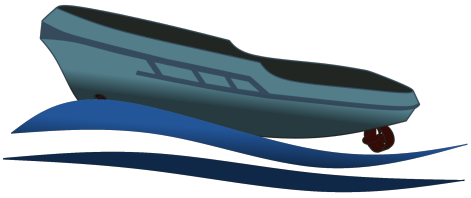
Teknisk rapport

Oscar Hermansson, Oskar Jonsson, Jonatan Siönäs,
Axel Ståhlbom, Johannes Wenngren, Tim Wiik

20 december 2021

Version 1.0



Status

Granskad	Samtliga	2021-12-16
Godkänd		

Projektidentitet

Beställare: Fredrik Ljungberg, Linköpings universitet
E-post: fredrik.ljungberg@liu.se

Kund: Jonas Linder, ABB Corporate Research
E-post: jonas.x.linder@se.abb.com

Handledare: Gustav Zetterqvist
E-post: gustav.zetterqvist@liu.se

Kursansvarig: Daniel Axehill och Gustaf Hendeby
E-post: daniel.axehill@liu.se, gustaf.hendeby@liu.se

Projektdeltagare

Namn	**Ansvar**	**E-post**
Oscar Hermansson	Dokumentansvarig	osche354@student.liu.se
Oskar Jonsson	Hårdvaruansvarig	oskjo275@student.liu.se
Jonatan Siönäs	Designansvarig	jonsi043@student.liu.se
Axel Ståhlbom	Mjukvaruansvarig	axest992@student.liu.se
Johannes Wenngren	Projektledare	johwe475@student.liu.se
Tim Wiik	Testansvarig	timwi572@student.liu.se

INNEHÅLL

DOKUMENTHISTORIK

Version	Datum	Utförda förändringar	Utförda av	Granskad
0.1	2021-012-06	Första utkast	Samtliga	Samtliga
0.2	2021-12-08	Andra utkast	Samtliga	Samtliga
0.3	2021-12-14	Tredje utkast	Samtliga	Samtliga
1.0	2021-12-16	Första revision	Samtliga	Samtliga

1 INLEDNING

Denna tekniska rapport dokumenterar projektet *Modellbaserad styrning av småskalig ytfarkost* i kursen TSRT10. De sex gruppmedlemmarna läser sista året på antingen civilingejörsprogrammet Teknisk fysik och elektroteknik eller Maskinteknik.

Detta dokument ämnar underlätta arbetet för nästa projektgrupp genom att förklara vad som gjorts i projektet. Detta inkluderar främst hur olika system implementerats, vilken teori som ligger bakom och varför vissa val har gjorts. I slutet av dokumentet lyfts tekniska framgångar, existerande problem och förbättringsförslag för att tydliggöra vad som är implementerat och vad som kan behöva implementeras eller undersökas.

1.1 Bakgrund

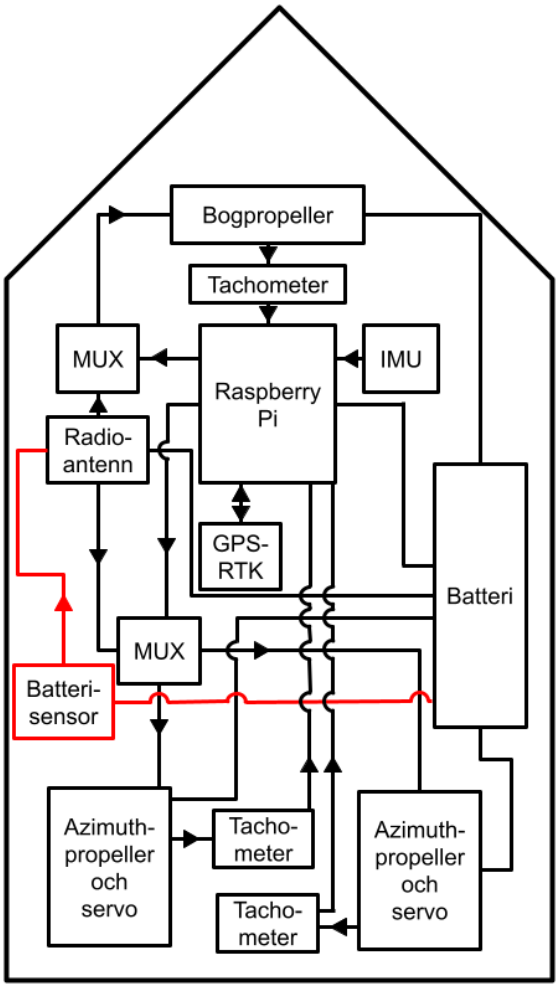
Farkosten och modellen som användes i projektet togs fram av Gustav Zetterqvist och Fabian Steen i deras examensarbete [1] och projektets mål var att vidareutveckla denna till en helt autonomt styrd ytfarkost.

1.2 Syfte

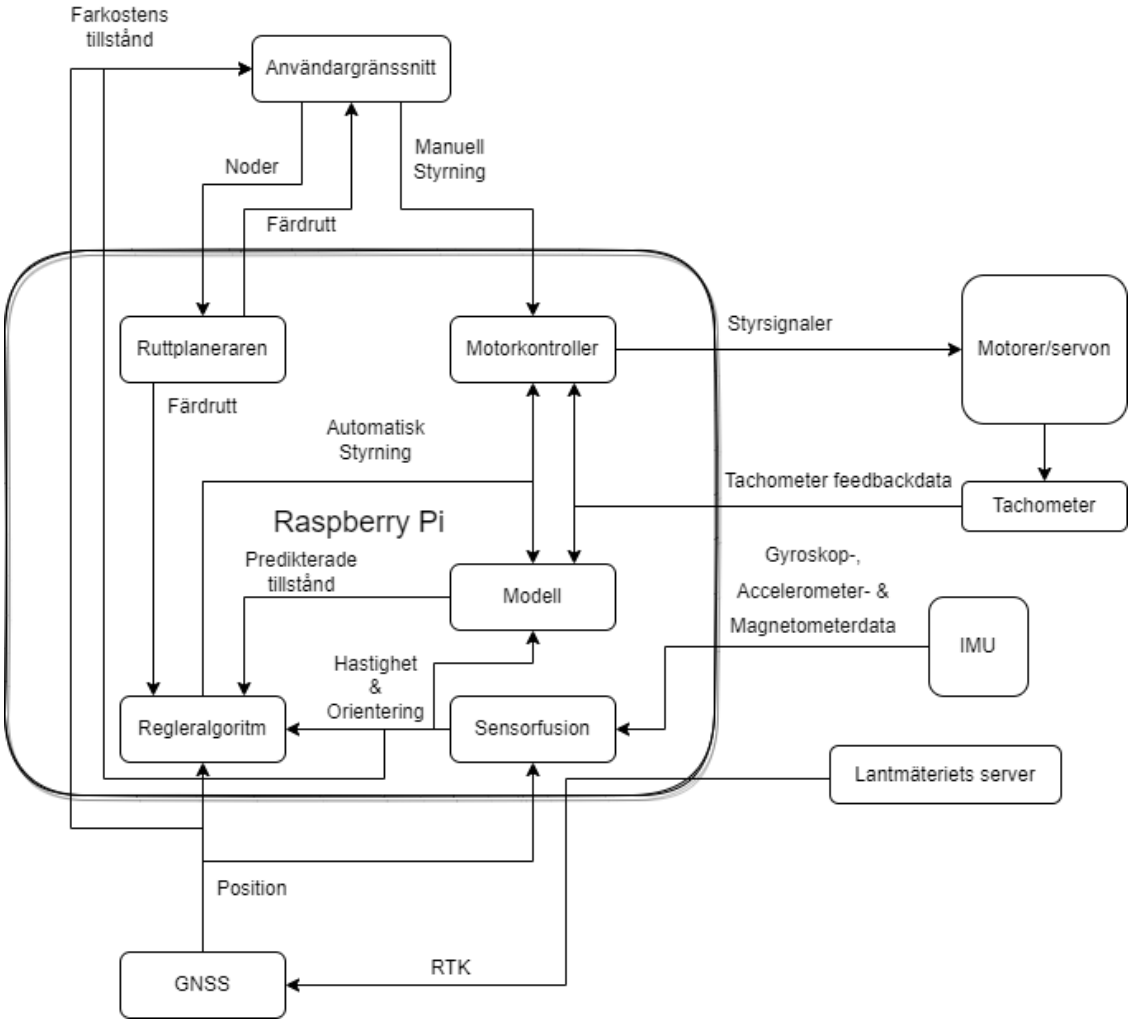
Syftet med projektet var att förenkla utvecklingen av nya autonoma fartyg och att kunna styra fartyg autonomt med önskad precision.

2 SYSTEMET

En översiktlig systemskiss för de komponenter som finns på farkosten visas i Figur 1 och en systemskiss för hur systemets olika moduler kommunicerar med varandra och vad det är för information som de olika modulerna skickar visas i Figur 2.



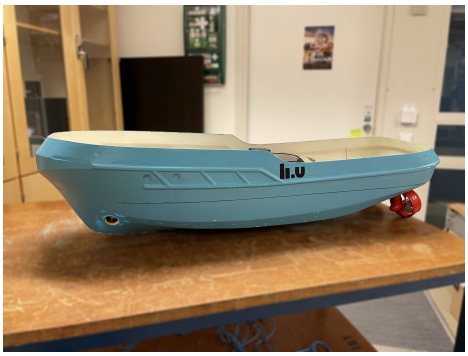
Figur 1: Systemskiss farkostens komponenter



Figur 2: Flödesschema för systemets olika moduler.

2.1 Hårdvara

Bilder på farkostens utseende kan ses i Figur 3, dessa har hämtats ur Zetterqvist och Steens examensarbete [1]. Nedan följer text, tabeller och bilder som ämnar beskriva dimensionen på farkosten, hårdvaran och in-/utportarna på den ombordplacerade datorn.

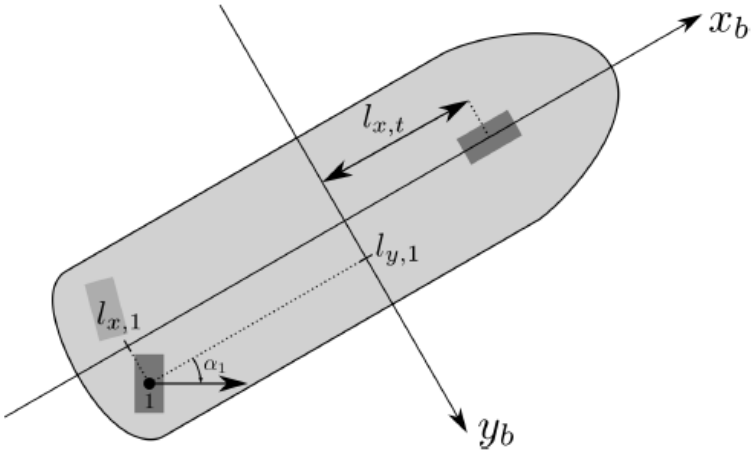



Figur 3: Bild av systemet

2.1.1 *Komponenter och dimensioner*

$l_{x,1}$, $l_{y,1}$ och $l_{x,t}$ från Tabell 1 beskriver propellernas position med hänsyn till origo i koordinatsystemet x_b och y_b, placerad i båtens mitt enligt Figur 7. Skeppets dimension och vissa komponenters placering kan ses i Tabell 1 och i Figur 1. Farkostens komponenter är:

- Två stycken roterbara roderproppellrar och tillhörande servon av typen ”DOMAN Digital Servom RS1513MD”. Roderpropellrarna drivs av varsin borstlös DC-motor av typen ”G-Power BL-Motor, type 3536”.
- En bogpropeller driven av en borstlös motor av typen ”G-Power BL-Motor, type 3536”.
- En GNSS-mottagare med stöd för RTK (Real Time Kinematic). Modulen är en u-blox ZED-F9P och har en extern antenn av typ MagmaX2.
- En IMU (Inertial Measurement Unit) av typen MPU9250 för mätning av vinkelhastighet, acceleration och magnetfält.
- Tachometer av typen ”Eagle Tree, Brushless motor rpm sensor v2” för mätning av propellrarnas rotationshastighet.
- En X8R - FrSky RC-mottagare för användning vid manuell styrning av skeppet.
- En Raspberry Pi 4 för beräkningar.
- Flera litiumjonbatterier med spänningen på 11.1 V och kapacitet på 2200 mAh (Turnigy 2200mAh 3S 25C LiPo Pack), varav endast ett åt gången kopplas in och strömförsörjer båten.
- Propellermotorerna är kopplade till varsin ESC ("Electronic Speed Controller"eller elektrisk hastighetsregulator på svenska) av typen Graupner HoTT +T 35 BEC.
- En spänningsregulator som genererar 5.0 volt (och max 5 Ampere) bland annat till Raspberry Pi:en.
- En spänningsregulator som genererar 7.0 volt (och max 20 Ampere) till servomotorerna.
- En FrSky lipo-sensor för övervakning av batterinivån i farkosten.



