Examensarbete, Högskolan på Åland, Utbildningsprogrammet för Maskinteknik

BYGGBESKRIVNING FÖR EN ARDUINO-BASERAD AUTOPILOT

Anpassningsbar för alla fartygstyper och kompatibel med NMEA 0183

André Haglund

2021:27

Datum för godkännande: 12.05.2021
Handledare: Kjell Dahl

EXAMENSARBETE
Högskolan på Åland

Utbildningsprogram:	Maskinteknik
Författare:	André Haglund
Arbetets namn:	Byggbeskrivning för en Arduino-baserad autopilot - Anpassningsbar för alla fartygstyper och kompatibel med NMEA 0183
Handledare:	Kjell Dahl
Uppdragsgivare:	

Abstrakt

En komplett byggbeskrivning av en billig Arduino-baserad autopilot som av användaren kan anpassas efter behov och fartygstyp. Inklusive komponentlista, ritningar, tabeller, programkod och hjälp med felsökning och anpassning.

Nyckelord (sökord)

Gör-det-själv, autopilot, arduino

Högskolans serienummer:	ISSN:	Språk:	Sidantal:
2021:27	1458-1531	Svenska	157 sidor

Inlämningsdatum:	Presentationsdatum:	Datum för godkännande:
20.04.2021	12.05.2021	12.05.2021

DEGREE THESIS
Åland University of Applied Sciences

Study program:	Mechanical Engineering
Author:	André Haglund
Title:	How to Build an Arduino-based Autopilot – Modifiable to Suit Any Ship and Compatible with NMEA 0183
Academic Supervisor:	Kjell Dahl
Technical Supervisor:	Åland University of Applied Sciences

Abstract

This is a thesis about how to construct an affordable Arduino-based autopilot that can be user modified to suit any type of vessel. Description of everything needed to build a working autopilot, as well as how to adapt it and solve possible errors are included.

Keywords

arduino, DIY, autopilot

Serial number:	ISSN:	Language:	Number of pages:
2021:27	1458-1531	Swedish	157 pages

Handed in:	Date of presentation:	Approved on:
20.04.2021	12.05.2021	12.05.2021

INNEHÅLLSFÖRTECKNING

1. INLEDNING

Efter att under årens lopp ha fått flertalet förfrågningar om jag kunde tillhandahålla en byggbeskrivning till autopiloten som jag använder i min egen fritidsbåt (och tidigare båtar) så valde jag att göra den som ett slutarbete i maskinteknik på Högskolan på Åland. Jag hoppas att detta arbete ger läsaren tillräckligt bra handledning för att själv kunna konstruera en billig och välfungerande autopilot till sin båt, som inte är låst till någon specifik tillverkares kringutrustning och med möjlighet att själv anpassa den till att fungera optimalt med avseende på fartygets svängradie, hastighet och storlek.

Handledaren till detta arbete på Högskolan på Åland har varit Kjell Dahl och utifrån har jag fått hjälp främst av signaturen ”Kuttmoped” och Jack Edwards samt även från forumen GitHub och Cruisersforum.com, samt sidor som inte längre existerar idag.

Programvaran och byggbeskrivningen i detta arbete hör till Öppen Källkod, vilket betyder att den är tillgänglig att använda, läsa, modifiera och vidaredistribuera för den som vill.

För att få använda autopiloten måste användaren godkänna att det är på sitt eget ansvar att på ett förnuftigt sätt använda autopiloten och att alternativ styrning alltid måste finnas tillgänglig i fartyget.

Denna slutgiltiga version (ca version 172) som byggbeskrivningen grundar sig på har dock visat sig att fungera väldigt väl och utan störningar från generator, startmotor, VHF, sökljus och annan utrustning som kan återfinnas i ett fartyg. Med det sagt – lämna aldrig autopiloten oövervakad.

1.1 Bakgrund

Jag har alltid haft ett stort intresse av automatisering och främst radiostyrning av först mindre bilar, båtar och flygplan som senare utvecklats till automatisering av hemmet, byggandet av större drönare och under de senaste 12 åren också utvecklandet av denna autopilot.

Den främsta orsaken till att detta projekt påbörjades var att när jag studerade vid Umeå Universitet pendlade jag till skolan från Österbotten med egen fritidsbåt, eftersom

biljettpriserna, och även tidsåtgången med färjan mellan Umeå och Vasa var ofördelaktiga. I början hade jag en litet snabbare båt som avverkade de 84 sjömilen upp till Umeå centrum på mindre än 4 h men jag hade redan då tänkt tanken på att bygga en egen autopilot efter att jag läst Henrik Florins slutarbete ”Autopilot Prototype” (Florin, 2010).

Efter att ha renoverat en träbåt som gör 7 knop och konstaterat att de färdiga autopilots-alternativen som fanns på marknaden långt överskred min budget så intensifierades arbetet med detta projekt och sedan 2015 har min båt körts med min egen autopilot. Uppskattningsvis har 800 h arbete lagts ned på utvecklingen av denna autopilot.

Henrik Florins arbete (Florin, 2010) är orsaken till att Arduino valdes som enkortsdator i detta projekt. Jag kände även till Raspberry Pi men för 10 år sedan fanns ingen färdigutvecklad autopilot-programvara till Rasberry Pi, som det gör idag med t.ex. PyPilot (D'Epagnier, 2021).

1.2 Syfte

Syftet med slutarbetet är att tillhandahålla en byggbeskrivning till min autopilot som jag under många år lovat att färdigställa till de som önskat att bygga en egen. Samtidigt bör detta vara, om man använder de komponenter som framkommer i detta arbete, det billigaste alternativet ännu år 2021.

1.3 Metod

Denna byggbeskrivning visar läsaren med bilder och text, steg för steg hur autopiloten sätts ihop, samt hur programvaran anpassas efter egna behov. Inga desto mera tidigare kunskaper av programmering eller svagströmselektronik krävs. Viss lödning och grundläggande datakunskap är det enda läsaren bör behärska för att klara av att bygga denna autopilot.

1.4 Avgränsningar

Arbetet inkluderar lättaste, billigaste och smidigaste lösningen att problemfritt få autopiloten att stabilt fungera i båten. Läsaren kan sedan själv anpassa utseende och vidareutveckling med t.ex. fjärrstyrning och vindkompensation på egen hand. I koden ingår möjlighet att via ett rf24-kort styra autopiloten men på grund at stabilitetsproblem inkluderas inte detta i

slutarbetet utan dokumentation, kopplingsschema och mjukvara hittas på Google Drive, länk finns i Bilaga 1. Förklaringar av koden i detalj görs ej utan endast det som är relevant för båtägaren presenteras. Dock finns hela koden komplett som bilaga till detta arbete så att denna autopilot alltid ska kunna byggas i framtiden, oavsett om länkarna fungerar eller inte.

1.5 Definitioner

- Arduino: Ett mikrokontrollerkort med öppen hårdvara
- NMEA 0183: En standard som bygger på seriell dataöverföring enligt RS422 inom marin elektronik
- ECDIS: Electronic chart display and information system, standard för elektroniska sjökort.
- Plotter: Gör om vektorgrafik till bilder på en skärm eller för utskrift. Syftar i det här arbetet på en navigator för båtar.
- PWM: Pulse Width Modulation (pulsbreddsmodulering på svenska). Spänningsreglering genom att sätta på och stänga av spänningen ett visst antal gånger per sekund.

2. TEORI

Nedan följer en kort beskrivning av hur en autopilot och dess komponenter fungerar.

2.1 PID

2.1.1 Vad är en PID-regulator?

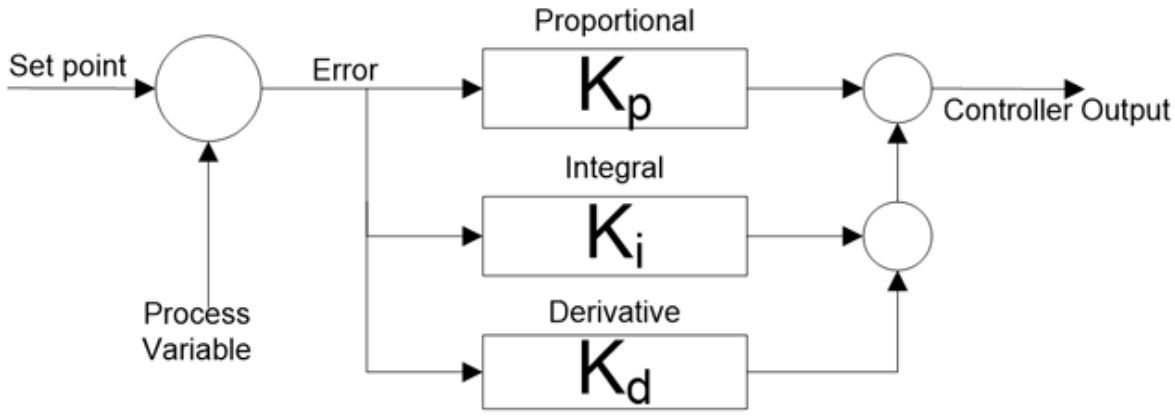
En PID-regulator är en typ av en regulator där en linjär kombination av proportionell, integrerande och deriverande verkan av ett reglerfel används för beräkning av en styrsignal. Det uppskattas att PID-regulatorer används i över 90 % av alla reglerkretsar, och den mest bekanta PID-regulatorn som de flesta kommer i kontakt med dagligen är farthållaren i bilen (Häggblom, 2021). Det finns mycket information om dessa på internet och läsaren kan lätt fördjupa sig inom området med hjälp av Google eller t.ex. genom att läsa boken ”Modern Reglerteknik” av Bertil Thomas.

