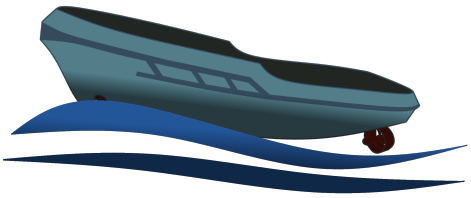
Designspecifikation

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

8 december 2021

Version 1.0



Status

Granskad	Johannes	2021-10-21
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	2021-09-17	Första utkast	Samtliga	Axel, Oscar
0.2	2021-10-11	Andra utkast	Samtliga	Axel, Oscar
0.3	2021-10-13	Tredje utkast	Samtliga	Oscar, Jonatan
0.4	2021-10-17	Fjärde utkast	Samtliga	Johannes
1.0	2021-10-21	Första revision	Samtliga	Johannes

1 INLEDNING

Detta dokument ämnar beskriva hur projektet Modellbaserad styrning av småskalig ytfarkost i kursen TSRT10 kommer att genomföras tekniskt, samt presentera de förkunskaper och metoder som kommer användas.

Farkostens fysiska design är fastslagen sedan tidigare, därav kommer projektet huvudsakligen att fokusera på designen hos mjukvaran i form av modeller, en regulator och ett användargränssnitt. Modeller av typen Greybox och eventuellt Blackbox kommer tas fram för farkosten och till dessa kommer datainsamlingsexperiment att designas med målet att få fram modellernas parametrar på så kort tid som möjligt då detta är högst önskvärt vid användande av större och dyrare farkoster. Färdruttsplanering och modellbaserad reglering kommer att utvecklas och implementeras för autonom körning av farkosten.

2 ÖVERSIKT AV DESIGNEN

Designen av farkosten är uppdelad i olika delsystem. Dessa är farkosten, modellen, sensorfusionen, färdruttsplaneringen, regleringen och användargränssnittet. De sammanfattas kort nedan:

- Farkostens hårdvara och design var färdig vid start av projektet och inga större förändringar tänker göras. Den nuvarande konstruktionen gör det möjligt att kunna samla in data och köra autonomt.
- Fokus för modelleringen ligger på att utveckla och validera en Greybox-modell med tre frihetsgrader men i mån av tid kan även en Blackbox-modell undersökas. Modellen kommer behövas för att utveckla regulatorn i simulering, använda modellbaserad styrning och rörelseplanering.
- Sensorfusionen kommer kunna ge information om farkostens orientering. Denna information är väsentlig vid skattningen av parametrar i modellen samt styrning av farkosten.
- Färdruttsplaneringen bygger på att noder innehållande information om: position, orientering och hastighet för farkosten skapas av en användare. Utifrån dessa noder kommer en färdrutt beräknas med krav på vinkel och hastighet i noderna, dvs färdruttsplaneraren kommer ha gränser för värden såsom kurvatur och acceleration som den måste förhålla sig efter.
- Regleringen är modellbaserad så en bra modell kommer ge bättre kvalitet på regleringen. Med reglering så kommer skeppet kunna följa den beräknade färdrutten och kunna köra autonomt.
- Användargränssnittet kommer göra det enklare för användaren att förbereda, logga data och övervaka en autonom körning. Via användargränssnittet finns också möjligheten att byta från autonom till manuell körning där farkosten då kan styras via tangentbordet.

3 DELSYSTEM

I det här kapitlet presenteras de olika delsystemen i projektet.

3.1 Farkosten

Farkostens utseende och hårdvara ändrades senast av Gustav Zetterqvist och Fabian Steen under examensarbetet [1]. Under tiden mellan examensarbetets slut och detta projekts start har vissa förändringar gjorts. GNSS-mottagarens (Global Navigation Satellite System) uppdateringsfrekvens har ökats från 1 till 8 Hz och utväxlingen mellan styrservona och roderpropellrarnas riktning har minskats så att roderpropellrarna rör sig mindre men noggrannare. Farkosten har en dimension på cirka 0.99m ×0.3m och väger cirka 13 kg, ett par bilder på farkosten kan ses i Figur 1, dessa har hämtats ur Zetterqvist och Steens examensarbete [1].

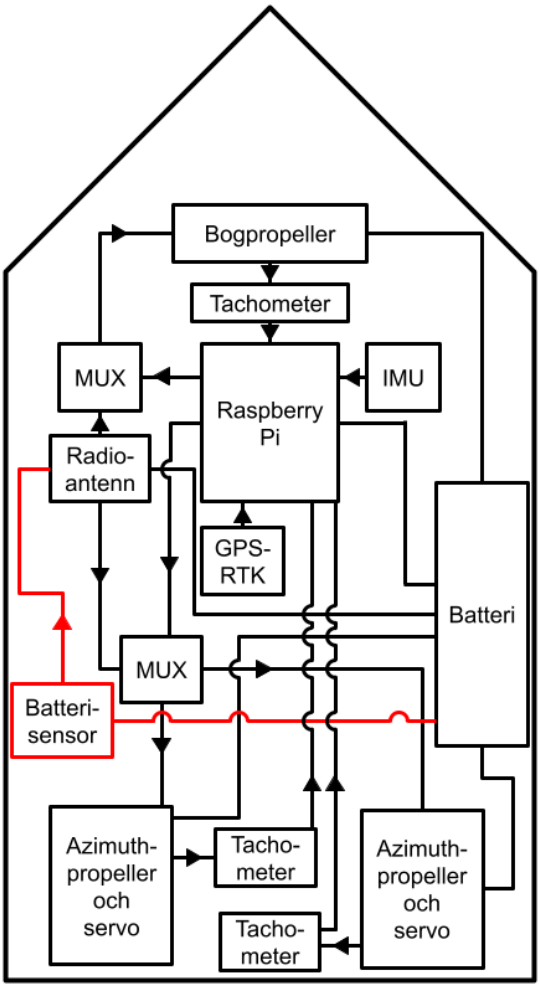
Figur 1: Bild av systemet

Skeppets dimension och vissa komponenters placering kan ses i Tabell 1 och i Figur 2. I början av projektet ser farkostens komponenter ut som:

- Två stycken roterbara roderproppellrar och tillhörande servon. Roderpropellrarna drivs av varsin borstlös 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 med RTK (Real Time Kinematic). Modulen är en u-blox ZED-F9P och har en MagmaX2 extern antenn.
- En IMU (Inertial Measurement Unit) av typen MPU9250 för mätning av vinkelhastighet, acceleration och magnetfält
- Tachometer för propellrarna.
- En X8R - FrSky RC mottagare för användning vid manuell styrning av skeppet.
- En Raspberry Pi 4
- 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 har 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) till Raspberry Pi:en.

- En spänningsregulator som genererar 7.0 volt (och max 20 Ampere) till servomotorerna.