Figur 4: Beskrivning över aktuatorernas placering och rotation

Tabell 1: Tabell för farkostens fysiska egenskaper.

Komponent	Specifikationer
Skrovlängd	0.99 m
Skrovbredd	0.30 m
Roderpropeller, styrbord	$l_{x,1} = -0.39$ m, $l_{y,1} = 0.07$ m
Roderpropeller, barbord	$l_{x,2} = -0.39$ m, $l_{y,2} = -0.07$ m
Bogpropeller	$l_{x,t} = 0.37$ m

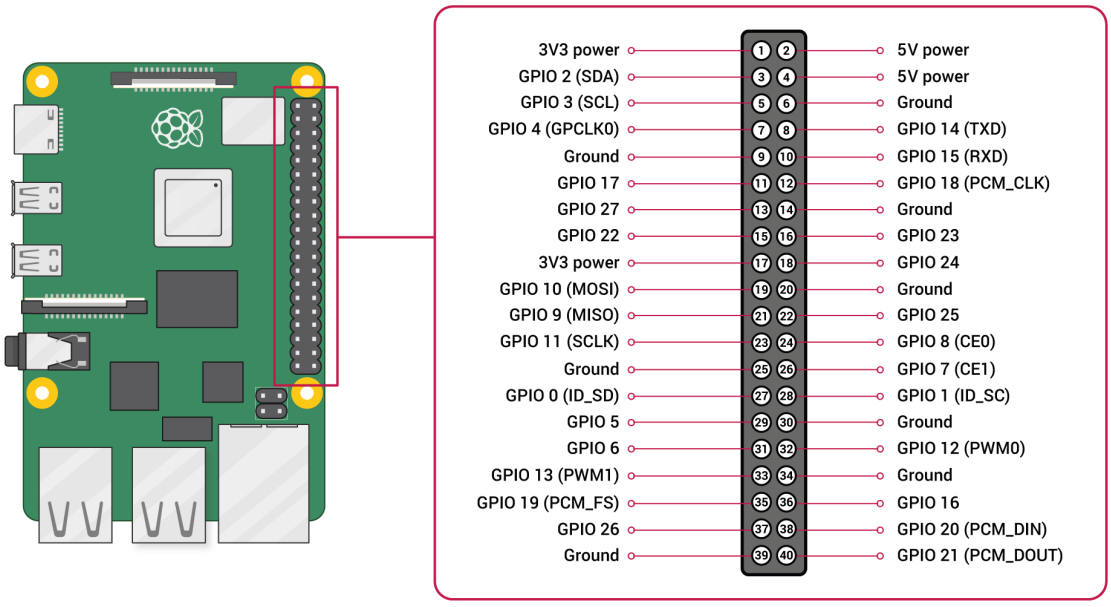
2.1.2 *Raspberry Pi I/O portar*

En Raspberry Pi 4 med operativsystemet Ubuntu användes som plattform för all mjukvara ombord och ingående moduler. På Raspberry Pi:en är modellen och reglersystemet implementerade och där genomförs alla beräkningar. All kod implementerad på Raspberry Pi:en är skriven i Python 3. Vid körning kan styrsignaler och mätdata från sensorer lagras på Raspberry Pi:en.

I Tabell 2 och Figur 5 redovisas alla anslutningar till Raspberry Pi:ens in- och utgångar. Jord (GND) ansluts mellan varje modul som kommunicerar med Raspberry Pi:en. Notera även att ESC betyder ”Electronic Speed Controller” eller elektrisk hastighetsregulator på svenska.

Tabell 2: Raspberry pi anslutningar

Raspberry pi pin	Ansluten till	Kommentar
GPIO 2 (SDA)	IMU SDA	I2C-data från IMU
GPIO 3 (SCL)	IMU SCL	I2C-klocka till IMU
GPIO 0	GNSS-mottagare RX	UART-data från Raspberry Pi till GNSS-mottagare
GPIO 1	GNSS-mottagare TX	UART-data från GPS till Raspberry Pi
GPIO 14	GNSS-mottagare RXD2	RTCM korrektionsdata från Raspberry Pi till GNSS-mottagaren
GPIO 15	GNSS-mottagare TXD2	RTCM korrektionsdata från GNSS-mottagaren till Raspberry Pi
GPIO 17	Tachometer 1	Läser av tachometer till roderpropeller nummer 1
GPIO 27	Tachometer 2	Läser av tachometer till roderpropeller nummer 2
GPIO 22	Tachometer T	Läser av tachometer till bogpropellern
GPIO 16	ESC till roderpropeller numer 1	Styr hastigheten på roderpropeller 1
GPIO 20	ESC till roderpropeller numer 2	Styr hastigheten på roderpropeller 2
GPIO 21	ESC till bogpropeller	Styra hastigheten på bogpropellern
GPIO 19	Servo till roderpropeller nummer 1	Styr vinkel på roderpropeller 1
GPIO 26	Servo till roderpropeller nummer 2	Styr vinkel på roderpropeller 2



Figur 5: Schema över Raspberry Pi 4b:s pin-positioner.

3 MODELLERING

Modellen som används i farkosten är baserad på den modell som har tagits fram av Zetterqvist och Steen i deras examens arbete [1] där vindkraften har försummats.

Först presenters generell teori för modelleringen som sedan följs av underrubriker med djupare förklaringar för modellens olika delar. Ekvationer och definitioner är hämtade från [1] och [2].

3.1 Teori

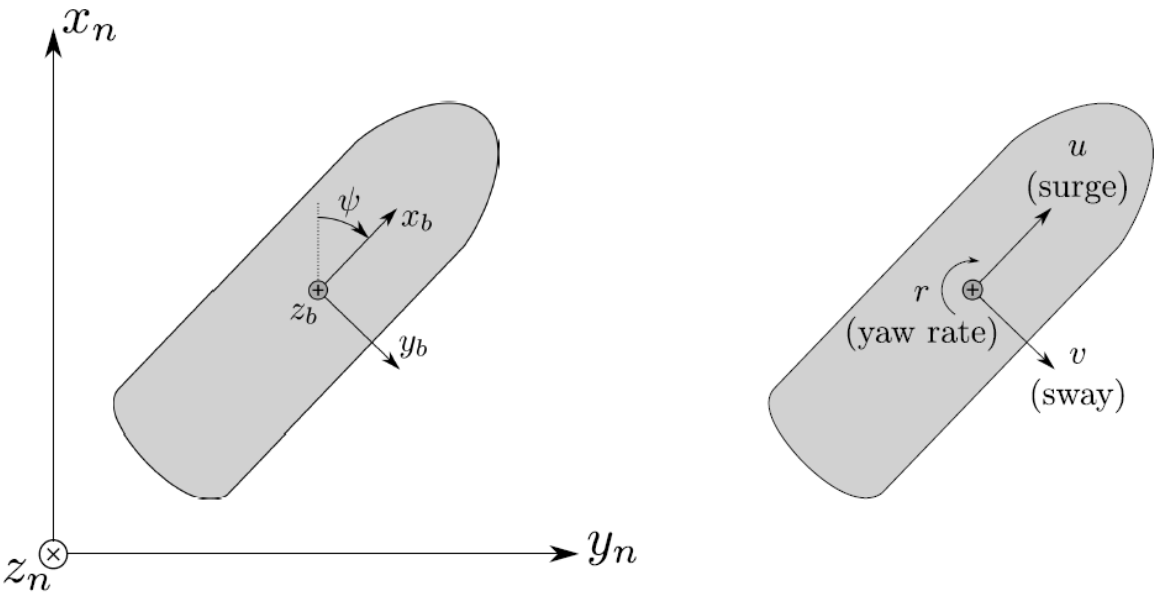
För att beskriva farkostens position och rörelser används två referensplan, ett statiskt $\boldsymbol{n}$-frame (globalt koordinatsystem) och ett som utgår från fartyget, $\boldsymbol{b}$-frame (farkostens koordinatsystem) som syns definierade i Figur 6. Med hjälp av dessa kan följande notation användas.

$$\boldsymbol{\eta} = \begin{bmatrix} x_\eta & y_\eta & \psi \end{bmatrix}^T \tag{1}$$

$$\boldsymbol{\nu} = \begin{bmatrix} u & v & r \end{bmatrix}^T \tag{2}$$

Där $\boldsymbol{\eta}$ och $\boldsymbol{\nu}$ är tillstånden i den tillståndsmodell som har använts och kan ses i Figur 6. Därtill definieras externa krafter enligt

$$\boldsymbol{\tau} = \begin{bmatrix} \tau_x & \tau_y & \tau_N \end{bmatrix}^T \tag{3}$$



Figur 6: Beskrivning över det lokala koordinatsystemet

3.1.1 *Kinematik*

Kinematiken för modellen av farkosten definieras enligt

$$\dot{\boldsymbol{\eta}} = \boldsymbol{R}(\psi)\boldsymbol{\nu} \tag{4}$$

där $\boldsymbol{\eta}$ farkostens position, $\boldsymbol{\nu}$ är hastigheten i farkostens referensplan och $\boldsymbol{R}$ är rotationsmatrisen enligt

$$\boldsymbol{R}(\psi) = \begin{bmatrix} cos\psi & -sin\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{5}$$

3.1.2 *Kinetik*

Kinetiken används för att beskriva hur farkosten påverkas av externa krafter och definieras enligt

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} = \boldsymbol{\tau} \tag{6}$$

där $\boldsymbol{M}$, $\boldsymbol{C}$ och $\boldsymbol{D}$ matriser för tröghetsmomentet, corioliskrafter och dämpningen av farkosten. $\boldsymbol{\tau}$ är de externa krafterna som verkar på farkosten. $\boldsymbol{M}$ och $\boldsymbol{C}$ har delats upp i en stelkroppsdel och en del för hydrodynamiska effekter enligt

$$\boldsymbol{M} = \boldsymbol{M}_{SK} + \boldsymbol{M}_{HD} \tag{7a}$$

$$\boldsymbol{C}(\boldsymbol{\nu}) = \boldsymbol{C}_{SK}(\boldsymbol{\nu}) + \boldsymbol{C}_{HD}(\boldsymbol{\nu}). \tag{7b}$$

Då båten är symmetrisk längs z_b-x_b-planet, som ses i Figur 6, kan *surge* frånkopplas från *sway* och *yaw* och då farkostens tyngdpunkt antas vara samma som båtens geometriska mittpunkt kan $\boldsymbol{M}$ beskrivas som

$$\boldsymbol{M} := \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I_z \end{bmatrix} + \begin{bmatrix} -X_{\dot{u}} & 0 & 0 \\ 0 & -Y_{\dot{v}} & 0 \\ 0 & 0 & -N_{\dot{r}} \end{bmatrix} = \begin{bmatrix} m_{11} & 0 & 0 \\ 0 & m_{22} & 0 \\ 0 & 0 & m_{33} \end{bmatrix}. \tag{8}$$

Här är m massan, I_z tröghetsmoment kring z-axeln och parametrarna $X_{\dot{u}}$, $Y_{\dot{v}}$ och $N_{\dot{r}}$ är hydrodynamiska effekter. Här är också m_{ii} en omskrivning av de ingående termerna för att underlätta notation.