2.1.2 Styrning till önskad kurs

I denna autopilot används PID för att räkna ut hur rodret skall reagera på olika kursändringar (avsiktliga eller av sjögång påverkade), enligt:

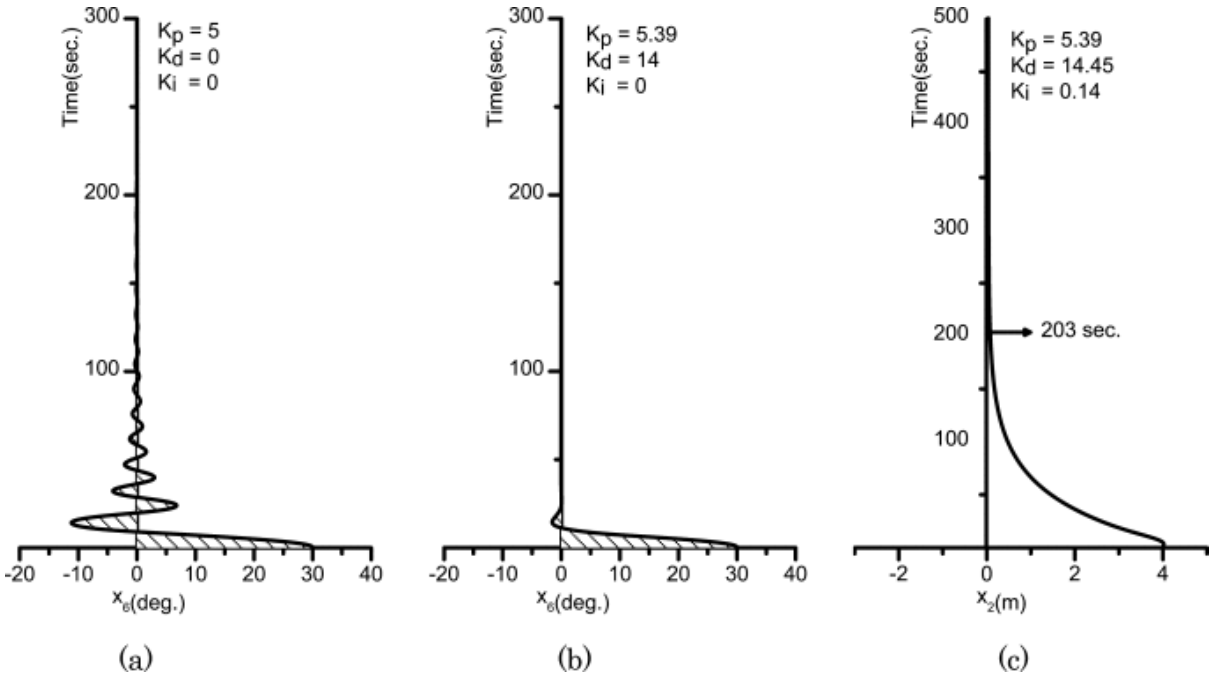
Heading Error = Current Heading – Heading Setpoint
Rudder Command(Proportional) = K * Heading Error

K är konstanten som P,I och D-delen i regulatorn multipliceras med, vilket figur 1 nedan visar:



Figur 1, blockschema av en PID-regulator (Kansagara, 2021)

I min egen båt, (som gör ca 7 knop), använder jag ett K-värde på 0,8, vilket betyder att autopiloten korrigerar rodervinkeln med 0,8 * 10° = 8° då båten befinner sig 10° från önskad kurs. Då båten återigen färdas på önskad kurs kommer kursfelet, Heading Error, att sjunka till 0°. Eftersom båten fortfarande i det skedet håller på att gira så måste rodret kompensera för detta genom att styra i motsatt riktning, och det görs med hjälp av regulatorns deriverande del, D. Den deriverande delen bestämmer alltså hur snabbt båten girar, vilket i mjukvaran beskrivs som ”Bearing Rate”. Nu har båten hittat tillbaka till den önskade kursen men avdrift från vind och sjö kan fortfarande leda till att båten förflyttas, utan att autopiloten vet om det, bort från den önskade kursen. Detta löses med den integrerande delen som eliminerar statiska reglerfel och kompenserar för störningar i reglerprocessen. Man kan alltså sammanfatta det som att P-delen gör så att båten följer önskad kurs (a)), men oscillerar över kurslinjen, D-delen gör att oscillationerna över kurslinjen försvinner (b)) och I-delen gör att båten hålls på kurslinjen trots yttre påverkan (c)), vilket bra illustreras i figur 2.



Figur 2, P,I och D-delens inverkan på fartygsstyrning (Sung-Soo Kim, 2015)

Detta ger då:

Rudder Command = Rudder Command Proportional – Rudder Command differential + Rudder Command Intergral

Vilket i mjukvaran blir:

PID_output = PID_Ks[0] * (PID_Ks[1] * heading_error - PID_Ks[2] * differential_error + integral_error, där: integral_error = integral_error + PID_Ks[3] * heading_error

Kom ihåg att vara mycket försiktig vid modifiering av I-delen i koden eftersom det lätt leder till instabilitet i autopiloten.

2.1.3 Inställning av en PID-regulator

Det, enligt mig, lättaste sättet att ställa in en PID-regulator på är att använda sig av Ziegler-Nichols metod. För att tillämpa den metoden på autopiloten behövs en mobiltelefon eller sjökortsplotter som sparar spåret man seglat med båten.

Det hela går ut på att man först sätter I och D-värdena till 0 för att sedan öka P-värdet tills instabilitet uppstår. När det inträffar mäter man frekvensen, f_0, på svängningen och följer sedan tabell 1:

Tabell 1. Ziegler Nichols metod. (Ellis, u.d.)

	K_P	K_I	K_D
P-regulator	0,5 K_{MAX}	0	0
PI-regulator	0,45 K_{MAX}	1,2 f_0	0
PID-regulator	0,6 K_{MAX}	1,2 f_0	1,2 f_0

Mer om detta i avsnitt 3.7.1 Anpassning av mjukvaran.

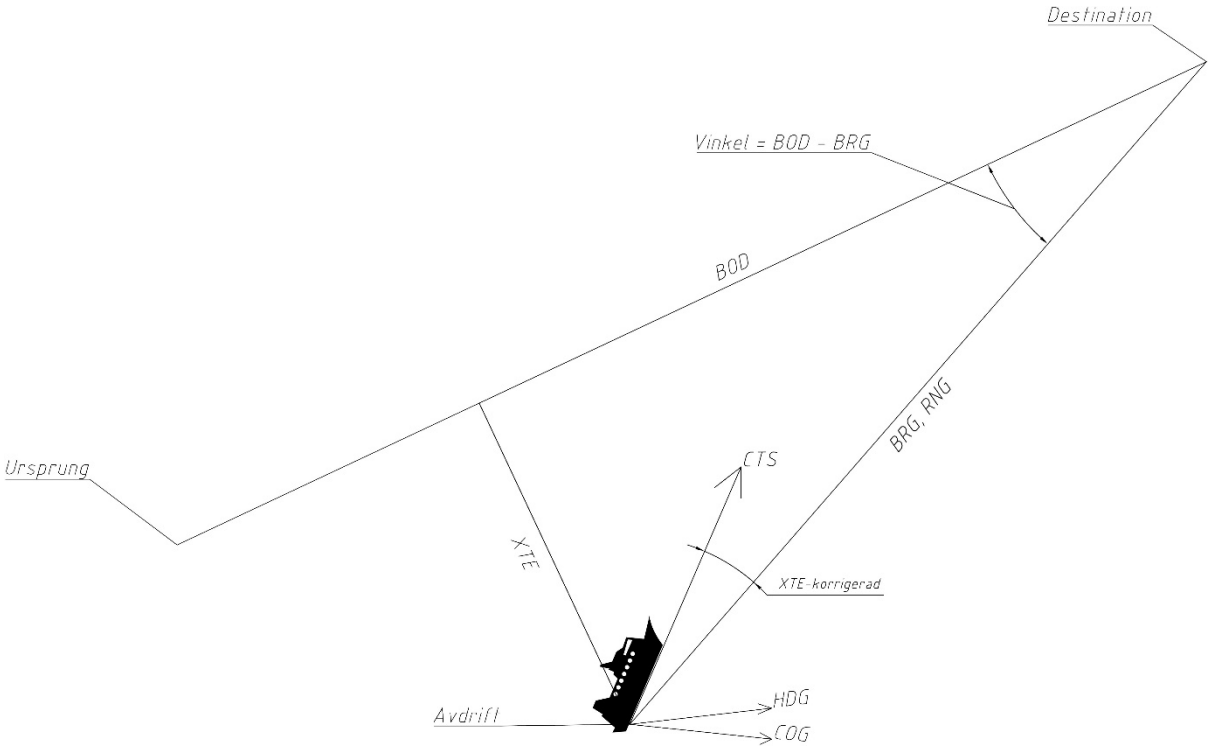
2.2 Styrning med hjälp av GPS

För att komma till sin nästa waypoint i ruttplanen måste man styra en kurs över grund, COG, den är inte nödvändigtvis den samma som kurs som båten i verkligheten måste färdas med för att nå målet. För att nå målet måste autopiloten ha en riktning att styra, HTS, och en nuvarande riktning, HDG. HDG får autopiloten från gyrokompassen och HTS räknas ut från GPS-data. GPS:en tillhandahåller autopiloten med:

- Riktning till destinationen, BOD
- Riktning till destinationen från nuvarande position, BRG

- Avstånd till destinationen från nuvarande position, RNG
- Kurs över grund, COG

I figur 3 framgår detta mera tydligt:



Figur 3. Geometrisk beskrivning av en autopilot.

Med hjälp av informationen från GPS:en är det nu möjligt att räkna ut avståndet till den önskade kurslinjen från vår nuvarande position:

XTE = RNG * sin(BOD - BRG)

För att få kursen båten skall styras efter subtraheras eller adderas vinkeln ”XTE-korrigerad” från BRG:

CTS = BRG + XTE-korrigerad

XTE-korrigerad bestäms av en PID-regulator enligt:

$$\text{XTE-korrigerad} = K_1 * \text{XTE} - K_2 (\text{XTE}_{ny} - \text{XTE}_{gammal})/(\text{Tidpunkt}_{ny} - \text{Tidpunkt}_{gammal}) + \text{XTE-korrigerad}_{gammal} + K_3 * \text{XTE-korrigerad}$$

Detta ger en kurs att styra, CTS, vilket INTE är den riktning man vill att båten skall styras med utan den riktning man vill färdas med.

