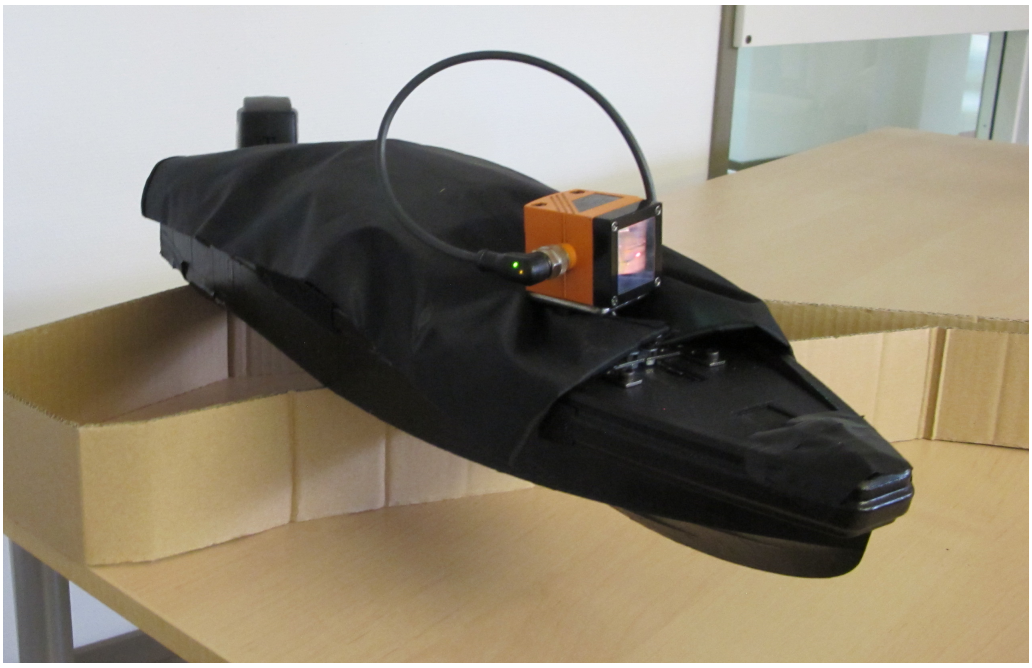
CHALMERS



Reglering av båt vid tilläggning

SSYX05

Michel Company

Fredrik Andersson

Institution för Signaler och System

Avdelningen för reglerteknik, automation och mekatronik

Chalmers tekniska högskola

Göteborg, Sverige 2015

Summary

The purpose of this thesis was to examine the possibilities of achieving some level of autonomy for leasure boats. This was made by designing a radio controlled boat and then develop a control system. The project was carried out at the offices of consulting company Broccoli Engineering in Gothenburg.

The hull have been designed in Catia V5 and then printed in plastic by a 3d-printer. The hull has then been fitted with an electric outboard motor and a bowthruster. For steering, a video game control was used, connected via Bluetooth to a Raspberry Pi. For automated control, an optical distance sensor and an orientation sensor were also connected to the Raspberry Pi.

A graphical user interface was develeoped to visualize the distance ahead that the sensor measures. Lastly control algorithms were developed for distance control towards a dock. Tests were carried out in a pool under ideal conditions.

Sammanfattning

Detta projekt syftade till att undersöka möjligheter att för fritidsbåtar få en nivå av autonomitet, likt moderna bilar. Det gjordes genom att konstruera en radiostyrd båt och sedan utveckla ett reglersystem för den. Projektet utfördes för konsultbolaget Broccoli Engineering på deras kontor i Göteborg.

Båtens skrov har konstruerats i Catia V5 och sedan skrivits ut i plast med hjälp av en 3d-skrivare. Skrovet har sedan bestyckats med en elektrisk utombordsmotor och bogpropeller. För styrning användes en tv-spelskontroll kopplad via Bluetooth till en Raspberry Pi. Till den kopplades också en avståndsgivare och en lägessensor för reglering.

Ett grafiskt gränssnitt togs fram för att visualisera det avstånd som givaren mäter upp framöver, likt en radarbild. Till sist utvecklades regleralgoritmer för reglering av avståndet gentemot brygga. Tester utfördes i en vattenbassäng under ideala förhållande, det vill säga utan vind och vågor.

Förord

Examensarbete är ett av delmomenten i utbildningen till mekatronikingenjör på Chalmers och omfattar 15 högskolepoäng. Följande rapport beskriver examensarbetet utfört på uppdrag hos Broccoli Engineering.

Vi vill tacka vår handledare Tobias Olsson på Broccoli Engineering samt Björn Bergholm och Amela Kobaslic som gav oss möjligheten att få göra vår idé till verklighet.

Ett stort tack går också till Lichron teknikgymnasium som hjälpte oss med 3D-utskrift av båten.

Tack också till Bertil Thomas, vår handledare på Chalmers och Manne Stenberg, vår examinator.

Göteborg 8 Juni 2015

Innehåll

Om inget annat anges så är bilderna författarnas egna.

Förkortningar

RC - Radio Controlled
PWM - Pulse-Width Modulation
CAD - Computer Aided Design
ESC - Electronic Speed Controller
I^2C - Inter-Integrated Circuit
SPI - Serial Peripheral Interface
UART - Universal asynchronous receiver/transmitter
ARM - Advanced RISC Machine
USB - Unversal Serial Bus
HDMI - High Definition Multi-media Interface
MHz - Megahertz
GB - Gigabyte
RGB - Red Green Blue
PLA - Polylaktid
OpenGL - Open Graphics Library
GUI - Graphical User Interface
AV - Actual Value - Ärvärde
SP - Set Point - Börvärde

1 INLEDNING

1.1 Bakgrund

Utvecklingen av autonoma system i bilbranschen går framåt i raska steg så frågeställningen uppstod; varför är båtbranschen så långt efter i utvecklingen?

Broccoli Engineering AB var lika nyfikna på frågeställningen och därmed kunde projektet utföras på deras kontor i Göteborg.

1.2 Syfte

Syftet med projektet är att utvärdera möjligheten att med hjälp av givare och sensorer lägga till en båt vid kaj. Detta för att hjälpa föraren lägga till vid kaj för att undvika person- och materialskador som kan uppkomma när den mänskliga faktorn är inblandad.

1.3 Precisering av uppgift och frågeställning

- Vilka indata behövs för att autonomt reglera båten?
  - Avståndsmätning, gyroskop, accelerometer, radar, GPS?
- Hur stor beräkningskapacitet krävs för att utvärdera indata och beräkna regleralgoritmen. D.v.s. Vad ska användas för typ av microcontroller.
- Vilka drivdon behövs för att styra båten?
  - Räcker det att styra båten med bara en motor eller krävs det också en bogpropeller?
- Kommer resultaten från vårt projekt vara tillräckliga för att verifiera om implementering i riktig båt är möjlig?
  - Är det framtagna systemet tillräckligt pålitligt?
  - Kommer systemet fungera i icke ideala förhållanden?

1.4 Delmål

På grund av begränsade resurser i tid och kapital utvärderas systemet endast på en radiostyrd modell av en båt. Projektet delas upp i ett antal steg där vart och ett agerar som en avgränsning.

1.4.1 Steg 1

Det första målet är att få en fungerande radiostyrd båt och visa upp avstånds-data i ett simpelt grafiskt gränssnitt.

1.4.2 Steg 2

Ta in data från avståndsgivare och reglera utsignalen till motorn och bogpro-pellern så att båten autonomt kan röra sig mot en plan yta (bryggan) och stanna samt hålla sig på ett visst avstånd från bryggan.

1.4.3 Steg 3

Reglera motor och bogproppeller så att båten hittar till bryggan och ställer sig i rätt vinkel emot bryggan samt kompenserar för yttre faktorer så som vind och vågor.

2 VAL AV KOMPONENTER

Det finns idag flera system som kan reglera ett fartygs position med hjälp av olika typer av givare och sensorer tillsammans med GPS. För bilar finns idag autonoma system som hjälper föraren att parkera. Utifrån dessa aspekter föddes projektet att utvärdera möjligheten att autonomt lägga till en fritidsbåt vid kaj.

För att lyckas med detta krävs det ett antal komponenter som nedan beskrivs kortfattat.

2.1 Raspberry Pi

En Raspberry Pi 2 Model B valdes som plattform. Det är en ARM baserad microdator med en 900MHz Quad-Core processor och 1GB RAM. Den har flertalet USB ingångar så att mus och tangentbord lätt kan kopplas in samt en HDMI ingång för en skärm. Detta tillsammans med operativsystemet Raspbian som är ett linuxsystem gör det väldigt lätt att programmera på.

Utöver detta så är den både billig, cirka 350 kr i inköp, och storleksmässigt passar den lätt i modellbåten.

2.2 Adafruit PWM HAT

Adafruits PWM HAT valdes för PWM styrda komponenter, till exempel servon. Styrkretsen heter PCA9685, den är en I^2C-buss kontrollerad 16 kanals kontroller med 12 bitars upplösning.

2.3 AD-omvandlare

Då servona och avståndsgivaren använder analogsignaler krävs möjligheten att ta emot dessa. Detta går inte att göra i microdatorn utan en separat krets krävs. Valet av krets föll på en MCP3204 från Microship. Kretsen har 4 analogingångar

med 14-bitars upplösning vilket är tillräckligt för projektet. Kretsen använder SPI för att kommunicera med andra kretsar vilket är bra då stöd för detta protokoll finns på Raspberry Pi.

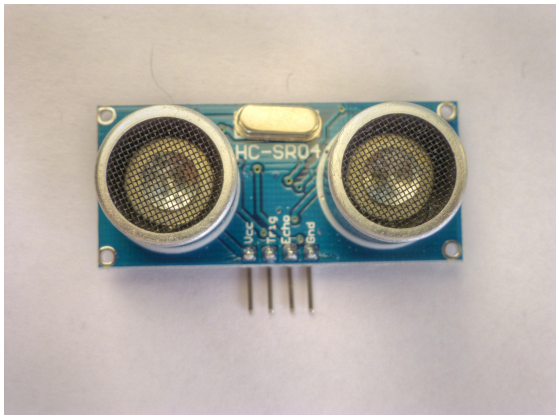
2.4 Lägessensor

För att få mer utförlig orientering av båtens läge, och kunna reglera mot yttre faktorer som vind och vatten, används en "Absolute Orientation Sensor" BNO055 från BOSCH. Detta är en givare som kombinerar tre olika sensorer (accelerometer, gyroskop och magnetometer) genom att utvärdera datan från de olika sensorerna i en integrerad microcontroller kan givaren beräkna sensorns orientering i rymden. Datan kan skickas från givaren via UART eller I^2C.

2.5 Avståndsgivare

För att mäta avståndet fram till bryggan utvärderades två olika avståndsgivare då deras praktiska prestanda var okänd. Den ena använder sig av ultraljud för att mäta avståndet medan andra använder sig av laser.

2.5.1 HC-SR04



Figur 1: Ultraljudsgivare - Bild: Nevit Dilmen

Denna givare ger ett mätavstånd på 2 cm - 400 cm med en noggrannhet på 3 mm. Den fungerar genom att en trigger-puls på 10 μs skickas till den varpå givaren då skickar ut åtta 40 kHz ljudsignaler och räknar tiden det tar tills de studsar tillbaka till givaren. Den har en mätvinkel på 15 grader vilket är bra nog för ändåmålet. Som alternativ går det att montera givaren på ett servo.

2.5.2 ifm O1D00

Figur 2: Optisk avståndsgivare. Bild: ifm electronic - www.ifm.com

Optisk avståndsgivare med ett mätavstånd på 0.2 m - 10 m. Avståndet ges ut från givaren i form av en analog signal på 0V-10V. Den har en sampelhastighet på 50 Hz. Eftersom givarens minsta mätavstånd är 20 cm så får den monteras längre bak i modellbåten, vilket ur tyngdpunktssyfte är att föredra. Givaren ska dock matas med 18V-30V vilket betyder att det isåfall krävs en DC/DC omvandlare.

Valet av avståndsgivare föll på den optiska avståndsgivaren ifmO1D00 på grund av tidigare erfarenhet med just den här modellen och att ultraljudsgivaren var för långsam.

2.6 Styrdon

För att styra båten på ett realistiskt vis används flera olika styrdon. För att styra dessa användes en tv-spelskontroll. Styrdonen beskrivs här kortfattat.

2.6.1 Utombordare

För att mer efterlikna den konstruktion som återfinns i fritidsbåtar valdes en motor monterad i ett hölje som sitter bakom båten likt en utombordare. Modellen heter Graupner GTX 480. Motorn är en borstlös trefasmotor som drivs av en ESC. Varvtalet på motorn regleras genom att skicka olika pulslängder till ESC, detta görs via en PWM signal. För att styra roderriktningen av utombordaren används ett servo. Dess vinkelposition styrs av en PWM signal där längden av pulsen på signalen styr vinkeln på servot. Det har också en återkoppling där ärvärdet av vinkeln ges av en analogsignal.

2.6.2 Electronic Speed Controller

Då motorn är en trefasmotor så måste den drivas av en ESC, Electronic Speed Controller. Valet föll på en SC8-RTR ESC. Den är vattentät och kyls av en fläkt istället för vattenkylning som flera andra ESC. Detta underlättar då installation av vattenkylning kan skippas. ESCn har olika lägen som går att programmera. Den går att köra fram och bak utan att behöva programmeras om. Den har också flera säkerhetsfunktioner som Low voltage cut-off skydd, överhettningsskydd, signalförlustskydd och motorblockeringskydd.

2.6.3 Bogpropeller

För att underlätta regleringen av båten används en bogpropeller. Det är en mindre propeller monterad i fören av skrovet för att underlätta både kursändring och sidledsförflyttning. Valet föll på en Bow Thruster 108-20. Detta är den minsta bogpropellern för RC modeller på marknaden. Diametern på tuberna är 12 mm. Konstruktionen består av en tub på vardera sida med en impeller i mitten. Den drivs av en borstad likströmsmotor som styrs med en PWM signal.

2.6.4 Servo

Valet av servo föll på Batan S1213 eftersom det är ett analogt återkopplat servo. De matas med 5 VDC och styrs av en PWM signal där längden av pulsen reglerar vinkeln på servot. Rotationsvinkeln är cirka 180° och längden på pulserna är cirka 500 ms - 2500 ms. Det skickar också tillbaka en analog spänningssignal på det aktuella vinkelvärdet.

2.7 Skrov

För skrovet så valdes CAD programmet Catia V5 som programvara för design och konstruktion. Catia V5 valdes på grund av tidigare erfarenheter och goda kunskaper. Inspiration för designen kom från de moderna skroven med rak stäv.

2.8 Mjukvara

2.8.1 Programmeringsspråk

Programmeringsspråket C valdes på grund av tidigare erfarenheter och goda kunskaper i språket. Dess snabbhet vägde också in i beslutet.

2.8.2 Grafiskt gränssnitt

För att kunna skapa ett grafiskt gränssnitt så användes biblioteket GTK+ 3. Det är en gratis multi-plattforms verktygslåda, skrivet i C och består av en mängd underbibliotek. Det fungerar genom att man skapar och placerar moduler, så kallade Widgets, med diverse funktioner så som: fönster, knappar, områden för att rita linjer och så vidare. Dessa Widgets kan man sedan ge händelser till. Väldigt lättanvändbart.

2.8.3 Programvara

MATLAB Simulink valdes som programvara för framtagande av regleralgoritmer på grund av tidigare erfarenheter och goda kunskaper med programvaran.

3 METOD

På grund av begränsade resurser i tid och kapital utvärderas systemet endast på en radiostyrd modell av en båt. Det betyder att motorn är elektrisk istället för bensin/diesel driven men reglertekniskt är det ingen större skillnad då dagens moderna utombordare är elektroniskt styrda. Tester utfördes i en vattenbassäng med dimensionerna 3x1,8 meter.

4 GENOMFÖRANDE

Projektet delas upp i flera steg som där varje moment successivt ökar autonomiteten av modellen.

4.1 Koppling och kommunikation

Innan de reglertekniska bitarna av projekten börjar så måste alla komponenter kopplas samman och fungera ihop.

4.1.1 Elschema

Systemet drivs av ett 7.2 V batteri. Då systemet behöver tre olika spänningsnivåer omvandlas spänningen till 5 VDC och 24VDC. 5 VDC används till matning av Raspberry Pi, AD-omvandlaren samt matning till PWM kortet. 24 V används till avståndsgivaren. 7.2 V används för drivning av bogpropellern och utombordaren.

4.1.2 Kommunikation

Raspberry Pi använder flera olika protokoll för att kommunicera med de andra kretsarna. SPI används för att läsa mätdata från analogingångarna på AD-omvandlaren.

I^2C används för att styra PWM utgångarna från PCA9685 kretsen som sitter på PWM-haten.

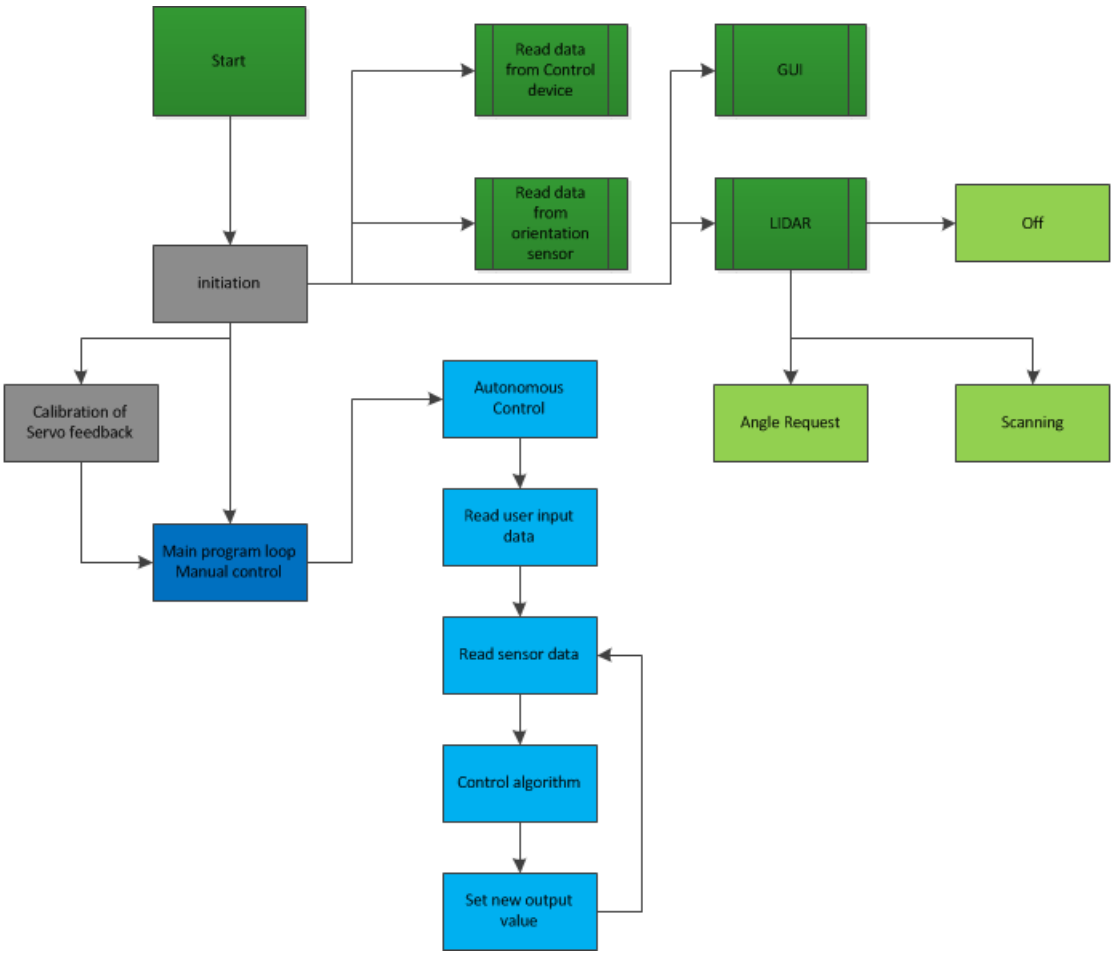
UART används för att skicka och ta emot data från lägessensorn.