Planerad utveckling av farkosten är att implementera en batterisensor för att mäta batterinivån. En översiktlig systemskiss för de komponenter som finns på farkosten visas i Figur 2. $l_{x,1}$, $l_{y,1}$ och $l_{x,t}$ från Tabell 1 beskriver propellernas position med hänsyn till origo i koordinatsystemet x_b och y_b enligt figur 5.



Figur 2: Systemskiss med befintliga komponenter i svart och förändringar som ämnas utföras i rött.

3.1.1 *Raspberry Pi*

En Raspberry Pi 4 med operativsystemet Ubuntu används som plattform för all mjukvara och ingående moduler. På Raspberry Pi:en kommer modellen och reglersystemet att implementeras och alla beräkningar genomföras. All kod som implementeras på Raspberry Pi ska vara i Python.

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	$l_{x,1} = -0.39$ m, $l_{y,1} = -0.07$ m
Bogpropeller	$l_{x,t} = 0.37$ m
Effektiv roderarea	$A_r = 0.001$ m^2

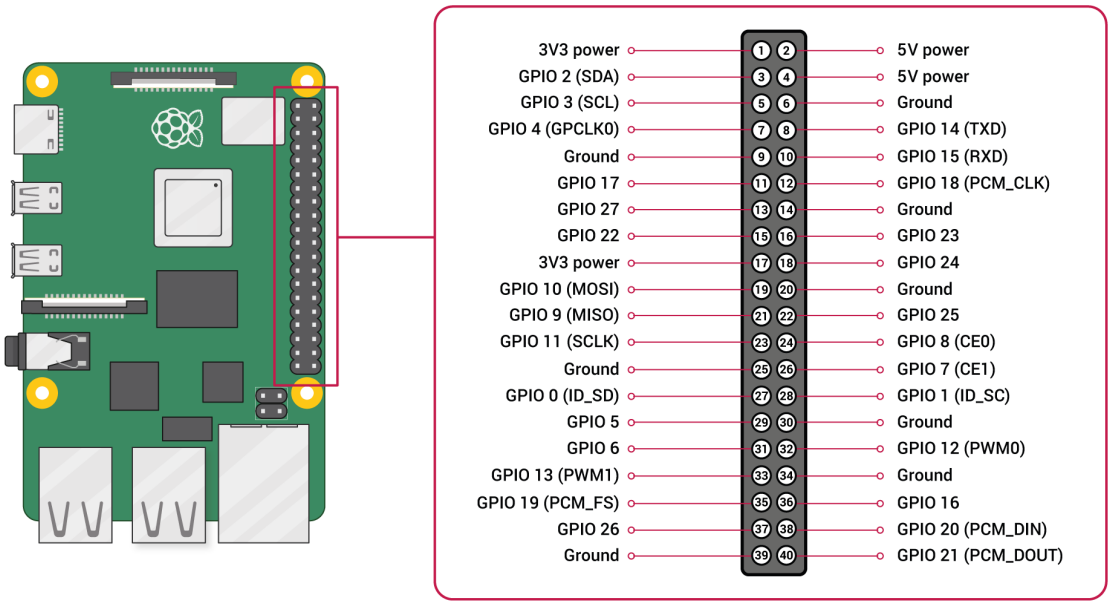
Tabell 1: Tabell för farkostens fysiska egenskaper.

Vid körning så kommer data att lagras på Raspberry Pi:en och om användaren önskar så kan datan även loggas på den externa dator där användargränssnittet körs på.

I Tabell 2 och Figur 3 redovisas alla anslutningar till Raspberry Pi:ens in- och utgångar samt GPIO pinnar. Eventuella kopplingar till batterisensorn inte är med i denna tabell. Jord (GND) skall även anslutas mellan varje modul som kommunicerar med Raspberry Pi:en. Notera även att ESC betyder ”Electronic Speed Controller”eller elektrisk hastighetsregulator på svenska.

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

Tabell 2: Raspberry pi anslutningar



Figur 3: Schema över Raspberry Pi 4b:s pin-positioner.

3.1.2 *Robot Operation System*

Raspberry Pi:en kommer att använda sig av Robot Operating System (ROS) för att kommunicera mellan modulerna och styra hela systemet. Data skickas mellan så kallade ”subscribe”- och ”publish”-noder. Genom dessa noder kan data behandlas och skickas runt. I tabellen nedan redovisas systemets noder.

Nod	Publish topic	Subscribe topics	Script
Ship		ship_command, velocity_command, angle_command	Ship_run.py
IMU	imu/data_raw, imu/mag, imu/mag_calibrated		IMU_talker.py
GPS	gps_chatter		gps_talker.py
RPM	rpm_chatter		rpm_talker.py
Sensorfusion	imu/sens_fus	gps_chatter, imu/data_raw imu/mag_calibrated	Sensorfusion.py
RTCM		gps_rtk/rtcm_data	rtcm2gps.py

Tabell 3: Noder i ROS-nätverket med tillhörande topics

3.2 Modell

Den Greybox-modell som utvecklats av Zetterqvist och Steen [1] skall utvärderas och valideras. För att skatta parametrarna används lämpligen minstakvadratmetoden på skattningsfelet, detta innebär att man vill minimera (1), enligt Ljung och Glad [2].

$$V_N(\vartheta 0) = \frac{1}{N}\sum_{k=1}^{N} \|y(t_k) - \hat{y}(t_k|\vartheta 0)\|_2^2 \tag{1}$$

Där $\vartheta 0$ är parametrarna som ska skattas, y är mätvärden på tillståndet (mer specifikt skattningsdata), N är antal mätvärden och $\hat{y}$ är det modellerade tillståndet. Det $\vartheta 0$ som ger minsta $V(\vartheta 0)$ är de skattade parametrarna och benämns $\hat{\vartheta 0}_N$, enligt

$$\hat{\vartheta 0}_N = \arg\min_{\vartheta 0} V_N(\vartheta 0) \tag{2}$$

Ljung och Glad [2] skriver att minimeringen vanligtvis görs med hjälp av en iterativ metod baserad på Newton-Raphsons metod. Zetterqvist och Steen [1] väljer att definiera $V_N(\vartheta 0)$ med en viktmatris W, se (3), vilket låter olika fel att viktas olika, vill man att alla fel viktas lika skulle man alltså sätta $W = I$, detta innebär att man gör en WLS-skattning (Weighted Least Square) av parametrarna. W kan alltså ses som en justerbar parameter.

$$V_N(\vartheta 0) = \frac{1}{N}\sum_{k=1}^{N} (y(t_k) - \hat{y}(t_k|\vartheta 0))^T W (y(t_k) - \hat{y}(t_k|\vartheta 0)) \tag{3}$$

Denna skattning av parametrarna $\vartheta 0$ kommer att utföras med hjälp av MatLab genom att skapa en idnlgrey-modell och sedan lösa den med hjälp av lsqnonlin-solvern som skapar en värdefunktion på samma sätt som i (3) och hittar sedan $fixed7"00302\vartheta$ som minimerar denna värdefunktion [3].

