkdir build && cd build
conan install .. \
  --options balder:with_doc=True \
  --build missing \
  --settings compiler.libcxx=libstdc++11
conan build ..
```

You need to activate virtual environments to run the built application.

```
. activate.sh     # Opensplice environment variables
. activate_run.sh # Dynamic libraries added to LD_LIBRARY_PATH
bin/balder
```

4.2.4 Packaging into artifacts

The project builds into various artifacts on Linux.

- *.AppImage* executable, build target `appimage`, only for `x86_64/amd64` architecture.
- *.deb* package, build target `package_it`.
- *.tar.gz* archive, build target `package_it`.
- `conan package ..` will build all supported targets.

Warning: When building with conan, the *.deb* package will not be built with system package dependencies. This is because the system packages likely are not the versions as the built executable expects. Also, the package is not bundled with the necessary libraries and resources in order to run.

The AppImage packaging uses linuxdeploy with linuxdeploy-plugin-qt to create an executable AppImage. The deployment tools are downloaded by CMake automatically. The packaging creates an *balder-<version>-<arch>.AppImage*. ImageMagick is needed and the virtual run environment must be active, since the dynamic libraries must be available on `LD_LIBRARY_PATH`, and also conan-installed binary `qmake` when running linuxdeploy.

```
apt-get install -y imagemagick
cd build
. activate_run.sh
conan package ..
```

- Instead of `conan package ..`, it is possible to build only a selected target with `. activate.sh && cmake --build . --target appimage`, or `package_it`.
- By default, the application loads a bundled config file. It can be overridden by setting the environment variable: `export OSPL_URI=file:///path/to/ospl.xml`.
- To run an AppImage, `fuse` may be required, alternatively, run the application with the following flag: `./balder-1.0.0_x86_64.AppImage --appimage-extract-and-run`.
- Internally, `conan package` calls `cmake --build . --target appimage` and will create the *AppImage*, which should contain almost all dependencies for running the application. The file can be run as a desktop application or from the command line.
- The contents of `AppDir` in the build directory is packaged in the *AppImage* file.
- `AppRun` is the entry script of the application.
- To "Install" an AppImage, perhaps AppImageLauncher is useful.
- Need to build on "oldest" distro you want to support to ensure standard library compliance.
- The only supported architecture is `x86_64/amd64`, because the used linuxdeploy appimages are not pre-built for arm-based CPUs.
- When building in a container, set environment variable `APPIMAGE_EXTRACT_AND_RUN=1`
- It is possible to use `--appimage-extract` and then create a symlink on `PATH` to the extracted `AppRun`.

4.3 Windows

4.3.1 Prerequisites

Prerequisites using `conan`. Most commands expect you to run with elevated privileges. We make use of chocolatey package manager for windows:

```
powershell -Command Set-ExecutionPolicy Bypass -Scope Process -Force; \
  [System.Net.ServicePointManager]::SecurityProtocol = \
  [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; \
  iex ((New-Object System.Net.WebClient).DownloadString( \
  'https://chocolatey.org/install.ps1'))
```

```
choco install -y python3 Wget
choco install -y cmake --installargs '"ADD_CMAKE_TO_PATH=System"'
choco install -y git.install --params "/GitAndUnixToolsOnPath"
python -m pip install setuptools wheel conan win-unicode-console
```

Microsoft Visual Studio build tools if they are not already installed. These commands must to be run with `cmd.exe`. **Note** that the `vs_buildtools.exe` will run in the background. These steps are not necessary if you have Visual Studio with C++ compilers installed.