4.2 Programmet

Figur 3 visar en översiktsbild över programmets struktur. Det första som sker när programmet startar är initiering och konfiguration av den hårdvara som

används. Därefter startar ytterligare 4 trådar för hantering av GUI, läsning av data från lägessensorn, avståndsmätningshantering samt läsning av kontrollenheten. Därefter kan användaren välja att starta programmet eller göra en inställning av servonas ändlägesposition.

När programmet är startat och huvudloopen körs kan användaren manuellt styra båten eller välja att låta regleraloritmen styra båten mot en vald position nära en brygga.



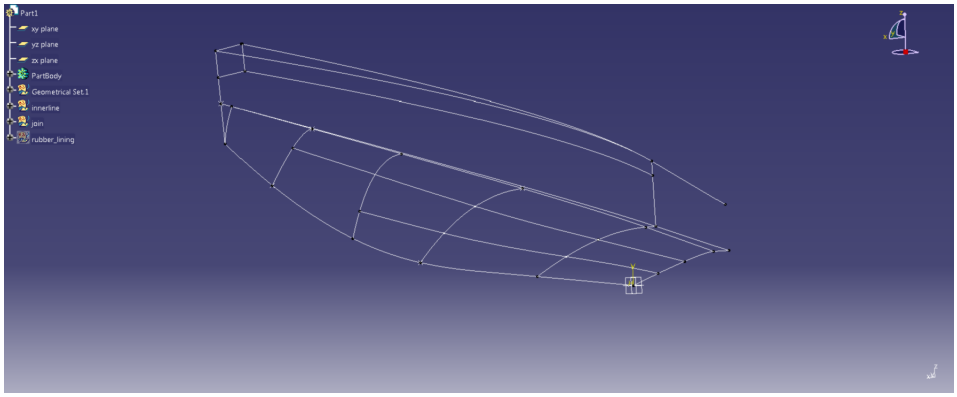
Figur 3: Översikt av programmet.

4.3 Konstruktion av skrov

Innan skrovet ritades upp så samlades information om längd, bredd och vinkel på v-botten in från moderna båtar. Denna information användes sedan som riktlinjer när modellskrovet ritades upp.

4.3.1 Skelettstruktur

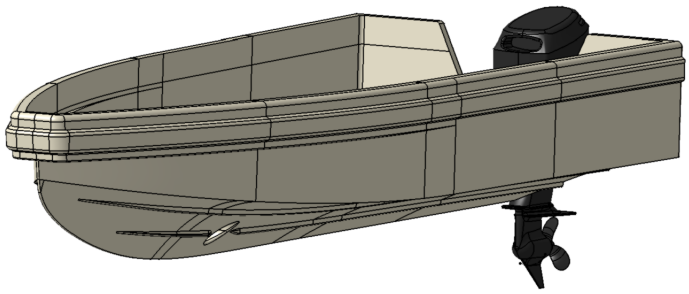
Skrovet börjar livet som en struktur av linjer som ritas med hjälp av Splines-verktyget i Catia V5. Köl ritades först i en lämplig längd och sedan lades sido-splines in med vissa mellanrum.



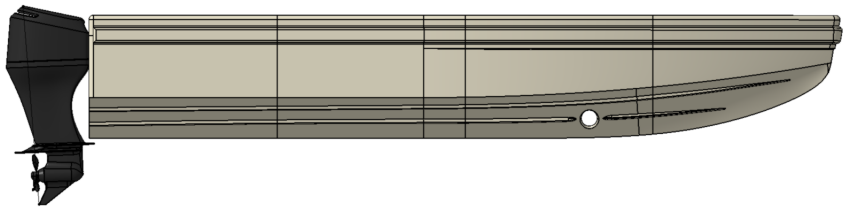
Figur 4: Båtens struktur.

4.3.2 Ytor

Med hjälp av linjerna byggdes sedan ytor upp. Detta gjorde med verktygen Fill och Multi-Section Surface. För att få en jämn och fin yta så användes främst G2-continuity (Curvature). Först togs ytterytan fram och sedan användes verktyget Offset inåt i skrovet för att få innerytan. Dessa två ytorna kopplades sedan ihop för att få en stängd modell.



Figur 5: ISO-vy.

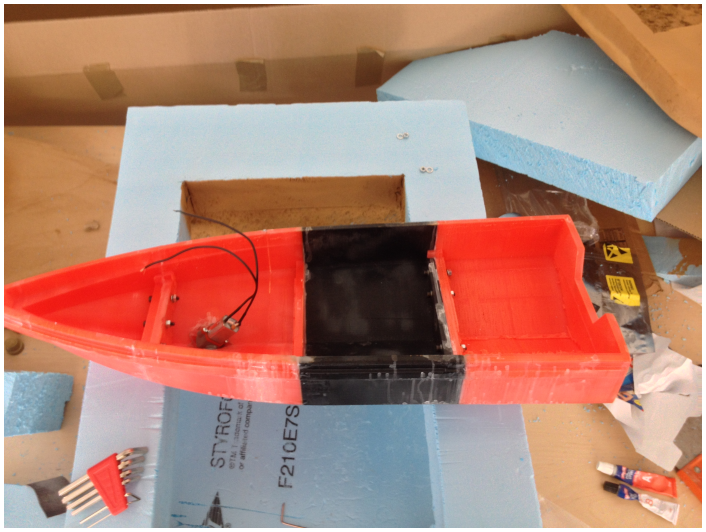


Figur 6: Sido-vy.

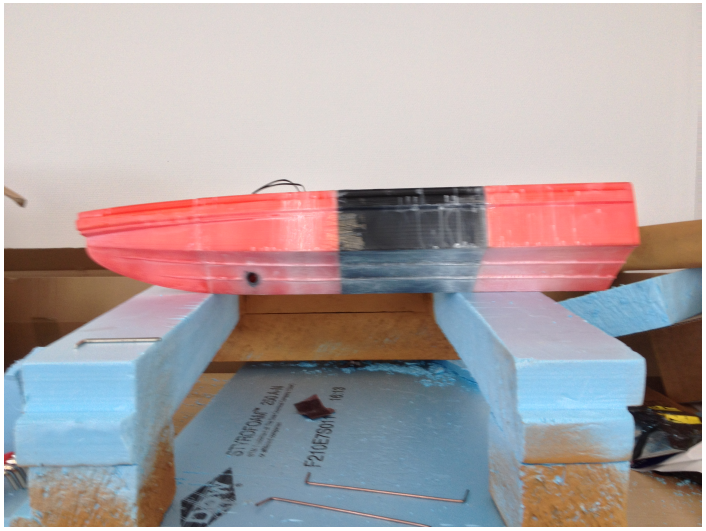
På grund av begränsad volym i 3d-skrivaren så fick skrovet delas upp i fyra ungefärligt lika stora delar och sedan användes skruv och mutter samt 2-komponentslim för att fästa delarna i varandra.

4.3.3 Färdig båt

Den färdiga skrovet anlände i 4 delar av PLA-plast. De målades sedan med en primerfärg avsedd för plast.



Figur 7: Skrovbild ovanför (inklusive bogpropellermotor).



Figur 8: Skrovbild sida.

Skrovet målades sedan med en svart färg för att göra den mer estetiskt tilltalande samt för ytterligare tätning.

Figur 9: Skrovbild färdigmålad.

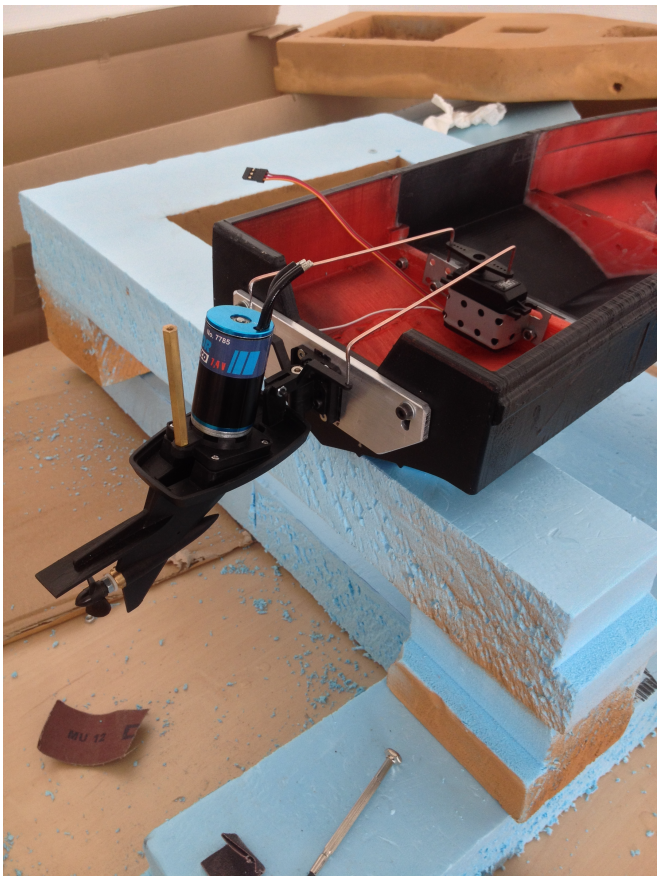
Den färdiga båten har dimensionerna cirka 580x200x100 mm

4.4 Brygga

En brygga skars ut i Styrofoam med en form som skall efterlikna en verklig brygga med y-bommar.

4.5 Integrera hårdvaran med modellbåten.

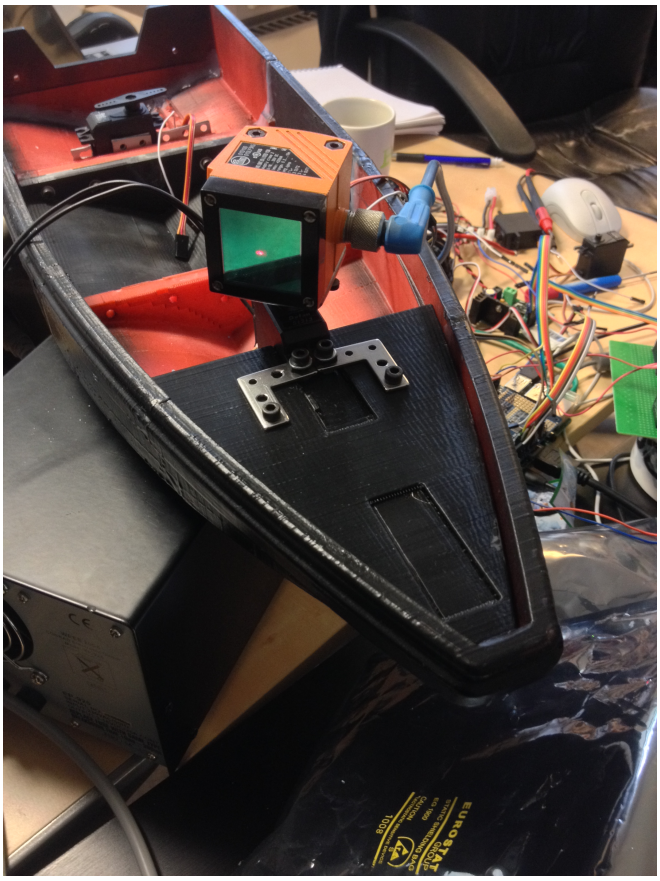
Med skrovet färdigbyggd och målat återstod att installera all hårdvara i den. Först ut var bogproppellern. Hål i skrovet var ritat i CAD med korrekt diameter så det som återstod var att fästa den med 2-komponentslim. Sedan användes badrumssilikon på insidan för ytterligare tätning.



Figur 10: Installation av motor.

Motorhöljet är kopplat till ett servo med två metallpinnar för styrning. Fästet sitter fastskruvat på en aluminiumplatta med skenor för skruvarna så att höjden på motorn går att justera. Detta är viktigt så att propellern får arbeta i ett så lite turbulent vattenflöde som möjligt.

Istället för en RC-kontroll används en tv-spelskontroll för att styra båten. Kontrollen har två små joystick-spakar där den ena styr servot som är kopplat till motorhöljet. Ett analogt reglage styr utsignalen till själva motorn, likt ett gasreglage och en digital knapp sköter växlingen mellan framåtdrift och backdrift. Den andra joystick-spaken används för att styra bogpropellern. Kontrollen kopplas med en Bluetooth-adapter till Raspberry Pi.



Figur 11: Avståndsgivare monterad i fören.

I figur 11 visas den utvalda avståndsgivaren monterad på ett servo fram över i båten.

4.6 Grafiskt gränssnitt

För att visualisera data så skapades ett grafiskt gränssnitt i C med hjälp av GTK+ 3 biblioteket. Idén var att relativt enkelt visa avståndet till bryggan grafiskt samt annan nyttig information så som ställvinkel på motorn, kurs och kompass. Detta skall visas på en display, som i en färdig produkt ska sitta i förarkonsolen på en riktig båt.

Ett GTK-fönster skapades först. Sedan fästes ett rutnät till detta fönster för att lättare kunna placera ut de kommande modulerna. En modul för att rita ut 2D-grafik fästes i mitten, ovanför modulen som ritar ut en importerad bild från Catia V5 av båten.

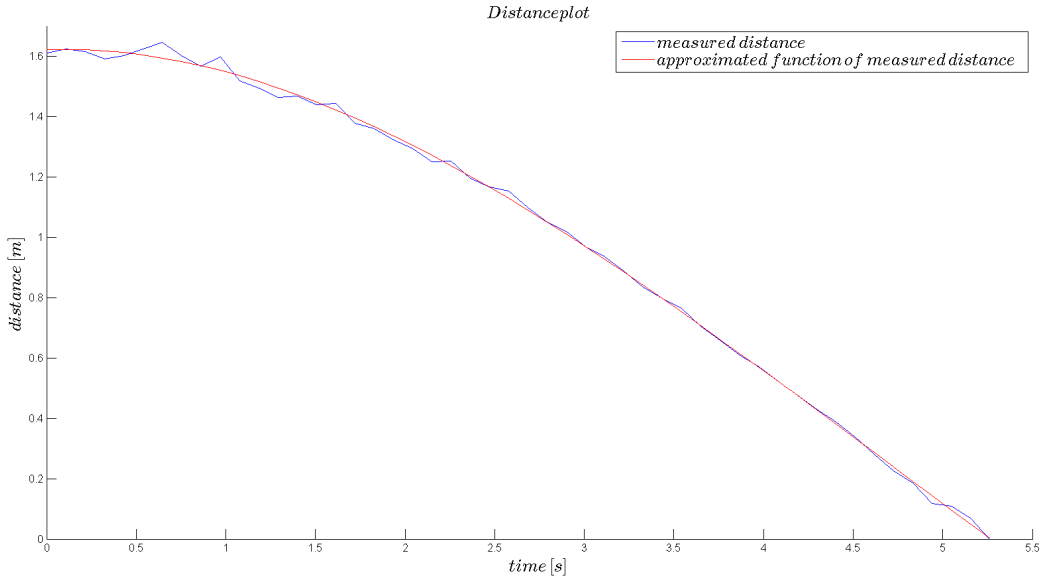
I grafikmodulen ritas avståndet ut till hinder så som bryggan, vilket den optiska avståndsgivaren mäter upp. Det görs genom att låta linjer lika långa som uppmätt avstånd (med en skalningsfaktor) svepa fram och tillbaka. De gamla värdena släcks långsamt ut genom att minska värdet på linjernas RGB-alpha kanal allteftersom, likt en radarbild.

4.7 Framtagning av regleralgoritmer.

Nedan följer beskrivning av hur arbetet med regleralgoritmerna för motor och bogpropeller har fortskridit.

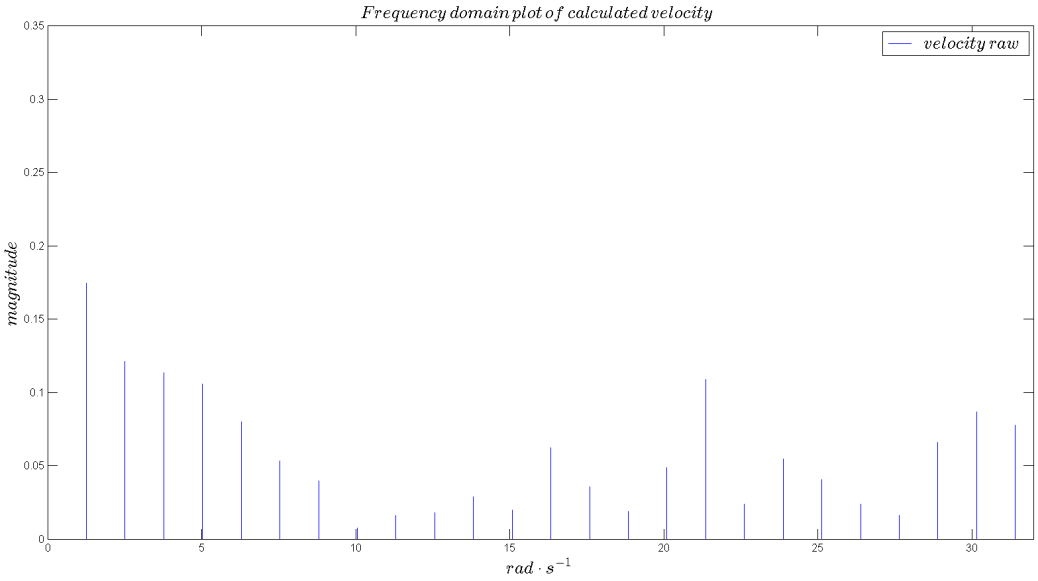
4.7.1 Dimensionering av lågpassfilter för hastighetsmätning

Då den råa indatan från den optiska givaren var för flukturerande för att användas i en regulator implementerades ett lågpassfilter på indatan.



Figur 12: Avståndsplott

Figur 12 visar hur avståndet minskar med tiden då båten kör en sträcka framåt. Här ses både den råa indatan och en approximerad funktion av samma indata. Funktionen är framtagen med hjälp av minstakvadratmetoden.