$\boldsymbol{C}(\boldsymbol{\nu})$ är en matris som kan beskrivas som en funktion av $\boldsymbol{M}$ och $\boldsymbol{v}$ enligt

$$\boldsymbol{C}(\boldsymbol{\nu}) := \begin{bmatrix} 0 & 0 & -mv \\ 0 & 0 & mu \\ mv & -mu & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & Y_{\dot{v}}v \\ 0 & 0 & -X_{\dot{u}}u \\ -Y_{\dot{v}}v & X_{\dot{u}}u & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & -m_{22}v \\ 0 & 0 & m_{11}u \\ m_{22}v & -m_{11}u & 0 \end{bmatrix}. \tag{9}$$

Den hydrodynamiska dämpningen beskrivs som

$$\boldsymbol{D}(\boldsymbol{\nu}) := \begin{bmatrix} d_{11}(\boldsymbol{\nu}) & 0 & 0 \\ 0 & d_{22}(\boldsymbol{\nu}) & d_{23}(\boldsymbol{\nu}) \\ 0 & d_{32}(\boldsymbol{\nu}) & d_{33}(\boldsymbol{\nu}) \end{bmatrix} \tag{10}$$

$$d_{11}(\boldsymbol{\nu}) = X_u - X_{|u|u}|u| \tag{11a}$$
$$d_{22}(\boldsymbol{\nu}) = Y_v - Y_{|u|v}|u| - Y_{|v|v}|v| - Y_{|r|v}|r| \tag{11b}$$
$$d_{23}(\boldsymbol{\nu}) = Y_r \tag{11c}$$
$$d_{32}(\boldsymbol{\nu}) = N_v - N_{|v|v}|v| \tag{11d}$$
$$d_{33}(\boldsymbol{\nu}) = N_r - N_{|r|r}|r| \tag{11e}$$

Värdena på parametrarna för att beräkna $d_{11}(\boldsymbol{\nu})$-$d_{33}(\boldsymbol{\nu})$ har tagits fram experimentellt av Zetterqvist och Steen [1] vilket kommer användas som initiala gissningar för skattningen. För att kunna garantera stabilitet hos farkosten så linjäriseras $\boldsymbol{C}(\boldsymbol{\nu})$ och $\boldsymbol{D}(\boldsymbol{\nu})$ för en konstant *surge*-hastighet vilket leder till

$$\begin{bmatrix} m_{11} & 0 & 0 \\ 0 & m_{22} & 0 \\ 0 & 0 & m_{33} \end{bmatrix} \dot{\boldsymbol{\nu}} + \begin{bmatrix} X_u & 0 & 0 \\ 0 & Y_v - Y_{|u|v}|U_0| & Y_r + m_{11}U_0 \\ 0 & N_v - (m_{11} - m_{22})U_0 & N_r \end{bmatrix} \boldsymbol{\nu} = \boldsymbol{\tau}. \tag{12}$$

För att det här linjära systemet ska vara stabilt krävs att den högra matrisen behöver vara positivt definit vilket leder till

$$X_u > 0 \tag{13a}$$
$$Y_v > 0 \tag{13b}$$
$$N_r > 0 \tag{13c}$$
$$(Y_v - Y_{|u|v}|U_0|)N_r - (m_{11}U_0 + Y_r)(N_v - (m_{11} - m_{22})U_0) > 0. \tag{13d}$$

Enligt Zetterqvist och Steen [1] gäller det att om sista ekvationen i (13) är uppfyllt för $U_0 = 0$ samt att den högsta *surge*-hastigheten som kan uppnås är $U_0 = U_{max}$, så kan stabilitet bevisas för alla hastigheter $0 \leq u \leq U_{max}$ om $m_{11} < m_{22}$.

3.1.3 *Propellerkraft*

De krafter som verkar på fartyget under färd definieras enligt

$$\boldsymbol{\tau} = \boldsymbol{\tau}_c + \boldsymbol{\tau}_{env} \tag{14}$$

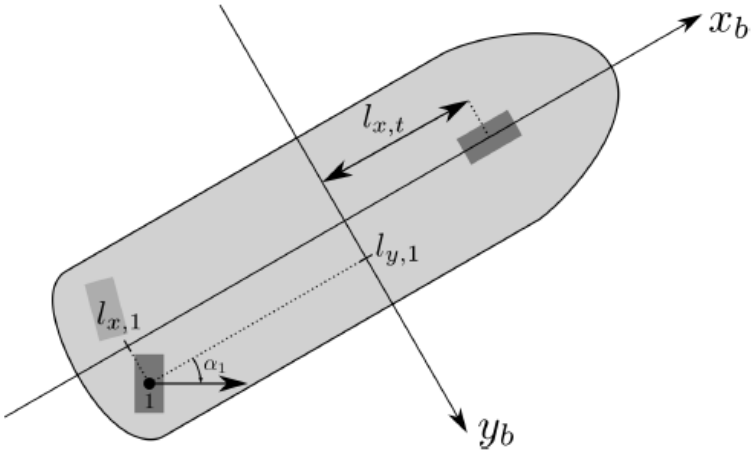
där modellen har förenklats till att endast de krafter som orsakas av proppellrarna påverkar. Alltså är $\boldsymbol{\tau}_{env} = 0$ och kraftvektorn ges då av

$$\boldsymbol{\tau} = \boldsymbol{\tau}_c = \boldsymbol{\tau}_{az} + \boldsymbol{\tau}_t + \boldsymbol{\tau}_r. \tag{15}$$

där index *az* är roderpropellrarna, index *t* är bogpropellern och index *r* är roderkrafterna. Roderpropellrarnas bidrag till kraftvektorn ges av ekvation (16) där μ_{az} är en positiv konstant.

$$\boldsymbol{\tau_{az}} = \mu_{az} \begin{bmatrix} \sum_{i=1}^{2} n_i^2 \, cos(\alpha_i) \\ \sum_{i=1}^{2} n_i^2 \, sin(\alpha_i) \\ \sum_{i=1}^{2} n_i^2 \, (l_{x,i} sin(\alpha_i) - l_{y,i} cos(\alpha_i)) \end{bmatrix} \tag{16}$$

Propellrarnas placering och vinkeldefinitionen α_i presenteras i Figur 7



Figur 7: Beskrivning över aktuatorernas placering och rotation

där α_i är vinkeln på roderpropellrarna och n_i anger varvtalet för respektive propeller. Denna modell förutsätter att propellrarnas placering i $\boldsymbol{b}$-frame är kända, vilket de är (se Tabell 1). Kraften från bogpropellern kan uttryckas som

$$\boldsymbol{\tau}_t = \mu_t n_t^2 \begin{bmatrix} 0 \\ 1 \\ l_{x,t} \end{bmatrix} \tag{17}$$

Farkosten har inget roder men propellrarnas hölje resulterar i en kraft som en liknar roderkraft. Den slutliga roderkraften ser då ut som

$$\boldsymbol{\tau}_r = \begin{bmatrix} \sum_{i=1}^{2} -F_N(\alpha_i, \boldsymbol{\nu}) \sin(\alpha_i) \\ \sum_{i=1}^{2} -F_N(\alpha_i, \boldsymbol{\nu}) \cos(\alpha_i) \\ \sum_{i=1}^{2} F_N(\alpha_i, \boldsymbol{\nu})(l_{x,i} \cos(\alpha_i) + l_{y,i} \sin(\alpha_i)) \end{bmatrix} \tag{18}$$

med

$$F_N(\alpha_i, \boldsymbol{\nu}) = \frac{1}{2} \rho_w A_r F'_N(\alpha_i, \boldsymbol{\nu})(u^2 + v^2) \tag{19}$$

$$F'_N(\alpha_i, \boldsymbol{\nu}) = C_N \sin(\alpha_i - atan2(v, u)) \tag{20}$$

där ρ_w är densiteten på vatten, α är rodervinkeln och C_N en positiv konstant. $l_{x,i}$, $l_{y,i}$ och A_r hittas i Tabell 1.

3.1.4 *Tillståndsmodell*

Med hjälp av ekvationerna i avsnitten ovan kan en tillståndsmodell uppföras enligt

$$\begin{cases} \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\vartheta}) \\ \boldsymbol{y} = \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{\vartheta}) \end{cases} \tag{21}$$

där $\boldsymbol{f}$ och $\boldsymbol{h}$ ges enligt

$$\begin{cases} \dot{\boldsymbol{x}} = \begin{bmatrix} cos(\psi)u - sin(\psi)v \\ sin(\psi)u + cos(\psi)v \\ r \\ \frac{1}{m_{11}}\left(m_{22}vr - X_u u + X_{|u|u}|u|u + \tau_X(\boldsymbol{u},\boldsymbol{\vartheta})\right) \\ \frac{1}{m_{22}}\left(-m_{11}ur - Y_v v - Y_r r + Y_{|u|v}|u|v + Y_{|v|v}|v|v + Y_{|r|v}|r|v + \tau_Y(\boldsymbol{u},\boldsymbol{\vartheta})\right) \\ \frac{1}{m_{33}}\left((m_{11} - m_{22})uv - N_v v + N_{|v|v}|v|v - N_r r + N_{|r|r}|r|r + \tau_N(\boldsymbol{u},\boldsymbol{\vartheta})\right) \end{bmatrix} \\ \boldsymbol{y} = \begin{bmatrix} x & y & \psi & r \end{bmatrix}^T \end{cases} \tag{22}$$

$$\boldsymbol{x} = \begin{bmatrix} x & y & \psi & u & v & r \end{bmatrix}^T \quad \text{Tillstånd} \tag{23}$$

$$\boldsymbol{u} = \begin{bmatrix} \alpha_1 & \alpha_2 & n_1 & n_2 & n_t \end{bmatrix}^T \quad \text{Insignal} \tag{24}$$

$$\boldsymbol{y} = \begin{bmatrix} x & y & \psi & r \end{bmatrix}^T \quad \text{Utsignal} \tag{25}$$

$$\begin{aligned} \boldsymbol{\vartheta} = \begin{bmatrix} m_{11} & m_{22} & m_{33} & X_u & X_{|u|u} & Y_v & Y_r & Y_{|u|v} \ldots \\ \ldots Y_{|v|v} & Y_{|r|v} & N_v & N_r & N_{|v|v} & N_{|r|r} & C_N & \mu_{az} & \mu_t \end{bmatrix}^T. \end{aligned} \quad \text{Okända parametrar} \tag{26}$$

Där $\boldsymbol{\vartheta}$ är den vektor med okända modellparametrar som ska skattas.

3.2 Parameterskattning

Skattningen av parametrarna gjordes med funktionen *idnlgreyest* som finns i MATLAB:s *system identification toolbox*. *Grey-box*-modellen, som är parametriserad i ϑ har definierats som ett *idnlgrey*-objekt som sedan *idnlgreyest* skattade parametervärdena av genom att minimera en värdefunktion beroende på prediktionsfelet. De skattade parametrarna gavs då som

$$\hat{\boldsymbol{\vartheta}}_N = \arg\min_{\boldsymbol{\vartheta}} V_N(\boldsymbol{\vartheta}) \tag{27}$$

där $\hat{\boldsymbol{\vartheta}}_N$ är de parametrar som minimerar värdefunktionen $V_N(\boldsymbol{\vartheta})$ med N stycken mätvärden. För denna typ av skattning behövs oftast bra initiala gissningar på parametervärdena. Först användes parametervärdena från [1] som initial gissningar för att ta fram parametervärdena som syns i Tabell 3. Dessa parametervärden användes slutligen som gissningar för att få fram parametervärdena som syns i Tabell 4. De datainsamlingsexperiment som utförts har designats med målet att excitera alla intressanta moder i systemet. Detta för att göra den resulterande modellen så allmängiltig som möjligt.

Den skattade modellen validerades sedan mot tre datainsamlingar speciellt framtagna för validering ämnade att testa modellens prestanda på de tre viktigaste situationerna, köra rakt , köra rakt i sidled och köra i loop. Vid implementationen av modellen i MATLAB kunde inte alla bivillkor implementeras och vissa antaganden har gjorts. De bivillkor som implementerats var att X_u, Y_v, N_r och $m_{22} - m_{11}$ skulle vara större än noll. Vid implementationen har m_{22} bytts ut mot $m_{22} - m_{11}$ som skattades vilket har definierats till m_{22d}. I implementationen har också $\mu_{az} = 1.435 \cdot 10^{-6}$ och $\mu_t = 4.84 \cdot 10^{-8}$ antagits vara kända från [1] och skattas inte.