För att få en kompassriktning att styra båten med så att båten färdas efter CTS, subtraheras XTE-korrigerad från BOD enligt:

CTS = BOD - XTE-korrigerad

Felet mellan riktningen man vill färdas i och riktningen man egentligen färdas i är kurs-felet, CE. För att få CE subtraheras COG från CTS enligt:

CE = CTS – COG

HTS = CE + HDG, eller:

För att få en kurs att styra som kompenserar för kompassfel, ström och vind:

HTS = BOD – XTE-korrigerad – COG + HDG

2.2.1 NMEA0183:s stora felmarginal

NMEA 0183 sänder information om avståndet till den ursprungliga kursen med en noggrannhet av maximalt 0,01 – vilket i detta fall mäts i sjömil. 0,01 sjömil, som är 18,5 meter, är ett alltför stort avstånd med en liten båt i Österbottens trånga farvatten (Bennet, 21). För att kompensera för detta beräknas avståndet till den ursprungliga kursen, XTE, med hjälp av BOD, waypointens positioner och båtens latitud och longitud. Detta minskar felmarginalen ytterligare 10 gånger till ca 1,8 meter. Längden på en grad latitud där båten geografiskt befinner sig på jorden måste därför uppges närheten av rad 200 i autopilotens första kod-flik.

3. GENOMFÖRANDE

3.1 Införskaffning av komponenter

Komponentlistan nedan är för den simplaste versionen av autopiloten, inklusive nödstyrning, för installation i båtar med 12V elsystem som tidigare hade vajerstyrning till rodret eller utombordaren/drevet. Hydraulstyrning fungerar givetvis lika bra eftersom autopiloten är byggd för att kontrollera en elmotor. Om elmotorn då sedan driver en pump eller ett linjärt ställdon spelar ingen roll.

GPS-styrning med ruttföljning från t.ex. Garmins plotter är inkluderat men autopiloten fungerar även helt utan GPS.

De flesta komponenterna går att hitta på Ebay, AliExpress och i elektronikaffärer. Kompassen och motorstyrningen rekommenderas att köpas direkt från Pololu.com eller dess återförsäljare för att minimera risken med att få en piratkopia, det samma gäller för Arduinon. Vill man vidareutrusta autopiloten med fjärrstyrning är det också viktigt att man får tag på originalet av nrf24l01-kortet.

En lödkolv av god kvalitet och lödtenn, flussmedel, krympslang, lim (smältlim), borrar i vanliga storlekar samt en bågfil är bra att ha för konstruktionen. Vidare behövs en dator som kan köra programmet Arduino IDE.

Innan man bestämmer sig för vilka komponenter som man behöver är det viktigt att fundera på vilka funktioner man vill ha. Via knappsats, eller lösa knappar, kan man välja:

0. Autopilot av
1. Styr enligt nuvarande kompasskurs
2. Styr enligt NMEA0183 insignal, om man håller in knappen kommer båten att styra nuvarande GPS-kurs
3. Kryss – för segelbåtar – ej testat
4. -10° från kursen, -100° från kursen om nummer segelbåtsprogrammet valts.
5. Båten styrs med autopilotens ratt

6. Samma som 4, fast +, girar mot styrbord alltså
7. -1° från kursen
8. Byter information på displayen
9. +1° till kursen
10. Knapparna ”*” och ”#” girar babord respektive styrbord så länge knapparna hålls intryckta, återgår sedan till föregående program

Man behöver således inte 12 st knappar för att kunna använda autopiloten, och man behöver heller ingen ratt om man inte vill. Vad knapparna gör kan givetvis flyttas om i mjukvaran. En bild på ett exempel av hur den färdiga autopiloten kan tänkas att se ut visas här i figur 4:

Figur 4. Färdig autopilot med nödstyrningen till vänster.

3.1.1 Komponentlista

Alla priser är från mars 2021.

- 1 st: Arduino Mega 2560 Rev3 – 35 €
- 3 st: Lådor, i metall, en med måtten 200*70*150 mm, en med måtten 150*70*150 mm och en med måtten 200*120*60 mm. Till autopiloten, roderstyrningen och nödstyrningen. Lådan till autopiloten måste vara i metall för att förhindra störningar. Måtten är minimimått, mindre lådor än så blir väldigt svåra att bygga autopiloten i. Ca 15 €/st.
- 1 st: Pololu High-Power Simple Motor Controller G2 18v15 – 33 €
- 1 st: Pololu Qik 2s12v10 Dual Serial Motor Controller - 65 € (OBS! Denna kan användas till både elektrisk styrning med ett linjärt ställdon eller till den hydrauliska versionen av autopiloten om man inte väljer att via ett relä och en solenoidventil

koppla över till nödstyrning från autopiloten). Denna komponent ersätter föregående Motor Controller G2 18v15 i punktlistan. Man behöver endast antingen eller.

- 1 st: Pololu AltIMU-10 v3 gyrokompass – 22 €
- 1 st: Linjärt ställdon, om hydraulstyrning och pump saknas, i samma längd som tidigare vajerstyrning, med en dragkraft på t.ex. 600N. Hastigheten bör inte underskrida 2,3°/s och rodret måste kunna svänga från 35° styrbord till 35° babord för optimal funktion (GL, 2017). Jag rekommenderar en modell som har beteckningen LX600 – 65 €
- 1 st: Pololu 5V, 5A Step-Down Voltage Regulator D24V50F5 – 12 €
- 50 st: Arduino Jumper Cables (av varierande längder, köp flest av längden 40 cm) – 3 €
- 2 st: 10 kΩ vridpotentiometrar – 2 €
- 1 st: 12 knappars knappsats (eller separata knappar om man föredrar det) – 1,5 €
- 3 st: RJ45 panelanslutningar (behövs inte men gör livet lättare om man vill kunna ta hem autopiloten från båten) – 4 €
- 1 st: Kopplingsplatta – fördelar spänningen på ett smidigt sätt till komponenterna, med t.ex. 200 uttag – 1 €
- 1 st: MAX3232 turn TTL level conversion board, för användning av GPS med NMEA 0183 signal – 0,1 € eller:
- 1 st: LCD display, LCD Board 2004 20*4 LCD 20X4 5V – 4 €
- 1 st: Joystick, 2 Position 2NO Momentary Type Monolever Joystick – 8 €
- 2 st: Relä, med 5 st anslutningar – 6 €
- 2 st: Relä, med 14 st anslutningar, inklusive hållare, MY4NJ small Electromagnetic relay, 4NO 4NC – 6 €
- ? meter: Nätverkskabel, för anslutning till gyrokompassen (installation ovanpå styrhyttens tak t.ex.) samt för anslutning av roderstyrningen och rodervinkelsensorn.
- ? meter: Förtennad kabel till autopiloten och nödstyrningen från batteriet, från nödstyrningen till rodermotorn, och från batteriet till roderstyrningen.
- 1 st: 5 Ω motstånd, för bakgrundsbelysningen till LCD-displayen – 0,1 €
- 1 st: Kondensatorer, t.ex. 100 nF, 47000 nF samt 220 µF - 0,1 €
- 1 st: Rodervinkelssensor, Honeywell RTY090LVNAA, det finns olika versioner av denna, vissa med färdig arm och vissa utan, och europeiska versionen har

anslutningen omvänd. Det spelar hur som helst ingen roll för funktionen, bara man får tag på en sensor som kan roteras +- 45° och ger en utsignal på +4,5 V. - 35 €

- ? st: Strömanslutningar, till autopiloten och till nödstyrningen. Enligt design och önskemål.
- Kretskortskontaktdon – för att fästa kablarna i mellan olika komponenter. Kablarna kan lödas fast direkt om man hellre vill det. T.ex. ”female header” på engelska. 50 st – 2 €
- 1 st: Motstånd, 4,7 Ω – 0,1 €
- 1 st: Vippströmbrytare, ON-ON, med tre ben – 0,6 €
- 2 st: Vippströmbrytare, ON-ON, med sex ben (godkänd för lika många ampere som rodermotorn) – 4 €
- (1 st: Vippströmbrytare, ON-ON, med 12 ben, istället för 2 st med 6 ben)

3.2 LCD och knappsats

De komponenter som behövs för att komma igång med projektet är: Arduino Mega 2560, en 10k Ohm potentiometer, strömkälla (rekommenderas starkt att en dyr stabil strömkälla som den i komponentlistan införskaffas), LCD-display, kopplingsplatta samt kablar för anslutning mellan de olika komponenterna.

3.2.1 Ihopkoppling av de första komponenterna

Nedan följer vilka anslutningsnummer på Arduinon som komponenterna ansluts till. Dessa kan i viss mån flyttas så länge som portnumret även uppdateras i programvaran. Det är väldigt viktigt detta kontrolleras flera gånger eftersom om det inte blir rätt leder det till många timmars felsökning.

För att förhindra störningar är det också **ytterst viktigt** att strömkällan och alla komponenter jordas i samma punkt i lådan som autopiloten installeras i, detta görs enkelt genom att borra ett hål i metallådan i vilket man skruvar i en skruv som sedan alla kablar jordas i.

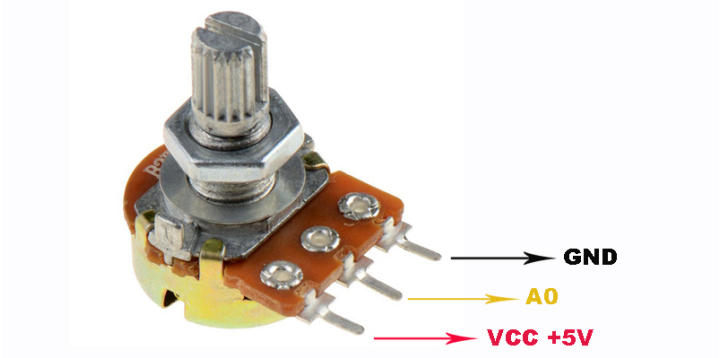
3.2.1.1 LCD-displayen

LCD-displayen ansluts till Arduinon och strömförsörjningen enligt tabell 2.

Tabell 2. Anslutning av LCD till Arduino.