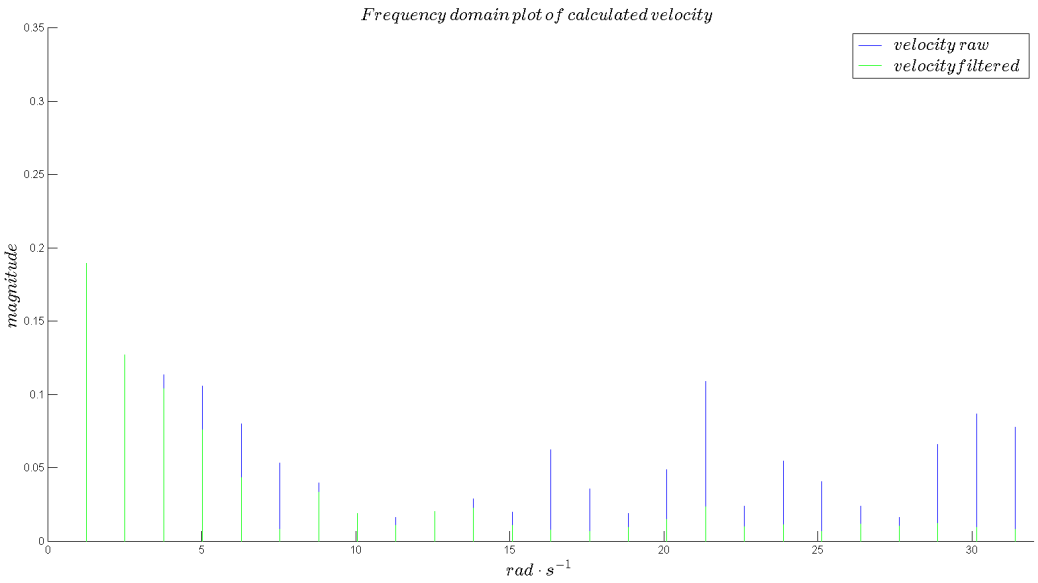
Figur 13: Frekvensdomän analys

I ovanstående graf har Fast Fourier Transform använts på den loggade hastighetsdatan från stegsvarskörningen. Där hastigheten har beräknats genom $\frac{ds}{dt}$ från avståndsgivaren och tiden mellan samplerna. Brytfrekvensen valdes till 5 rad/s då signalen lider av höga frekvenser som skulle filtreras bort. Detta gav värdet 0,2 sekunder på tidskonstanten T. Lågpassfiltrets överföringsfunktion blev således:

$$G_{lp} = (\frac{1}{1 + 0{,}2 \cdot s}) \tag{4.1}$$

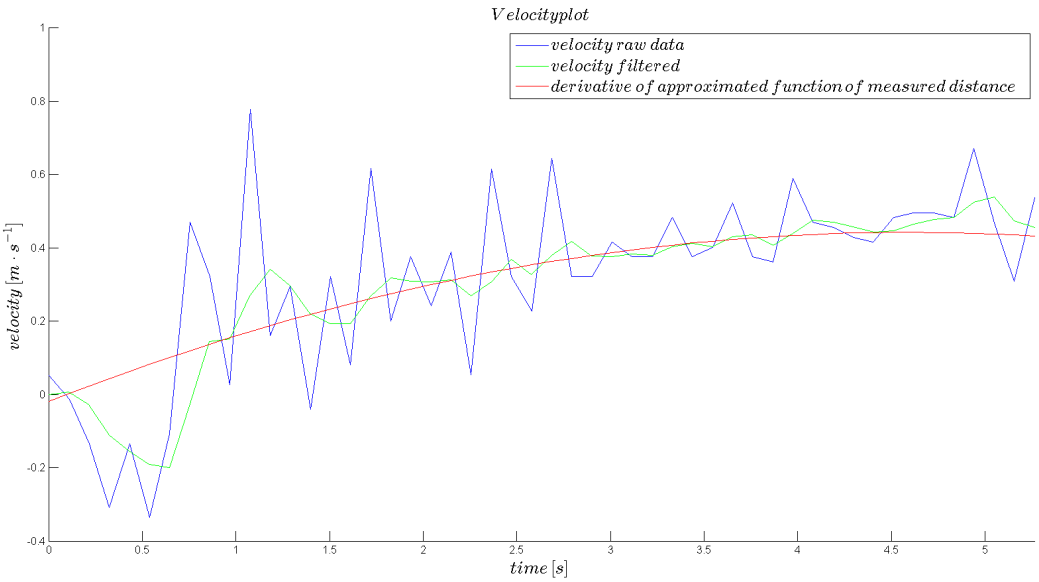
Denna transformerades med bilineär transform med samplingsintervallet 0,1 sekunder som sedan gav följande differensekvation:

$$y[n] = 0{,}6 \cdot y[n-1] + 0{,}2 \cdot u[n] + 0{,}2 \cdot u[n-1] \tag{4.2}$$



Figur 14: Lågpassfiltret - filtrerade frekvenser

Ovan jämförs den lågpassfiltrerade signalens frekvensinnehåll med den råa signalens frekvensinnehåll som även hittas i figur 13.



Figur 15: Hastighetsplott

I figur 15 visas indatan för hastighet, den lågpassfiltrerade indatan för hastighet och en approximerad funktion för hastighet. Den funktionen är framtagen genom att derivera den approximerade funktionen för avstånd som återfinns i figur 12.

4.7.2 Reglering av motor

För att reglera båtens positionering gentemot bryggan måste båtens överföringsfunktion beräknas först. Där insignalen är programmets styrsignal till ESCn och utsignalen är båtens hastighet. För att beräkna överföringsfunktionen gjordes ett stegsvar.

Med stegsvaret för en process eller ett system menas hur utsignalen ändrar sig, som en funktion av tiden, efter en momentan (stegformad) ändring av insignalen.[1]

Stegsvaret gjordes genom att låta båten köra en rak sträcka från ena änden av bassängen till den andra änden samtidigt som den optiska avståndsgivaren samlade in avståndsdata mot en fixerad punkt vid bassängens slutände. Med

hjälp av den insamlade datan kunde ett stegsvar på med båtens hastighet över tid plottas upp. Ur datan approximerades en funktion med hjälp av minsta kvadratmetoden som sedan användes som stegsvar. Detta då utvärderingsmöjligheterna av det verkliga systemet var begränsade.

Ur figur 15 kan man se att stegsvaret motsvarar en första ordningens process med en tidskonstant. Sambandet mellan in- och utsignal kan då beskrivas med differentialekvationen:

$$T \cdot \dot{y} + y(t) = K \cdot u(t) \tag{4.3}$$

Där T är en konstant som motsvarar tiden det tar för stegsvaret att nå 63% av slutvärdet och K är systemets förstärkning.

Laplacetransformering ger:

$$T \cdot Y(s) \cdot s + Y(s) = K \cdot U(s) \tag{4.4}$$

Överföringsfunktionen definieras som utsignalen genom insignalen. Omstuvningen av det matematiska uttrycket ger:

$$\frac{Y(s)}{U(s)} = G_{sys} = \frac{K}{1 + T \cdot s} \tag{4.5}$$

Där T uppmättes till 1,827 sekunder och K till 0,44. Systemet har också en dödtid L som uppmättes till 0,15 sekunder. Överföringsfunktionen för dödtid är:

$$G(s)_{dtid} = e^{-L \cdot s} \tag{4.6}$$

Detta ger den slutgiltiga överföringsfunktionen för systemet:

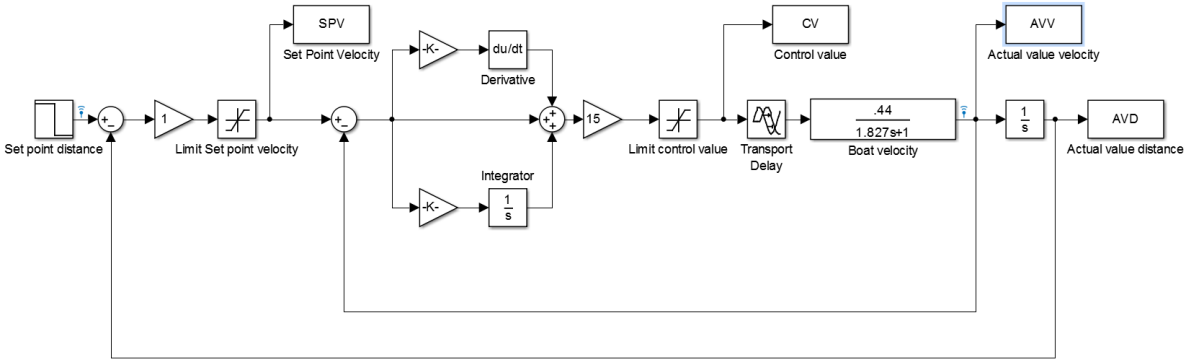
$$G_{sys} = \frac{0{,}44}{1 + 1{,}827 \cdot s} \cdot e^{-0{,}15 \cdot s} \tag{4.7}$$

4.7.3 Dimensionering av regulator

För dimensionering av regulatorn utvärderades några olika metoder på systemet som Ziegler-Nichols stegsvarsmetod, Lambdametoden [2] och Chiens, Hrones och Reswicks 0% översvängsmetod (benämns i fortsättningen som CHR) [3].

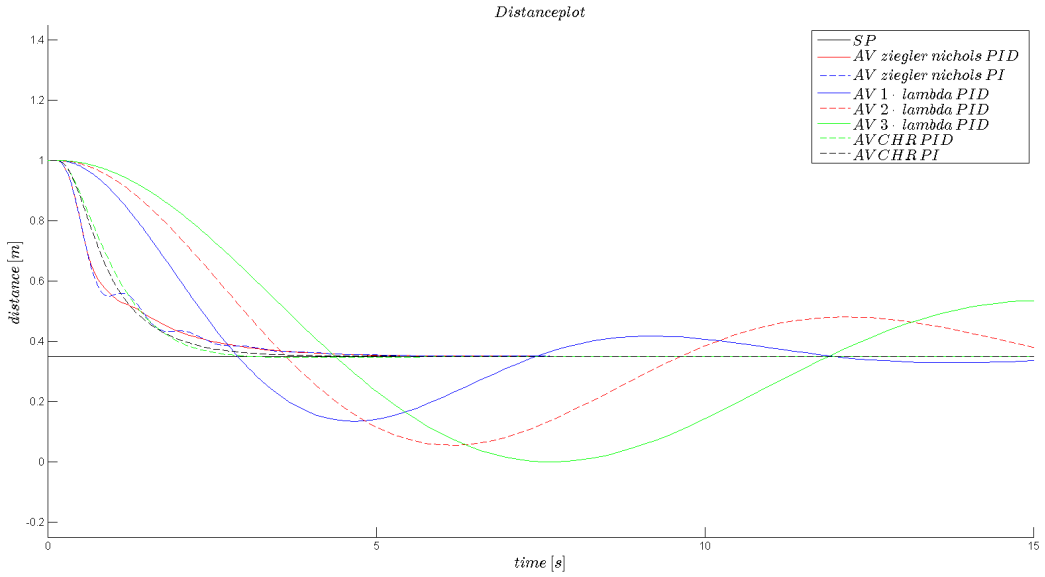
Regulatorns överföringsfunktion ser ut som följande:

$$G_{pid} = K(1 + \frac{1}{T_i \cdot s} + T_d \cdot s) \tag{4.8}$$



Figur 16: Simulink modell av systemet och regulator.

I figur 16 visas Simulink modellen som användes vid simulering av de olika regulatormetoderna. Den inre loopen reglerar hastighet medan den yttre loopen är avståndsreglering. Resultatet av simuleringarna ses nedan i figur 17.



Figur 17: Jämförelse av de olika regulatormetoderna.

Parametrarna för Ziegler-Nichols stegsvarsmetod beräknades enligt:

Regulator	K	T_i	T_d
PI	$\frac{0,9 \cdot T}{K_p \cdot L}$	$3,3 \cdot L$	-
PID	$\frac{1,2 \cdot T}{K_p \cdot L}$	$2 \cdot L$	$\frac{L}{2}$

Där K_p är den statiska förstärkningen, L är dödtiden och T är tiden det tar för processens stegsvar att nå 63% av slutvärdet. Detta gav följande regulatorer:

$$G_{ZN-pi} = 24,9(1 + \frac{1}{0,495 \cdot s}) \tag{4.9}$$

$$G_{ZN-pid} = 33,2(1 + \frac{1}{0,3 \cdot s} + 0,075 \cdot s) \tag{4.10}$$

Parametrarna för Lambdametoden beräknades enligt:

Regulator	K	T_i	T_d
PID	$\frac{0,5 \cdot L + T}{K_p \cdot (\lambda + 0,5 \cdot L)}$	$0,5 \cdot L$ + T	$\frac{L \cdot T}{L + 2 \cdot T}$

Där K_p är den statiska förstärkningen, L är dödtiden och T är tiden det tar för processens stegsvar att nå 63% av slutvärdet. Detta gav följande regulator:

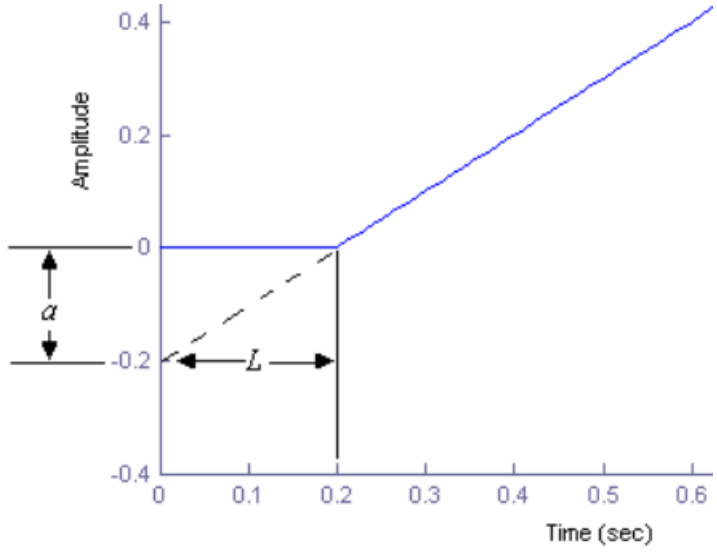
$$G_{Lambda-pid} = \frac{1,9}{0,44 \cdot \lambda + 0,033}(1 + \frac{1}{1,9 \cdot s} + 0,072 \cdot s) \tag{4.11}$$

Där värdet på λ kan justeras så regulatorns beteende ändras enligt:

$\lambda = T$ för aggressiv reglering
$\lambda = 2 \cdot T$ för säker reglering
$\lambda = 3 \cdot T$ för robust reglering

Parametrarna för CHRs metod beräknades enligt:

Regulator	K	T_i	T_d
PID	$\frac{0,6}{a}$	T	0,5·L



Figur 18: Illustrativ beräkning av konstanten a.

Där a räknas ut enligt ovan figur 18, L är dödtiden och T är tiden det tar för processens stegsvar att nå 63% av slutvärdet. Detta gav följande regulator:

$$G_{CHR-pid} = 15(1 + \frac{1}{1,827 \cdot s} + 0,075 \cdot s) \tag{4.12}$$

Ziegler-Nichols metod gav ingen översväng men däremot ett insvängningsförlopp som ej var önskvärt. Lambda metoden gav oacceptabelt beteende oavsett värde på λ. CHRs metod gav däremot ett önskat beteende.

Verkliga tester visade dock att CHRs PI regulator fungerade snäppet bättre än motsvarande PID regulator.

Därmed fick den slutgiltiga regulatorn följande utseende:

$$G_{pi} = 15(1 + \frac{1}{1,827 \cdot s}) \tag{4.13}$$

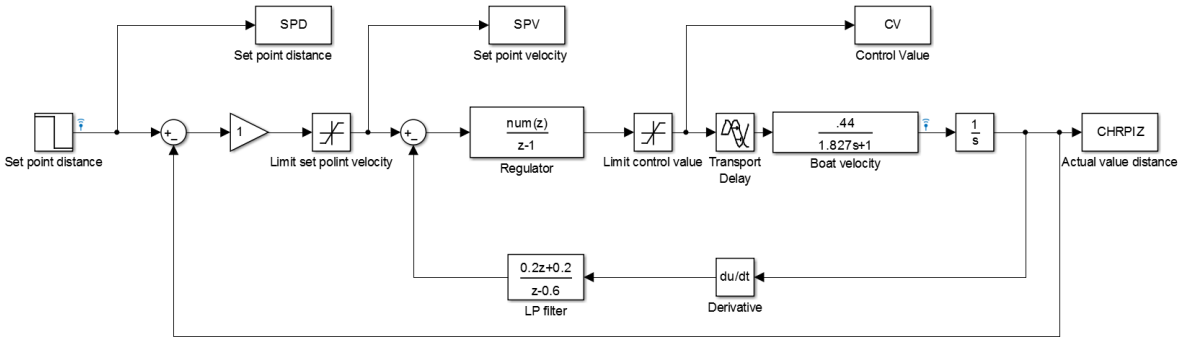
4.7.4 Tidsdiskret transform

Eftersom Laplacetransformen är tidskontinuerlig medan en dator är tidsdiskret krävs det att överföringsfunktionen transformeras. Bilineär transform användes till detta ändamål.

Denna transform, som även kallas *Tustins approximation*, görs genom att ersätta alla s med $2(z-1)/h(z+1)$ i den analoga överföringsfunktionen. [4]

Där h är samplingsintervallet. Detta gav följande differensekvation för samplingsintervallet 0,1 sekunder:

$$y[n] = y[n-1] + 15{,}41 \cdot u[n] - 14{,}59 \cdot u[n-1] \tag{4.14}$$



Figur 19: Simulink modell av systemet och diskret regulator.

4.7.5 Reglering av bogpropeller

För att reglera kursvinkel implementerades en P-regulator. Ett stegsvar gjordes där insignal är PWM-signalen till bogpropellerns motor och utsignal är kursvinkel. Stegsvaret motsvarar en första ordningens process med en tidskonstant. Slutnivån uppskattades till 0,00046. Tiden T det tar för processen att nå 63% av slutnivån uppmättes till 10 sekunder. Dödtiden L uppmättes till 3 sekunder. Detta gav överföringsfunktionen:

$$G_{bog-sys} = \frac{0.00046}{1 + 10 \cdot s} \cdot e^{-3 \cdot s} \tag{4.15}$$

Förstärkningen på regulatorn testades fram till värdet 350.

4.8 Styrning av kurs och riktning under reglering

För att styra båtens kurs mäts den aktuella riktningen med lägessensorn och jämförs sedan med den av användaren valda kursen. Skillnaden mellan värdena visar hur fel båten ligger. För att korrigera felet styrs utombordsmotorn till motsatt kursvinkel om hastigheten är positiv. Om hastigheten är negativ vrids motorn åt samma håll som felet ligger. Detta görs för att ge en vridande kraft åt rätt håll.

Samtidigt används också bogpropellern för att rotera båten mot rätt kurs. Denna styrs med den PI-regulator som tagits fram. Fördelen med att använda bogpropellern är att ingen hastighet framåt eller bakåt krävs för att rotera båten.