Ett alternativt sätt för att skatta parametrarna istället för att använda en idnlgrey-modell är att göra som Zetterqvist och Steen [1] och använda sig av multiple shooting för att estimera parametrarna.

Båda de föreslagna metoderna kräver initiala gissningar för parametrarna, hur bra dessa gissningar behöver vara är inte känt men det bör räcka med de initiala gissningar som använts av Zetterqvist och Steen [1]. Alla parametrar bör kunna bestämmas på en gång så länge data som innehåller mätvärden som kan jämföras med predikterade värden, alltså att y mäter samma sak som modellen avser att prediktera, det vill säga $\hat{y}$. För att skattningen av parametrarna ska vara bra behöver datan excitera alla relevanta moder i systemet vilket kommer undersökas när optimala datainsamlings test ska tas fram. En vidareutveckling i denna aspekt vore att utöka dessa vektorer att även innehålla hastigheterna u och v, detta har undersökts lite grann och kommer sannolikt göras genom att de skattas med sensorfusion av GNSS med RTK och accelerometerdata. För att vidareutveckla den befintliga modellen så är troligtvis en bra utgångspunkt att se om det går att få modellen att fungera med uppmätta data även då roderkrafterna, $\tau 0_r$, modeleras enligt det som Zetterqvist och Steen föreslagit [1], i detta fall kan man sedan även se över ett par approximationer som gjordes för denna del av modellen i [1].

Den befintliga modellen kan användas som grund för en simuleringsmiljö där ytterligare modeller samt ett regleringssystem kan utvecklas. I det här kapitlet återges en beskrivning av den ovan nämnda modellen, alla ekvationer är inhämtade från [1] och [4]. Simuleringsmiljön kommer att implementeras i MatLab/Simulink

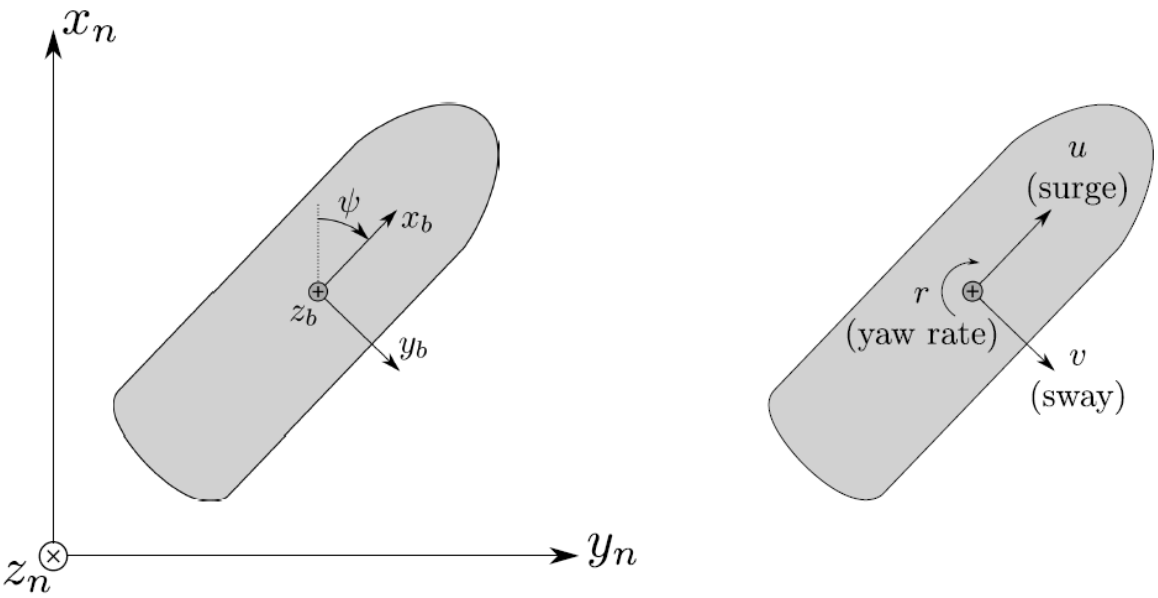
medan modellen sedan kommer att konverteras till Python för att implementeras på fartygets system och vara kompatibelt med Robot Operating System. Den befintliga modellen antar lugnt vatten och låga hastigheter för goda skattningar av hastigheter i de tre frihetsgraderna: surge, sway och yaw, vilket syns i Figur 4. Två referensplan beskriver fartygets position och rörelser. Ett statiskt n-frame (globalt koordinatsystem) och ett som utgår från fartyget, b-frame (farkostens koordinatsystem). Med hjälp av dessa kan följande notation användas.

$$\eta 0 = \begin{bmatrix} x_\eta & y_\eta & \psi \end{bmatrix}^T \tag{4}$$

$$\nu 0 = \begin{bmatrix} u & v & r \end{bmatrix}^T \tag{5}$$

Där $\eta 0$ och $\nu 0$ är tillstånden i den tillståndsmodell som har använts och kan ses i Figur 4.
Därtill definieras externa krafter enligt.

$$\tau 0 = \begin{bmatrix} \tau_x & \tau_y & \tau_N \end{bmatrix}^T \tag{6}$$



Figur 4: Beskrivning över det lokala koordinatsystemet

3.2.1 *Kinematik*

Kinematiken för farkosten definieras enligt

$$fixed7"00307\eta = R(\psi)\nu 0 \tag{7}$$

där $\eta 0$ farkostens position, $\nu 0$ är hastigheten i farkostens referensplan och R är rotationsmatrisen enligt

$$R(\psi) = \begin{bmatrix} cos\psi & -sin\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{8}$$

3.2.2 *Kinetik*

Kinetiken är med för att beskriva hur farkosten påverkas av externa krafter och definieras

$$M\dot{\nu 0} + C(\nu 0)\nu 0 + D(\nu 0)\nu 0 = \tau 0 \tag{9}$$

där M, C och D matriser för tröghetsmomentet, corioliskrafter och dämpningen av farkosten. $\tau 0$ är de externa krafterna som verkar på farkosten. M och C har delats upp i en stelkroppsdel och en del för hydrodynamiska effekter enligt

$$M = M_{SK} + M_{HD} \tag{10a}$$

$$C(\nu 0) = C_{SK}(\nu 0) + C_{HD}(\nu 0). \tag{10b}$$

Då båten är symmetrisk längs z_b-x_b-planet, som ses i Figur 4, kan surge frånkopplas från sway och yaw och då farkostens tyngdpunkt är samma som båtens geometriska mittpunkt kan M beskrivas som

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

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.