Arduino anslutningsnummer	***LCD anslutningsnummer***	***Strömförsörjning***
-	1, VSS	Jord (minus)
-	2, VDD	+5 V
7, rekommenderas inte, PWM kontroll av kontrasten.	3, V0	Mittenbenet på en 10 kΩ pot. För kontroll av kontrast.
D39	4, RS	-
-	5, RW	Jord (minus)
D41	6, E	-
-	7, 8, 9, 10	-
D43	11, DB4	-
D45	12, DB5	-
D47	13, DB6	-
D49	14, DB7	-
	15, LED+	+5 V, i serie med ca 5 Ω motstånd (t.ex. 4,7 Ω)
	16, LED-	Jord (minus)

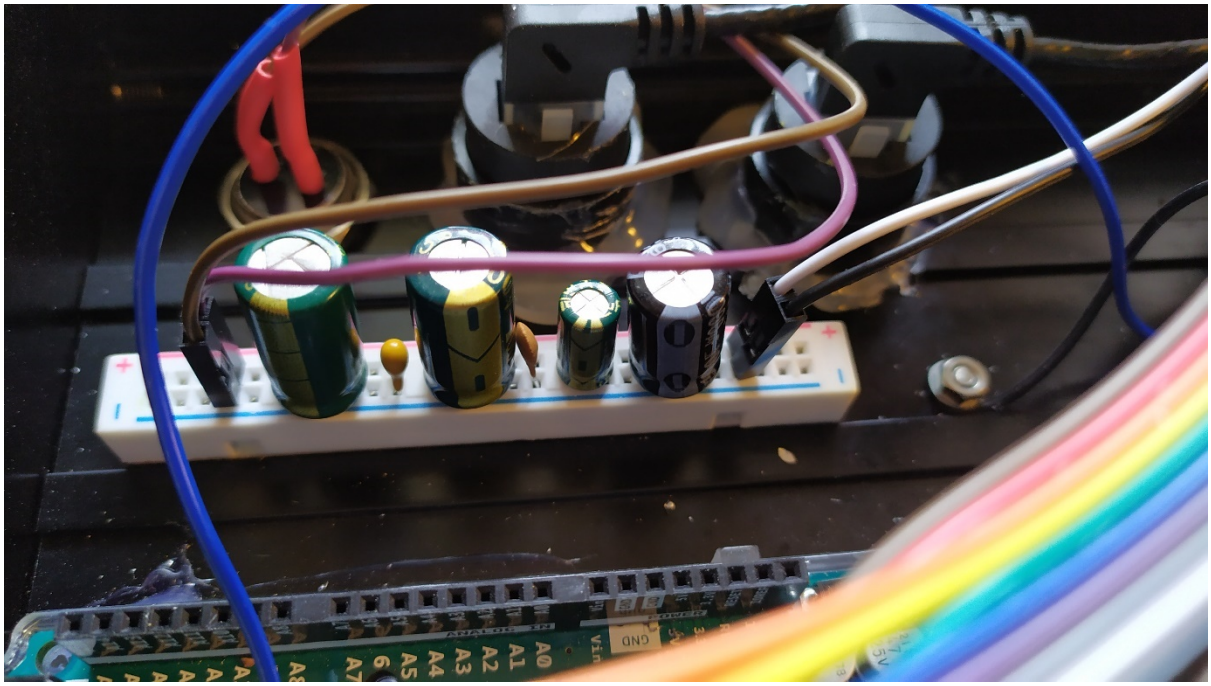
En potentiometer kopplas in enligt figur 5, med benet i mitten anslutet till komponenten man vill reglera.



Figur 5. Inkoppling av en potentiometer (S, 2021)

3.2.1.2 Strömkällan och kopplingsplatta

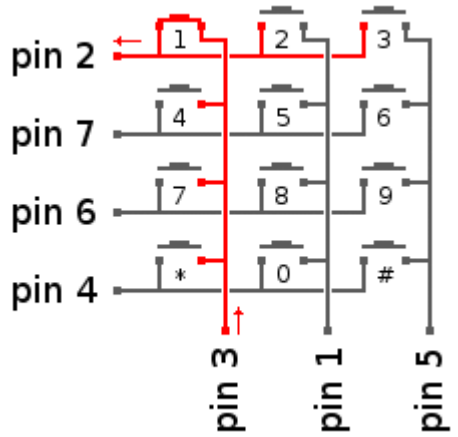
Strömkällan ansluts till båtens batteri och sedan ansluts utgångarna till en lämplig rad på kopplingsplatta. Innan strömförsörjningen från kopplingsplattan ansluts till Arduinon (anslutningarna +5V och GND) och övriga komponenter rekommenderas det att man installerar några parallellkopplade kondensatorer för att motverka störningar från t.ex. mobiltelefoner. Elektrolytkondensatorerna måste anslutas till plus och minus på rätt sätt., vilket illustreras i figur 6. De keramiska kondensatorerna har inte någon skillnad polaritet.



Figur 6. Parallellkoppling av kondensatorer med minus-benet kopplat till minus.

3.2.1.3 Knappsatsen

Knappsatsen med 12 knappar är uppbyggd enligt en matris som illustreras i figur 7. Vill man inte, eller behöver man inte ha alla funktioner som denna autopilot erbjuder så kan man istället ersätta knappsatsen med ett lämpligt antal knappar och ansluta dem så att de matchar anslutningsnumren på raderna och kolumnerna i figur 7.



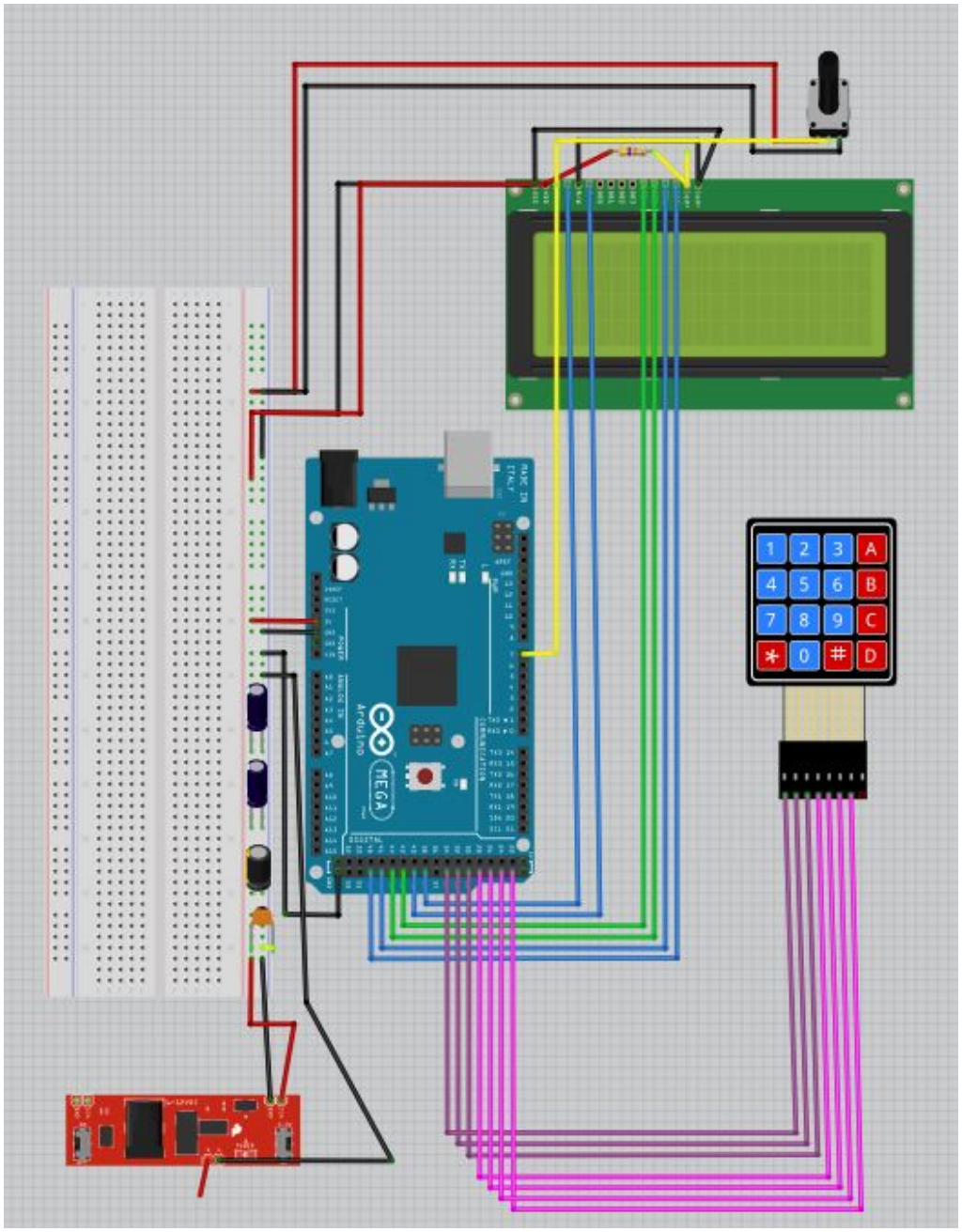
*Figur 7. Illustration av hur en 3*4 knappars knappsats är uppbyggd.* (Electronoobs, 2021)

I tabell 3 beskrivs hur knappsatsen ansluts till Arduinon. Kom ihåg att det är mycket viktigt att man kontrollerar att anslutningarna stämmer överens med koden.

Tabell 3. Anslutning av knappsats till Arduino

Arduino anslutningsnummer	***Knappsatsens anslutningsnummer*** *(från vänster till höger)*
35	1
33	2
31	3
29	4
27	5
25	6
23	7

Nu är det klart för att testas innan man går vidare i byggnationen och autopiloten borde se ut enligt figur 8. Knappsatsen som illustreras är inte den som används i verkligheten (16 istället för 12 knappar).