4.9 Avståndsmätning under reglering

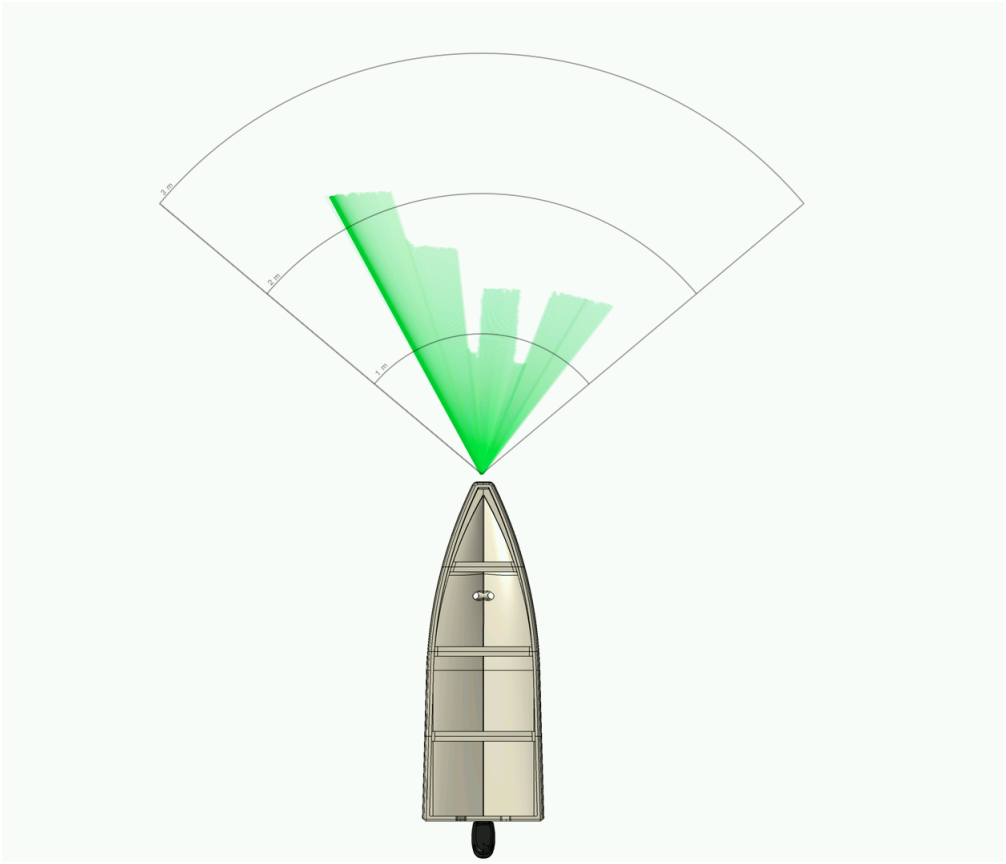
Avståndet fram till den valda positionen beräknas genom att vrida avståndmätaren mot den position användaren valt styra båten till. Givaren mäter då avståndet fram till bryggan och en beräkning utifrån avståndsmätarens uppmätta värde och den position användaren valt styra båten mot görs. Detta värde sparas sen och används i regleringen som det avstånd båten strävar mot att uppnå.

Under regleringen vrider sig givaren hela tiden mot den vinkel som användaren valt. Detta görs på liknande vis som för utombordsmotorn.

5 RESULTAT

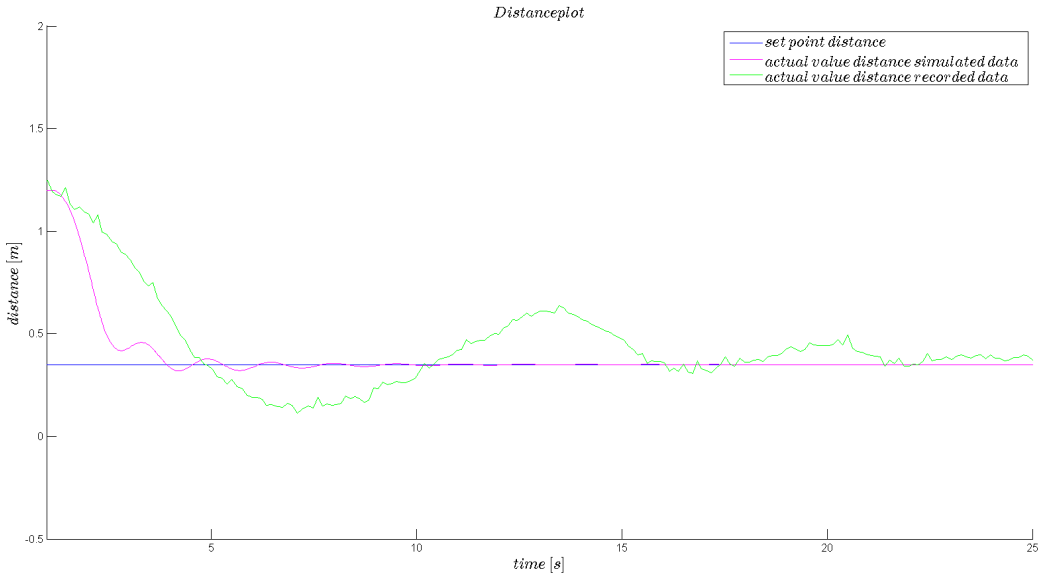
Det första steget var att ta fram en fjärrstyrd modellbåt, mäta avståndet till bryggan samt visa detta genom ett enkelt grafiskt gränssnitt på en display. Detta mål är uppfyllt.

Figur 20: Båten i sitt territorium.



Figur 21: Grafiskt gränssnitt - Här har systemet plottat ut hinder föröver (bl.a. bryggan och dess bommar)

Vidare var steg två att implementera regleralgoritmer för att reglera båtens position gentemot bryggan. Detta gjordes genom att reglera utsignalen till båtens motor. Bogpropellen användes för att hjälpa till vid kursfel.



Figur 22: Avståndsplott mot börvärde

I figur 22 visas hur båten reglerar sitt avstånd mot valt börvärde, i detta fall 0,35 meter. Som synes fås i verkligheten lite större översväng och oscillering kring börvärdet jämfört med den simulerade kurvan. En av orsakerna är att när båten saktar ned så får den sina egna svallvågor i aktern vilket skjutsar den lite framåt. Grafen visar också hur den diskreta regulatorn reagerar på en likadan börvärdesändring i Simulink. Den relativt höga samplingstiden skapar lite besvär för regulatorn vilket ger ett något udda insvängningförlopp.

Det sista steget, nummer 3, var att implementera regleralgoritmer som kunde kompensera för yttre faktorer som vind och vågor genom att använda sensorerna i BNO055-chipet. Tyvärr fanns ej tid för detta.

6 DISKUSSION

Här följer en kort analys av resultatet samt tankar och ytterligare idéer.

6.1 Huvudresultatet

Resultatet av detta projekt visar att det är klart möjligt att utveckla ett reglersystem för fritidsbåtar. Ytterligare utveckling krävs dock för reglering mot yttre störningar vilket, för ett system i en verklig båt, naturligtvis krävs.

För enkel reglering av avstånd emot brygga så fungerar det med en ensam avståndsgivare. För ett autonomt reglersystem som skall lägga till båten utan assistans samt klara av att parera yttre störningar, som till exempel sidovind och vågor, så är lägesgivare ett måste. För ytterligare utveckling så bör man använda en avståndsgivare som mäter i rummet och inte bara i ett plan. Till exempel kan en LIDAR användas för 3D-mappning av området runt båten. Då får systemet en klarare överblick över vad som finns för över. Alternativt kan avståndssensorer installeras både i akter och på sidorna.

Kraftigare hårdvara är också att rekommendera då processorn i Raspberry Pi jobbade vid sin bristningsgräns med processing av all data samtidigt som den hanterade det grafiska gränssnittet. En idé som testades under projektets gång var att skriva det grafiska gränssnittet i OpenGL istället för att använda GTK biblioteket. Detta då OpenGl hanterar grafik mycket bättre. Tyvärr fanns ej kunskaper nog om OpenGl för att få gränssnittet att fungera tillfredsställande. På grund av svårigheter med realtidsstyrning på Rasberry Pi kan det vara en fördel att använda en microcontroller för reglering och en separat dator för det grafiska gränssnittet.

Det var något svårt att mäta upp ett vettigt stegsvar på grund av motorns överdimensionerade kraft och den relativt korta sträcka som systemet kunde utvärderas på. Motorns höga varvtal vid den lägsta utsignalen gjorde även att regleringen blev besvärlig. En linjär acceleration på motorn hade gjort det mycket lättare. Bogpropeller installerades i båten och med facit i hand kan detta rekommenderas. Det gjorde att båten kunde hålla kursen lättare. För att reglersystemet skall kunna klara av yttre störningar så är bogpropeller

absolut nödvändig. Vid skrivandets stund är en PI-regulator under utveckling för bogpropellern då det är viktigt att undvika kvarstående fel vid kursreglering.

Att implementera systemet, som det ser ut idag, i en riktig båt hade fungerat men för ett komplett reglersystem för autonom tilläggning så bör det skrivas en funktion i programmet som väver samman datan för sidledsförflyttning från lägessensorn med regleralgoritmerna. En microcontroller bör användas för att ta hand om reglerberäkningen. Sist men inte minst bör avståndsmätningen ske i 3 dimensioner. En säkerhetsfunktion kan med fördel skrivas i programmet som avbryter regleringsläget och byter till manuell körning vid till exempel stora mätfel under, i sammanhanget, en längre tid eller om båten skulle hamna allt för mycket ur kurs.

Med dessa förbättringar så hade systemet fungerat även i icke-ideala förhållanden.

6.2 Framtida arbeten

Det finns stora möjligheter för fortsatt utveckling av detta projekt. I synnerhet finns det utrymme att utveckla regleralgoritmerna. Processen har olika stegsvar åt olika håll ty skrovet skär vattnet på olika sätt vid framåtdrift och bakåtdrift. På grund utav detta vore det fördelaktigt att implementera en till regulator för bakåtdrift alternativt mäta åt båda hållen och ta ett medelvärde på stegsvaret.

Ytterligare komponenter som kan tänkas installeras är ett riktat ekolod hade kunnat göra det möjligt att mäta upp hinder under vattenyta och därmed göra det möjligt att autonomt lägga till mot en klippa till exempel. Något som vid manuell körning kan ställa till med stora besvär och materialskador.

Andra funktioner som kan tänkas skrivas är till exempel en funktion som lägger båten tryckt mot kajen. Detta för att underlätta av- och ombordstigning eller lastning. Detta är bara en fråga om att skriva funktioner i koden då systemet är flexibelt.

Fler säkerhetfunktioner kan även implementeras. Tanken är dock att detta system, i en riktig båt, ska ha mänsklig övervakning. En manuell avbrytknapp kan då vara fullt tillräckligt. Framtida arbeten utvärderas även med fördel på en riktig båt, i verkliga naturförhållanden.

7 SLUTSATS

Projektet syftade till att undersöka möjligheten till att underlätta tilläggning av fritidsbåt vid brygga. Det råder inga tvivel om att det är möjligt att utföra.

I början av rapporten formulerades ett flertal frågeställningar av uppgiften. Att det skulle krävas indata för reglering var givet och under arbetets gång framkom det att endast avståndsgivare krävdes för reglering av motorkraft gentemot avstånd till bryggan. Datan från BNO055-chipet där bland annat gyroskop och accelerometer ingår behövs för reglering med hjälp av bogpropeller.

Valet av microcontroller föll på Raspberry Pi (egentligen en Single-board computer) och med facit i hand kan det konstateras att för smärtfri körning av det grafiska gränssnittet så behövs något kraftigare hårdvara alternativt goda kunskaper i OpenGL programmering.

En tredje frågeställning var om bogpropeller var nödvändigt. Frågan är intressant då många av dagens fritidsbåtar runt 7 meters klassen ej är utrustade med bogpropeller. Dock syns en markant ökning av bogpropeller som standardutrustning vid nyköp. Modellbåten utrustades därefter med en sådan propeller då den är en viktig kugge i reglering av kurs och emot yttre faktorer.

Sista frågeställningen som ställdes var om systemet är tillräckligt pålitligt. Systemet idag använder till exempel lågpassfilter för att filtrera bort mätfel, vilket är nödvändigt. Det finns dock naturligtvis utrymme för fortsatt förbättring.

Överlag finns det spännande möjligheter till fortsatt utveckling av projektet.

Referenser

[1] Bertil Thomas. "*Modern Reglerteknik*". I: Fjärde Upplagan. Liber AB, 2011, s. 29.

[2] Susanne Lundell. "Undersökning av inställningsmetoder för PID-regulatorer". Examensarb. Chalmers University of Technology, 2012.

[3] Justin Youney. "A COMPARISON AND EVALUATION OF COMMON PID TUNING METHODS". Examensarb. University of Central Florida, 2007.

[4] Bertil Thomas. "*Modern Reglerteknik*". I: Fjärde Upplagan. Liber AB, 2011, s. 347.