Tabell 3: Initiala värden på parametrarna för parameterskattningen

Initiala parametervärden
$m_{11} = 17.4590$
$m_{22d} = 0.0787$
$m_{33} = 0.5147$
$X_u = 1.0858$
$X_{u\|u\|} = -9.6613$
$Y_v = 0.0002$
$Y_r = -22.9853$
$Y_{u\|v\|} = -36.9627$
$Y_{v\|v\|} = -116.0589$
$Y_{r\|v\|} = 106.4577$
$N_v = -0.0542$
$N_r = 3.2602$
$N_{v\|v\|} = 1.5378$
$N_{r\|r\|} = 0.5897$
$C_N = 0.2$

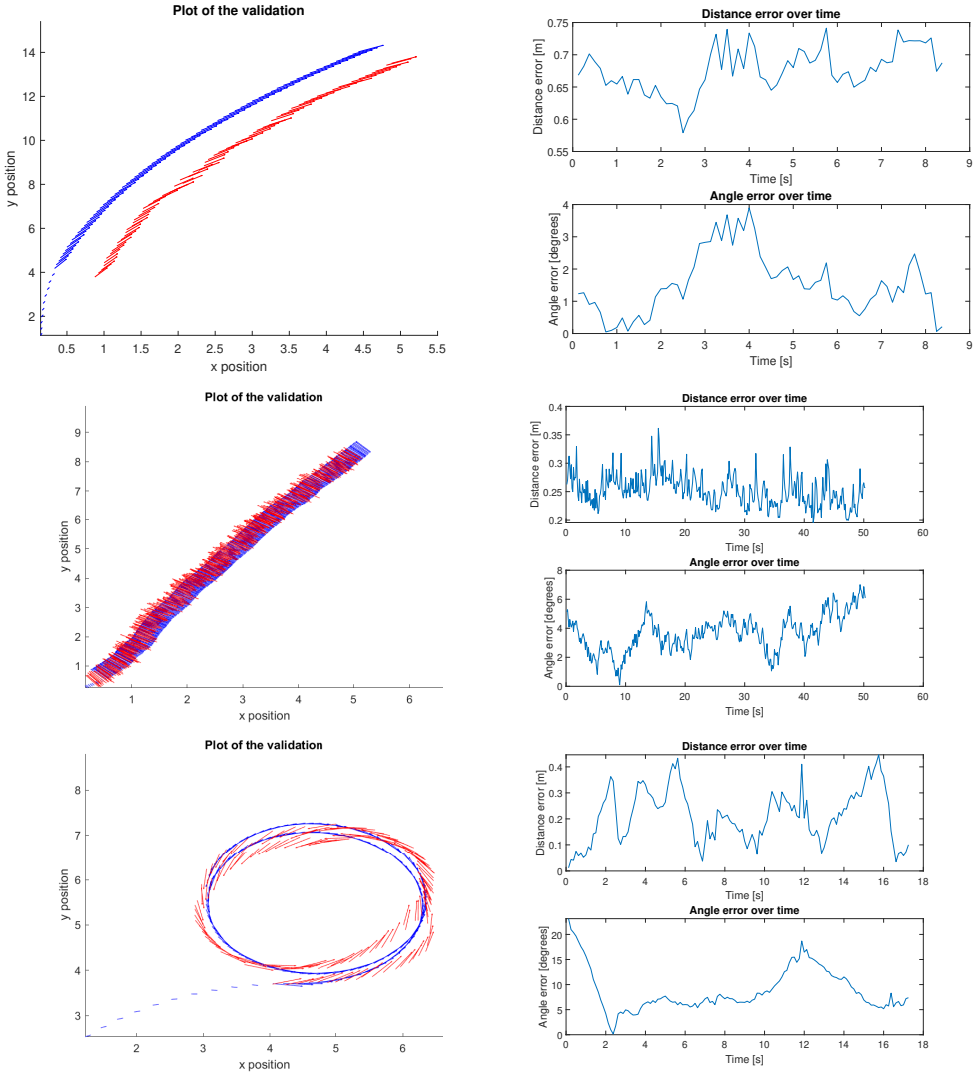
3.2.1 *Datainsamling*

För att kunna estimera en modell behövdes estimeringsdata som samlats in genom olika datainsamlingsexperiment. Datainsamlingarna var flera olika tester som ämnade att excitera en eller flera moder i systemet. Istället för en enda lång datainsamling gjordes flera små som sedan kombinerades vid estimeringen.

Datainsamlingtesten utformades i filer som definierar insignalerna rpm och vinklar för propellrarna, dessa var valda så att de skulle excitera så mycket av farkostens dynamik som möjligt på minimal tid och många olika varianter testades.

3.2.2 *Estimering och validering*

När estimeringsdata finns att tillgå så kan estimeringen göras vilket börjar med att de datainsamlingar som önskas användas för estimering kombineras till en datamängd. Preliminärt används hela modellen men roderkrafterna kan ignoreras ifall skattningen inte konvergerar eller tar för lång tid. När en modell har valts sätts parametrarnas initiala värden. Det är väsentligt att de ansätts väl så att parametrarna konvergerar och beräkningstiden minimeras. Exempel på validering av modellen, med roderkraft kan ses i Figur 8 där en datainsamling har använts för estimering och validerats mot tre valideringsdatainsamlingar för att testa kraven definierade i [3]. I Figur 8 har varje mätt datapunkt använts som initialvärde och farkostens tillstånd, 2.5 sekunder in i framtiden, predikterats. Detta motsvarar MPC:ns planeringshorisont samt hur kraven på modellen kontrolleras. Det här är anledningen till att graferna inte startar i samma punkt och att de första 2.5 sekunderna av den uppmätta datan är streckad. I Tabell 4 visas en tabell med de estimerade parametervärdena.



Figur 8: Validering av modellen mot rakt fram, sidled och looptest med det uppmätta i blått och det modellerade i rött. Strecken i den vänstra bilden visar farkostens orientering

Tabell 4: Estimerade värden på parametrarna

Estimerade parametervärden
$m_{11} = 19.7925$
$m_{22d} = 0.0262$
$m_{33} = 0.0008$
$X_u = 0.6732$
$X_{\|u\|u\|} = -10.4124$
$Y_v = 0.2210$
$Y_r = -23.6081$
$Y_{\|u\|v\|} = -52.1707$
$Y_{\|v\|v\|} = -229.2764$
$Y_{\|r\|v\|} = 134.5180$
$N_v = -0.8268$
$N_r = 2.9871$
$N_{\|v\|v\|} = 1.3526$
$N_{\|r\|r\|} = -1.9027$
$C_N = 1.5308$

4 SENSORFUSION

I farkosten används sensorfusion för att få en skattning av *yaw*-vinkel och hastighet i det globalt definierade koordinatsystemet. Eftersom dessa tre tillstånd är viktiga vid reglering och modellskapandet behövs noggranna skattningar. Dessa skattningar görs via två parallellt körande Kalman-filter.

4.1 Magnetometerkalibrering

För att få värdefulla mätningar från magnetometern behöver den kalibreras innan användning. Då magnetometern ska användas som en kompass behöver den mäta av jordens magnetfält. Det finns då enligt [4] två huvudsakliga effekter som kan störa mätningarna. Så kallade hård- och mjukjärneffekter. Hårdjärnseffekter kommer från magnetfält från komponenter eller material som är fästa i samma kropp som magnetometern. De orsakar en konstant förskjutning av de värden som magnetometern läser av. Mjukjärnseffekter kommer från att kringliggande material kan förstärka eller försvaga hur jordens magnetfält läses av av magnetometern. Dessa effekter orsakar att magnetfältets x och y komponenter skalas olika beroende på magnetometerns vinkel relativt jordens magnetfält. Dessa störningarna kan enligt [4] modelleras som att varje tvådimensionell mätning av magnetfältet är det riktiga magnetfältet multiplicerat med en symmetrisk 2×2 matris, $\boldsymbol{A}$.

För att kalibrera magnetometern roteras farkosten ett varv längs sin z-axel. Mätningarna bör då hamna på enhetscirkeln, men på grund av störningarna skapar de istället en ellips som inte är centrerad i origo. En ellips anpassas då till mätningarna genom minstakvadratmetoden. Ellipsen beskrivs av ekvationen:

$$\boldsymbol{p}_e = \boldsymbol{A}\boldsymbol{p}_c + \boldsymbol{b} \tag{28}$$

där $\boldsymbol{p}_e$ är en punkt på ellipsen, $\boldsymbol{p}_e$ en punkt på enhetscirkeln och $\boldsymbol{b}$ är ellipsens centerpunkt. Förskjutningen och skalningsmatrisen som genererar den anpassade ellipsen inverteras sedan för att skapa en transform som omvandlar mätningar på ellipsen till enhetscirkeln, se (29).

$$\boldsymbol{p}_c = \boldsymbol{A}^{-1}(\boldsymbol{p}_e - \boldsymbol{b}) \tag{29}$$

Denna transform används sedan på alla vidare mätningar från magnetometern för att ta bort störningseffekterna.

4.2 Vinkelskattning

Vid skattning av *yaw*-vinkeln är det magnetometern och gyroskopet i IMU:n (*Inertial Measurement Unit*) som används tillsammans med ett Kalman-filter. För skattning följs algoritm 7.1 som definieras i Gustafsson, [5]. Den tidsdiskreta tillståndsmodell som används i filtret är

$$\begin{cases} x_k = \tan^{-1}(\frac{-Mag_y}{Mag_x}) \\ x_{k+1} = x_k + T_s u_k \\ y = x_k \end{cases} \tag{30}$$

där Mag_x och Mag_y är mätvärdet från magnetometern i x_b- respektive y_b-led, x_k är *yaw*-vinkeln vid sampel k, u_k är styrsignalen som i den här modellen är en mätning av rotationshastigheten i x_b-y_b-planet som från fås gyroskopet och T_s är samplingstiden som antas vara konstant $1/8[s]$. Denna metod approximerar farkosten till att ligga plant i vattnet vilket kan leda till något felaktiga mätningar men fastställdes experimentellt vara tillräckligt nogrann för denna implementation. Kalman-filtret är uppbyggt av en tidsuppdatering som predikterar tillståndet i nästa sampel och korrigeras sedan av en mätuppdatering för att få skattningen i sampel k.

Tidsuppdateringen definieras som

$$\hat{x}_{k+1|k} = \hat{x}_{k|k} + T_s u_k \tag{31}$$

$$P_{k+1|k} = P_{k|k} + T_s^2 Q \tag{32}$$

där P är skattningens varians och $Q = 0.0008$ är variansen på bruset från gyroskopets mätningar. Detta görs enbart om den uppmätta styrsignalen ligger inom intervalet $[-2, 2]\frac{rad}{s}$, annars antas styrsignalen vara $0\ \frac{rad}{s}$.

Mätuppdateringen definieras som

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + \frac{P_{k|k-1}}{P_{k|k-1} + R}(y_k - \hat{x}_{k|k-1}) \tag{33}$$

$$P_{k|k} = P_{k|k-1} - \frac{P_{k|k-1}^2}{P_{k|k-1} + R} \tag{34}$$

där $R = 0.002$ är variansen på bruset från magnetometervärdena. Som för tidsuppdateringen kommer detta enbart ske om den euklidiska normen av Mag_x och Mag_y ligger inom intervalet $[0, 2]$. Detta medför att om ingen mätuppdatering görs så blir tidsuppdateringens prediktion det skattade tillståndet för den senaste uppdateringen. Värdena för Q och R har utgått från den faktiskt uppmätta variansen på bruset för mätningar men har justerats för bättre resultat.

4.3 Hastighetsskattning

För skattningen av hastighet följer ett liknande tillvägagångssätt som för skattningen av *yaw*-vinkel där x_k är hastigheten i x_n- och y_n-led mätt från GNSS vid sampel k och u_k är accelerationen mätt i x_b- och y_b-led. GNSS-mätvärdena görs om till approximativa hastighetsmätningar som

$$\begin{cases} Vel_{x_k} = \frac{GNSS_{x_k} - GNSS_{x_{k-1}}}{T_s} \\ Vel_{y_k} = \frac{GNSS_{y_k} - GNSS_{y_{k-1}}}{T_s} \end{cases} \tag{35}$$

där $GNSS_{x_k}$ och $GNSS_{y_k}$ är mätvärdet från GNSS:en i x_n- och y_n-led vid sampel k. GNSS-värdenas noggrannhet har förbättrats genom att använda RTK (Real Time Kinematic). Skillnaderna mellan hastighetsskattningen och *yaw*-vinkelskattningen är att det nu är två tillstånd istället för ett och att en rotationsmatris introduceras i tillståndsmodellen för att göra om accelerationsmätningarna från lokal acceleration till global acceleration. Tillstånden, styrsignalerna och utsignalerna är definierade som

$$\begin{cases} \hat{\boldsymbol{x}}_k = \begin{bmatrix} Vel_{x_n} \\ Vel_{y_n} \end{bmatrix} \\ \hat{\boldsymbol{u}}_k = \begin{bmatrix} Acc_{x_b} \\ Acc_{y_b} \end{bmatrix} \\ \boldsymbol{y_k} = \hat{\boldsymbol{x_k}} \end{cases} \tag{36}$$

Tidsuppdateringen definieras som

$$\hat{\boldsymbol{x}}_{k+1|k} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \hat{\boldsymbol{x}}_{k|k} + T_s \begin{bmatrix} cos(\psi_k) & sin(\psi_k) \\ -sin(\psi_k) & cos(\psi_k) \end{bmatrix} \boldsymbol{u}_k \tag{37}$$

$$\boldsymbol{P}_{k+1|k} = \boldsymbol{P}_{k|k} + T_s^2 \boldsymbol{Q} \tag{38}$$

där ψ är den skattade *yaw*-vinkel vid sampel k medans Q och P är variansen för bruset på accelerometern och variansen på skattningen, som för skattningen av *yaw*-vinkel men fetstilta för att förtydliga att det är en vektor eller matris. Q är definierat som

$$\boldsymbol{Q} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.1 \end{bmatrix}$$

Om någon av de uppmätta accelerationerna ligger utanför intervalet $[-2,2]\frac{m}{s^2}$ antas styrsignalen vara 0 $\frac{m}{s^2}$.