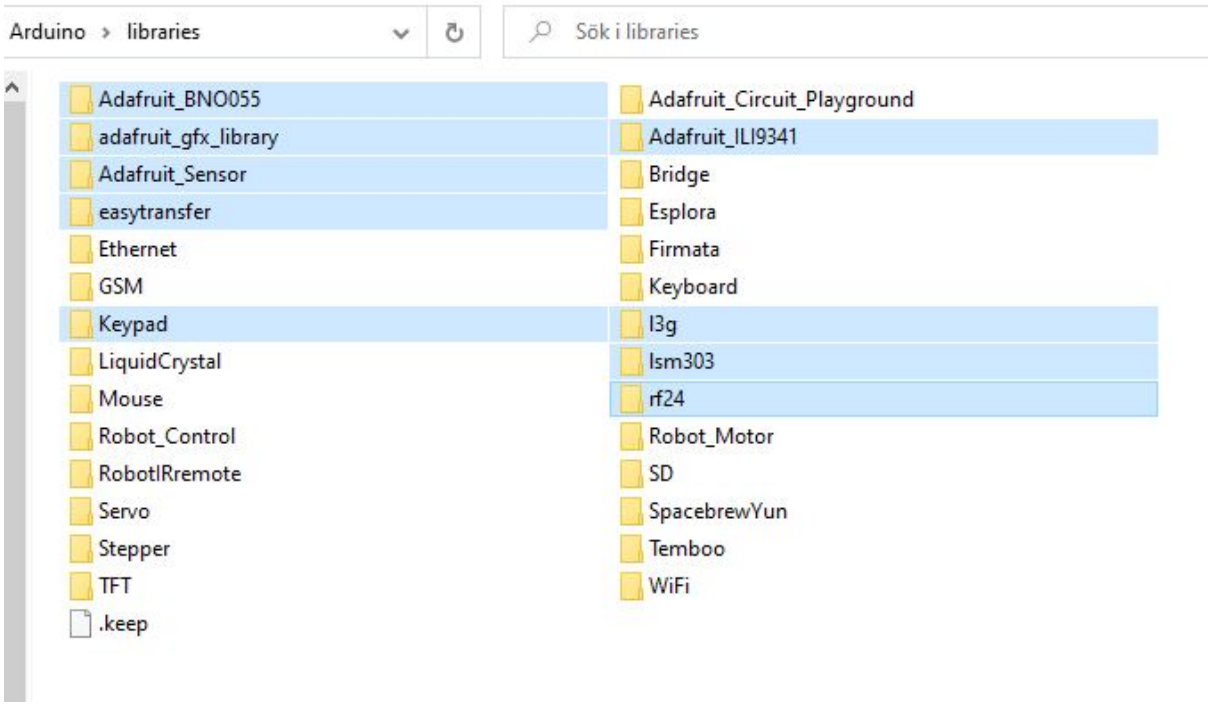
Figur 8. Klart för första testen.

3.3 Arduino IDE

Innan man kan provköra autopiloten för att verifiera att projektet gått rätt hittills så måste man bekanta sig med Arduino IDE. Programmet krävs för att kunna överföra programvaran till Arduino Mega 2560, och det kan gratis laddas ned från Arduinos hemsida.

3.3.1 Bibliotek

Då programmet är installerat måste man först skapa en mapp för detta projekt, lämpligen i Arduino IDE:s installationsmapp. Följande steg är att installera de bibliotek (Libraries) som fattas för att installera komponenterna som inte redan finns med i grundinstallationen. Detta görs enklast genom att ladda ned dem från länken till Google Drive i Bilaga 1, och helt enkelt kopiera in dem i Libraries-mappen. Skulle länken försvinna så är de bibliotek som fattas i grundinstallation markerade i figur 9 och går att enkelt hitta i Arduino IDE:s program under ”Skiss -> Inkludera bibliotek -> Hantera bibliotek”.



Figur 9. Bibliotek som fattas i grundinstallationen.

3.3.2 Test

Ladda ned innehållet i mappen ”Knappsats_test” från Google Drive, alternativt kopiera in texten som finns i bilaga 2 i en ny flik i Arduino IDE.

Följande steg är att i Arduino IDE välja ”Verktyg -> Kort -> Arduino Mega 2560”. Välj efter detta ”Skiss -> Verifiera/kompilera”. Slutligen återstår bara att välja ”Ladda upp”, som är knapp två från vänster under menyraden.

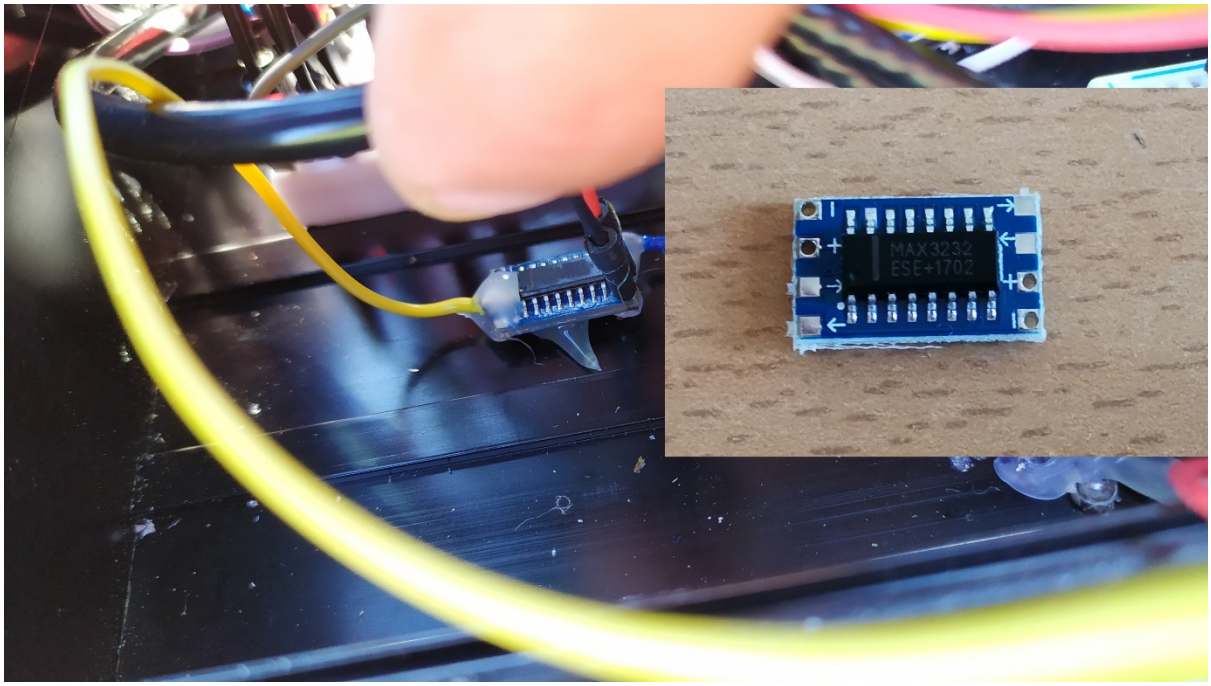
När detta är gjort skall man starta Seriell Monitor, som finns under ”Verktyg -> Seriell monitor. Välj ”Baud Rate” 57600. Tryck, håll in och släpp alla knappar. Resultatet skall nu visas på både datorn och LCD-skärmen.

Nu är det dags att montera ihop återstående komponenter av autopiloten.

3.4 Återstående komponenter

3.4.1 NMEA 0183

För att få GPS-indata krävs ett litet kretskort som heter MAX3232 och som finns fotograferat i figur 10.



Figur 10. MAX3232, NMEA 0183 till Arduino TTL.

Inkopplingen sker genom att man ansluter NMEA 0183 utsignalen (+) till in-pilen på vänstra sidan av kretskortet, med beteckningen ”RS232”. På högra sidan av kretskortet ansluts strömförsörjningen och utsignalen. NMEA 0183 utsignalen markerad med ett minustecken ansluts till jord och den omvandlade NMEA 0183 utsignalen från MAX3232-kortet ansluts till plats nummer ”RX3” på Arduinon.

MAX3232-kortet stöder 3,3-5 V spänning men utvecklar hög värme om det ansluts till 5 V. Jag rekommenderar att man använder Arduinons 3,3 V utgång till strömförsörjningen. Sammanfattat ansluts MAX3232 enligt beskrivningen i tabell 4.

Tabell 4. Anslutning av MAX3232 till Arduino.

Arduino anslutningsnummer	***MAX3232 anslutningsnummer*** *(från vänster till höger)*	***Strömförsörjning***	***Från GPS/PC***
-	Input (på RS232-sidan av kortet)	-	NMEA +
3,3 V	+ (på TTL-sidan av kortet)	-	-
-	- (på TTL-sidan av kortet)	Jord	-
RX3	Output (på TTL-sidan av kortet)	-	-

3.4.2 Gyrokompassen

Mjukvaran stöder kompasser med benämningen BNO055, IMU9 v2 och v3, IMU10 v3 och IMU9 v5. Sedan 2016 har jag använt IMU10 v3 med goda resultat men användare av den vid långvarig svår sjögång rapporterar om bristande funktion. Detta kan man lösa genom att installera den på t.ex. ett stabiliserat kamerastativ men man måste självklart ta i beaktande störande magnetfält vid en sådan installation. Jag har aldrig upplevt dessa problem i finländska farvatten.

Gyrokompassen ansluts enligt tabell 5 och installeras på så långt avstånd som möjligt från störande magnetfält och gärna i båtens centerlinje, med kretskortets anslutningar (bakre delen) riktade mot aktern. Gyrokompassen är inte vattentät och om den skall installeras utanpå båten måste den göras vattentät genom att t.ex. gjutas in i epoxi. Om man gör detta så slutar samtidigt IMU-versionernas barometer att fungera.

Tabell 5. Anslutning av AltIMU-10 v3 till Arduino.

Arduino anslutningsnummer	***AltIMU-10 v3 anslutningsnummer***	***Strömförsörjning***
SDA 20	SDA	-
SCL 21	SCL	-
-	Vin	+5 V

-	GND	Jord

3.4.3 Rodrets motorkontroll

Under årens lopp har jag använt olika motorkontroller och haft olika lösningar för nödstyrningen ombord på min båt men har de senaste åren använt mig av Pololus G2 18v15 motorkontroll samt relästyrd nödstyrning. Detta har fungerat bäst och varit den billigaste lösningen. Om man vill så finns även alternativet med mjukvarustyrd omkoppling till nödstyrningen med hjälp av en Pololu Qik 2s12v10 *dual motor controller* men detta förutsätter att inget fel med mjukvaran inträffar när man vill gå över till nödstyrning. Båda motorkontrollerna ansluts på samma sätt enligt tabell 6. Mer ingående byggbeskrivning kommer i kapitel 3.5.