$C(\nu 0)$ är en matris som kan beskrivas som en funktion av M och v enligt

$$C(\nu 0) := \begin{bmatrix} 0 & 0 & -mv \\ 0 & 0 & mu \\ mv & -mu & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & Y_{\dot{v}}v \\ 0 & 0 & -X_{\dot{u}}u \\ -Y_{\dot{v}}v & X_{\dot{u}}u & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & -m_{22}v \\ 0 & 0 & m_{11}u \\ m_{22}v & -m_{11}u & 0 \end{bmatrix}. \tag{12}$$

Den hydrodynamiska dämpningen beskrivs som

$$D(\nu 0) := \begin{bmatrix} d_{11}(\nu 0) & 0 & 0 \\ 0 & d_{22}(\nu 0) & d_{23}(\nu 0) \\ 0 & d_{32}(\nu 0) & d_{33}(\nu 0) \end{bmatrix} \tag{13}$$

$$d_{11}(\nu 0) = X_u - X_{|u|u}|u| \tag{14a}$$

$$d_{22}(\nu 0) = Y_v - Y_{|u|v}|u| - Y_{|v|v}|v| - Y_{|r|v}|r| \tag{14b}$$

$$d_{23}(\nu 0) = Y_r \tag{14c}$$

$$d_{32}(\nu 0) = N_v - N_{|v|v}|v| \tag{14d}$$

$$d_{33}(\nu 0) = N_r - N_{|r|r}|r|. \tag{14e}$$

Värdena på parametrarna för att beräkna $d_{11}(\nu 0)$-$d_{33}(\nu 0)$ 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 M, $C(\nu 0)$ och $D(\nu 0)$ för en konstant surge hastighet vilket leder till

$$\begin{bmatrix} m_{11} & 0 & 0 \\ 0 & m_{22} & 0 \\ 0 & 0 & m_{33} \end{bmatrix} \dot{\nu 0} + \begin{bmatrix} X_u & 0 & 0 \\ 0 & Y_v - Y_{|u|v|}|U_0| & Y_r + m_{11}U_0 \\ 0 & N_v - (m_{11} - m_{22})U_0 & N_r \end{bmatrix} \nu 0 = \tau 0. \tag{15}$$

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

$$X_u > 0 \tag{16a}$$

$$Y_v > 0 \tag{16b}$$

$$N_r > 0 \tag{16c}$$

$$(Y_v - Y_{|u|v|}|U_0|)N_r - (m_{11}U_0 + Y_r)(N_v - (m_{11} - m_{22})U_0) > 0. \tag{16d}$$

Enligt Zetterqvist och Steen [1] gäller det att om sista ekvationen i (16) 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 $m_{11} < m_{22}$.

3.2.3 *Propellerkrafter*

De krafter som verkar på fartyget under färd definieras enligt

$$\tau 0 = \tau 0_c + \tau 0_{env} \tag{17}$$

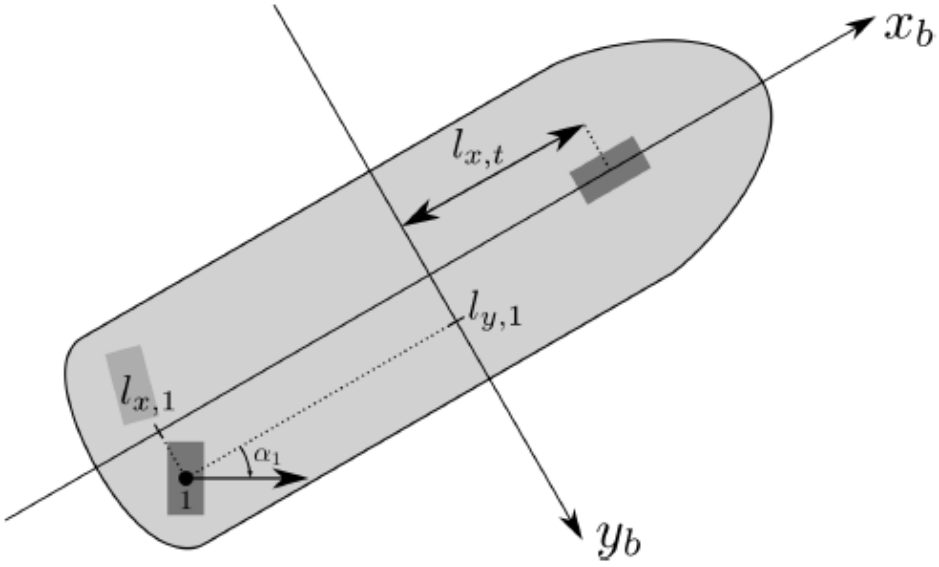
där modellen har förenklats till att endast de krafter som orsakas av proppellrarna påverkar. Alltså är $\tau 0_{env} = 0$ och kraftvektorn ges då av

$$\tau 0 = \tau 0_c = \tau 0_{az} + \tau 0_t. \tag{18}$$

där index az är roderpropellrarna och index t är bogpropellern. Roderpropellrarnas bidrag till kraftvektorn ges av ekvation (19) där μ_{az} är en positiv konstant.

$$\tau 0_{az} = \mu_{az} \begin{bmatrix} \sum_{i=1}^{2} n_i^2 \ cos(\alpha_i) \\ \sum_{i=1}^{2} n_i^2 \ sin(\alpha_i) \\ \sum_{i=1}^{2} n_i^2 \ (l_{x,i} sin(\alpha_i) - l_{y,i} cos(\alpha_i)) \end{bmatrix} \tag{19}$$

Propellrarnas placering och vinkeldefinitionen α_i presenteras i Figur 5. Bogpropellerns bidrag till kraftvektorn ges av



Figur 5: 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 positioner i b-frame är kända, vilket de är (se Tabell 1). I (20) är μ_t en positiv konstant och n_t propellerns varvtal.

$$\tau 0_t = \mu_t n_t^2 \begin{bmatrix} 0 \\ 1 \\ l_{x,t} \end{bmatrix} \tag{20}$$

3.2.4 *Tillståndsmodell*

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

$$\begin{cases} fixed7"00307x = \vartheta f(x,u,0) \\ y = \vartheta h(x,0) \end{cases} \tag{21}$$

där f och h ges enligt

$$\begin{cases} fixed7"00307x = \begin{bmatrix} cos(\psi)u - sin(\psi)v \\ sin(\psi)u + cos(\psi)v \\ r \\ \frac{1}{m_{11}}\left(m_{22}vr - X_u u + X_{|u|u}|u|u + \tau_X(\vartheta u, 0)\right) \\ \frac{1}{m_{22}}\left(-m_{11}ur - Y_v v - Y_r r + Y_{|u|v}|u|v + Y_{|v|v}|v|v + Y_{|r|v}|r|v + \tau_Y(\vartheta u, 0)\right) \\ \frac{1}{m_{33}}\left((m_{11} - m_{22})uv - N_v v + N_{|v|v}|v|v - N_r r + N_{|r|r}|r|r + \tau_N(\vartheta u, 0)\right) \end{bmatrix} \\ y = \begin{bmatrix} x & y & \psi & r \end{bmatrix}^T \end{cases} \tag{22}$$

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

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

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

$$\vartheta 0 = \begin{bmatrix} m_{11} & m_{22} & m_{33} & X_u & X_{|u|u} & Y_v & Y_r & Y_{|u|v} \ldots \\ \ldots & Y_{|v|v} & Y_{|r|v} & N_v & N_r & N_{|v|v} & N_{|r|r} & \kappa & C_N & \mu_{az} & \mu_t \end{bmatrix}^T. \quad \text{Okända parametrar} \tag{26}$$

Där $\vartheta 0$ är en vektor med okända modellparametrar som ska skattas.