A Bilaga 1

```
1  /************************boat.c************************
2   * Program for autonomous boat docking
3   * a Bachelor thesis at Broccoli engineering AB.
4   * Gothenburg Sweden 2015
5   *
6   * Authors: Michel Company, Fredrik Andersson
7   *
8   * Version 1.0
9   *
10  * Date: 2015-06-02
11  *
12  * Chalmers University of Technology
13  * Bachelor of Science in mechatronics engineering
14  *
15  *
16  ***************************************************/
17
18
19 /********************libraries************************/
20 #include <bcm2835.h>    /*C library for Broadcom BCM 2835
       as used
21                 in Raspberry Pi. By Mike McCauley*/
22 #include <stdio.h>      //input/output
23 #include <stdlib.h>     //Standard library
24 #include <string.h>     //String handling
25 #include <stdint.h>     //integer types
26 #include <unistd.h>     //Used for UART
27 #include <fcntl.h>      //Used for UART
28 #include <termios.h>    //Used for UART
29 #include <wiringPi.h>   /*Useful functions for the
       raspberry pi.
30                 By Gordon Henderson*/
31 #include <wiringPiSPI.h>  /*Useful functions for the
       raspberry pi SPI
32                 interface. By Gordon Henderson*/
33 #include <errno.h>      //Error numbers
34 #include <sys/ioctl.h>
35 #include <linux/joystick.h> /*for reading joystick input
```

```
    devices.
36                By Vojtech Pavlik*/
37  #include "PCA9685.h"    //Register definitions for PCA9685
38  #include "bno055.h"     //Register definitions for BNO055
39  #include "DualShock3.h"   //Register definitions for DS3
40  #include <math.h>     //Math functions
41  #include <gtk/gtk.h>    //GTK library
42  /*****************************************************/
43
44
45  //gcc -Wall -o "%e" "%f"  -lbcm2835 -lwiringPi `pkg-config
        --libs gtk+-3.0`  `pkg-config --cflags gtk+-3.0`  -lm
46
47  #define SQUAREfil 0
48  #define CIRCLEfil 1
49  #define TRIANGLEfil 2
50  #define CROSSfil 3
51  /********************Definitions**********************/
52  #define PI 3.14159265   //PI definition
53  #define pi 3.14159265   //pi definition
54
55
56  #define JOY_DEV "/dev/input/js0" //file location for DS3
       input data.
57
58  //Thread keys
59  #define I2cKey 0    //Thread locking key for I2C
60  #define SPIKey 1    //Thread locking key for SPI
61
62  //GUI definitions
63  #define GTKCenterX 0
64  #define GTKCenterY 0
65  #define BoatImgWidth 165
66  #define BoatImgHeight 479
67  #define gtk_distance_factor 175
68
69  //Communication definitions
70  //SPI
71  #define CS_MCP3204  6       // PIN for chip Select
72  #define SPI_CHANNEL 0   //channel
73  #define SPI_SPEED   10000   // 1MHz
```

```
74
75 //I2c
76 #define I2CPWMAdress 0x40 //I2C address for PCA9685
77
78 //PWM Channel
79 #define RudderAngleServo 0
80 #define BowThrusterLeft 10
81 #define BowThrusterRight 11
82 #define OutboardMotor 3
83 #define LIDARAngleServo 4
84 //ADC Channel
85 #define LIDARDistanceSensor 0
86 #define LIDARAngleServoFeedback 1
87 #define RudderAngleServoFeedback 2
88
89 //Direction
90 #define Forward 1
91 #define Reverse -1
92
93 //BowThruster
94 #define BowThrusterEnable 16 // enable pin
95
96 //BNO055
97 #define BNO055Reset 21 //  PIN for BNO055 reset
98
99 //LIDAR mode
100 #define Off 0
101 #define Scanning 1
102 #define AngleRequest 2
103 /*******************Global variables***************/
104 //GUI variables
105 //GTK window
106 gint ScreenWidth = 0 ;
107 gint ScreenHeight = 0;
108
109 //GTK Drawing
110 int DrawingWidth = 0;
111 int DrawingHeight = 0;
112
113 //Pointer to image
114 cairo_surface_t *image;
```

```
115
116 //User Set Points
117 float UserSetPointX = GTKCenterX;
118 float UserSetPointY = GTKCenterY/2;
119
120 //Program control
121 int started =0;
122 //DS3
123 struct Joystick{
124 volatile char* button;
125 volatile int * axis;
126 };
127
128
129 int joy_fd;
130 int num_of_axis=0;
131 int num_of_buttons=0;
132
133
134 struct Joystick DS3;
135
136 //Coordinate system for plotting data in the GUI
137 struct CoordinateSystem{
138   float *X;
139   float *Y;
140   int n;
141   int latest;
142 };
143
144 struct CoordinateSystem LIDARCoordinateSystem;
145
146
147 // Struct used when reading analog feedback from servo and
       storing in a file
148 struct ServoAngle {
149   int ServoPWMChannel;
150   unsigned char ServoADCChannel;
151   FILE *fptrZeroAngle;
152   FILE *fptrRightAngle;
153   FILE *fptrLeftAngle;
154   char* FileNameZeroAngle;
```

```
155   char* FileNameRightAngle;
156   char* FileNameLeftAngle;
157 };
158
159 // Lidar control variables
160 int LIDARMode=0;
161 float LIDARAngleRequest = 0.0;
162 int LIDARAngleReached  = 0;
163
164
165 //Uart filestream ptr
166 int uart0_filestream = -1;
167
168 //BNO055
169
170 int BNO055_READ_LINEAR_ACCEL_ACCESS=0;
171 int BNO055_READ_EULER_ACCESS=0;
172 int BNO055_READ_MAG_ACCESS =0;
173 short int BNO055_LINEAR_ACCEL[3];
174 short int BNO055_EULER[3];
175 short int BNO055_MAG[3];
176
177
178
179
180 /************************Functions
      **********************************/
181 int init();              //IO and libraries initiation
182 void initUART();           //UART initiation
183 int ReadMCP3004(unsigned char) ;  //Reading analog data
      from MCP3204
184 int initDS3();             //initiation of control device
185 void initPCA9685();          //initiation of PCA9685 PWM
      controler
186 void SetPWMDutyCycle(int ,int);   //Set PWM Freq
187 void initBNO055();           //initiation of BNO055
      absolute orientation sensor
188 void UARTTX(char*, int);       //UART Transmit
189 int UARTRX( char*);          //UART Read
190 static void do_drawing(cairo_t *);      //GUI handling
191 int ReadCalibrationData( unsigned char ); //Read Analog
```

```
    Servo feedback end point
192 void CalibrationMode();           //Read Analog Servo
    feedback end point
193 void AngleCalibration(struct ServoAngle); //Read Analog
    Servo feedback end point
194 void AutonomControl ();           //Control loop
195 float OutBoardMotorScale(float);      //Scale value for
    Outboard motor from control loop
196 /*
    ****************************************************************
    */
197
198
199 /**************PI_THREAD (BNO055ReadData)
    ****************************
200  * Description: Read the BNO055 data via UART
201  *
202  * Setting BNO055_READ_EULER_ACCESS to 0
203  * will cause a new read
204  *
205  *
206  *
207  *
208  *
209  * Conditions: init(), initUART() and initBNO055(); needs
     to be called
210  * prior to starting this thread.
211  *
212  ***************************************************************
     */
213 PI_THREAD (BNO055ReadData){
214   char UartRx[128];
215   char UartTx[128];
216   int RxLength=0;
217   while(1){
218
219     UartTx[0]=BNO055_UART_START_BYTE;
220     UartTx[1]=BNO055_UART_READ;
221     UartTx[2]=BNO055_LINEAR_ACCEL_DATA_X_LSB_ADDR;
222     UartTx[3]=0x06;
223
```

```
224       UartTx[2]=BNO055_EULER_H_LSB_ADDR;
225       UARTTX( UartTx,4);
226
227       usleep(30000);
228       RxLength=UARTRX(UartRx);
229       if((UartRx[0]==0xBB) && BNO055_READ_EULER_ACCESS==0){
230
231         BNO055_EULER[0] = (( UartRx[2] ) | (UartRx[3]<< 8));
232         BNO055_EULER[1] = (( UartRx[4] ) | (UartRx[5]<< 8));
233         BNO055_EULER[2] = (( UartRx[6] ) | (UartRx[7]<< 8));
234
235         BNO055_READ_EULER_ACCESS=1;
236       }
237     /*
238       UartTx[2]=BNO055_MAG_DATA_X_LSB_ADDR;
239       UARTTX( UartTx,4);
240
241       usleep(30000);
242       RxLength=UARTRX(UartRx);
243       if((UartRx[0]==0xBB) && BNO055_READ_MAG_ACCESS==0){
244
245         BNO055_MAG[0] = (( UartRx[2] ) | (UartRx[3]<< 8));
246         BNO055_MAG[1] = (( UartRx[4] ) | (UartRx[5]<< 8));
247         BNO055_MAG[2] = (( UartRx[6] ) | (UartRx[7]<< 8));
248         BNO055_READ_MAG_ACCESS=1;
249       }
250       * */
251     }
252 }
253
254 //
255 /**********************PI_THREAD(LIDAR)
       ****************************
256  * Description: Controlling the angle off the distance
       sensor and
257  * measure the distance.
258  *
259  *
260  * Input: Mode, sets the different modes, Scanning, angle
       request and off
261  *
```

```
262   *
263   *
264   *
265   * Conditions: init(),  initBNO055(); needs to be called
266   * prior to starting this thread.
267   *
268   ****************************************************************
        */
269 PI_THREAD(LIDAR){
270
271    int direction= 0;
272    float SensorAngle=0;
273    float Distance = 0;
274    int angle=300;
275    int i = 0;
276    int j = 0;
277    int NewRef=0;
278    float k=0;
279    float m=0;
280    float kr=0;
281    float mr=0;
282    float RightLimit =-PI/4;
283    float LeftLimit = PI/4;
284    float RightAngleNewRef=0;
285    float LeftAngleNewRef=0;
286    float tol = 0.4;
287    FILE *fptrZeroAngle;
288    FILE *fptrRightAngle;
289    FILE *fptrLeftAngle;
290    int ZeroAngle;
291    int RightAngle;
292    int LeftAngle;
293
294
295
296    LIDARCoordinateSystem.n=200;
297    LIDARCoordinateSystem.X=calloc( LIDARCoordinateSystem.n,
           sizeof( float ) );
298    LIDARCoordinateSystem.Y=calloc( LIDARCoordinateSystem.n,
           sizeof( float ) );
299
```

```
300
301
302
303     while (started != 1);
304
305     while(i<=LIDARCoordinateSystem.n){
306       LIDARCoordinateSystem.X[i]=DrawingWidth/2;
307       LIDARCoordinateSystem.Y[i]=DrawingHeight;
308       i++;
309     }
310     i=0;
311
312
313     //Reading angles from files
314     fptrZeroAngle = fopen("LIDARZeroAngle.txt ", "r");
315     if (fptrZeroAngle == NULL)
316     {
317       printf("LIDARAZeroAngle.txt does not exists \n");
318       return 0;
319     }
320     fscanf(fptrZeroAngle,"%i",&ZeroAngle);
321     fclose(fptrZeroAngle);
322
323     fptrRightAngle = fopen("LIDARRightAngle.txt", "r");
324       if (fptrRightAngle == NULL)
325     {
326       printf("LIDARRightAngle.txt does not exists \n");
327       return 0;
328     }
329     fscanf(fptrRightAngle,"%i",&RightAngle);
330     fclose(fptrRightAngle);
331     fptrLeftAngle = fopen("LIDARLeftAngle.txt", "r");
332       if (fptrLeftAngle == NULL)
333     {
334       printf("LIDARALeftAngle.txt does not exists \n");
335       return 0;
336     }
337     fscanf(fptrLeftAngle,"%i",&LeftAngle);
338     fclose(fptrLeftAngle);
339
340
```

```
341
342   k=((-PI/2.0)-(PI/2))/(RightAngle-LeftAngle);
343   m=(PI/2.0)-(k*LeftAngle);
344
345
346   while(1){
347     switch(LIDARMode){
348       case Off:
349         while(LIDARMode==Off){
350           usleep(100000);
351         }
352         break;
353
354       case Scanning:
355
356         while(LIDARMode==Scanning){
357
358           if(NewRef!=1){
359             piLock (SPIKey) ;
360             SensorAngle=(((ReadMCP3004(
                   LIDARAngleServoFeedback))*k)+m);
361             piUnlock (SPIKey) ;
362           }
363
364           if(NewRef==1){
365             SensorAngle=angle*k+m;
366
367           }
368           piLock (SPIKey) ;
369           Distance=ReadMCP3004(LIDARDistanceSensor);
370           piUnlock (SPIKey) ;
371
372           Distance=Distance*(175.0/740.0);
373
374           if(Distance>(gtk_distance_factor*3)){
375             Distance=gtk_distance_factor*3;
376           }
377           LIDARCoordinateSystem.X[i]=DrawingWidth/2+(-sin(
                SensorAngle)*Distance);
378           LIDARCoordinateSystem.Y[i]=DrawingHeight+(-cos(
                SensorAngle)*Distance);
```

```
379             LIDARCoordinateSystem.latest = i;
380
381
382
383             if (direction == 0){
384                angle++;
385                if(SensorAngle<=(-PI/4)){
386
387                  if(NewRef==0){
388                   RightAngleNewRef=angle;
389
390
391                 }
392                 direction = 1;
393                }
394             }
395
396             if (direction == 1){
397                angle--;
398                  if(SensorAngle>=(PI/4)){
399
400                  if (NewRef==0 && RightAngleNewRef!=0){
401                   LeftAngleNewRef=angle;
402                   k=(-0.5*3./RightAngleNewRef)*(1/(1-(
                          LeftAngleNewRef/RightAngleNewRef)));
403                   m=(3.14159265/4)-(k*LeftAngleNewRef);
404                   NewRef=1;
405                 }
406                  direction = 0;
407
408                }
409             }
410             piLock (I2cKey) ;
411             SetPWMDutyCycle(LIDARAngleServo ,angle);
412             piUnlock (I2cKey) ;
413
414             usleep(20000);
415
416
417             if(i<=LIDARCoordinateSystem.n){
418               i++;
```

```
419             }
420             if (i>LIDARCoordinateSystem.n){
421               i=0;
422             }
423
424           }
425
426
427           break;
428
429         case AngleRequest:
430         tol=0.02;
431         kr=((-PI/2.0)-(PI/2))/(RightAngle-LeftAngle);
432         mr=(PI/2.0)-(kr*LeftAngle);
433
434           while(LIDARMode==AngleRequest){
435             piLock (SPIKey) ;
436             SensorAngle=((ReadMCP3004(
                   LIDARAngleServoFeedback))*kr+mr);
437             piUnlock(SPIKey);
438             j=0;
439             while(j<3){
440
441               piLock (SPIKey) ;
442               SensorAngle=SensorAngle+((ReadMCP3004(
                     LIDARAngleServoFeedback))*kr+mr);
443               piUnlock(SPIKey);
444               j++;
445             }
446             SensorAngle=SensorAngle/4.0;
447
448             if((((LIDARAngleRequest)+tol)<=(SensorAngle)) ||
                    (((LIDARAngleRequest)-tol)>=(SensorAngle)))
                  {
449               LIDARAngleReached =0;
450               if (SensorAngle<LIDARAngleRequest){
451                 piLock (I2cKey) ;
452                 angle=angle-1;
453                 SetPWMDutyCycle(LIDARAngleServo ,angle);
454                 piUnlock (I2cKey) ;
455               }
```

```
456             if (SensorAngle>LIDARAngleRequest){
457               piLock (I2cKey) ;
458               angle=angle+1;
459               SetPWMDutyCycle(LIDARAngleServo ,angle);
460               piUnlock (I2cKey) ;
461             }
462
463           }
464           piLock (SPIKey) ;
465           Distance=ReadMCP3004(LIDARDistanceSensor);
466           piUnlock (SPIKey) ;
467
468           Distance=Distance*(175.0/740.0);
469
470           if(Distance>(gtk_distance_factor*3)){
471             Distance=gtk_distance_factor*3;
472           }
473           LIDARCoordinateSystem.X[i]=DrawingWidth/2+(-sin(
                SensorAngle)*Distance);
474           LIDARCoordinateSystem.Y[i]=DrawingHeight+(-cos(
                SensorAngle)*Distance);
475           LIDARCoordinateSystem.latest = i;
476
477           if(i<=LIDARCoordinateSystem.n){
478             i++;
479           }
480           if (i>LIDARCoordinateSystem.n){
481             i=0;
482           }
483
484           if((((LIDARAngleRequest)+tol)>=(SensorAngle)) &&
                  (((LIDARAngleRequest)-tol)<=(SensorAngle))){
485             LIDARAngleReached=1;
486             usleep(10000);
487           }
488         }
489         break;
490       default :
491         break;
492     }
493   }
```

```
494 }
495
496 /**********************PI_THREAD (ReadDS3)
       ****************************
497  * Description: Reading the input from control device,
       once every 10 ms
498  *
499  *
500  *  Input:
501  *
502  *  Output:  write a new value to DS3 every 10 ms
503  *
504  *
505  * Conditions: initDS3()
506  *
507  ***************************************************************
       */
508
509 PI_THREAD (ReadDS3) {
510
511   struct js_event js;
512
513   while(1){
514       // read the joystick state
515     read(joy_fd, &js, sizeof(struct js_event));
516
517       // see what to do with the event
518     switch (js.type & ~JS_EVENT_INIT)
519     {
520       case JS_EVENT_AXIS:
521         DS3.axis    [ js.number ] = js.value;
522         break;
523       case JS_EVENT_BUTTON:
524         DS3.button [ js.number ] = js.value;
525         break;
526     }
527
528   usleep(10000);
529   }
530 }
531 /*******************************************************
```

```
532 * Description: Handles the re-calling of drawing area
533 ******************************************************/
534 static gboolean time_handler(GtkWidget *widget) {
535
536
537 gtk_widget_queue_draw(widget);
538 return TRUE;
539 }
540 /******************************************************
541 * Description: Executes the drawing function
542 ******************************************************/
543 static gboolean draw_event(GtkWidget *widget, cairo_t *cr,
        gpointer data) {
544   do_drawing(cr);
545   return TRUE;
546 }
547
548 /******************************************************
549 * Description: Drawing function
550 ******************************************************/
551 static void do_drawing(cairo_t *cr) {
552
553   gint i;
554   float j ;
555
556   //Set up angles and radius for the radar grid.
557   double radius = gtk_distance_factor;
558   double angle1 = -40.0 * (PI/180.0);
559   double angle2 = (180+40) * (PI/180.0);
560
561   cairo_select_font_face(cr, "Arial",
         CAIRO_FONT_SLANT_NORMAL,CAIRO_FONT_WEIGHT_BOLD);
562   cairo_set_font_size(cr, 10.0);
563
564
565   cairo_set_source_rgba(cr,0.39, 0.58,0.92 ,0.5);
566   cairo_set_line_width(cr, 4.0);
567
568   //Plot distance lines from LIDARCoordinateSystem struct.
569   i = 0;
570
```

```
571       while(i<=LIDARCoordinateSystem.n){
572       j=i*1.0;
573
574
575
576       cairo_move_to(cr, (DrawingWidth/2),(DrawingHeight-15))
             ;
577
578
579
580       if(((LIDARCoordinateSystem.latest-1)-i)>=0){
581         cairo_line_to(cr, LIDARCoordinateSystem.X[((
              LIDARCoordinateSystem.latest-1)-(i))],
              LIDARCoordinateSystem.Y[(LIDARCoordinateSystem.
              latest-1)-(i)]);
582
583       }
584       if(((LIDARCoordinateSystem.latest-1)-i)<0){
585         cairo_line_to(cr,LIDARCoordinateSystem.X[((
              LIDARCoordinateSystem.latest-1)-i+(
              LIDARCoordinateSystem.n+1))],
              LIDARCoordinateSystem.Y[((LIDARCoordinateSystem.
              latest-1)-i+(LIDARCoordinateSystem.n+1))]);
586
587       }
588
589
590       cairo_set_source_rgba(cr,0, .8,0.2,((1.0)/j)+0.1);
591
592       i++;
593       cairo_stroke(cr);
594     }
595
596
597       cairo_arc(cr,0,1,0,LIDARCoordinateSystem.X[
             LIDARCoordinateSystem.latest+1],
             LIDARCoordinateSystem.Y[LIDARCoordinateSystem.
             latest+1]);
598       cairo_stroke(cr);
599
600
```

```
601     //Plotting user set point dot
602     //cairo_set_source_rgba(cr,0,0,0,1);
603     //cairo_arc(cr, UserSetPointX, UserSetPointY, 1, 0, 2
           *PI);
604     //cairo_arc(cr, 0, 0, 1, 0, 2 *PI);
605     //cairo_stroke(cr);
606
607
608     //Plotting user set point as boat
609     cairo_set_source_surface (cr, image, UserSetPointX-8,
           UserSetPointY);
610     cairo_paint (cr);
611     //Plot radar guidelines
612     cairo_set_source_rgba(cr,0,0,0,0.4);
613     cairo_set_line_width(cr,1.5);
614
615     cairo_arc_negative(cr, DrawingWidth/2, DrawingHeight
           -15,radius, angle1,angle2);
616     cairo_stroke(cr);
617
618     cairo_arc_negative(cr, DrawingWidth/2, DrawingHeight
           -15,2*radius, angle1,angle2);
619     cairo_stroke(cr);
620     cairo_arc_negative(cr, DrawingWidth/2, DrawingHeight
           -15,3*radius, angle1,angle2);
621     cairo_stroke(cr);
622
623     cairo_arc(cr, DrawingWidth/2, DrawingHeight-15, 3*
           radius, angle1,angle1);
624     cairo_line_to(cr, DrawingWidth/2, DrawingHeight-15);
625
626     cairo_arc(cr, DrawingWidth/2, DrawingHeight-15, 3*
           radius, angle2,angle2);
627     cairo_line_to(cr, DrawingWidth/2, DrawingHeight-15);
628     cairo_stroke(cr);
629     //Plot radar text
630     cairo_move_to(cr, (DrawingWidth/2.0)-(
           gtk_distance_factor*(cos(40.0*(PI/180.0))))+5,(
           DrawingHeight-25)-(gtk_distance_factor*(sin(40.0*(
           PI/180.0)))));
631     cairo_rotate(cr, -45.0*(PI/180.0));
```

```
632       cairo_show_text(cr,"1 m");
633       cairo_rotate(cr, 45.0*(PI/180.0));
634
635       cairo_move_to(cr, (DrawingWidth/2.0)-(2*
             gtk_distance_factor*(cos(40.0*(PI/180.0))))+5,(
             DrawingHeight-25)-(2*gtk_distance_factor*(sin
             (40.0*(PI/180.0)))));
636       cairo_rotate(cr, -45.0*(PI/180.0));
637       cairo_show_text(cr,"2 m");
638       cairo_rotate(cr, 45.0*(PI/180.0));
639
640       cairo_move_to(cr, (DrawingWidth/2.0)-(3*
             gtk_distance_factor*(cos(40.0*(PI/180.0))))+5,(
             DrawingHeight-25)-(3*gtk_distance_factor*(sin
             (40.0*(PI/180.0)))));
641       cairo_rotate(cr, -45.0*(PI/180.0));
642       cairo_show_text(cr,"3 m");
643       cairo_rotate(cr, 45.0*(PI/180.0));
644
645
646  }
647
648  /**********************PI_THREAD (GUI)
        ****************************
649   * Description: Main GTK thread