Mätuppdateringen definieras som

$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{P}_{k|k-1}(\boldsymbol{P}_{k|k-1} + \boldsymbol{R})^{-1}(\boldsymbol{y}_k - \hat{\boldsymbol{x}}_{k|k-1}) \tag{39}$$

$$\boldsymbol{P}_{k|k} = \boldsymbol{P}_{k|k-1} - \boldsymbol{P}_{k|k-1}(\boldsymbol{P}_{k|k-1} + \boldsymbol{R})^{-1}\boldsymbol{P}_{k|k-1} \tag{40}$$

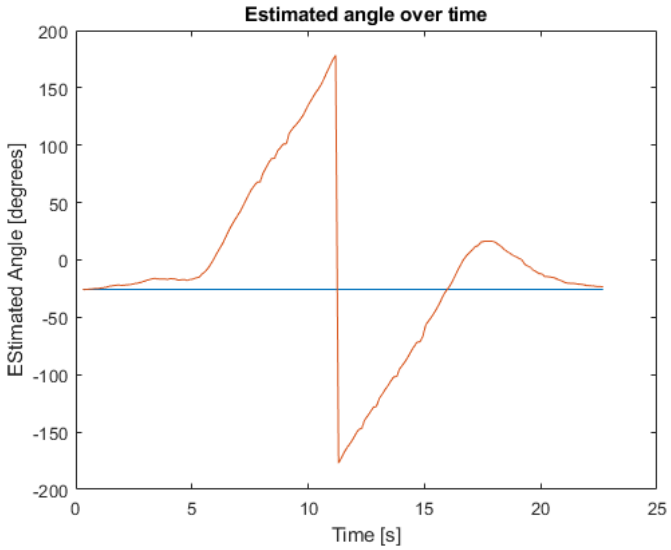
där R är variansen på bruset från GPS:en i x_b- och y_b-led och ser ut som

$$\boldsymbol{R} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$$

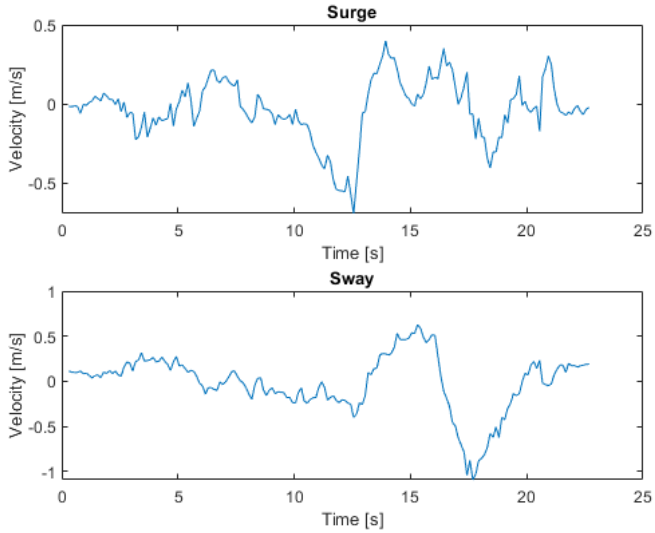
Detta görs bara om de approximativa hastighetsmätningarna ligger inom intervalet $[-3,3]\frac{m}{s}$. Som för skattningen av *yaw*-vinkeln är $\boldsymbol{Q}$ och $\boldsymbol{R}$ justerade från den faktiskt mätta variansen på bruset för att få bättre resultat på skattningen.

4.4 Resultat

Resultatet av sensorfusionen från körningen som användes för att kontrollera kraven på sensorfusionen kan ses i Figur 9 och 10 som är den skattade *yaw*-vinkeln samt *surge* och *sway*. Körningen roterade farkosten ett helt varv och som resultatet visar fungerar skattningen av yaw-vinkeln mycket bra och det skiljer ungefär 2 grader från den första skattade vinkeln och den sista skattade vinkeln. Hastigheterna har rimliga värden och de stämmer bra överens med hur farkosten rörde sig under körningen.



Figur 9: Den skattade *yaw*-vinkeln under körningen



Figur 10: Skattningen av *surge* och *sway* under körningen

5 FÄRDRUTTSPLANERING

För autonom körning så behövs en bana som farkosten kan följa. En sådan kan definieras i det grafiska användargränssnittet som utvecklats för farkosten, se avsnitt 7. Detta görs genom att i det grafiska gränssnittet placera noder som innehåller position och vinkel. Med minst två stycken noder skapas en sluten färdrutt som uppfyller nodernas positions- och vinkelkrav. Interpolationerna görs på farkostens Raspberry Pi som sedan skickar den resulterande banan till det grafiska gränssnittet för visualisering.

5.1 Noder

En nod N_k, definieras enligt

$$\boldsymbol{N_k} = \begin{bmatrix} P_{x_k} \\ P_{y_k} \\ \rho_k \\ v_{abs_k} \end{bmatrix} \tag{41}$$

där P_x och P_y är positionen i x- och y-led, ρ är vinkeln som farkostens önskas åka igenom noden. Dessutom är v_{abs} den fart som farkosten önskas passera noden med men denna används inte i regleringen då funktioner för att utnyttja den behöver utvecklas.

5.2 Interpolation

En tredjegradsekvation ansätts mellan varje nod enligt:

$$x(\delta) = a_4\delta^3 + a_3\delta^2 + a_2\delta + a_1 \tag{42}$$
$$y(\delta) = b_4\delta^3 + b_3\delta^2 + b_2\delta + b_1 \tag{43}$$

där δ är en fiktiv variabel så att $(x(0), y(0))$ är startnoden och $(x(1), y(1))$ slutnoden. För att vinklarna, θ_0 och θ_1, ska överensstämma med resultatet fås då följande konstanter:

$$a_1 = x(0) \tag{44}$$
$$b_1 = y(0) \tag{45}$$
$$a_2 = k\cos(\theta_0) \tag{46}$$
$$b_2 = k\sin(\theta_0) \tag{47}$$
$$a_3 = 3x(1) - 3a_1 - 2a_2 - c\cos(\theta_1) \tag{48}$$
$$b_3 = 3y(1) - 3b_1 - 2b_2 - c\sin(\theta_1) \tag{49}$$
$$a_4 = -2x(1) + 2a_1 + a_2 + c\cos(\theta_1) \tag{50}$$
$$b_4 = -2y(1) + 2b_1 + b_2 + c\sin(\theta_1) \tag{51}$$

Här är c och k godtyckliga konstanter. För att uppnå en relativt kort och mjuk bana testas olika kombinationer av dessa. Ett antal, N, punkter av x samt y beräknas därefter. Det euklidiska avståndet, l, mellan dessa punkter beräknas som skattning på kurvlängden och kurvaturen beräknas för varje punkt enligt:

$$c = \frac{x'(\delta)y''(\delta) - y'(\delta)x''(\delta)}{(x'(\delta)^2 + y'(\delta)^2)^{\frac{3}{2}}} \tag{52}$$

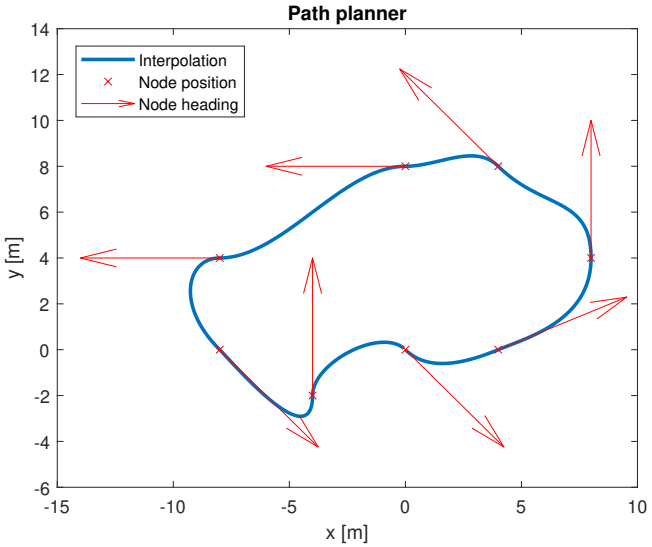
Därefter används den kombination av c och k som ger lägst resultat på värdefunktionen:

$$V = \gamma_1 l + \gamma_2 \max\{c\} \tag{53}$$

Där γ är viktkoefficienter. Experimentellt har det fastställts att $N = 50$, $\gamma_1 = 1$ och $\gamma_2 = 20$ ger bra beteende för detta projekt, där c och k testas som tolv olika värden var, dessa är logaritmiskt spridda mellan 10^{-1} och 10^2. Ett exempel på hela ruttplaneringen syns i Figur 11 i avsnittet 5.3.

5.3 Resultat

Nedan presenteras resultatet av färdruttsplaneraren. Som visas i Figur 11, tas en kontinuerlig, sammanhängande färdrutt utifrån de noder som angivits där orienteringen för varje nodpassage är i överenstämmelse med vad som önskats.



Figur 11: Interpolationer mellan noder

6 REGLERING

Nedan beskrivs de reglerstategier som testats på farkosten.

6.1 Tillståndsåterkoppling

En regulator som använder sig av tillståndsåterkoppling med olinjär kompensering är implementerad. Regulatorn tar hänsyn till tillståndsavvikelser för avstånd från referensbana och vinkelfel för farkostens kurs. Farkostens position används för att projicera till referensbanan, således hittas den punkt på referensbanan som är närmast farkosten. Denna punkt används för att räkna ut skillnaden på referensbanans och farkostens rikting. Med referensbanans vinkel vid närmsta punkten som $c(s)$. Vidare är C_{NL} en olinjär kompenseringsfaktor och K_i är finjusteringsparametrar för regulatorn. Den önskade kurvaturen, w, på farkostens färdbana sätts då enligt

$$w = c(s) - K_1 C_{NL} d_e - K_2 \psi_e \tag{54}$$

$$C_{NL} = 1 - \frac{\psi_e^2}{6} + \frac{\psi_e^4}{120} - \frac{\psi_e^6}{5040} \tag{55}$$

Där d_e är närmaste avståndet från farkosten till referensbanan och ψ_e är vinkelfelet mellan punkten på referensbanan och farkosten. Den olinjära kompenseringsfaktorn kommer från Taylor-utveckling av $\frac{sin\psi_e}{\psi_e}$. Styrvinkeln översätts sedan till rodervinkel genom följande ekvation

$$\alpha_i = tan^{-1}(w) \tag{56}$$

Ekvation härleds från en *kinematik single track* bilmodell, från en föreläsning av Erisk Frisk i kursen TSFS12 [6] på Linköpings universitet. Notera att båda roderpropellrar sätts till samma vinkel med denna regulator. En konstant

propeller-hastighet på 800 RPM användes också. Utifrån test i simuleringsmiljön valdes designparametrarna som $K_1 = 2$ och $K_2 = 3$.

6.2 MPC

MPC står för Model Predictive Control och betyder här att modellen som beskrivs i avsnitt 3 används för att förutse farkostens färdväg. Regleringen kan då ställas upp som ett optimeringsproblem där differensen mellan färdväg och referensbana ska minimeras med avseende på farkostens styrsignaler.