3.3 Sensorfusion

Mätsignaler från farkostens IMU och GNSS-mottagare ska användas för att med hjälp av sensorfusion skatta yaw-vinkeln och fartygets hastighet. GNSS-mottagaren har en uppdateringsfrekvens på 8 Hz och publiceras genom ROS. Dess noggrannhet har förbättras genom att använda RTK. Detta minskar felmarginalen markant som uppskattas vara ett par centimeter. IMU-data publiceras genom ROS med en uppdateringsfrekvens satt till 8 Hz. IMU:n har möjligheten att mäta rotationshastigheter, linjära accelerationer och det omgivande magnetfältets styrka och riktning. Mätdata kan sedan sparas på Raspberry Pi:en och kan nås genom att koppla upp en extern dator på samma nätverk som Raspberry Pi:en.

För att få noggranna mätningar från magnetometern, vilket behövs för en bra skattning av orienteringen, behöver magnetometern kalibreras innan användning. Detta görs genom att samla in datan från den då båten roteras ett varv. En ellips anpassas sedan till datapunkterna med hjälp av Minstakvadratmetoden. Ellipsen representeras som den symmetriska 2×2 matris som multiplicerat med alla punkter på enhetscirkeln genererar den anpassade ellipsen. Genom att multiplicera magnetometervärdena med inversen till denna matris fås omskalade, kalibrerade mätningar av magnetfältet.

Från Gustafsson, [5], presenteras en algoritm för ett Kalmanfilter som används för att skatta yaw-vinkel. Den tidsdiskreta tillståndsmodell som används i filtret är

$$\begin{cases} x_{k+1} = x_k + Tu \\ y = x_k \end{cases} \tag{27}$$

där x_k är yaw-vinkeln vid sampel k, u är styrsignalen som i den här modellen är en mätning av rotationshastigheten i x_b-y_b-planet och T är samplingstiden som antas vara konstant. Yaw- och rotationshastigheten

skattas genom datan från IMU. Yaw-vinkeln tas fram genom att beräkna tangens mellan magnetometerdatan i x_b-led och y_b-led medan rotationshastigheten är gyroskop-datan i z_b-led. Denna modell används för att göra tidsuppdateringen av Kalmanfiltret med de värden från sampel k för att estimera vad tillståndet borde vara i nästa sampel.

Tidsuppdatering görs enbart om styrsignalen ligger inom ett rimligt intervall för att bortse från dåliga värden. Tidsuppdateringen vid sampel k görs enligt (28) och (29), som kommer från algoritm 7.1 i [5], där Q är skattning av variansen på bruset för gyroskop-mätnignarna och P är variansen på det skattade tillståndet.

$$\hat{x}_{k+1|k} = \hat{x}_{k|k} + T_k u_k \tag{28}$$

$$P_{k+1|k} = P_{k|k} + T_k^2 Q \tag{29}$$

Nästa del av Kalmanfiltret är att göra en mätuppdatering där det estimerade tillståndet jämförs med mätvärdet och uppdaterar skattning av den verkliga yaw-vinkeln. Likt tidsuppdateringen sker detta enbart om det mätta värdet ligger inom rimliga gränser. Om inget rimligt eller något mätvärde alls noteras så skippas mätuppdateringen och skattningen av yaw-vinkeln kommer vara estimeringen från tidsuppdateringen. Mätuppdateringen vid sampel k görs enligt (30) och (31), som kommer från algoritm 7.1 i [5], där R är skattningen av variansen på bruset för magnetometerdatan.

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + \frac{P_{k|k-1}}{P_{k|k-1} + R}(y_k - \hat{x}_{k|k-1}) \tag{30}$$

$$P_{k|k} = P_{k|k-1} - \frac{P_{k|k-1}^2}{P_{k|k-1} + R} \tag{31}$$

Vid skattningen av hastighet följer ett liknande tillvägagånssätt som för skattningen av yaw-vinkeln och detta Kalmanfilter kommer köras parallellt med det tidigare nämnda. En diskretmodell skapas med hastigheten i x_b- och y_b-led som tillstånd och accelerationen i x_b- och y_b-led som insignal, som

$$\hat{x}_{k+1|k} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \hat{x}_{k|k} + T_k \begin{bmatrix} cos(\psi) & sin(\psi) \\ -sin(\psi) & cos(\psi) \end{bmatrix} u_k \tag{32}$$

$$P_{k+1|k} = P_{k|k} + T_k^2 Q \tag{33}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + P_{k|k-1}(P_{k|k-1} + R)^{-1}(y_k - \hat{x}_{k|k-1}) \tag{34}$$

$$P_{k|k} = P_{k|k-1} - P_{k|k-1}(P_{k|k-1} + R)^{-1} P_{k|k-1}. \tag{35}$$

Accelerationsmätningarna transformeras till artificiella mätningar i det globala koordinatsystemet genom att använda den, av det första filtret, skattade yaw-vinkeln i en rotationsmatris som sedan appliceras på mätningarna. När data från IMU om accelerationen kommer så görs en tidsuppdatering och när mätningen av hastigheten kommer görs en mätuppdatering. För hastigheten i x_b- och y_b-led skapas artificiella mätningar genom att skillnaden i position i x_b- och y_b-led mellan två sampel divideras med samplingstiden.

Sensorfusionsmodulen läser av data från IMU och GNSS med hjälp av ROS genom en så kallad ”subscribe”-nod. Därefter genomförs beräkningar för att sedan skicka vidare data med ROS ”publish”-funktionen så att informationen kan användas av styrenheten. Både rå sensordata och tillståndsskattningar loggas också i en bag-fil som kan läsas och analyseras efter körningen.

3.4 Färdruttsplanering

För att skapa färdrutten och eventuellt definiera färdbart vatten så kommer ett användargränssnitt att användas. Färdrutten för farkosten ska skapas genom att noder med position, vinkel, och hastighet för farkosten ska specificeras. Med ett antal, minst två, av dessa noder utplacerade kommer en färdrutt mellan dem tas fram genom interpolation, som görs på Raspberry Pi:en. För att hitta en färdrutt som är så enkel för båten som möjligt kommer flera interpolationer göras för att sedan välja den som ger minst kurvatur utan att ta för lång omväg. För att räkna ut vad som är för mycket omväg kan ruttplaneraren tillåta att interpolationen är lika lång som det euklidiska avståndet mellan intepolationspunkterna plus en procentsats (till exempel 15%). Denna interpolation kommer skapa en färdrutt bestående av punkter som kommer användas för regleringen.