650   ***************************************************************
        */
651  PI_THREAD(GUI){
652
653     char output[32] = {0};
654       // Initiate pointers
655       GtkWidget *window;
656       GtkWidget *fixed;
657       GtkWidget *da;
658     GtkWidget *img;
659
660
661
662       gtk_init (0, 0);
663
664       GdkRGBA color = {1.0, 1.0, 1.0, 1.0};
```

```
665
666       // Creating main window and set background colour and
             fullscreen.
667       window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
668       gtk_widget_override_background_color(window,
             GTK_STATE_FLAG_NORMAL, &color);
669
670
671       gtk_window_set_position(GTK_WINDOW(window),
             GTK_WIN_POS_CENTER);
672     gtk_window_fullscreen(GTK_WINDOW(window));
673
674     //Retrieve current screen data.
675     ScreenWidth=gdk_screen_width();
676     ScreenHeight=gdk_screen_height();
677
678
679
680     // Attach grid to main window for widget positioning.
681       fixed = gtk_fixed_new();
682
683     // Attach a background image to grid.
684     img = gtk_image_new_from_file("boat_top_view.bmp");
685     gtk_fixed_put(fixed, img, (-BoatImgWidth/2)+(ScreenWidth
        /2), ScreenHeight-BoatImgHeight );
686       gtk_widget_set_size_request(fixed,ScreenWidth,
             ScreenHeight);
687       gtk_container_add(GTK_CONTAINER(window),fixed);
688
689       DrawingHeight = ScreenHeight-BoatImgHeight;
690       DrawingWidth = ScreenWidth;
691
692       UserSetPointX=ScreenWidth/2;
693
694     // Create the drawing area and set widget size.
695       da = gtk_drawing_area_new();
696       gtk_fixed_put(fixed,da,0,10);
697     gtk_widget_set_size_request(da,DrawingWidth,
        DrawingHeight);
698     g_signal_connect(G_OBJECT(da), "draw", G_CALLBACK(
        draw_event), NULL);
```

```
699
700
701     // Create pointer to user set point image
702     image = cairo_image_surface_create_from_png ("
          boat_top_view_scaled.png");
703
704
705     // Start the graphic interface.
706     gtk_widget_show_all(window);
707
708     // Request update of drawing area.
709     gint id3 = g_timeout_add(1000/24, (GSourceFunc)
          time_handler, da);
710
711     gtk_main ();
712
713
714     g_source_remove(id3);
715       return 0;
716 }
717
718
719 /**************void UARTTX(char*tx_buffer, int size )
       ******************
720  * Description: main, manual control. main program loop
721  *
722  *  Inputs:
723  *
724  * Output:
725  *
726  * Conditions: .
727  *
728  *  *
       ************************************************************
       */
729 int main (void){
730   float BowThrusterLeftDuty = 0;
731   float BowThrusterRightDuty = 0;
732   float RudderAngleServoCtrlValue=0;
733   float OutBoardMotorDuty=0;
734   int MotorDirection = 0;
```

```
735   float k=0;
736   float m=0;
737   //Initate IO and libraies
738   printf("Initiating IO...\n");
739   if (init()!=1){
740     printf("failed to initate\n");
741     return(0);
742   }
743   printf("Configuring UART..\n");
744
745   // Initiateate the UART
746   initUART();
747
748   // Initiate The BNO055 sensor
749   initBNO055();
750
751   // Initiate the PCA9685 (PWM module)
752   initPCA9685();
753
754
755   // Initiate the DS3 Controller
756   if(initDS3()==1){
757     printf( "DS3 Controller initiated\n" );
758   }
759
760   if(initDS3()!=1){
761     printf( "Couldn't open joystick\n" );
762     return(0);
763   }
764
765   //Initiate the GUI Interface
766   if (piThreadCreate (GUI) != 0){
767     printf ("GUI failed to startn");
768     return(0);
769   }
770
771
772   if (piThreadCreate (BNO055ReadData) != 0){
773     printf ("BNO055ReadData failed to start\n");
774     return(0);
775   }
```

```
776
777   if (piThreadCreate (ReadDS3) != 0){
778     printf ("ReadDS3 failed to start\n");
779     return(0);
780   }
781
782   if (piThreadCreate (LIDAR) != 0){
783     printf ("LIDAR failed to start\n");
784     return(0);
785   }
786
787
788   piLock (I2cKey) ;
789   SetPWMDutyCycle(RudderAngleServo ,0);
790   SetPWMDutyCycle(BowThrusterLeft,0);
791   SetPWMDutyCycle(BowThrusterRight ,0);
792   SetPWMDutyCycle(OutboardMotor ,0);
793   SetPWMDutyCycle(LIDARAngleServo ,0);
794   piUnlock (I2cKey) ;
795
796   printf ("All systems go... \npress start to enter main
         program.\npress select to enter calibration mode\n");
797
798
799
800   while(DS3.button[START]==0){
801
802     if(DS3.button[SELECT] ==1){
803       system("clear");
804       printf("Entering calibration mode.\npress X for
            LIDAR Servo.\npress O for Rudder Servo. \nPress
            start to abort.\n");
805       CalibrationMode();
806     }
807   }
808   while(DS3.button[START]==1);
809   started =1;
810
811   //Main program loop
812     while(1)
813   {
```

```
814     //Program is in manual mode by default
815     if (DS3.button[R1]==1){
816       LIDARMode=Scanning;
817     }
818
819     if (DS3.button[L1]==1){
820       LIDARMode=Off;
821     }
822     if(DS3.button[START]==1){
823       AutonomControl ();
824     }
825
826     //BowThruster
827     bcm2835_gpio_write(BowThrusterEnable, HIGH);
828     //Thruster left
829     if (DS3.axis[RIGHT_STICK_X]<=0){
830       if (BowThrusterRightDuty!=0){
831         BowThrusterRightDuty=0;
832         piLock (I2cKey) ;
833         SetPWMDutyCycle(BowThrusterRight,(int)
              BowThrusterRightDuty);
834         piUnlock (I2cKey) ;
835       }
836       if(BowThrusterLeftDuty != (abs(DS3.axis[
             RIGHT_STICK_X])*(4095.0/32767.0))){
837         BowThrusterLeftDuty = (abs(DS3.axis[RIGHT_STICK_X
              ])*(4095.0/32767.0));
838         piLock (I2cKey) ;
839         SetPWMDutyCycle(BowThrusterLeft ,
              BowThrusterLeftDuty);
840         piUnlock (I2cKey) ;
841       }
842     }
843     //Thruster right
844     if (DS3.axis[RIGHT_STICK_X]>0){
845       if (BowThrusterLeftDuty!=0){
846         BowThrusterLeftDuty=0;
847         piLock (I2cKey) ;
848         SetPWMDutyCycle(BowThrusterLeft,(int)
              BowThrusterLeftDuty);
849         piUnlock (I2cKey) ;
```

```
850         }
851         if(BowThrusterRightDuty != DS3.axis[RIGHT_STICK_X
               ]*(4095.0/32767.0)){
852           BowThrusterRightDuty = DS3.axis[RIGHT_STICK_X
                 ]*(4095.0/32767.0);
853           piLock (I2cKey) ;
854           SetPWMDutyCycle(BowThrusterRight ,(int)
                 BowThrusterRightDuty);
855           piUnlock (I2cKey) ;
856
857         }
858       }
859       //Rudder Steering
860       if(DS3.button[L2_DIGITAL]==0){
861         k=((280-566)/(2.0*32767));
862         m=280-32767*k;
863         if((DS3.axis[LEFT_STICK_X]*k+m)!=
               RudderAngleServoCtrlValue){
864           RudderAngleServoCtrlValue=((DS3.axis[LEFT_STICK_X
                 ]*k+m));
865           piLock (I2cKey) ;
866           SetPWMDutyCycle(RudderAngleServo ,(int)
                 RudderAngleServoCtrlValue);
867           piUnlock (I2cKey) ;
868         }
869       }
870
871       // Speed and direction control
872       if((DS3.button[UPP]==1) && (DS3.button[DOWN]==0)&&
              MotorDirection!=Forward){
873         piLock (I2cKey) ;
874         SetPWMDutyCycle(OutboardMotor ,0);
875         piUnlock (I2cKey) ;
876         MotorDirection=Forward;
877       }
878
879       if((DS3.button[UPP]==0) && (DS3.button[DOWN]==1)&&(
              MotorDirection!=Reverse)){
880         piLock (I2cKey) ;
881         SetPWMDutyCycle(OutboardMotor ,0);
882         piUnlock (I2cKey) ;
```

```
883         MotorDirection=Reverse;
884       }
885
886
887       if(MotorDirection==Forward){
888         k=(750.0-450.0)/(2.0*32767.0);
889         m=600;
890         if(DS3.axis[R2_ANALOG]*k+m != OutBoardMotorDuty){
891           OutBoardMotorDuty=DS3.axis[R2_ANALOG]*k+m;
892           piLock (I2cKey) ;
893           SetPWMDutyCycle(OutboardMotor ,OutBoardMotorDuty);
894           piUnlock (I2cKey) ;
895         }
896       }
897
898       if(MotorDirection==Reverse){
899         k=-(450.0-150.0)/(2.0*32767.0);
900         m=300;
901         if(DS3.axis[R2_ANALOG]*k+m != OutBoardMotorDuty){
902           OutBoardMotorDuty=DS3.axis[R2_ANALOG]*k+m;
903           piLock (I2cKey) ;
904           SetPWMDutyCycle(OutboardMotor ,OutBoardMotorDuty);
905           piUnlock (I2cKey) ;
906         }
907       }
908
909
910       if(DS3.button[L2_DIGITAL]==1){
911         k=(DrawingWidth)/(32767.0*2);
912         m=(DrawingWidth-32767*k);
913         UserSetPointX=DS3.axis[LEFT_STICK_X]*k+m;
914         k=(DrawingHeight-4)/(32767.0*2);
915         m=DrawingHeight-k*32767;
916         UserSetPointY=DS3.axis[LEFT_STICK_Y]*k+m;
917       }
918
919
920       if (BNO055_READ_EULER_ACCESS==1){
921         BNO055_READ_LINEAR_ACCEL_ACCESS=0;
922         BNO055_READ_EULER_ACCESS = 0;
923       }
```

```
924       usleep(9000);
925     }
926   }
927
928
929   /**************void UARTTX(char*tx_buffer, int size )
          ******************
930    * Description: Send data via UART
931    *
932    *  Inputs: tx_buffer, pointer to buffer to send.
933    *  size, number of bytes to send
934    *
935    *
936    * Output:
937    *
938    * Conditions: initUART() needs to be called prior to
939    * executing this function.
940    *
941    *  *
          ************************************************************
          */
942
943   void UARTTX(char*tx_buffer, int size ){
944     //----- TX BYTES -----
945     int count = 0;
946
947     char *p_tx_buffer_start=&tx_buffer[0];
948     char *p_tx_buffer_end=&tx_buffer[size];
949     if (uart0_filestream != -1)
950     {
951       count = write(uart0_filestream, &tx_buffer[0], (
             p_tx_buffer_end- p_tx_buffer_start));   //
             Filestream, bytes to write, number of bytes to
             write
952
953       if (count < 0)
954       {
955         printf("UART TX error\n");
956       }
957     }
958   }
```

```
959
960
961
962 /***************int UARTRX( char* rx_buffer)
       ************************
963  * Description: Read UART input port buffer.
964  *
965  *  Inputs: rx_buffer, pointer to buffer for storing data.
966  *
967  *
968  * Output: rx_length, number of bytes read.
969  *
970  * Conditions: initUART() needs to be called prior to
971  * executing this function.
972  *
973  *  *
       ************************************************************
       */
974 int UARTRX( char* rx_buffer){
975   //----- CHECK FOR ANY RX BYTES -----
976
977   if (uart0_filestream != -1)
978   {
979     // Read up to 255 characters from the port if they are
          there
980
981     int rx_length = read(uart0_filestream, (void*)
         rx_buffer, 128);    //Filestream, buffer to store
         in, number of bytes to read (max)
982     if (rx_length < 0)
983     {
984       //An error occured (will occur if there are no bytes
            )
985     }
986     else if (rx_length == 0)
987     {
988       //No data waiting
989     }
990     else
991     {
992       //Bytes received
```

```
993         rx_buffer[rx_length] = '\0';
994       }
995       return(rx_length);
996     }
997     return(0);
998 }
999
1000
1001 /**********void SetPWMDutyCycle(int Channel ,int DutyCycle
       )***********
1002  * Description: Sets the PWM duty cycle of the channel
       whiht the
1003  * value of DutyCycle
1004  *
1005  *  Inputs: Channel, the channel that is to be changed.
1006  *  DutyCycle, the new DutyCycle for the channel
1007  *
1008  *
1009  * Output:
1010  *
1011  * Conditions: init() and initPCA9685() needs to be called
       prior to
1012  * executing this function I2cKey needs to be locked befor
       calling
1013  * this function, and unlocked after return.
1014  *
1015  *  *
       ************************************************************
       */
1016 void SetPWMDutyCycle(int Channel ,int DutyCycle){
1017   char wbuf[32];
1018   wbuf[0]=(Channel0_ON_L + Channel_MULTIPLYER * (Channel))
       ;
1019   wbuf[1]=0 & 0xFF;
1020   bcm2835_i2c_write (wbuf,2);
1021   wbuf[0]=(Channel0_ON_H + Channel_MULTIPLYER * (Channel))
       ;
1022   wbuf[1]=0 & 0x8;
1023   bcm2835_i2c_write (wbuf,2);
1024
1025   wbuf[0]=(Channel0_OFF_L + Channel_MULTIPLYER * (Channel)
```

```
        );
1026    wbuf[1]=(DutyCycle & 0xFF);
1027    bcm2835_i2c_write (wbuf,2);
1028    wbuf[0]=(Channel0_OFF_H + Channel_MULTIPLYER * (Channel)
        );
1029    wbuf[1]= DutyCycle >> 8;
1030    bcm2835_i2c_write (wbuf,2);
1031
1032
1033  }
1034
1035
1036  /*************************void CalibrationMode()
       ************************
1037   * Description: Mode for calibrate the endpoint of the
       servos
1038   *
1039   * Inputs:
1040   *
1041   *
1042   *
1043   * Output:
1044   *
1045   * Conditions:
1046   *
1047   *  *
       ****************************************************************
       */
1048  void CalibrationMode(){
1049    struct ServoAngle LIDARServoAngle;
1050    struct ServoAngle RudderServoAngle;
1051
1052    LIDARServoAngle.ServoADCChannel =
        LIDARAngleServoFeedback;
1053    LIDARServoAngle.ServoPWMChannel = LIDARAngleServo;
1054    LIDARServoAngle.FileNameZeroAngle=(char*)malloc( 50*
        sizeof( char ) );
1055    LIDARServoAngle.FileNameRightAngle=(char*)malloc( 50*
        sizeof( char ) );
1056    LIDARServoAngle.FileNameLeftAngle=(char*)malloc(50*
        sizeof( char ) );
```

```
1057    LIDARServoAngle.FileNameZeroAngle="LIDARZeroAngle.txt\0"
           ;
1058    LIDARServoAngle.FileNameRightAngle="LIDARRightAngle.txt
           \0";
1059    LIDARServoAngle.FileNameLeftAngle="LIDARLeftAngle.txt\0"
           ;
1060
1061    RudderServoAngle.ServoADCChannel =
           RudderAngleServoFeedback;
1062    RudderServoAngle.ServoPWMChannel = RudderAngleServo;
1063    RudderServoAngle.FileNameZeroAngle=calloc( 100, sizeof(
           char ) );
1064    RudderServoAngle.FileNameRightAngle=calloc( 100, sizeof(
            char ) );
1065    RudderServoAngle.FileNameLeftAngle=calloc( 100, sizeof(
           char ) );
1066    RudderServoAngle.FileNameZeroAngle="RudderZeroAngle.txt
           \0";
1067    RudderServoAngle.FileNameRightAngle="RudderRightAngle.
           txt\0";
1068    RudderServoAngle.FileNameLeftAngle="RudderLeftAngle.txt
           \0";
1069
1070    while(DS3.button[START]==0){
1071      if (DS3.button[CROSS]){
1072        printf("LIDAR calibration\n");
1073        while(DS3.button[CROSS]==1);
1074        AngleCalibration(LIDARServoAngle);
1075        system ("clear");
1076        printf("calibration mode.\npress X for LIDAR Servo.\
               npress O for Rudder Servo. \nPress start to enter
                main program.\n");
1077
1078      }
1079      if (DS3.button[CIRCLE]){
1080        printf("Rudder calibration\n");
1081        while(DS3.button[CIRCLE]==1);
1082        AngleCalibration(RudderServoAngle);
1083        system ("clear");
1084        printf("calibration mode.\npress X for LIDAR Servo.\
               npress O for Rudder Servo. \nPress start to enter
```

```
                main program.\n");
1085        }
1086      }
1087    }
1088
1089
1090
1091    /**********void AngleCalibration(struct ServoAngle
          ServoAngleStr)*****
1092     * Description:Read the analog feedback servo and store
           the value in files
1093     *
1094     * Inputs:  servo to be read
1095     *
1096     *
1097     *
1098     * Output:
1099     *
1100     * Conditions: initPCA9685()
1101     *
1102     *  *
           ************************************************************
           */
1103    void AngleCalibration(struct ServoAngle  ServoAngleStr){
1104
1105      int NewCalValue = 0;
1106
1107      SetPWMDutyCycle(ServoAngleStr.ServoPWMChannel ,0);
1108      printf("Press Square to calibrate LeftAngle\nPress
           Triangle to calibrate Zeroangle\nPress Circle to
           calibrate RightAngle\nPress Select to return to
           calibration menu\n" );
1109
1110        while(DS3.button[SELECT]==0){
1111
1112
1113        if(DS3.button[TRIANGLE]){
1114          printf("Reading ZEROANGLE...\n");
1115          ServoAngleStr.fptrZeroAngle = fopen(ServoAngleStr.
              FileNameZeroAngle, "w");
1116          NewCalValue=ReadCalibrationData(ServoAngleStr.
```

```
            ServoADCChannel);
1117        fprintf(ServoAngleStr.fptrZeroAngle, "%i\n",
              NewCalValue);
1118        while(DS3.button[TRIANGLE]==1);
1119        printf ("ZEROANGLE set\n");
1120        printf ("New ZEROANGLE: %i \n",NewCalValue);
1121        fclose(ServoAngleStr.fptrZeroAngle);
1122
1123      }
1124
1125
1126      if(DS3.button[SQUARE]){
1127        printf("Reading LEFTANGLE...\n");
1128        ServoAngleStr.fptrLeftAngle = fopen(ServoAngleStr.
              FileNameLeftAngle, "w");
1129        NewCalValue=ReadCalibrationData(ServoAngleStr.
              ServoADCChannel);
1130        fprintf(ServoAngleStr.fptrLeftAngle, "%i\n",
              NewCalValue);
1131        while(DS3.button[SQUARE]==1);
1132        printf ("LEFTANGLE set\n");
1133        printf ("New LEFTANGLE: %i \n",NewCalValue);
1134        fclose(ServoAngleStr.fptrLeftAngle);
1135
1136      }
1137
1138      if(DS3.button[CIRCLE]){
1139        printf("Reading RIGHTANGLE...\n");
1140        ServoAngleStr.fptrRightAngle = fopen(ServoAngleStr.
              FileNameRightAngle, "w");
1141        NewCalValue=ReadCalibrationData(ServoAngleStr.
              ServoADCChannel);
1142        fprintf(ServoAngleStr.fptrRightAngle, "%i\n",
              NewCalValue);
1143        while(DS3.button[CIRCLE]==1);
1144        printf ("RIGHTANGLE set\n");
1145        printf ("New RIGHTANGLE: %i \n",NewCalValue);
1146        fclose(ServoAngleStr.fptrRightAngle);
1147      }
1148
1149    }
```

```
1150    return;
1151  }
1152
1153  /************ReadCalibrationData(unsigned char adcChannel)
        ********************
1154   * Description: Read analog feedback from servo n times
        and return the
1155   * mean value
1156   *
1157   *Inputs: Channel, the channel that is to be read.
1158   *
1159   *
1160   *
1161   * Output: mean value
1162   *
1163   * Conditions: init() and ReadMCP3004 needs to be called
        prior to
1164   * executing.
1165   *
1166   ***********************************************************************
        */
1167
1168  int ReadCalibrationData(unsigned char adcChannel){
1169    int i = 0;
1170    int CalValue=0;
1171    int n=50;
1172    while(i<= n){
1173      piLock (SPIKey) ;
1174      CalValue=CalValue+ReadMCP3004(adcChannel);
1175      piUnlock (SPIKey) ;
1176      usleep(6000);
1177      i++;
1178    }
1179    return(CalValue/(n+1));
1180  }
1181
1182  /************ReadMCP3004(unsigned char adcChannel)
        ********************
1183   * Description: Read analog adcChannel data from MCP3004
1184   *
1185   *  Inputs: Channel, the channel that is to be read.
```

```
1186   *
1187   *
1188   *
1189   * Output: 14 bit data in adcValue.
1190   *
1191   * Conditions: init() needs to be called prior to
1192   * executing this function SPIKey needs to be locked befor
           calling
1193   * this function, and unlocked after return.
1194   *
1195   ****************************************************************
          */
1196
1197  int ReadMCP3004(unsigned char adcChannel){
1198    unsigned char buff[3];
1199    int adcValue = 0;
1200    buff[0] = 0x06 | ((adcChannel & 0x07) >> 7);
1201    buff[1] = ((adcChannel & 0x07) << 6);
1202    buff[2] = 0x00;
1203    digitalWrite(CS_MCP3204, 0);  // Low : CS Active
1204    wiringPiSPIDataRW(SPI_CHANNEL, buff, 3);
1205    buff[1] = 0x0F & buff[1];
1206    adcValue = ( buff[1] << 8) | buff[2];
1207    digitalWrite(CS_MCP3204, 1);  // High : CS Inactive
1208    return adcValue;
1209  }
1210
1211
1212
1213  /*****************************int init()
          ****************************
1214   * Description: IO and libraries initiation
1215   *
1216   *  Inputs:
1217   *
1218   *
1219   * Output: 1 if success. 0 if fail.
1220   *
1221   * Conditions:
1222   *
1223   ****************************************************************
```

```
1224      */
1225  int init(){
1226
1227    // sets up the bcm2835 library
1228    if (!bcm2835_init()){
1229      return -1;
1230    }
1231
1232    //Initate the i2c Interface
1233    bcm2835_i2c_begin();
1234    bcm2835_i2c_set_baudrate (100000);
1235      bcm2835_i2c_setSlaveAddress(I2CPWMAdress);
1236
1237
1238      // sets up the wiringPi library
1239      if (wiringPiSetup () < 0) {
1240        fprintf (stderr, "Unable to setup wiringPi: %s\n",
             strerror (errno));
1241        return -1;
1242    }
1243
1244    // Enable control for bow thruster
1245    bcm2835_gpio_fsel(BowThrusterEnable,
        BCM2835_GPIO_FSEL_OUTP);
1246    bcm2835_gpio_write(BowThrusterEnable, LOW);
1247
1248    //Initate BNO055Reset as Output
1249    bcm2835_gpio_fsel(20, BCM2835_GPIO_FSEL_OUTP);
1250
1251    bcm2835_gpio_fsel(21, BCM2835_GPIO_FSEL_OUTP);
1252
1253    //Initate the SPI
1254    pinMode(CS_MCP3204, OUTPUT);
1255
1256     if(wiringPiSPISetup(SPI_CHANNEL, SPI_SPEED) == -1)