MPC utgår från en tidsdiskret version av modellen som skattar systemets framtida tillstånd. Minimering sker sedan av en målfunktion med vikt på felaktiga tillstånd och storleken på styrsignaler under en given mängd tidsinstanser. Målfunktionen ges av (57) enligt Enqvist, Glad, Gunnarsson med flera [7].

$$J_N(\boldsymbol{x}(k)) = \sum_{j=0}^{N-1} ||\boldsymbol{z}(k+j) - \boldsymbol{r}(k+j)||^2_{\boldsymbol{Q_1}} + ||\boldsymbol{u}(k+j)||^2_{\boldsymbol{Q_2}} \tag{57}$$

$$\text{OCP} \begin{cases} \hat{\boldsymbol{u}}_N & = \arg\min_{\boldsymbol{u}_N} J_n(\boldsymbol{x}(k)) \\ s.t & \begin{cases} \boldsymbol{x}(0) = \boldsymbol{x}_0 \\ \hat{\boldsymbol{u}}_N \in [\boldsymbol{u}_{min}, \boldsymbol{u}_{max}] \text{ for } k = 0, ..., N-1] \\ \boldsymbol{x} \in [\boldsymbol{x}_{min}, \boldsymbol{x}_{max}] \text{ for } k = 0, ..., N-1] \end{cases} \end{cases} \tag{58}$$

I (57) är N mängden tidsinstanser att optimera över och väljs så att samplingstiden gånger N är ungefär insvängningstiden för det önskade slutna systemet. Vidare är $\mathbf{z}(k)$ de tillstånden som ska styras och ges av $\boldsymbol{z}(k) = \boldsymbol{M}\boldsymbol{x}(k)$, $\boldsymbol{r}(k)$ är referensvärde vid tidpunkt k, $\mathbf{u}(k)$ är styrsignaler vid tidpunkt k och $\boldsymbol{Q_1}$ samt $\boldsymbol{Q_2}$ är viktmatriser som kan användas som designparametrar. Elementen i $\boldsymbol{Q_i}$ kommer att väljas med hårda straff på avvikelse från tillstånden utifrån att styrsignalerna används fritt inom det tillåtna intervallet. Övriga krav på dessa viktmatriser är att $\boldsymbol{Q_1}$ skall vara positivt semidefinit medan $\boldsymbol{Q_2}$ skall vara positivt definit. Målfunktionen optimeras vid varje tidpunkt vilket medför att regleringen blir dynamisk och kan hantera plötsliga förändringar i omgivningen. Optimeringsproblemet är definierat i (58) där styrsignalen i tiden k ansätts som det elementet ur $\hat{\boldsymbol{u}}_N$ som är närmast i tiden.

6.2.1 *Implementerad MPC*

Den implementerade MPC-styrningen bygger på en linjärisering och approximation av den framtagna modellen. Fördelen med linjärisering av modellen är att MPC-problemet kan skrivas om till ett kvadratiskt programmeringsproblem, för vilket det finns effektiva numeriska lösare. Nackdelen med linjäriseringen är att viktig dynamik för farkosten lätt försvinner.

För att förenkla problemet används en annan kinematik i MPC modellen. Denna kinematik är samma som används i tillståndsåterkopplingsregulatorn där båten antas ha en hastighet framåt och att det alltid är effektivast att röra sig framåt (i motsats till i sidled). Tillstånden för position och orientering byts då ut mot avstånd till referensbanan, d_e, och vinkelfel, ψ_e, för farkostens kurs, se Figur 12. Den resulterade kinematiken samt en linjäriseringen av den beskrivs i (59).

$$\begin{bmatrix} \dot{d}_e \\ \dot{\psi}_e \end{bmatrix} = \begin{bmatrix} u \cdot \sin\psi_e \\ r \end{bmatrix} \approx /|\psi_e| \text{ antas litet}/ \approx \begin{bmatrix} u \cdot \psi_e \\ r \end{bmatrix} \tag{59}$$

där u är båtens hastighet framåt och r dess vinkelhastighet. Denna ändring minskar också tillståndets dimension vilket gör att problemet kräver mindre beräkningskraft för att lösas. För att linjäriseringen ska fungera måste dock båtens hastighet framåt, u vara nollskild, vilket var anledningen till att u lades till bland de reglerade tillstånden och gavs en nollskild referens.

För att linärisera modellens kinetik behövs enligt [8] en stationär punkt, $\boldsymbol{x_{stat}, u_{stat}}$, där modellen kan linjäriseras runt. Att en punkt är stationär betyder att tillståndens tidsderivata där är 0. Om modellen ges av $\dot{\boldsymbol{x}} = \boldsymbol{f(x,u)}$ är den linjäriserade modellen $\dot{\bar{\boldsymbol{x}}} = \boldsymbol{A\bar{x} + B\bar{u}}$ där $\boldsymbol{A}$ har elementen a_{ij} och $\boldsymbol{B}$ elementen b_{ij} definierade enligt:

$$a_{ij} = \frac{\partial f_i}{\partial x_j},\ b_{ij} = \frac{\partial f_i}{\partial u_j} \tag{60}$$

och där $\bar{\boldsymbol{x}} = \boldsymbol{x} - \boldsymbol{x_{stat}}, \bar{\boldsymbol{u}} = \boldsymbol{u} - \boldsymbol{u_{stat}}$.

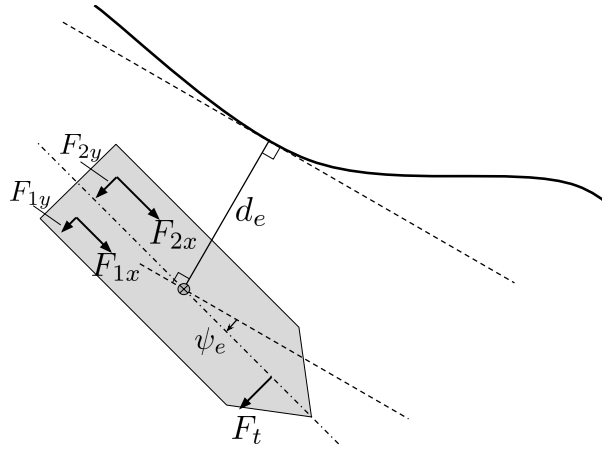
I början av arbetet med att utveckla MPC-styrningen försökes den olinjära modellen från avsnitt 3 linjäriseras direkt runt farkostens nuvarande tillstånd, det vill säga de tillstånd som gavs från sensorfusionen (se avsnitt 4) och övriga givare. Mycket tid spenderades på detta men inga framsteg gjordes. Det första tillvägagångssättet som fungerade var att alltid linjärisera runt en fix stationär punkt där $u = 0, v = 0, r = 0$. Notera dock att denna punkt endast användes vid linjärisering av modelens kinetik, kinematiken antar fortfarande en hastighet framåt. På grund av tidsbrist gjordes valet att fortsätta utveckla endast detta tillvägagångssätt.

Ett problem som uppkommer vid linjärisering runt denna punkt är att modellens kvadratiskt dämpade krafter, det vill säga krafter på formen $C \cdot |x|x$, har en derivata lika med noll. Inverkan av dessa krafter försvinner då vid linjäriseringen. Samma problem gör även att corioliseffekten inte kommer med i modellen. För att undvika detta approximeras dessa krafter med linära dämpade krafter på formen $C_l \cdot x$. Konstanten C_l sätts, i detta projekt, till att minimera skillnaden mellan den kvadratiska och den linjära kraften på det interval, $|x| \leq x_{max}$, där tillståndet x förväntades ligga under en körning, det vill säga:

$$C_l = \arg\min_{C_l} \int_{-x_{max}}^{x_{max}} (C_l \cdot x - C \cdot |x|x)^2 dx = 3/4 \cdot x_{max} \cdot C \tag{61}$$

Denna metod fungerar dock inte för att behålla corioliseffekten. Den används därför inte i den implementerade MPC:n.

Den sista approximationen som görs är att låta styrsignalerna i MPC:n vara kraftkomponenterna i båtens lokala koordinatsystem från respektive propeller istället för varvtal och vinkel, se Figur 12 för en illustration av dessa krafter. På detta sätt blir deras inverkan linjär. Notera att kraften framåt (x-led) från propeller 1 är en separat styrsignal från kraften framåt från propeller 2. Det samma gäller kraftkomponenterna i sidoled (y-led) från varje propeller. En fördel med att dela upp krafterna mellan propellrarna, jämfört med att endast låta den totala kraften och momentet på farkosten vara styrsignal, är att vikterna och begränsningarna i optimeringsproblmet kan anpassas till att prioritera eller begränsa en eller flera propellrar. Till exempel kan bogpropellern inte generera lika mycket kraft vid dess högsta varvtal som de andra propellrarna, MPC:n kan då ta hänsyn till detta när styrsignalerna planeras. De separata vikterna gör också att det inte uppstår problem från att farkosten har en överstyrning. Det vill säga från att det finns flera kombinationer av styrsignaler som kan generera samma totala kraft på farkosten. Detta då en kombination kommer generera ett mindre värde på målfunktionen, $\boldsymbol{J}_N$ se (57). Efter att MPC:n räknat ut styrsignalen omvandlades krafterna till motsvarande varvtal och vinklar. Denna omvandling är baserad på modellen i avsnitt 3 och visas i (62).



Figur 12: Illustration av de kinematiska tillstånden och de krafter som används i MPC:n.

$$
\begin{aligned}
n_1 &= \sqrt{\frac{1}{\mu_{az}}\sqrt{F_{1x}^2 + F_{1y}^2}} \cdot \operatorname{sgn} F_{1x} \\
n_2 &= \sqrt{\frac{1}{\mu_{az}}\sqrt{F_{2x}^2 + F_{2y}^2}} \cdot \operatorname{sgn} F_{2x} \\
n_t &= -\sqrt{\frac{|F_t|}{\mu_t}} \cdot \operatorname{sgn} F_t \\
\alpha_1 &= \arctan\left(F_{1y}/F_{1x}\right) \\
\alpha_2 &= \arctan\left(F_{2y}/F_{2x}\right)
\end{aligned}
\tag{62}
$$

Den resulterade reducerade modellen som används för MPC är beskriven i (63).

$$
\dot{\boldsymbol{x}}_{MPC} = \begin{bmatrix}
u_{lin} \cdot \psi_e \\
r \\
\frac{1}{m_{11}}(X_{|u|u} \cdot \frac{3}{4} \cdot u_{max} - X_u)u + \frac{1}{m_{11}}(F_{1x} + F_{2x}) \\
\frac{1}{m_{22}}((Y_{|v|v} \cdot \frac{3}{4} \cdot v_{max} - Y_v)v - Y_r r) + \frac{1}{m_{22}}(F_{1y} + F_{2y} + F_t) \\
\frac{1}{m_{33}}((N_{|v|v} \cdot \frac{3}{4} \cdot v_{max} - N_v)v + (N_{|r|r} \cdot \frac{3}{4} \cdot r_{max} - N_r)r) + \frac{1}{m_{33}}(-l_{y1}F_{1x} + l_{x1}F_{1y} - l_{y2}F_{2x} + l_{x2}F_{2y} + l_{xt}F_t)
\end{bmatrix}
\tag{63}
$$

Här är $\boldsymbol{x}_{MPC} = \begin{bmatrix} d_e & \psi_e & u & v & r \end{bmatrix}^T$ och u_{lin} är den hastighet framåt som båten förväntas ha. Notera att modellen är linjär och kan skrivas på formen $\dot{\boldsymbol{x}}_{MPC} = \boldsymbol{A}_{MPC}\boldsymbol{x}_{MPC} + \boldsymbol{B}_{MPC}\boldsymbol{c}_f$ där $\boldsymbol{c}_f = \begin{bmatrix} F_{1x} & F_{1y} & F_{2x} & F_{2y} & F_t \end{bmatrix}^T$

För att använda modellen med MPC måste den vara i diskret tid. Detta görs genom Eulers metod i (64). Notera att T_s är sampeltiden på 1/8 sekunder:

$$
\boldsymbol{x}_{MPC}(t + T_s) = \left(\boldsymbol{I}_{5\times5} + T_s \cdot \boldsymbol{A}_{MPC}\right)\boldsymbol{x}_{MPC}(t) + T_s \cdot \boldsymbol{B}_{MPC}\boldsymbol{c}_f(t) \triangleq \boldsymbol{F}_{MPC}\boldsymbol{x}_{MPC}(t) + \boldsymbol{G}_{MPC}\boldsymbol{c}_f(t) \tag{64}
$$

6.2.2 *Kvadratiskt programmering*

På Raspberry pi:en används Python-modulen `osqp` för att lösa MPC-problemet. Modulens algoritm finns beskriven i [9]. Minimeringsproblemet från (58) är implementerat som följande:

$$J_n(\boldsymbol{x}(k)) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x} + \boldsymbol{q}^T\boldsymbol{x} \tag{65}$$
$$s.t: \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$$

Nedan beskrivs de matriser som används i (65) för den MPC som beskrivs tidigare i avsnitt 6.2.1.