För interpoleringen kommer kubisk spline i 2 dimensioner användas, vilket innebär att en kubisk spline behöver göras i både x- och y-led. Två noder, k och $k+1$, definieras enligt (36) och (37) där P_x och P_y är positionen i x- och y-led, ρ är vinkeln som farkostens önskas åka igenom noden med och slutligen är v_{abs} den absoluta hastigheten som farkosten önskas åka igenom noden med.

$$N_k = \begin{bmatrix} P_{x_k} \\ P_{y_k} \\ \rho_k \\ v_{abs_k} \end{bmatrix} \tag{36}$$

$$N_{k+1} = \begin{bmatrix} P_{x_{k+1}} \\ P_{y_{k+1}} \\ \rho_{k+1} \\ v_{abs_{k+1}} \end{bmatrix} \tag{37}$$

Mellan dessa två noder kommer två stycken kubiska spline göras som parametriserats med variabeln δ som går mellan 0 och 1 vid nod k respektive $k+1$. Ekvationerna för kubisk spline i x- och y-led ser ut som (38) och (39).

$$x(\delta) = \alpha_1 + \alpha_2\delta + \alpha_3\delta^2 + \alpha_4\delta^3 \tag{38}$$

$$y(\delta) = \beta_1 + \beta_2\delta + \beta_3\delta^2 + \beta_4\delta^3 \tag{39}$$

Det finns åtta okända parametrar som behöver bestämmas där (38) och (39) måste upfylla bivilkoren (40) där L är det euklidiska avståndet mellan noderna.

$$x(0) = P_{x_k} \tag{40a}$$

$$x(1) = P_{x_k+1} \tag{40b}$$

$$y(0) = P_{y_k} \tag{40c}$$

$$y(1) = P_{y_k+1} \tag{40d}$$

$$\nabla(0) = \rho_k \tag{40e}$$

$$\nabla(1) = \rho_{k+1} \tag{40f}$$

$$L \cdot 1.15 > \int_0^1 \sqrt{x'(\delta)^2 + y'(\delta)^2}\, d\delta \tag{40g}$$

$$1 > \frac{x'(\delta)y''(\delta) - y'(\delta)x''(\delta)}{(x'(\delta)^2 + y'(\delta)^2)^{\frac{3}{2}}} \tag{40h}$$

Ekvationerna för att räkna ut längden och krökningen av interpolationen kommer från [6]. För att få olika interpolationer som inte bryter mot bivilkoren får enbart parametrarna α_3, α_4, β_3 och β_4 ändras. Om till exempel α_3 antar värdet 1 så kommer α_4 ändras för att bivilkoren ska uppfyllas, samma relation gäller för β_3 och β_4 också. För att få flera interpolationer kommer parametrarna var för sig och i tur och ordning ansättas värdena 1 och -1. Dessa interpolationer diskretiseras med jämt utspridda diskretiseringspunkter där antalet punkter väljs så att det euklidiska avståndet mellan två punkter inte blir för stort (max 1 cm vid relevant skala). Till sist väljs en interpolation som minimerar en värdefunktion som viktar längden och den summerade krökningen i varje punkt som ser ut som (41) där l_{disk} är det euklidiska avståndet mellan två diskretiseringspunkter och c_{disk} är krökningen i varje diskretiseringspunkt. Parametrarna γ_1 och γ_2 är justerbara parametrar som behöver justeras för att få önskat val av interpolation.

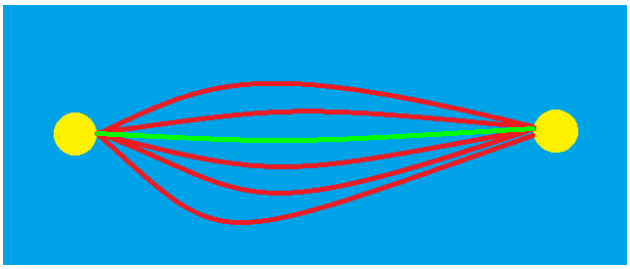
$$V = \gamma_1 \sum l_{disk} + \gamma_2 \sum c_{disk} \tag{41}$$

För en given kubisk spline kommer varje diskretiseringspunkt också att ha ett referensvärde på hastigheten som beräknas med en linjär interpolation mellan referenshastigheten i nod k och $k+1$. Denna interpolation beräknas enligt

$$v_{abs_{ref}}(n) = \frac{N-n}{N} v_{abs_k} + \frac{n}{N} v_{abs_{k+1}} \tag{42}$$

där n är index för en disktretiseringspunkt som går mellan 0 och N,där N är antalet diskretiseringspunkter i en specifik kubiska spline.

Ifall det finns tid, ska systemet utvecklas för att kunna hantera hinder i färdruttsplaneringen. I det fallet kommer det fortfarande att köra flera interpolationer och sedan välja den bästa av de som inte kör på, eller för nära, ett hinder. För att placera ut hinder och definiera vart båten inte kan köra ska användaren genom att dra linjer kunna specificera en yta och sedan välja om det är den omslutna ytan eller komplementet som är icke-färdbart. I Figur 6 syns ett exempel på hur ruttplaneraren skulle kunna planera en rutt mellan två noder genom att interpolera ett antal vägar mellan dem och sedan välja den som passar bäst. Vad som passar bäst beror sedan på omständigheterna, vad har noderna för riktningar, finns det hinder, etc.



Figur 6: Exempelskiss på hur ruttplaneraren skulle kunna fungera.

3.5 Reglersystem

Regulatorn ska kunna följa en sekvens av noder, tillhandahållet från användargränsitt. Regulatorn kommer vara av typen Model Predictive Control (MPC) vilket är en diskret regulator som optimerar styrsignalen över

en begränsad mängd tidsinstanser i framtiden samt hanterar bivillkor på insignaler och tillstånd. Det som ska regleras är avståndet från den planerade rutten, vinkelfelet och hastigheten. För att detta ska fungera väl ställs det höga krav på modellens förmåga att prediktera framtida tillstånd. Om det predikterade tillståndet inte stämmer bra överens med de faktiska framtida tillstånden kommer regleringen inte prestera acceptabelt eller till och med bli instabil. MPC utgår i regel från en tidsdiskret modell. Med den diskreta modellen skattas systemets tillståndet och minimerar en målfunktion med vikt på felaktiga tillstånd och storleken på styrsignalerna under en given mängd tidsinstanser enligt (43), från Enqvist, Glad, Gunnarsson med flera [7].

$$J_N(x(k)) = \sum_{j=0}^{N-1} \|z(k+j) - r(k+j)\|^2_{Q_1} + \|u(k+j)\|^2_{Q_2} \tag{43}$$

$$\text{OCP} \begin{cases} \hat{u}_N & = \arg\min_{u_N} J_n(x(k)) \\ s.t & \begin{cases} x(0) = x_0 \\ \hat{u}_N \in [u_{min}, u_{max}] \text{ for } k = 0, ..., N-1] \\ x \in [x_{min}, x_{max}] \text{ for } k = 0, ..., N-1] \end{cases} \end{cases} \tag{44}$$

I (43) är N mängden tidsinstanser att optimera över och väljs så att samplingstiden gånger N är ungefär insvängningstiden för det önskade slutna systemet. z(k) är de mätbara tillstånden vid tidpunkt k, r(k) är referensvärden vid tidpunkt k, u(k) är styrsignalerna vid tidpunkt k och Q_1 samt Q_2 är viktmatriser som kan användas som designparametrar. Elementen i 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 Q_1 skall vara positiv-semidefinit medan Q_2 skall vara positiv-definit. Målfunktionen optimeras vid varje tidpunkt vilket medför att regleringen blir dynamisk och kan hantera plötsliga förändringar i omgivningen. Optimeringsproblemet är definierat i (44) där styrsignalen i tiden k ansätts som det elementet ur $\hat{u}_N$ som är närmast i tiden.