```
mkdir C:\TEMP && cd C:\TEMP
wget https://aka.ms/vs/16/release/vs_buildtools.exe
vs_buildtools.exe --quiet --norestart --wait --nocache \
  --installPath C:\BuildTools \
  --add Microsoft.VisualStudio.Workload.MSBuildTools \
  --add Microsoft.VisualStudio.Workload.VCTools --includeRecommended
setx path "%path%;C:\BuildTools\Common7\Tools"
```

Tip: You may need to start new command window sessions between commands to load the new PATH variables.

4.3.2 Documentation prerequisites on Windows *(optional)*

```
choco install -y doxygen.install emacs plantuml
choco install -y graphviz.portable --force # Maybe optional: 2.44 broken, downgrades to␣
↪2.38
python -m pip install -r docs/requirements.txt --upgrade
emacs -Q --batch -l docs/emacs-install-pkgs.el
```

4.3.3 Building and running

To install dependencies and build the application you can run the following commands:

```
mkdir build
cd build
conan install .. \
  --options balder:with_doc=True \
  --build missing
conan build ..
```

You need to use a virtual environment to run the application. `activate.bat` sets `OSPL_URI` and `OSPL_HOME` environment variables.

```
activate.bat
cd bin
balder.exe
```

4.3.4 Packaging into installer and archive

The project is set up with packaging into an executable installer (*.exe.*) and an archive (*.tar.gz*) using the build target named `package_it`. The installer is made with NSIS through CPack. NSIS and can be installed with chocolatey:

```
choco install -y nsis
```

The packager uses `windeployqt.exe`, which is bundled with Qt. Create the artifacts with:

```
cd build
conan package ..
# or
activate_run.bat
cmake --build . --config Release --target package_it
```

- By default, the application loads a bundled configuration file `ospl.xml`. It can be overridden by setting the environment variable: `set OSPL_URI=file://C:\\path\\to\\ospl.xml`.
- There is also a batch script `ospl_env.bat`, which can be run before the application in the same session on the command line. It sets `OSPL_URI` and `OSPL_HOME`.

Tip: If the package is installed with docs, `WIN + "Balder Documentation'"` should link to the bundled html documentation.

4.4 Android

Warning: We offer only highly experimental Android support.

Note: The android compilation is only tested on a Linux platform.

You need to have Android SDK and NDK installed. The recipe itself uses it, but if some dependencies need building, environment variables need to be on path. The following command will help in that regard:

```
mkdir build_help && cd build_help
conan install "android-cmdline-tools/[>=6858069]@joakimono/testing" \
  --generator virtualenv \
  --profile=../tools/profiles/android_amd64_api29 \
  --build missing
. activate.sh
cd ..
```

The build process makes use of QtAndroidCMake to create an APK bundle. During the build process, the created *.apk* can be installed directly to a connected Android device with the conan option `with_install=True`, (CMake: `WITH_INSTALL=ON`).

This project are bundled with some pre-made profiles in `tools/profiles` for common architectures. Below we show how to compile for `x86_64` with API level 29 (Android 10):

```
mkdir build && cd build
conan install .. \
  --profile=../tools/profiles/android_amd64_api29 \
  --build missing
conan build ..
```

Warning: The packaging uses a self-signed keystore `data/balder.keystore` for its call to `add_qt_android_apk()` in `CMakeLists.txt`. This should be replaced with your own signing keystore and information updated accordingly in `CMakeLists.txt`. See self signing on how ot create a keystore.

Internally the `conan build ..` is almost equivalent to the following commands:

```
source activate.sh
cmake \
  -DCMAKE_BUILD_TYPE=Release \
  -DCMAKE_TOOLCHAIN_FILE=$CONAN_CMAKE_TOOLCHAIN_FILE \
  -DANDROID_PLATFORM=$ANDROID_PLATFORM \
  -DANDROID_ABI=$ANDROID_ABI \
  -DANDROID_STL=$ANDROID_STL \
  -DCMAKE_FIND_ROOT_PATH_MODE_PACKAGE=ON \
  ..
cmake --build . --parallel 4
```

4.5 Troubleshooting

- If the AppImage fails with `.. symbol lookup error ... version Qt_5_PRIVATE_API`, you forgot `. activate_run.sh` before `conan package ..`, delete the build folder and try again.
- If there is an issue compiling qt, similar to this. Solution was to `unset CPATH`
- `. deactivate.sh` does not properly unset `OSPL_HOME`, which may cause issues, use `unset OSPL_HOME`

SINTEF

Teknologi for et bedre samfunn