1257    {
1258      fprintf (stdout, "wiringPiSPISetup Failed: %s\n",
           strerror(errno));
1259      return -1 ;
1260    }
```

```
1261    return(1);
1262  }
1263
1264  /*************************void initUART()
        **********************
1265   * Description: initiates the UART interface
1266   *
1267   *  Inputs:
1268   *
1269   * Output:
1270   *
1271   * Conditions:
1272   *
1273   ************************************************************
        */
1274  void initUART(){
1275
1276    //-------------------------
1277    //----- SETUP USART 0 -----
1278    //-------------------------
1279    //At boot up, pins 8 and 10 are already set to UART0_TXD
          , UART0_RXD (ie the alt0 function) respectively
1280
1281
1282    //OPEN THE UART
1283    //The flags (defined in fcntl.h):
1284    //  Access modes (use 1 of these):
1285    //    O_RDONLY - Open for reading only.
1286    //    O_RDWR - Open for reading and writing.
1287    //    O_WRONLY - Open for writing only.
1288    //
1289    //  O_NDELAY / O_NONBLOCK (same function) - Enables
          nonblocking mode. When set read requests on the file
          can return immediately with a failure status
1290    //                        if there is no input immediately
           available (instead of blocking). Likewise, write
          requests can also return
1291    //                        immediately with a failure
          status if the output can't be written immediately.
1292    //
1293    //  O_NOCTTY - When set and path identifies a terminal
```

```
          device, open() shall not cause the terminal device to
           become the controlling terminal for the process.
1294    uart0_filestream = open("/dev/ttyAMA0", O_RDWR |
          O_NOCTTY | O_NDELAY);    //Open in non blocking read/
          write mode
1295    if (uart0_filestream == -1)
1296    {
1297      //ERROR - CAN'T OPEN SERIAL PORT
1298      printf("Error - Unable to open UART.  Ensure it is not
              in use by another application\n");
1299    }
1300
1301    //CONFIGURE THE UART
1302    //The flags (defined in /usr/include/termios.h - see
          http://pubs.opengroup.org/onlinepubs/007908799/xsh/
          termios.h.html):
1303    //  Baud rate:- B1200, B2400, B4800, B9600, B19200,
          B38400, B57600, B115200, B230400, B460800, B500000,
          B576000, B921600, B1000000, B1152000, B1500000,
          B2000000, B2500000, B3000000, B3500000, B4000000
1304    //  CSIZE:- CS5, CS6, CS7, CS8
1305    //  CLOCAL - Ignore modem status lines
1306    //  CREAD - Enable receiver
1307    //  IGNPAR = Ignore characters with parity errors
1308    //  ICRNL - Map CR to NL on input (Use for ASCII comms
          where you want to auto correct end of line characters
           - don't use for bianry comms!)
1309    //  PARENB - Parity enable
1310    //  PARODD - Odd parity (else even)
1311    struct termios options;
1312    tcgetattr(uart0_filestream, &options);
1313    options.c_cflag = B115200 | CS8 | CLOCAL | CREAD;   //<
          Set baud rate
1314    options.c_iflag = IGNPAR;
1315    options.c_oflag = 0;
1316    options.c_lflag = 0;
1317    tcflush(uart0_filestream, TCIFLUSH);
1318    tcsetattr(uart0_filestream, TCSANOW, &options);
1319    return;
1320  }
1321
```

```
1322
1323
1324
1325 /*************************void initDS3()
       **********************
1326  * Description: Opens control device from linux JOY_DEV in
       read
1327  * only mode
1328  *
1329  *  Inputs:
1330  *
1331  * Output: returns 1 if sucsess, -1 fail.
1332  *
1333  * Conditions:
1334  *
1335  ************************************************************
       */
1336 int initDS3(){
1337   char name_of_joystick[80];
1338   if( ( joy_fd = open( JOY_DEV , O_RDONLY)) == -1 )
1339   {
1340     return -1;
1341   }
1342   ioctl( joy_fd, JSIOCGAXES, &num_of_axis );
1343   ioctl( joy_fd, JSIOCGBUTTONS, &num_of_buttons );
1344   ioctl( joy_fd, JSIOCGNAME(80), &name_of_joystick );
1345   DS3.axis =  calloc( num_of_axis, sizeof( int ) );
1346   DS3.button =  calloc( num_of_buttons, sizeof( char ) );
1347   fcntl( joy_fd, F_SETFL, O_NONBLOCK ); // use non-
       blocking mode
1348   return (1);
1349 }
1350
1351
1352
1353
1354 /*************************void initPCA9685()
       **********************
1355  * Description: Resets the PWM controller and configure it
       for
1356  * a PWM freq defined in PrescaleVal
```

```
1357  *
1358  *  Inputs:
1359  *
1360  * Output: returns 1 if success, -1 fail.
1361  *
1362  * Conditions:
1363  *
1364  ************************************************************
        */
1365
1366 void initPCA9685(){
1367   char wbuf[32];
1368     //reset
1369     wbuf[0]=MODE1;
1370     wbuf [1]=0x00;
1371     bcm2835_i2c_write (wbuf,2);
1372     wbuf[0]=MODE2;
1373     wbuf[1]=0x04;
1374   bcm2835_i2c_write (wbuf,2);
1375
1376     //SetPWMFreq
1377     uint8_t prescaleVal = (CLOCK_FREQ/4096/61)-1;
1378     wbuf[0]=MODE1;
1379     wbuf[1]=0x10;
1380     bcm2835_i2c_write (wbuf,2);
1381     wbuf[0]=PRE_SCALE;
1382     wbuf[1]=prescaleVal;
1383     bcm2835_i2c_write (wbuf,2);
1384     wbuf[0]=MODE1;
1385     wbuf[1]=0x80;
1386     bcm2835_i2c_write (wbuf,2);
1387     wbuf[0]=MODE2;
1388     wbuf[1]=0x04;
1389     bcm2835_i2c_write (wbuf,2);
1390 }
1391
1392
1393 /*************************void initBNO055()
       **********************
1394  * Description: Resets BNO055 sensor and configure it for
1395  * NDOF mode
```

```
1396   *
1397   * Inputs:
1398   *
1399   * Output:
1400   *
1401   * Conditions: initUART() and init() needs to be called
          prior to
1402   * executing this function
1403   *
1404   ************************************************************
          */
1405  void initBNO055(){
1406      char UartRx[128];
1407      char UartTx[128];
1408      int RxLength =0;
1409      printf("Initiating the BNO055 Orientation sensor... \n
             ");
1410      bcm2835_gpio_write(20, HIGH);
1411      bcm2835_delay(500);
1412      //reset the sensor
1413
1414      bcm2835_gpio_write(21, LOW);
1415      printf("Reseting the sensor... \n");
1416      bcm2835_delay(500);
1417      bcm2835_gpio_write(21, HIGH);
1418      bcm2835_delay(500);
1419
1420      printf("Atempting to configure the sensor... \n");
1421
1422      //configuring the sensor for NDOF mode
1423      while(UartRx[1]!=0x01){
1424
1425        bcm2835_gpio_write(21, LOW);
1426        printf("Reseting the sensor... \n");
1427        bcm2835_delay(500);
1428        bcm2835_gpio_write(21, HIGH);
1429        bcm2835_delay(500);
1430        //configuring the sensor for NDOF mode
1431        UartTx[0]=BNO055_UART_START_BYTE;
1432        UartTx[1]=BNO055_UART_WRITE;
1433        UartTx[2]=BNO055_OPR_MODE_ADDR;
```

```
1434        UartTx[3]=0x01;
1435        UartTx[4]=OPERATION_MODE_NDOF;
1436        UARTTX(UartTx,5);
1437        bcm2835_delay(500);
1438        //reading the response
1439        RxLength=UARTRX( UartRx);
1440      }
1441      if (UartRx[0]==0XEE && UartRx[1]==0X01 && RxLength==2)
             {
1442        printf("Success\n");
1443        return ;
1444      }
1445
1446      printf("failed to initate BNO055.\n");
1447      printf("Byte 1 : %i\n", UartRx[0]);
1448      printf("Byte 2 : %i\n", UartRx[1]);
1449      return ;
1450
1451  }
1452
1453  /**************************void AutonomControl ()
        **********************
1454   * Description: AutonomControl, this function will
         regulate the boat
1455   * towards the set points defined in the user interface.
1456   * the set points are the desired distance and the angle
         from the set point
1457   * and the current heading.
1458   *
1459   * Inputs: SetpointDistance, SetpointHeading.
1460   *
1461   * Output: The function will set LIDARmode to AngleRequest
          and
1462   * continuously write new value to LIDARAngleRequest.
1463   *
1464   *
1465   *
1466   * Conditions: All init functions needs to be executed and
          all thread
1467   * needs to be started before calling this function.
1468   *
```

```
1469  ************************************************************
        */
1470  void AutonomControl (){
1471    int j = 0;
1472    float k = 0;
1473    float m = 0;
1474    float dt=0;
1475    float ds=0;
1476    int RudderReverse= 1 ;
1477    float RudderAngle = 0;
1478    float timestamp=0;
1479    uint64_t TimestampVelo= 0;
1480    uint64_t TimestampAngleVelo= 0;
1481    uint64_t DelayTime = 1000*100; //10ms
1482    uint64_t ControlTime= 0;
1483
1484    FILE *Logging;
1485
1486    float BowThrusterDuty = 0;
1487    float BowThrusterLeftDuty  =0;
1488    float BowThrusterRightDuty  = 0;
1489
1490    float OutBoardMotorDuty=0;
1491    float RudderAngleServoCtrlValue = 0;
1492
1493    float BowThrusterLimit = 4095;
1494
1495    float OutBoardMotorForwardLimit = 10 ;
1496    float OutBoardMotorReverseLimit = -10;
1497
1498    //Constants for control loop
1499    float K1Velocity =15.410509031198689;
1500    float K2Velocity =-14.589490968801318;
1501
1502    //Constants for LP filter
1503    float LPFilterK1=0.6;
1504    float LPFilterK2=0.2;
1505
1506
1507    float OutboardMotorUnscaledDuty =0;
1508
```

```
1509
1510
1511    float SetpointDistance = 0.35;
1512    float DistanceTolerance = 1;
1513
1514    float SetpointVelocity =0 ;
1515    float SetpointAngleVelocity  = 0;
1516
1517    float ActualValueVelocity = 0;
1518    float ActualValueVelocityPre  =0;
1519    float ActualValueAngleVelo  =0;
1520    float ActualValueHeading = 0;
1521    float ActualValueDistance = 0;
1522
1523    float ErrorDistance  = 0;
1524    float S =0;
1525
1526    float ErrorHeading = 0;
1527    float ErrorVelocity = 0;
1528    float ErrorVelocityPrev = 0;
1529
1530    float ErrorAngleVelocity = 0;
1531    float ErrorAngleVelocityPrev=0;
1532
1533
1534    //User Set point heading
1535    float SetpointHeading = atan2(-1*(UserSetPointX-(
          DrawingWidth/2)),-1*(UserSetPointY-(DrawingHeight)))
          *(180/PI);
1536    //float SetpointHeading = 0;
1537
1538
1539    //change lidar mode
1540    LIDARMode=AngleRequest;
1541    LIDARAngleRequest=SetpointHeading*(PI/180.0);
1542    LIDARAngleReached=0;
1543
1544    while(BNO055_READ_EULER_ACCESS==0);
1545
1546    SetpointHeading=(BNO055_EULER[0]/16.0)-SetpointHeading;
1547    BNO055_READ_EULER_ACCESS=0;
```

```
1548
1549    if(SetpointHeading>360){
1550      SetpointHeading=SetpointHeading-360;
1551    }
1552    if(SetpointHeading<0){
1553      SetpointHeading=SetpointHeading+360;
1554    }
1555
1556    printf("SetpointHeading%f\n",SetpointHeading);
1557
1558
1559    piLock (I2cKey) ;
1560    SetPWMDutyCycle(RudderAngleServo ,(int)
           RudderAngleServoCtrlValue);
1561    SetPWMDutyCycle(BowThrusterLeft,(int)BowThrusterLeftDuty
           );
1562    SetPWMDutyCycle(BowThrusterRight ,(int)
           BowThrusterRightDuty);
1563    SetPWMDutyCycle(OutboardMotor ,(int)OutBoardMotorDuty);
1564    piUnlock (I2cKey) ;
1565
1566    //hold for laser to reach correct angle
1567    usleep(50000);
1568    while (LIDARAngleReached==0);
1569
1570    //read distance and calculate the set point
1571    //gtk_distance_factor
1572    piLock (SPIKey) ;
1573    SetpointDistance=((ReadMCP3004(LIDARDistanceSensor)
           *(1291.0/962.0))/1000)-((50+((sqrt(pow((UserSetPointX
           -(DrawingWidth/2.0)),2)+pow((UserSetPointY-
           DrawingHeight),2))))*(1000.0/190.0))/1000);
1574    piUnlock (SPIKey) ;
1575    timestamp =  bcm2835_st_read()/1000000;
1576
1577    //Data logging
1578    Logging= fopen("DataLoggingSQUARE.txt", "w");
1579    fprintf(Logging,"tid  SpAvstand Spkurs  Sphastighet
           Spvinkelhastighet AVAvstand AVkurs  AVhastighetFil
           AVhastighetRaw  AVvinkelhastighet EAvstand  Ekurs
           Ehastighet  Evinkelhastighet
```

```
        OutBoardMotorUnscaledDuty BowThrusterDuty j
        BowThrusterLeftDuty BowThrusterRightDuty\n");
1580
1581    while(DS3.button[START]==1) ;
1582
1583      while(DS3.button[START]==0){
1584
1585
1586        /************************control loop
              ******************/
1587        if(ControlTime+DelayTime<=bcm2835_st_read()){
1588
1589          /**************Angle Velocity and heading
                ****************/
1590          while (BNO055_READ_EULER_ACCESS==0); //hold for
                heading data
1591
1592          ActualValueAngleVelo= ((BNO055_EULER[0]/16.0)-
                ActualValueHeading)/(((float)(bcm2835_st_read()
                -TimestampAngleVelo))/1000000.0);
1593          if((BNO055_EULER[0]/16.0)==ActualValueHeading){
1594            ActualValueAngleVelo=0;
1595          }
1596          ActualValueHeading=BNO055_EULER[0]/16.0;
1597          BNO055_READ_EULER_ACCESS=0;
1598
1599          TimestampAngleVelo = bcm2835_st_read();
1600          ErrorHeading=SetpointHeading-ActualValueHeading;
1601          if(ErrorHeading<-180){
1602            ErrorHeading=360+ErrorHeading;
1603          }
1604          if(ErrorHeading>180){
1605            ErrorHeading=360-ErrorHeading;
1606          }
1607
1608
1609          /*****************************Rudder
                *************************/
1610          k=((280.0-420.0)/(22.0));
1611          m=426;
1612
```

```
1613          if(ActualValueVelocity>=0.025){
1614            j++;
1615            if(j>1){
1616              RudderReverse=1;
1617              j=2;
1618            }
1619          }
1620
1621          if(ActualValueVelocity<-0.025){
1622            j--;
1623            if(j<-1){
1624              RudderReverse=-1;
1625              j=-2;
1626            }
1627          }
1628
1629          RudderAngle=k*RudderReverse*ErrorHeading+m;
1630          if(fabsf(ErrorHeading)<=22){
1631            piLock (I2cKey) ;
1632            SetPWMDutyCycle(RudderAngleServo ,(int)(
                  RudderAngle));
1633            piUnlock (I2cKey) ;
1634          }
1635
1636          if(RudderReverse==1){
1637            if(ErrorHeading>22){
1638              piLock (I2cKey) ;
1639              SetPWMDutyCycle(RudderAngleServo ,280);
1640              piUnlock (I2cKey) ;
1641            }
1642
1643            if(ErrorHeading<-22){
1644              piLock (I2cKey) ;
1645              SetPWMDutyCycle(RudderAngleServo ,560);
1646              piUnlock (I2cKey) ;
1647            }
1648          }
1649
1650          if(RudderReverse==-1){
1651            if(ErrorHeading>22){
1652              piLock (I2cKey) ;
```

```
1653              SetPWMDutyCycle(RudderAngleServo ,560);
1654              piUnlock (I2cKey) ;
1655            }
1656
1657            if(ErrorHeading<-22){
1658              piLock (I2cKey) ;
1659              SetPWMDutyCycle(RudderAngleServo ,280);
1660              piUnlock (I2cKey) ;
1661            }
1662          }
1663
1664
1665          /**************Velocity and Distance
                 ****************/
1666
1667          piLock (SPIKey) ;
1668          S =((ReadMCP3004(LIDARDistanceSensor)
                 *(1291.0/962.0))/1000);
1669
1670          dt=(((float)(bcm2835_st_read()-TimestampVelo))
                 /1000000.0);
1671          ds=(S-ActualValueDistance);
1672
1673          //digital LP filter calculation
1674          ActualValueVelocity=(ActualValueVelocity*
                 LPFilterK1+LPFilterK2*(-ds/dt)+LPFilterK2*
                 ActualValueVelocityPre);
1675          ActualValueVelocityPre=-ds/dt;
1676
1677          ActualValueDistance=S;
1678          TimestampVelo=bcm2835_st_read();
1679          piUnlock (SPIKey);
1680
1681          ErrorDistance=SetpointDistance-ActualValueDistance
                 ;
1682
1683
1684          /*************Outboard motor control
                 ****************/
1685          //set point velocity
1686          if(fabsf(ErrorHeading)<=30){
```

```
1687
1688            if((fabsf(ErrorDistance))<=DistanceTolerance){
1689              SetpointVelocity=-1*ErrorDistance;
1690            }
1691
1692
1693            if((ErrorDistance)>DistanceTolerance){
1694              SetpointVelocity=-0.6;
1695            }
1696
1697            if((ErrorDistance)<-DistanceTolerance){
1698              SetpointVelocity=+0.6;
1699
1700            }
1701          }
1702          if(fabsf(ErrorHeading>30)){
1703            SetpointVelocity=0;
1704          }
1705          //error v
1706          ErrorVelocity=SetpointVelocity-ActualValueVelocity
                 ;
1707
1708          //Difference equation  PI
1709
1710          OutboardMotorUnscaledDuty=
                 OutboardMotorUnscaledDuty+(ErrorVelocity*
                 K1Velocity)+(ErrorVelocityPrev*K2Velocity);
1711          ErrorVelocityPrev=ErrorVelocity;
1712
1713          //forward Limit
1714          if(OutboardMotorUnscaledDuty>
                 OutBoardMotorForwardLimit){
1715            OutboardMotorUnscaledDuty=
                   OutBoardMotorForwardLimit;
1716          }
1717
1718
1719          //reverse Limit
1720          if(OutboardMotorUnscaledDuty<
                 OutBoardMotorReverseLimit){
1721            OutboardMotorUnscaledDuty=
```

```
                OutBoardMotorReverseLimit;
1722        }
1723
1724        OutBoardMotorDuty=OutBoardMotorScale(
               OutboardMotorUnscaledDuty);
1725
1726        piLock (I2cKey) ;
1727        SetPWMDutyCycle(OutboardMotor ,(int)
               OutBoardMotorDuty);
1728        piUnlock (I2cKey) ;
1729
1730
1731
1732        /*************BowThruster control***************/
1733        //setpoint angle velocity
1734
1735        if((fabsf(ErrorHeading<=15))){
1736          BowThrusterDuty=350*ErrorHeading;
1737        }
1738
1739        if(ErrorHeading<-15){
1740          BowThrusterDuty=-4095;
1741        }
1742
1743        if((ErrorHeading)>15){
1744          BowThrusterDuty=4095;
1745
1746        }
1747
1748
1749        if(BowThrusterDuty>BowThrusterLimit){
1750          BowThrusterDuty=BowThrusterLimit;
1751        }
1752        if(BowThrusterDuty<-BowThrusterLimit){
1753          BowThrusterDuty=-BowThrusterLimit;
1754        }
1755        //Left
1756
1757        if(BowThrusterDuty<=0){
1758          BowThrusterLeftDuty=-BowThrusterDuty;
1759          piLock (I2cKey) ;
```

```
1760            if(BowThrusterRightDuty!=0){
1761              SetPWMDutyCycle(BowThrusterRight ,0);
1762              BowThrusterRightDuty=0;
1763            }
1764            SetPWMDutyCycle(BowThrusterLeft ,(int)
                   BowThrusterLeftDuty);
1765            piUnlock (I2cKey) ;
1766          }
1767
1768          //right
1769          if(BowThrusterDuty>0){
1770            BowThrusterRightDuty=BowThrusterDuty;
1771            piLock (I2cKey) ;
1772            if(BowThrusterLeftDuty!=0){
1773              SetPWMDutyCycle(BowThrusterLeft ,0);
1774              BowThrusterLeftDuty=0;
1775            }
1776            SetPWMDutyCycle(BowThrusterRight ,(int)
                   BowThrusterRightDuty);
1777            piUnlock (I2cKey) ;
1778          }
1779          //Data logging
1780          fprintf(Logging,"%f %f  %f  %f  %f  %f  %f  %f  %f
                 %f  %f  %f  %f  %f  %f  %f  %i  %f  %f\n",((
               bcm2835_st_read()/1000000.0)-timestamp),
               SetpointDistance,SetpointHeading,
               SetpointVelocity,SetpointAngleVelocity,
               ActualValueDistance,ActualValueHeading,
               ActualValueVelocity,ActualValueVelocityPre,
               ActualValueAngleVelo,ErrorDistance,ErrorHeading
               ,ErrorVelocity,ErrorAngleVelocity,
               OutboardMotorUnscaledDuty,BowThrusterDuty,j,
               BowThrusterLeftDuty,BowThrusterRightDuty);
1781          LIDARAngleRequest=ErrorHeading*(-PI/180);
1782          ControlTime=bcm2835_st_read();
1783        }
1784        /************************End control loop
               *****************/
1785
1786        /*****************Distance Sensor direction
               *****************/
```

```
1787
1788
1789
1790
1791
1792    }
1793    piLock (I2cKey) ;
1794    SetPWMDutyCycle(RudderAngleServo ,(int)0);
1795    SetPWMDutyCycle(BowThrusterLeft,(int)0);
1796    SetPWMDutyCycle(BowThrusterRight ,(int)0);
1797    SetPWMDutyCycle(OutboardMotor ,(int)0);
1798    piUnlock (I2cKey) ;
1799
1800    fclose(Logging);
1801    while(DS3.button[START]==1);
1802  }
1803
1804
1805
1806
1807  /*****************float OutBoardMotorScale(float)
        ******************
1808   * Description: Scales the duty for the Outboard motor to
        a value that
1809   * can be handled by the ESC
1810   *
1811   * Inputs: OutboardMotorUnscaledDuty.
1812   * Output:
1813   ************************************************************
        */
1814  float OutBoardMotorScale(float OutboardMotorUnscaledDuty){
1815    if(OutboardMotorUnscaledDuty==0){
1816      return (450);
1817    }
1818
1819    if(OutboardMotorUnscaledDuty>0){
1820      return(481+OutboardMotorUnscaledDuty);
1821    }
1822
1823    if(OutboardMotorUnscaledDuty<0){
1824      return(388+OutboardMotorUnscaledDuty);
```

```
1825        }
1826
1827    }
```