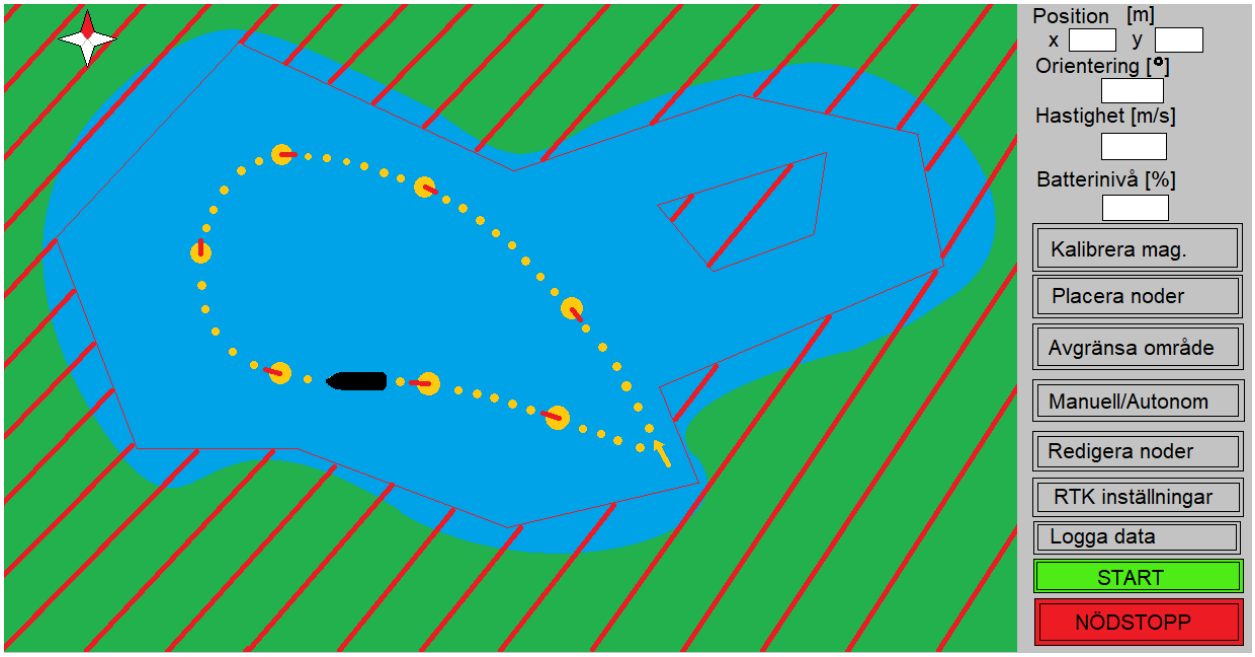
Fördelen med MPC är att bivillkor på insignaler och tillstånd är enkla att implementera samt att med en bra modell kan framtida tillstånd på grund av regleringen predikteras. Nackdelar med MPC är att den är modellberoende vilket ställer höga krav på modellen för att det ska fungera väl. Metoden är också beräkningstung då flera optimeringsproblem måste lösas varje iteration men detta kan hanteras med en kortare tidshorisont. För implementeringen av MPC i Python kommer oqsp, [8], att användas. Det är en ” open source ” bibliotek med verktyg som hanterar tillståndsmodeller och kvadratisk programmering.

Om MPC inte skulle fungera på en acceptabel nivå är alternativa reglerstrategier Pure Pursuit eller framkoppling med PID/LQ. Pure Pursuit är enklast att implementera och bör följa den planerade rutten väl men hastighetsregleringen kan bli problematisk. Med framkoppling så kan den framtagna modellen användas men regleringen bryr sig inte om att den framtida reglering blir bra utan bara att regleringen i en given tidsinstans blir bra.

3.6 Användargränssnitt

Syftet med användargränssnittet är att ge användaren ett enkelt sätt att förbereda och övervaka en körning av farkosten. En konceptskiss på användargränssnittet syns nedan i Figur 7. Då det är en konceptskiss så finns det givetvis en möjlighet att funktioner tas bort eller tillkommer. Idén är att man först avgränsar det område som farkosten kan färdas i, sedan placerar man ut en hamnnod och godtyckligt antal ruttnoder för farkosten att passera igenom. För varje nod kan farkostens önskade hastighet och bäring (orientering)

specificeras, utom för hamnnoden där hastigheten alltid är 0. Ruttplaneraren interpolerar sedan mellan de utsatta noderna, de mindre punkterna i Figur 7, och ger de nya punkterna sin egen önskade hastighet och vinkel. När man är nöjd med färdrutten kan magnetometern kalibreras, RTK startas och båten placeras i hamnen och sedan trycker man på start så ska den autonoma körningen startas. Användargränssnittet kan även användas för att slå på loggning av data och växla mellan autonom och manuell körning. Vid manuell körning styrs farkosten via tangentbordet. Kommunikationen mellan användargränssnittet och farkosten ska ske via Wi-Fi. Användargränssnittet kommer skrivas i matlab för garanterad funktionalitet tillsammans med ROS.



Figur 7: Konceptskiss för användargränssnittet.

4 TEST

Testen kommer initialt genomföras genom att styra farkosten via skriftliga kommandon skickade över SSH (Secure Shell), dessa kommer förberedas i förväg och manuellt skickas med lämpliga intervall. Raspberry Pi:en är sedan tidigare kapabel att logga samtlig data. När användargränssnittet utformats kommer farkosten kunna styras genom det vid testning.

Fjärrkontrollen kommer alltid finnas i beredskap och användas för att manuellt ta kontroll eller nödstoppa farkosten om någon form av risk uppstår.