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{c}_f^T(k) & \boldsymbol{c}_f^T(k+1) & \dots & \boldsymbol{c}_f^T(k+N-1) \end{bmatrix}^T$$
$$\boldsymbol{P} = \mathcal{G}^T\mathcal{M}^T\mathcal{Q}_1\mathcal{M}\mathcal{G} + \mathcal{Q}_2$$
$$\boldsymbol{q} = \mathcal{G}^T\mathcal{M}^T\mathcal{Q}_1(\mathcal{M}\mathcal{F}\boldsymbol{x}_{MPC} - \mathcal{R})$$
$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{I}_{5\times5} & & \\ & \ddots & \\ & & \boldsymbol{I}_{5\times5} \\ -\boldsymbol{I}_{5\times5} & & \\ & \ddots & \\ & & -\boldsymbol{I}_{5\times5} \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} \boldsymbol{c}_{f,max} \\ \vdots \\ \boldsymbol{c}_{f,max} \\ -\boldsymbol{c}_{f,min} \\ \vdots \\ -\boldsymbol{c}_{f,min} \end{bmatrix}$$

Där $\boldsymbol{c}_{f,max}$ är de största tillåtna krafterna och $\boldsymbol{c}_{f,min}$ är de minsta tillåtna krafterna. Notera att följande beteckningar används:

$$\mathcal{F} = \begin{bmatrix} \boldsymbol{I}_{5\times5} \\ \boldsymbol{F}_{MPC} \\ \vdots \\ \boldsymbol{F}_{MPC}^{N-1} \end{bmatrix}, \mathcal{G} = \begin{pmatrix} 0 & 0 & \cdots & \cdots & 0 \\ \boldsymbol{G}_{MPC} & 0 & \cdots & \cdots & 0 \\ \boldsymbol{F}_{MPC}\boldsymbol{G}_{MPC} & \boldsymbol{G}_{MPC} & \ddots & & \vdots \\ \vdots & & & & \vdots \\ \boldsymbol{F}_{MPC}^{N-2}\boldsymbol{G}_{MPC} & \cdots & \boldsymbol{F}_{MPC}\boldsymbol{G}_{MPC} & \boldsymbol{G}_{MPC} & 0 \end{pmatrix},$$

$$\mathcal{Q}_1 = \begin{bmatrix} \boldsymbol{Q}_1 & & \\ & \ddots & \\ & & \boldsymbol{Q}_1 \end{bmatrix}, \mathcal{Q}_2 = \begin{bmatrix} \boldsymbol{Q}_2 & & \\ & \ddots & \\ & & \boldsymbol{Q}_2 \end{bmatrix}, \mathcal{M} = \begin{bmatrix} \boldsymbol{M} & & \\ & \ddots & \\ & & \boldsymbol{M} \end{bmatrix}, \mathcal{R} = \begin{bmatrix} \boldsymbol{r}(k) \\ \boldsymbol{r}(k+1) \\ \vdots \\ \boldsymbol{r}(k+N-1) \end{bmatrix}$$

Se avsnitt 6.2 för mer information om matriserna som används.

6.2.3 *Val av tillåtna styrsignaler*

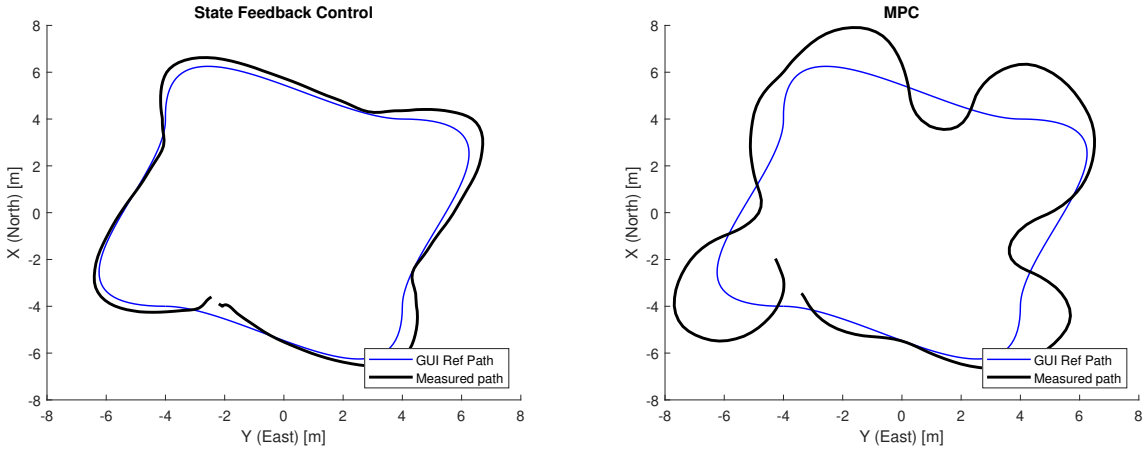
De största och minsta krafterna, $\boldsymbol{c}_{f,max}$ och $\boldsymbol{c}_{f,min}$, valdes så att propellermotorerna aldrig skulle rotera i mer än 3500 RPM. Roderpropellrarna skulle inte heller byta riktning eller vridas mer än 90 grader åt ett håll. Motiveringen till

att roderpropellrarna inte fick byta riktning var att detta fall inte modellerats lika bra som fallet när propellrarna drev farkosten framåt. För att införa denna begränsning i optimeringsproblemet behövde vinkel också begränsas till ± 90 grader. Jämförelsevis kunde roderpropellrarna på den riktiga farkosten vrida sig i ± 100 grader.

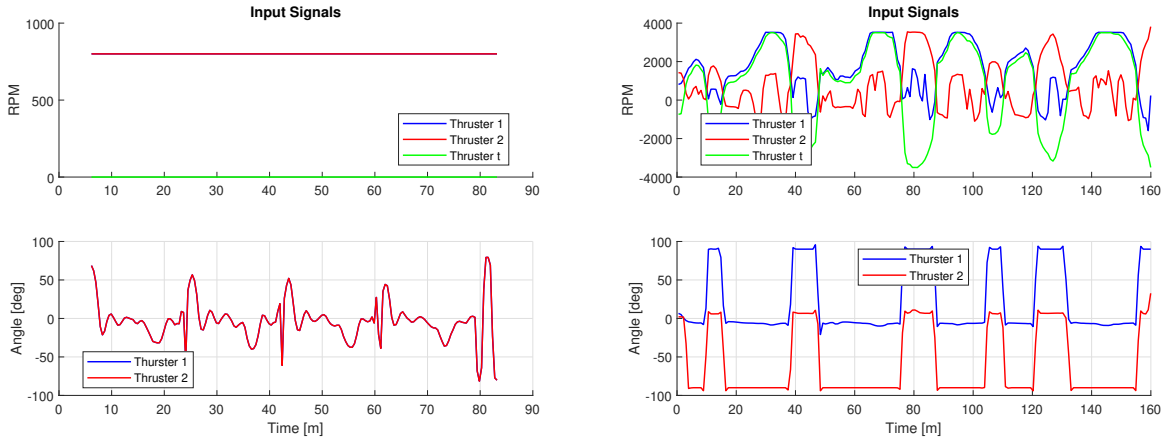
Dessa begränsningar gav MPC-styrningen en svaghet i att den inte kunde frambringa någon kraft för att bromsa båten. En lösning där fler styrkrafter var tillåtna skulle kunna förbättra regulatorns prestanda. Det fanns dock inte tid att utveckla och testa en sådan lösning.

6.3 Resultat från reglering

Både MPC:n och tillståndsåterkopplingen testades genom att låta farkosten följa den rutt som genererades när interpolationen, som beskrivs i avsnitt 5.2, gavs fyra noder representerade en 8 meters kvadrat. Farkostens uppnådda bana för tillståndsåterkoppling och MPC presenteras i Figur 13. Styrsignalerna som regulatorerna gav plottas i Figur 14.



Figur 13: Farkostens bana vid reglering baserad på tillståndsåterkoppling respektive MPC.



Figur 14: Styrsignalerna som gavs vid följning av referensbanan för tillståndsåterkoppling (till vänster) och MPC (till höger).

6.3.1 *Analys av resultat från reglering*

Det är tydligt från Figur 13 att tillståndsåterkopplingen presterade bättre i detta test. Detta attributerades till att dess relativa enkla design och få frihetsgrader gjorde att den inte behövde mycket finjusteringar av designparametrarna för att få ett bra resultat. Då MPC:n hade större frihet i hur den kunde allokera styrsignaler var den mer känsliga för hur dess designparametrar i Q_1 och Q_2 var inställde. Det fanns inte tillräckligt med tid för att göra flera test och hitta bättre värden på dessa matriser.

Utifrån MPC:ns styrsignal i Figur 14 går det att se att regulatorn skickade negativa RPM-värden till både roderpropeller 1 och 2. Detta betyder att kraven som sattes i optimeringsproblemet i (65) inte är uppfyllda. Vidare är det tydligt att MPC:n försökte använda bogpropellern mycket för att styra båtens riktning. Detta fungerar enligt modellen men när farkosten rör sig framåt blir bogproppelern i verklighet mycket mindre effektiv då mindre vatten rinner in till den. En sista egenskap som noterades var att MPC:n krävde snabba förändringar av vinklarna på roderpropellrarna. Det är inte garanterat att ställdonen på farkosten kan ändra vinklarna lika fort som MPC:n behöver.

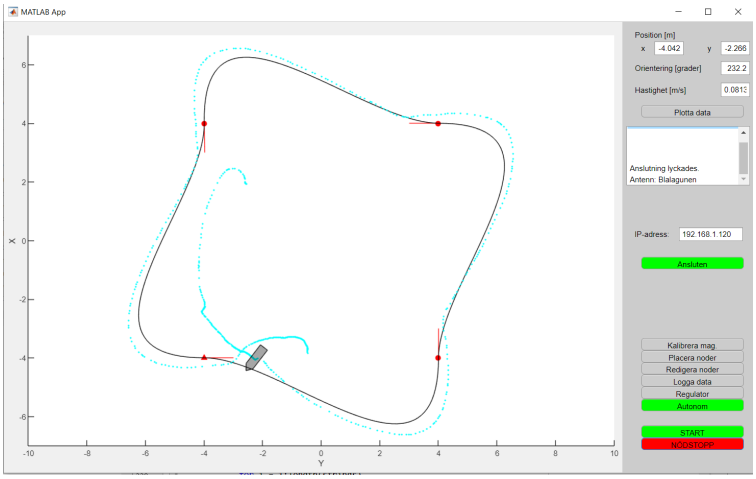
7 ANVÄNDARGRÄNSSNITT

Användargränssnittet är utvecklat i MATLAB App Designer och har som syfte att övervaka farkosten i realtid och underlätta genomförandet av tester.

7.1 Information som visas

Farkostens position, *yaw*-vinkel och hastighet redovisas i realtid i användargränssnittets övre högra del samt grafiskt genom att rita ut farkosten i ett koordinatsystem (se Figur 15). Utöver detta visas även de noder som användaren eventuellt valt att placera som röda cirklar med streck för att indikera vilken riktning de har, den första av dessa noder (startnoden) indikeras av en triangel istället för en cirkel. När 2 eller fler noder placerats interpolerar Raspberry Pi:en en bana mellan dessa noder, denna bana ritas även ut i koordinatsystemet som en svart linje. För att utvärdera

hur väl farkosten följt denna bana ritas gamla positioner ut som ljusblå prickar. Slutligen finns en textruta där korta medelanden skrivs ut, till exempel vilken RTK-antenn man valt att ansluta till.



Figur 15: Exempelbild på användargränssnittet under användning.

7.2 Funktionalitet

Utöver att visa den information som beskrivits i segmentet ovan, så finns det funktionalitet för att:

- genomföra magnetometerkalibrering
- redigera, lägga till och ta bort noder i en lista för att skapa en färdrutt
- logga data
- välja mellan tillståndsåterkoppling eller MPC vid autonom körning och justera dess parametrar
- starta autonom körning
- köra manuellt med tangentbordet
- få fram realtidsplottar för position, *yaw*-vinkel och hastighet

7.2.1 *Farkostens data*

Farkostens position, *yaw*-vinkel och hastighet tas emot som ROS-medelanden från Raspberry Pi:en, dessa medelanden publiceras 8 gånger per sekund. Alltså uppdateras informationen på användargränssnittet 8 gånger per sekund.