Tabell 6. Anslutning av motorkontroll till Arduino.

Arduino anslutningsnummer	***Motorkontrollens anslutningsnummer***	***Strömförsörjning***
TX2 16	RX	-
-	GND	Jord
-	**GND**	-12 V
-	**VIN**	+12 V
-	**OUT A** (eller M0) till rodermotorn	-
-	**OUT B** (eller M1) till rodermotorn	-

3.4.4 Rattstyrning

Rattstyrningen (krävs inte för att autopiloten skall fungera) är en 10 kΩ potentiometer med mittenbenet anslutet till ”A2” på Arduinon och respektive ben till +5 V och jord. Potentiometern skall anslutas så att spänningen ökar då man vrider den medurs.

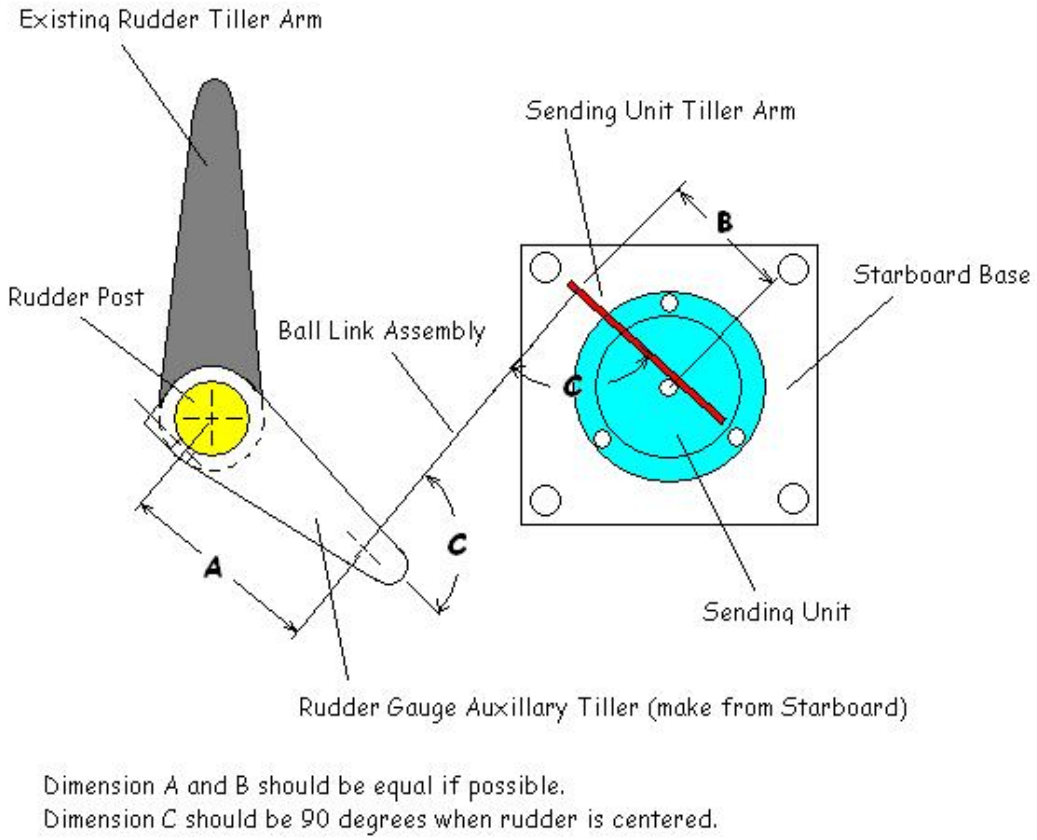
3.4.5 Rodervinkelssensor

Rodervinkelssensorn krävs inte för att autopiloten skall fungera men ger ett betydligt bättre resultat. Den ansluts till +5 V, jord och Arduinons analoga anslutning ”A4” (enligt medföljande bruksanvisning beroende på vilken modell man använder). Honeywells sensor

har en 3-vägs AMP-anslutning om man önskar att använda den istället för att löda fast ledningarna.

Längden på armen som monteras på rodervinkelsensorn skall ha samma längd som armen på hjärtstocken som styr rodrets vinkel. Exempel på detta illustreras i figur 11 och 12.

Figur 11. Installation av rodervinkelssensor i min båt och i ett riktigt handelsfartyg



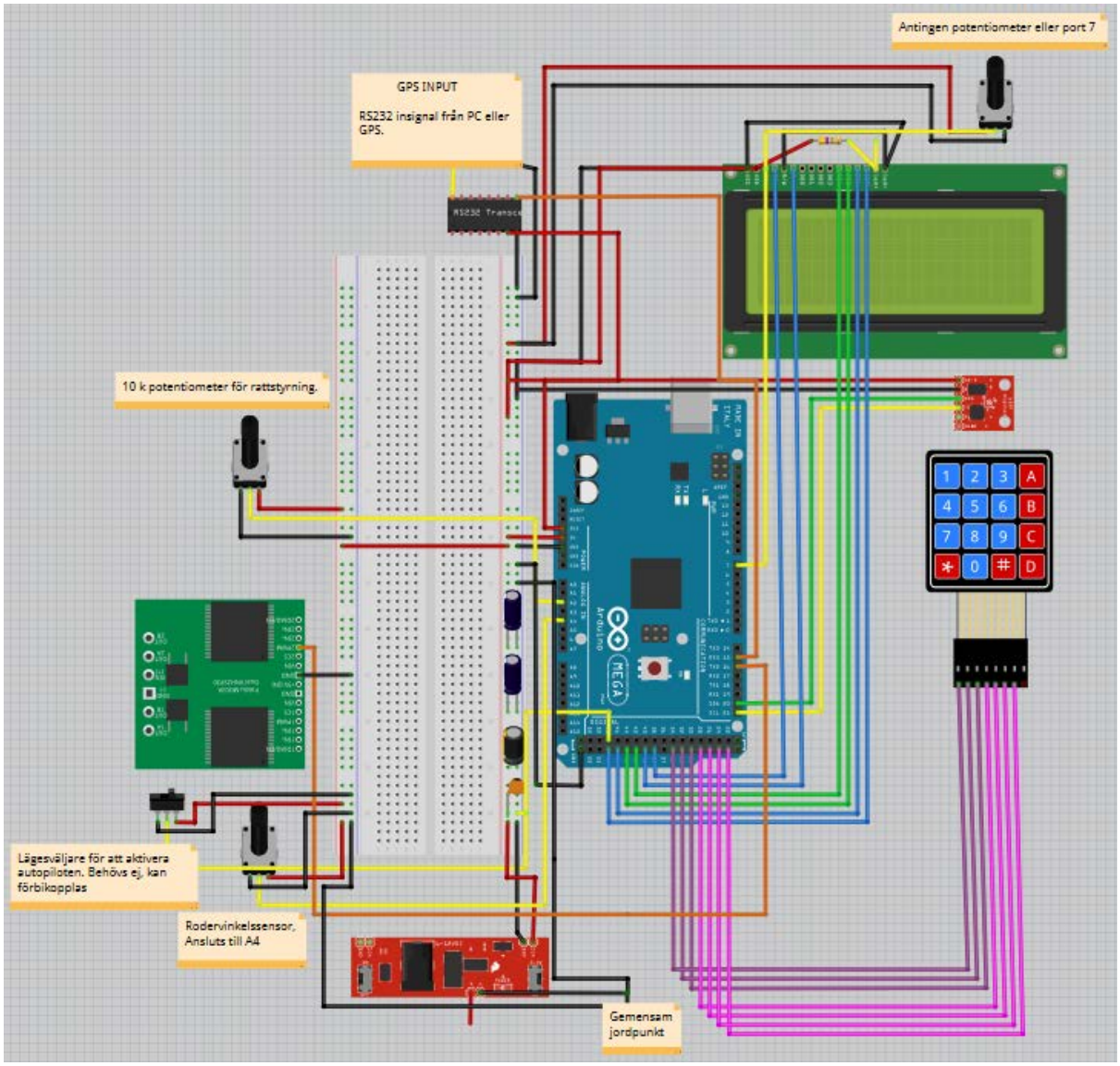
Figur 12. Exempel på installation av rodervinkelssensor (Johnson, 2007).

3.4.6 På/Av-knapp

Denna funktion behövs för att autopiloten skall fungera, men går att koppla förbi genom att ansluta anslutning nummer 48 på Arduinon till 5 V. Utan spänning på port 48 kommer autopiloten inte att starta. Vill man kunna stänga av autopiloten när man t.ex. kör med manuell styrning utan att bryta strömmen till den rekommenderas en ON-ON-vippströmbrytare med mittenbenet kopplat till anslutning nummer 48 och respektive ben på sidan om till 5 V och jord. Rodrets vinkel visas även med autopiloten avstängd och saknar man den funktionen i båten är det således en bra idé att låta autopiloten vara påslagen även om nödstyrningen används.

3.4.7

Nu är autopiloten färdig och skall ha följande utseende som presenteras i figur 13:



Figur 13. Skiss på färdig autopilot

3.5 Styrning av rodret

Autopiloten har visat sig att lida av besvärliga störningsbekymmer som under många år var svåra att hitta en lösning på. För att undvika detta och få en pålitlig autopilot måste autopiloten och roderstyrningen förses med separerad spänning.

3.5.1 Strömförsörjning av autopilot och roderstyrning

För att växla mellan nödstyrning och autopilot behövs en tvåvägs vippströmbrytare med sex ben. Strömförsörjningen kopplas till mittenbenen och ena sidans två ben skall förse nödstyrningen med ström och den andra sidans två ben skall kopplas till de två 14-bensreläna så att roderstyrningen strömsätts då man slår över från nöd- till automatisk styrning.