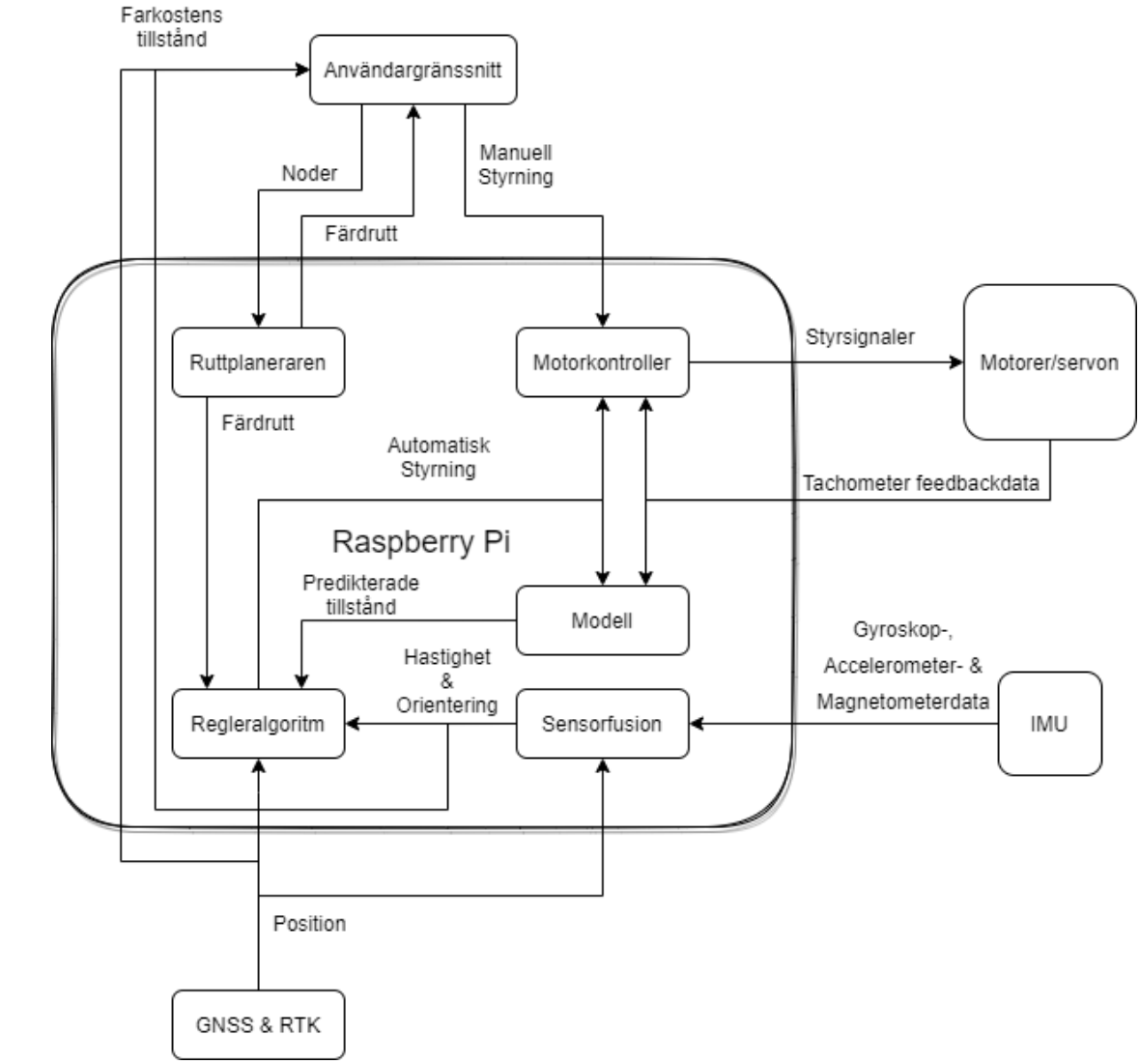
Preliminärt finns tre huvudsakliga test planerade:

- Datainsamling med syfte att genera stora mängder varierande data för att estimera och validera modeller med. Detta sker genom att farkosten styrs enligt ett visst förbestämt sätt under en kortare tid (ca 1 min) definierat i en bash-fil. Flera av dessa bash-filer ska testas för att se vilka styrsignaler som funkar bäst för att bestämma parametrar.
- Modelltestet ämnar undersöka specifika aspekter hos den modell som tas fram och används som en sista verifiering av dess kvalité.
- Slutgiltigt test av reglering och ruttplanering, detta innebär att den färdiga produkten testas. Vid detta test ämnas samtliga krav i kravspecifikationen [9] vara uppfyllda.

Förutom dessa test finns även utrymme att genomföra mer spontana experiment vid behov, till exempel tester i polen på innergården.

5 SYSTEMSKISS

Figur 8 visar hur systemets olika moduler kommer kommunicera med varandra och vad det är för information som de olika modulerna skickar. Värt att notera är att även om positionen från GNSS & RTK inte går igenom någon separat modul innan de anländer till användargränssnittet, så går den igenom Raspberry Pi:en.



Figur 8: Flödesschema för systemets olika moduler.

6 IMPLEMENTATIONSSTRATEGI

Innan datainsamlingen kan börja så behöver magnetometerns kalibreringskod skrivas och kod som hanterar vinkeln på trustrarna skrivas. När dessa är klara kan den första datainsamlingen utföras och därefter initial modellskattning och validering. För att kunna skatta parametrar i modellerna för farkosten behövs även sensorfusion för orientering och eventuellt hastighet. Parallellt med detta ska en simuleringsmiljö skapas där modellen från Steen och Zetterqvists examensarbete [1] ska implementeras för att kunna börja utvecklingen av regleringen. För att använda regleringen ska kod för att ta fram en färdrutt för farkosten skapas. Parallellt

med regleringen ska användargränssnittet utvecklas, detta kommer underlätta testningen och utvärderingen av regleralgoritmen då det kommer vara enklare att definiera en färdrutt och man kommer kunna få en bra översikt om hur bra regleringen följer färdrutten.

Implementation av delar i farkosten är tänkt att ske i små steg där varje implementation ska testas utförligt för att undvika onödiga problem i framtiden. Modellen är beroende av sensorfusionen och regleringen kommer behöva testas med färdrutten så dessa kommer implementeras först. Efter dessa implementationer har testats och fungerar kan allting implementeras på farkosten.

REFERENSER

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

[2] Torkel Glad Lennart Ljung. *Modellbygge och Simulering*. Studentlitteratur, 2004.

[3] URL: https://se.mathworks.com/help/ident/ref/nlgreyestoptions.html.

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

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

[6] Mats Neymark. *Matematisk analys: flera variabler*. Liber AB, 2017.

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

[8] B. Stellato, G. Banjac, P. Goulart m. fl. "OSQP: an operator splitting solver for quadratic programs". I: *Mathematical Programming Computation* 12.4 (2020), s. 637–672. DOI: 10.1007/s12532-020-00179-2. URL: https://doi.org/10.1007/s12532-020-00179-2.

[9] O. Hermansson, O. Jonsson, J. Siönäs m. fl. *Kravspecifikation, Model-based Control of Small-scale Surface Vessel*. 2021.