7.2.2 *Skapa färdrutt*

Ett av användargränssnittets viktigaste funktioner är att på ett smidigt sätt kunna skapa en färdrutt för farkosten att navigera längs. Detta har implementerats så att användaren placerar ut noder och deras riktning i koordinatsystemet

och när det finns två eller fler noder placerade med bestämda riktningar så skickas dessa till Raspberry Pi:en som interpolerar en färdrutt mellan dessa punkter i den ordning de placerats ut. Denna interpolation görs om varje gång en ny nod placeras ut och får sin vinkel bestämd samt när någon redigering på befintliga noder görs. När interpolationen är gjord så skickas den från Raspberry Pi:en till användargränssnittet så att användaren kan se färdrutten.

7.2.3 *Ställa in regulator*

Efter att färdrutten skapats måste användaren bestämma vilken regulator den önskar använda. Denna funktion är implementerad så att när användaren sparar regulatorn så skickas ett ROS-medelande om MPC valdes, ett annat om tillståndspterkoppling valdes. Dessa medelanden innehåller de respektive regulatorernas parametrar samt vilken regulator som valdes. Då färdrutten som regulatorn ska följa sparas i samband med att dessa medelanden tas emot är det viktigt att valet av regulator sparas efter att färdrutten färdigställts, även om inga ändringar önskas göras i själva regulatorn.

7.2.4 *Manuell körning*

Den manuella körningen via tangentbordet är begränsad av hastigheten hos användargränssnittet samt av tangenters binära (av/på) egenskap. Implementationen är gjord så att om tillståndet (nedtryckt/inte nedtryckt) hos en av tangenterna som styr farkosten ändras och autonomt läge inte är på, så skickas ett uppdaterat `ShipCommand` ROS-meddelandet till Raspberry Pi:en.

7.2.5 *Realtidsplottar*

Plottarna är till för att underlätta felsökning, de plottar endast data från och med att dialogrutan öppnades. Alternativt kan användaren ställa in hur långt tidsintervall den vill se plottad, i detta fall plottas datan från så lång tid tillbaka fram till den aktuella tiden.

7.3 Kända problem

Det går endast att starta loggningen av data, inte sluta. Vidare orsakar prestandaproblem hos användargränssnittet att informationen om farkosten inte hinner uppdateras 8 gånger i sekunden.

8 ROS

ROS, *Robot Operating System*, är ett system för att strukturera kod och kommunikation. Versionen som användes var *ROS Noetic*, detaljer finns att tillgå i dess dokumentation [10]. Två viktiga delar av *ROS* är *nodes* och *topics*, där en *node* är en oberoende tråd som utför en uppgift. Kommunikationen mellan olika *nodes* sker via *topics* som är en sorts kanal där en viss typ av data kan publiceras och läsas av olika *nodes*. Se Tabell 6 för en beskrivning av alla *topics* som används. Tabell 6 listar alla noder som körs.

Tabell 5: *ROS topics*

	Topic	**Innehåll**
1	*rpm_chatter*	Uppmätt varvtal från respektive trustor
2	*ship_command*	Önskade varvtal och styrutslag
3	*velocity_command*	Önskade varvtal
4	*angle_command*	Önskade styrutslag
5	*gps_rtk/rtcm_data*	RTK-data
6	*imu/data_raw*	Accelerometer- och gyroskopdata
7	*imu/mag*	Obehandlad magnetometerdata
8	*imu/mag_calibrated*	Behandlad magnetometerdata
9	*mag_cal_res*	Resultatet från kalibreringen av magnetometern
10	*gui_command*	Sammanslagning av *Topic* 11-13 och 20-21
11	*mag_command*	Start/stopp av mag. kal. från användargränssnittet
12	*log_command*	Start/stopp av loggning av data från användargränssnittet
13	*node_command*	Lista med noder från användargränssnittet
14	*path*	Lista med interpolerade punkter för färdrutt
15	*gps_chatter*	Koordinater från GNSS-mottageren i latitud och longitud
16	*local_coords*	Koordinater från GNSS-mottageren konverterade till lokala koordinater, x och y
17	*imu/sens_fus*	Hastighet och *yaw*-vinkel
18	*run*	Start/stopp av autonom körning från användargränssnittet
19	*control_param*	Regulatorparametrar från användargränssnittet
20	*rtk_command*	RTK-inställningar
21	*auto_command*	Låser upp/låser autonom körning

Tabell 6: *ROS nodes*

Node	**Syfte**	***Topics***-publicering	***Topics***-prenumeration
boat	Skickar signaler som styr båtens hårdvara	-	1,2,3,4
gui_com	Hanterar större delen av kommunikationen med GUI:t	9,14	10,11,12,13
rpm_node	Läser av varvtalen på motorerna	1	-
imu_node	Läser av data från IMU:n och omvandlar mag-datan	6,7,8	9
sens_node	Utför sensorfusionen	17	6,8,16
ctrl_node	Utför regleringen	2,18	6,14,16,17,18,19
ntrip_client	Tar emot RTCM-data	5	20
rtcm2gps	Skriver rtcm-data till gps:en	-	5
rtk_gps_node	Läser av gps och konverterar till lokala koordinater	15,16	-
MatLab_node	Gui:t (den enda noden som inte körs på båten)	2,11,12,13,18,19,20,21	9,14,16,17

9 SLUTSATSER

En stor mängd data har samlats in från flera datainsamlingsexperiment. En modell för farkosten har tagits fram vars resultat har bedömts tillräckligt bra för MPC, se Figur 8. Ett användargränssnitt för realtidsövervakning av farkosten har utvecklats.
Farkosten är kapabel interpolera en färdrutt, givet 2 eller fler noder, och sedan följa denna färdrutt med en felmarginal på 0.7 m.

Sammanfattningsvis har datainsamlingar gjorts för att ta fram en modell av forkosten. Modellen har sedan använts som grund vid utveckling av en modellbaserad regulator. En färdrutt kan definieras i det grafiska gränssnittet som farkosten ska följa. Rutten kan skickas trådlöst till Raspberry Pi:en på farkosten. Användaren kan aktivera autonom körning vilket får farkosten att börja köra längs den definierade färdrutten tills autonom körning inte längre önskas.

9.1 Tekniska framgångar

Under arbetet med projektet har två olika regulatorer utvecklats. En simuleringsmiljö, för evaluering av regulatorer och utveckling av nya, har också skapats. Det finns även fungerande MATLAB-kod för diverse plottning och analys av modellprestanda. Vidare har utvecklingen av ett grafiskt användargränssnitt, med realtidsåterkoppling av båtens tillstånd, lyckats. Detta användargränssnitt använder sig av ROS för kommunikation med mjukvaran som körs på farkosten.

9.2 Nuvarande problem

- Under några datainsamlingar märktes det att data Från GNSS-mottagaren till synes slumpmässigt stängs av och ingen positioneringsdata loggas. Detta har lett till bekymmer vid datainsamlingar och test av reglering. Den troligaste slutsatsen är glapp i kopplingen mellan GNSS-mottagaren och Raspberry Pi:en.
- RTK-data fås inte alltid vid start av mjukvaran på farkosten.
- Ibland så kan inte en skalningsmatris för magnetometerkalibrering tas fram vilket markeras som att matrisens element är NaN. När detta sker, upprepa magnetometerkalibreringen tills rimliga värden i matrisens element fås.
- Oregelbundna momuntära vinkelutslag och rotationshastigheter för roderpropellerna. Detta sker oavsett om en regulator är aktiv eller inte.
- Vid rotationshastigheter under 1200 rpm på roderpropellerna är mätningarna från tachometrarna inte längre tillförlitliga.
- Det grafiska gränssnittet är långsamt från början och prestandaproblemen blir värre desto mer som ritas ut. Av denna anledning bör det grafiska gränssnittet återställas med jämna mellanrum. En potentiell lösning är även att använda en kraftigare dator.
- Kommunikationen mellan GUI:t och båten är i vissa fall känsligt för vilken version av MATLAB som används (i detta projekt användes 2020b).
- Vid autonom körning krävs att båten positioneras efter startnoden innan start för korrekt banföljning.

- Vid autonom körning behöver säkerhetsbrytaren på handkontrollen aktiveras när slutnoden nås. Detta eftersom regleringen annars försöker återvända till sista punkten på banan.

9.3 Framtida arbete

9.3.1 *Sensorfusion*

Sensorfusionen kan utvecklas för att hantera nollskilda värden på *roll* och *pitch*. Detta skulle kunna ge bättre skattningar av *yaw* samt kompensera flr magnetometern.

9.3.2 *Modell*

En ordentlig undersökning av alternativa modeller som kan användas för MPC och datainsamlingsoptimering. En grundlig undersökning av alternativa modeller, både grey- och blackbox. Modeller som kan användas för MPC samt datainsamlingsoptimering. Här bör även olika typer av linjäriseringar undersökas för att utröna viktig dynamik.

9.3.3 *Användargränssnitt*

För användargränssnittet bör det implementeras en modell av fartyget för en mer realistisk ruttlanering sär farkostens förmåga att följa banan är säkerställd. Därtill bör funktionalitet vilken kan avgränsa ej farbart vatten och hinder i användargränssnittet implementeras. Dessa avgränsade områden bör vara definierade så att även ruttplaneringen kan ta hänsyn till det vid interpolering av referensbana.

9.3.4 *Regulatorer*

Implementationen av MPC-styrningen bör ses över efter möjliga fel som kan orsaka att begränsningarna på som sätts i optimeringsproblemet överskrids.

Modellen och optimeringsproblemet bör också uppdateras för att bättre hantera farkostens begränsningar i styrsignalerna.

Modellens linjäriseringar och approximationer bör uppdateras och undersökas och uppdateras för att ge bättre förutsättningar för regulatorn. Därtill kan möjligheter för att uppdatera implementeringen av begränsningarna i fartygets styrsignaler uppdateras.

Optimeringsproblemet kan undersökas både utifrån vilka verktyg som används och tidsoptimeras.

Den befintliga strukturen mellan användargränssnitt och regulatorer kräver att regulatorn parametrar skickas varje gång färdrutten uppdateras. Detta bör avhjälpas.

9.3.5 *Allmänt*

Fartygssystemet bör felsökas efter vad som gör att RTK-data ej mottages vid varje tidssteg då detta är en vital del av systemet.

Strukturera ROS-nätverket. Information läses av, ”subscribers”, av många olika script där det första som görs är att signalen skalas om eller justeras. Detta skulle kunna avhjälpas genom en centralnod som behandlar datan och skickar ut signaler med enheter och strukturer som bättre passar projektet.

REFERENSER

[1] G. Zetterqvist och F. Steen, *Modelling and Trajectory Planning for a Small-Scale Surface Ship*, 2021.

[2] T. I. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*. John Wiley & Sons, Ltd, 2011.

[3] O. Hermansson, O. Jonsson, J. Siönäs, A. Ståhlbom, J. Wenngren och T. Wiik, *Kravspecifikation, Model-based Control of Small-scale Surface Vessel*, 2021.

[4] T. Ozyagcilar, *Calibrating an eCompass in the Presence of Hard- and Soft-Iron Interference*, Freescale Semiconductor, 2015. URL: https://www.nxp.com/docs/en/application-note/AN4246.pdf.

[5] F. Gustafsson, *Statistical Sensor Fusion*. Studentlitteratur, 2010.

[6] E. Frisk, *Lecture 06 Ground Vehicle Motion Control*, 2021.

[7] M. Enqvist, T. Glad, S. Gunnarsson m. fl., *Industriell reglerteknik kurskompendium*. Linköping University, 2014.

[8] T. Glad och L. Ljung, *Reglerteori Flervariabla och olinjära metoder*. Studentlitteratur, 2003.

[9] B. Stellato, G. Banjac, P. Goulart, A. Bemporad och S. Boyd, "OSQP: an operator splitting solver for quadratic programs," *Mathematical Programming Computation*, årg. 12, nr 4, s. 637–672, 2020. DOI: 10.1007/s12532-020-00179-2. URL: https://doi.org/10.1007/s12532-020-00179-2.

[10] *ROS Documentation*. URL: http://wiki.ros.org/noetic.