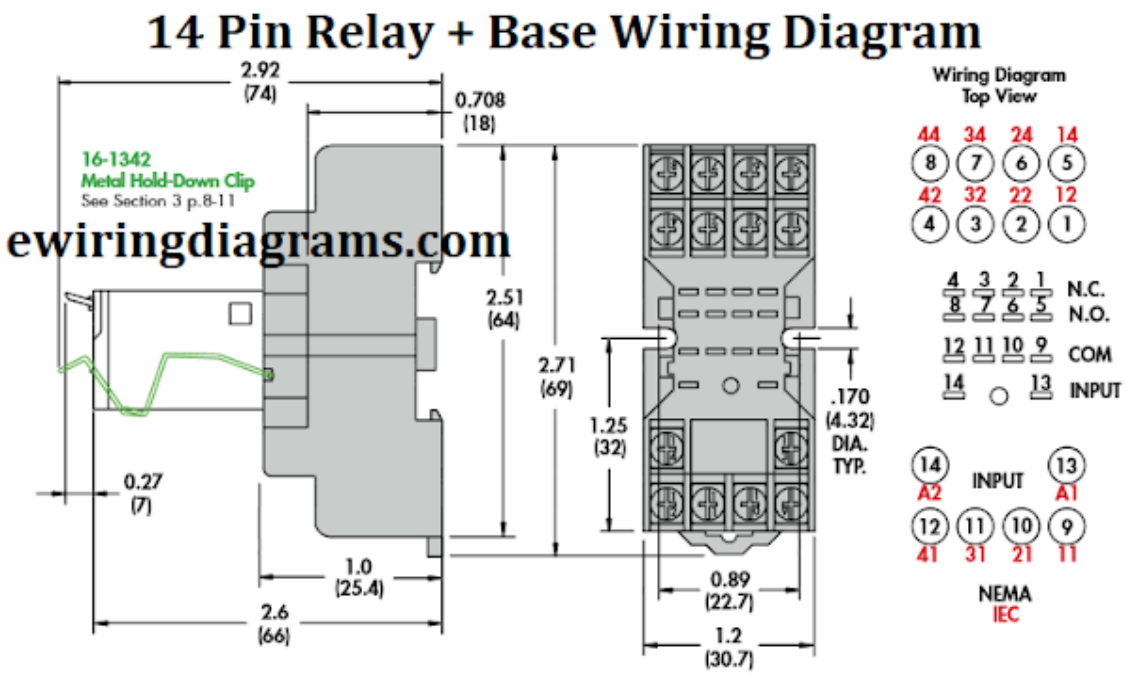
3.5.2 Avskiljning av roderstyrningen

Till de två 14-bensreläna, som aktiveras av vippströmbrytaren ovan, skall man koppla enligt tabell 7:

Tabell 7. Avskiljning av Arduinon och roderstyrningen.

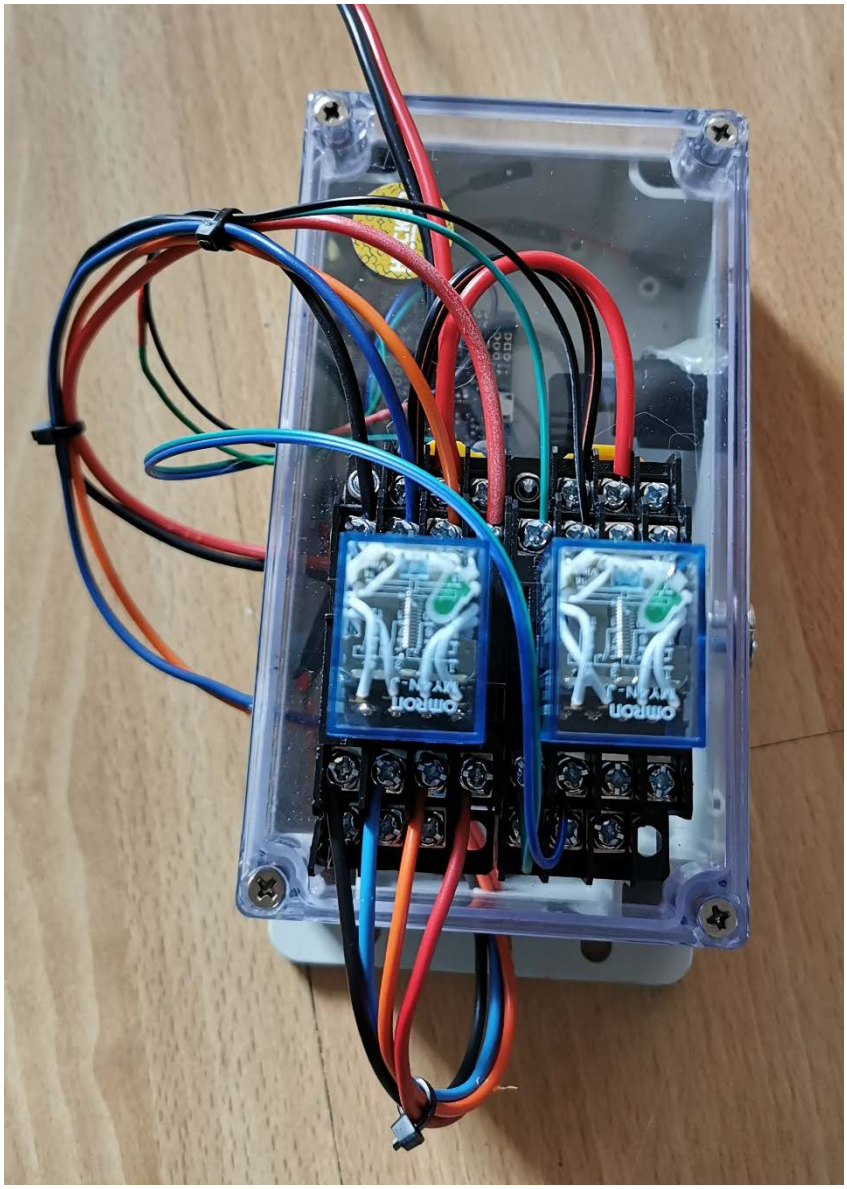
Från komponent:	***In till relä:***	***Ut från relä:***	***Till komponent:***
Vippströmbrytare 12 V	INPUT 13	-	-
Vippströmbrytare jord	INPUT 14	-	-
Vippströmbrytare 12 V	INPUT 13 (relä 2)	-	-
Vippströmbrytare jord	INPUT 13 (relä 2)	-	-
Batteri 12 V	COM 9	NO 5	Motorstyrning 12V
Batteri jord	COM 10	NO 6	Motorstyrning jord
Roderstyrning +	COM 11	NO 7	Rodermotor +
Roderstyrning -	COM 12	NO 8	Rodermotor -
Arduino jord	COM 9 (relä 2)	NO 5 (relä 2)	Motorstyrning GND
Arduino signal TX2	COM 10 (relä 2)	NO 6 (relä 2)	Motorstyrning RX

Benens nummer och position illustreras av figur 14:



Figur 14. Kopplingsschema för ett 14-bens relä. (Ullah, 2021)

Då styrningen av rodret är färdigt kan det se ut som i figur 15. Här saknas anslutningarna från batteriet och till roderstyrningen ännu, men alla andra kablar är anslutna.



Figur 15. Roderstyrning med reläfrånskiljning.

3.6 Nödstyrning

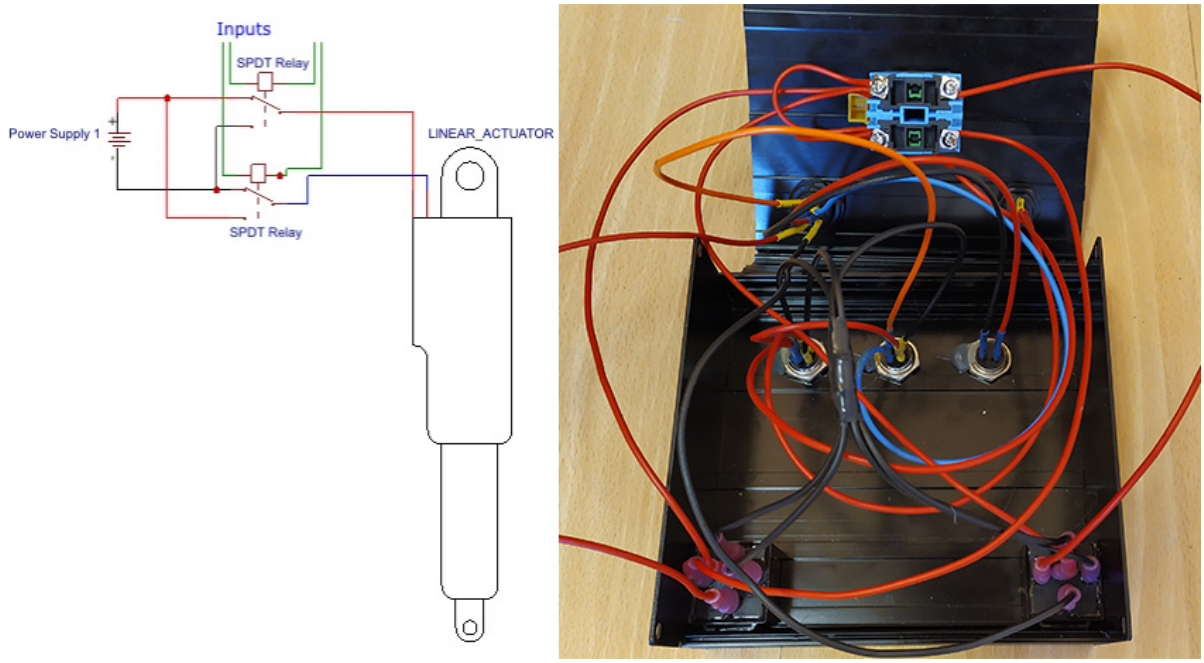
Det måste alltid finnas ett alternativ att styra båten med ifall autopiloten skulle få något fel, och om båten inte har den möjligheten att styras manuellt så måste en elektrisk nödstyrning installeras. Detta löses t.ex. smidigt med två stycken SPDT (Single-Pole Double-Throw) och en joystick. De ansluts enligt tabell 8:

Tabell 8. Inkoppling av nödstyrning.