B Bilaga 2

```
%Script for evaluating data and determine regulators


%% Read data from boat
%-------------------File to read from------------------%
filename = 'DataLogging4.txt';
delimiterIn = '\t';
headerlinesIn = 1;
Boat = importdata(filename,delimiterIn,headerlinesIn);

%------------------------------------------------------%


%----------------------Measured data-------------------%
t=1;
SPDistance = 2
SPHeading = 3
SPVelocity = 4
SPAngleVelocity = 5

AVDistance  = 6
AVHeading = 7
AVVelocityFiltered = 8
AVVelocityRaw = 9
AVAngleVelocity = 10

EDistance = 11
EHeading = 12
EVelocity = 13
EAngleVelocity = 14

ControlValueOutboardMotor = 15
ControlValueBowThruster = 16
%------------------------------------------------------%


%--------------------Figures---------------------------%

```

```
39 DistanceFig=1
40 VelocityFig=2
41 HeadingFig=3
42 AngleVelocityFig =4
43 %-------------------------------------------------------%
44
45
46
47 %------------------------PLOT-------------------------%
48 %Distance
49 figure(DistanceFig)
50 clf(DistanceFig)
51 hold on
52 plot(Boat.data(:,t),Boat.data(:,SPDistance),'b');
53 plot(Boat.data(:,t),Boat.data(:,AVDistance),'g');
54 plot(Boat.data(:,t),Boat.data(:,EDistance),'k');
55
56 str_legend = legend('$$Set \, point \, distance$$','
      $$Actual\, Value \, distance$$','$$Error \, distance$$'
      );
57 set(str_legend,'Interpreter','latex','FontSize',15,'
      Location','northeast');
58 title('$$Distance plot$$','Interpreter','latex','FontSize'
      ,15);
59 xlabel('$$time$$','Interpreter','latex','FontSize',15);
60 ylabel('$$distance$$','Interpreter','latex','FontSize',15)
      ;
61
62
63
64 %Velocity
65 figure(VelocityFig)
66 clf(VelocityFig)
67 hold on
68 plot(Boat.data(:,t),Boat.data(:,SPVelocity),'b')
69 plot(Boat.data(:,t),-Boat.data(:,AVVelocityFiltered),'g')
70 plot(Boat.data(:,t),-Boat.data(:,AVVelocityRaw),'r')
71 plot(Boat.data(:,t),Boat.data(:,EVelocity),'k')
72
73 str_legend = legend('$$Set \, point \, velocity$$','
      $$Actual\, value \, velocity \, filtered$$','$$ Actual
```

```
    \, value \, velocity \, raw$$','$$Error \, velocity$$')
    ;
74 set(str_legend,'Interpreter','latex','FontSize',15,'
    Location','northeast');
75 title('$$Velocity plot$$','Interpreter','latex','FontSize'
    ,15);
76 xlabel('$$time$$','Interpreter','latex','FontSize',15);
77 ylabel('$$velocity$$','Interpreter','latex','FontSize',15)
    ;
78
79
80
81 %Heading
82 figure(HeadingFig)
83 clf(HeadingFig)
84 hold on
85 plot(Boat.data(:,t),Boat.data(:,SPHeading),'b');
86 plot(Boat.data(:,t),Boat.data(:,AVHeading),'g');
87 %plot(Boat.data(:,t),Boat.data(:,EHeading),'k');
88 %plot(Boat.data(:,t),Boat.data(:,ControlValueBowThruster)
    ,'m');
89
90 str_legend = legend('$$Set \, point \, heading$$','
    $$Actual\, Value \, heading$$','$$Error \, heading$$');
91 set(str_legend,'Interpreter','latex','FontSize',15,'
    Location','northeast');
92 title('$$Heading plot$$','Interpreter','latex','FontSize'
    ,15);
93 xlabel('$$time$$','Interpreter','latex','FontSize',15);
94 ylabel('$$heading$$','Interpreter','latex','FontSize',15);
95
96
97
98 %Angle Velocity
99 figure(AngleVelocityFig)
100 clf(AngleVelocityFig)
101 hold on
102 plot(Boat.data(:,t),Boat.data(:,SPAngleVelocity),'b');
103 plot(Boat.data(:,t),Boat.data(:,AVAngleVelocity),'g');
104 plot(Boat.data(:,t),Boat.data(:,EAngleVelocity),'k');
105
```

```
106 str_legend = legend('$$Set \, point \, angle \, velocity$$
      ','$$Actual\, Value \, angle \, velocity$$','$$Error \,
       angle \, velocity$$');
107 set(str_legend,'Interpreter','latex','FontSize',15,'
      Location','northeast');
108 title('$$Angle \, velocity plot$$','Interpreter','latex','
      FontSize',15);
109 xlabel('$$time$$','Interpreter','latex','FontSize',15);
110 ylabel('$$angle \, velocity$$','Interpreter','latex','
      FontSize',15);
111
112 %-------------------------------------------------------%
113
114 %% -----------------Calculate regulator----------------%
115 s=tf('s');
116 format long
117
118 Ts=0.1 ; %Sampling time
119 L=0.15;  %Transport delay
120 Kstat=0.44; %Static amplification
121 T=1.827;  %63% of time to Kstat
122
123
124
125 %--------------------ziegler nichols -----------------%
126 %PID
127 Ti=2*L
128 Td=0.5*L
129 k=(1.2*T)/(Kstat*L)
130 %continuous
131 GZNPID=(1+(1/(s*Ti))+(Td*s))*k
132 %discret
133 HZNPID=c2d(GZNPID,(Ts),'tustin')
134 % HZNPID.num{1};
135 % HZNPID.den{1};
136
137 %PI
138 k=(0.9*T)/(Kstat*L)
139 Ti=3.3*L
140 Td=0
141
```

```
142 GZNPI=(1+(1/(s*Ti))+(Td*s))*k
143 %discret
144 HZNPI=c2d(GZNPI,(Ts),'tustin')
145 % HZNPI.num{1};
146 % HZNPI.den{1};
147
148 %-------------------------------------------------------%
149 %------------------------Lambda PID---------------------%
150 Ti=(0.5*L)+T
151 Td=(L*T)/(L+(2*T))
152 % K=((0.5*L)+T)/(Kstat*(lambda+(0.5*L)))
153
154
155
156 lambda=1*T; %speed
157 k1Lambda=((0.5*L)+T)/(Kstat*(lambda+(0.5*L)))
158
159 %continuous
160 G1Lamba=(1+(1/(s*Ti))+Td*s)*k1Lambda
161 %discret
162 H1Lambda=c2d(G1Lamba,(Ts),'tustin')
163 % H1Lambda.num{1};
164 % H1Lambda.den{1};
165
166 %2*T
167 lambda=2*T %speed
168 K2Lambda=((0.5*L)+T)/(Kstat*(lambda+(0.5*L)))
169
170 %continuous
171 G2Lamba=(1+(1/(s*Ti))+Td*s)*K2Lambda
172 %discret
173 H2Lambda=c2d(G2Lamba,(Ts),'tustin')
174 % H2Lambda.num{1};
175 % H2Lambda.den{1};
176
177 %3*T
178 lambda=3*T; %speed
179 k3Lambda=((0.5*L)+T)/(Kstat*(lambda+(0.5*L)))
180
181
182 %continuous
```

```
183 G3Lamba=(1+(1/(s*Ti))+Td*s)*k3Lambda
184 %discret
185 H3Lambda=c2d(G3Lamba,(Ts),'tustin')
186 % H3Lambda.num{1};
187 % H3Lambda.den{1};
188
189
190
191 %-------------------------------------------------------%
192 %--------------------------CHR--------------------------%
193 %PID
194 a=0.04
195 k=0.6/a
196 Ti=T
197 Td=0.5*L
198 %continuous
199 GCHRPID=(1+(1/(s*Ti))+Td*s)*k
200 %discret
201 HCHRPID=c2d(GCHRPID,(Ts),'tustin')
202 % HCHRPID.num{1};
203 % HCHRPID.den{1};
204 %PI
205
206 k=0.6/a
207 Ti=T
208 Td=0
209 %continuous
210 GCHRPI=(1+(1/(s*Ti))+Td*s)*k
211 %discret
212 HCHRPI=c2d(GCHRPI,(Ts),'tustin')
213  HCHRPI.num{1}
214  HCHRPI.den{1}
215 %-------------------------------------------------------%
216
217 %% Lp-filter
218 %-------------------File to read from-------------------%
219 clearvars Wk
220 filename = 'DataLogging.txt';
221 delimiterIn = '\t';
222 headerlinesIn = 1;
223 Boat = importdata(filename,delimiterIn,headerlinesIn);
```

```
224 %-------------------------------------------------------%
225 %--------------------Measured data----------------------%
226 t=1;
227 AVVelocityRaw = 9
228 %-------------------------------------------------------%
229
230 %-----------------------Figures-------------------------%
231
232 FrequencydomainVRaw=5
233 FrequencydomainVRawFil=6
234 VelocityRawFil =7
235 %-------------------------------------------------------%
236
237 %fft
238 n1=length(Boat.data(:,AVVelocityRaw))
239 vfft= (fft(Boat.data(:,AVVelocityRaw)/(n1)))';
240 absvfft=2*abs(vfft);
241
242 FS=10
243 Tlim=n1/FS;
244 Ts=Tlim/(n1-1);
245
246 % Frequency axis
247 for k=1 : (n1);
248 Wk(:,k) =(2*pi*FS*(k-1))/(n1);
249 end
250
251 % plot frequency domain data
252 figure(FrequencydomainVRaw)
253 clf(FrequencydomainVRaw)
254 stem(Wk,absvfft,'Color','b','Marker','none');
255 s=tf('s');
256
257 str_legend = legend('$$velocity \, raw$$');
258 set(str_legend,'Interpreter','latex','FontSize',15,'
       Location','northeast');
259 title('$$Frequrncy \, domain \, velocity \, plot$$','
       Interpreter','latex','FontSize',15);
260 xlabel('$$rad \cdot s^{-1}$$','Interpreter','latex','
       FontSize',15);
261 ylabel('$$magnitude$$','Interpreter','latex','FontSize'
```

```
    ,15);
262
263 %Lp filter
264 Lpfilter=1/(0.2*s+1)
265 tLP=linspace(0,(Boat.data(n1,t)),n1);
266 filV=lsim(Lpfilter,Boat.data(:,AVVelocityRaw),tLP);
267
268 % compare frequency domain data
269 Vfilfft=(fft(filV/n1));
270 figure(FrequencydomainVRawFil)
271 clf(FrequencydomainVRawFil)
272 hold on
273 stem(Wk,absvfft,'Color','b','Marker','none');
274 stem(Wk,2*abs(Vfilfft),'Color','g','Marker','none');
275
276 str_legend = legend('$$velocity \, raw$$','$$velocity
    filtered$$');
277 set(str_legend,'Interpreter','latex','FontSize',15,'
    Location','northeast');
278 title('$$Frequrncy \, domain \, velocity \, plot$$','
    Interpreter','latex','FontSize',15);
279 xlabel('$$rad \cdot s^{-1}$$','Interpreter','latex','
    FontSize',15);
280 ylabel('$$magnitude$$','Interpreter','latex','FontSize'
    ,15);
281
282 % compare time domain data
283 figure(VelocityRawFil)
284 clf(VelocityRawFil)
285 hold on
286 plot(Boat.data(:,t),(Boat.data(:,AVVelocityRaw)),'b')
287 plot(Boat.data(:,t),filV,'g')
288
289 str_legend = legend('$$velocity \, raw$$','$$velocity
    filtered$$');
290 set(str_legend,'Interpreter','latex','FontSize',15,'
    Location','northeast');
291 title('$$velocity \, plot$$','Interpreter','latex','
    FontSize',15);
292 xlabel('$$time$$','Interpreter','latex','FontSize',15);
293 ylabel('$$velocity$$','Interpreter','latex','FontSize',15)
```

```
    ;
294
295 % Digital transformation
296 LPDigitalt=c2d(Lpfilter,(0.1),'tustin')
297 % LPDigitalt.num{1}
298 % LPDigitalt.den{1}
299 %-------------------------------------------------------%
```

C Bilaga 3