Från komponent:	***In till relä:***	***Ut från relä:***	***Till komponent:***

Batteri 12V	-	-	Ena ingången på joysticken
Batteri 12V	-	-	Andra ingången på joysticken
Batteri 12V	87	-	-
Batteri 12V	87 (relä 2)	-	-
Batteri jord	87a	-	-
Batteri jord	87a (relä 2)	-	-
Batteri jord	85	-	-
Batteri jord	85 (relä 2)	-	-
Ena utgången på joysticken	86	-	-
Andra utgången på joysticken	86 (relä 2)	-	-
-	-	30	Rodermotor + (eller tvärtom beroende på önskad riktning)
-	-	30 (relä 2)	Rodermotor - (eller tvärtom beroende på önskad riktning)
Pololu motorstyrning, ERR			Aktivering av strömförsörjning till nödstyrningen samt summer via relä (om man vill)

Nödstyrningen installeras i en separat metallåda för att motverka magnetfälten som skapas av dess relän. Den förses med spänning från en tvåvägs strömbrytare med de två mittenbenen anslutna till båtens strömkälla, med ena sidan ansluten till joystick och relän samt andra sidan ansluten till autopilotens DC-DC omvandlare. Nödstyrningen kan nu se ut som i figur 13.

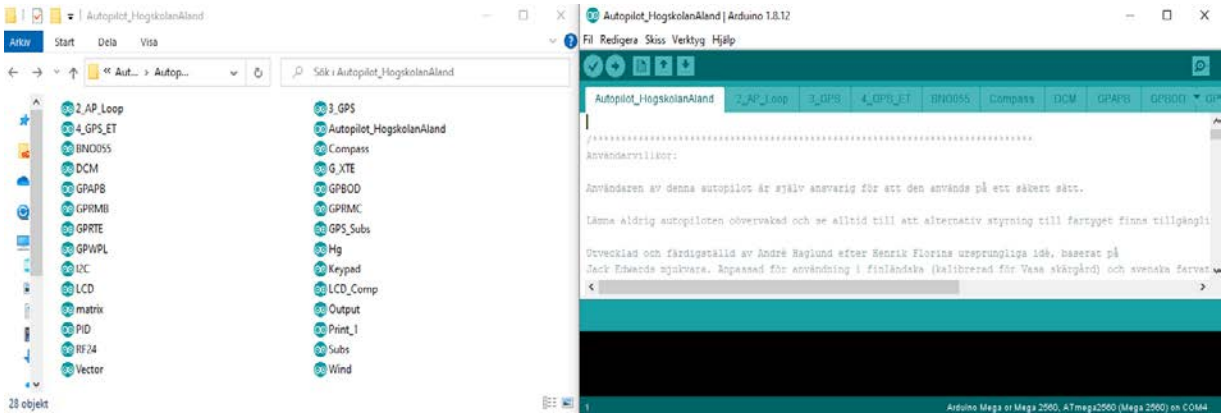


Figur 16. Nödstyrning med relän samt ritning över kopplingarna.

Vill man automatiskt slå över till nödstyrningen kan man via utgången ”ERR” på motorstyrningen (gäller alla motorstyrningar som beskrivs i detta arbete) ansluta ett 3,3 V relä som aktiverar strömförsörjningen till nödstyrningen, samtidigt som den t.ex. aktiverar en 12 V summer som via en ljudsignal uppmärksammar operatören om att något gått fel.

3.7 Installation av mjukvaran

För att autopiloten skall fungera måste man skapa en mapp på datorn som man lämpligen döper till ”Autopilot”. I mappen kommer det att finnas ett antal programfiler enligt figur 17. Dessa går antingen att ladda ned från Google Drive (Bilaga 1) eller att skapa själva genom att välja ”Fil -> Ny” i Arduino IDE och klistra in koden från respektive del i bilagorna. När detta är gjort och filerna är uppladdade till Arduinon så skall autopiloten starta och fungera.



Figur 17. Filerna som behövs för att köra autopiloten.

3.7.1 Anpassning av mjukvaran

Direkt när man öppnat huvudfilen i autopiloten, ”AP_HA.ino” kommer man vid ca rad 30 till ett avsnitt som heter ”Väljs av användaren”. Här framgår det i koden tydligt vad de olika parametrarna kontrollerar och användaren kan välja vad som passar bäst. Kom ihåg ” MagVar_default” vid rad 61 eftersom trots att det är en gyrokompass så sänder den en magnetisk kompassriktning till autopiloten och detta behöver därför korrigeras.

3.7.1.1 PID

PID parametrarna är idag anpassade för en båt med en längd av 10 meter och en topphastighet på 7 knop. I koden visas detta som ” PID_Ks[4] = {2, .4, 2, .0005}”, där första värdet multipliceras med alla tre efterföljande. ”.4” betyder att rodret kompenserar med 2 * 0,4 * 10° vid en kursavvikelse på 10°. Följande tal, ”2”, beskriver hur många grader rodret kommer att svänga med för att förhindra att man svänger över kurslinjen då autopiloten tagit båten tillbaka till önskad kurs. Om man har 10° roderutslag och girar med en hastighet av 6° per sekund så kommer autopiloten att svänga rodret med 10° - 2 * 6° = -2° i motsatt riktning då giren är avklarad. Sista värdet är det integrerande värdet och beräknas enligt:

integral_error = integral_error + PID_Ks[3] * heading_error

Detta värde begränsas till +-10° i koden för att inte orsaka problem.

Man kan utgå från sig själv för att komma fram till det andra, proportionella värdet i PID-regulatorn (alltså 0,4 i exemplet ovan) genom att själv fundera över hur många grader man skulle vilja svänga rodret vid en kursavvikelse på 10°. PID-regulatorn utvecklades delvis just

av Nicolas Minorsky observationer av hur en rorsman styrde ett fartyg i början av 1900-talet. (Minorsky, 1922)

3.7.1.2 Avläsning av NMEA (för att få GPS och ruttföljning att fungera)

Först av allt måste man kontrollera att GPS:en fungerar, detta kan man bekräfta t.ex. genom att kontrollera att UTC-klockan tickar i en av autopilotens displayer. Om den inte gör det och man inte vill använda GPS-styrning måste man i PID-fliken runt rad 357 ändra ”GPS_Was_Available = true” till false.

Om möjligt, välj sedan i inställningarna till plottern/ECDIS maximalt antal decimaler den sänder NMEA-data med och skapa och aktivera en rutt att följa.

I närheten av rad 113 finns det något som heter ” boolean print_NMEA = 0; //1 = ja, 0 = nej”, detta behövs för att veta hur många decimaler som plottern/ECDIS sänder data med om man är osäker.

Under fliken GPRMB återfinns sedan vid rad 41:

```
string1 = data_RMB[10];
    NMEA_TO_FLOAT(3);
returns float3
    Range_Destination = float3;
```

”RMB[10]” betyder det 10:e ordet i RMB-strängen , string1 är nya namnet på RMB[10] och ” NMEA_TO_FLOAT(3)” är antalet decimaler (3) som datan innehåller. Modifiera så att det passar aktuell inkommande data. Om UTC-fungerar i displayen så har autopiloten i varje fall kontakt med GPS:en och felet ligger inte i inkopplingen av MAX3232-kortet.

Kontrollera även övrig data under GPRMB-fliken vid problem.

3.7.1.3 Längden av en grad latitud

Anges i meter vid ca rad 200. Detta är viktigt för att få en exakt autopilot men behöver inte modifieras vid användning inom Norden.

3.7.1.4 Rodrets hastighet

I närheten av rad 400 börjar ett avsnitt i koden som kontrollerar rodermotorns hastighet. Väljer man en för liten motorspeed så kommer inte rodret att röra på sig då små korrektioner av kursen krävs eftersom spänningen till rodret blir för låg. Det är inte heller rekommenderat att med ett snabbt fartyg använda en för hög roderhastighet. Detta kan dock kompenseras för genom att modifiera ”kryss”-läget i autopiloten, ursprungligen tänkt för segling, till att t.ex. begränsa rodermotorns hastighet här till 20 %. I dagsläget är det inställt på 50 % av ovan inställd roderhastighet och återfinns på rad 58; ” float Tack_rudder_speed = .5”.

3.7.1.5 Kalibrering av kompassen

Data från kalibrering av kompassen förs in i koden vid rad 500. Mer om detta i avsnitt 3.7.2

3.7.1.6 Namnge båten

Man kan definiera ett eget namn på sitt fartyg (maximalt 20 tecken) vid rad 700.

3.7.2 Kalibrering av kompassen

I bilagorna finns kod som heter ”Kalibrering av kompassen” (bilaga 31). Gör som tidigare och ladda upp denna till Arduinon. Rotera sedan gyrokompassen i alla riktningar, X, Y och Z och avläs värdena på skärmen. För sedan in dessa vid rad 500 i koden till autopiloten. Kalibreringen görs helst så långt ifrån magnetiska föremål som möjligt.

I biblioteket som tillhör kompassen, lsm303, finns även där en kalibreringsfil, med den skillnaden att autopiloten måste vara ansluten till datorn för att se vilka värden man får.

Då kompassen är kalibrerad och autopilot-koden uppdaterad och laddar man upp den till autopiloten igen.

Nu skall allting fungera och övriga modifikationer i kodens andra flikar är upp till användaren att modifiera.

4 PROBLEM OCH FELSÖKNING

4.1 Omstarter

De största problemen jag haft med autopiloten har varit de oförklarliga omstarterna av Arduinon som jag omöjligen kunde hitta orsaken till.

Efter att ha försökt komma till rätta med problemen genom att installera optiska isolatorer på signalöverföringen till roderstyrningen utan resultat blev jag tvungen att försöka hitta en lösning på annat sätt. Kondensatorer av olika värden installerades på likströmsmotorn som styr rodret men även detta gav inga positiva resultat. Till slut kom jag fram till att problemet måste bero på induktiv återledning från rodermotorn.

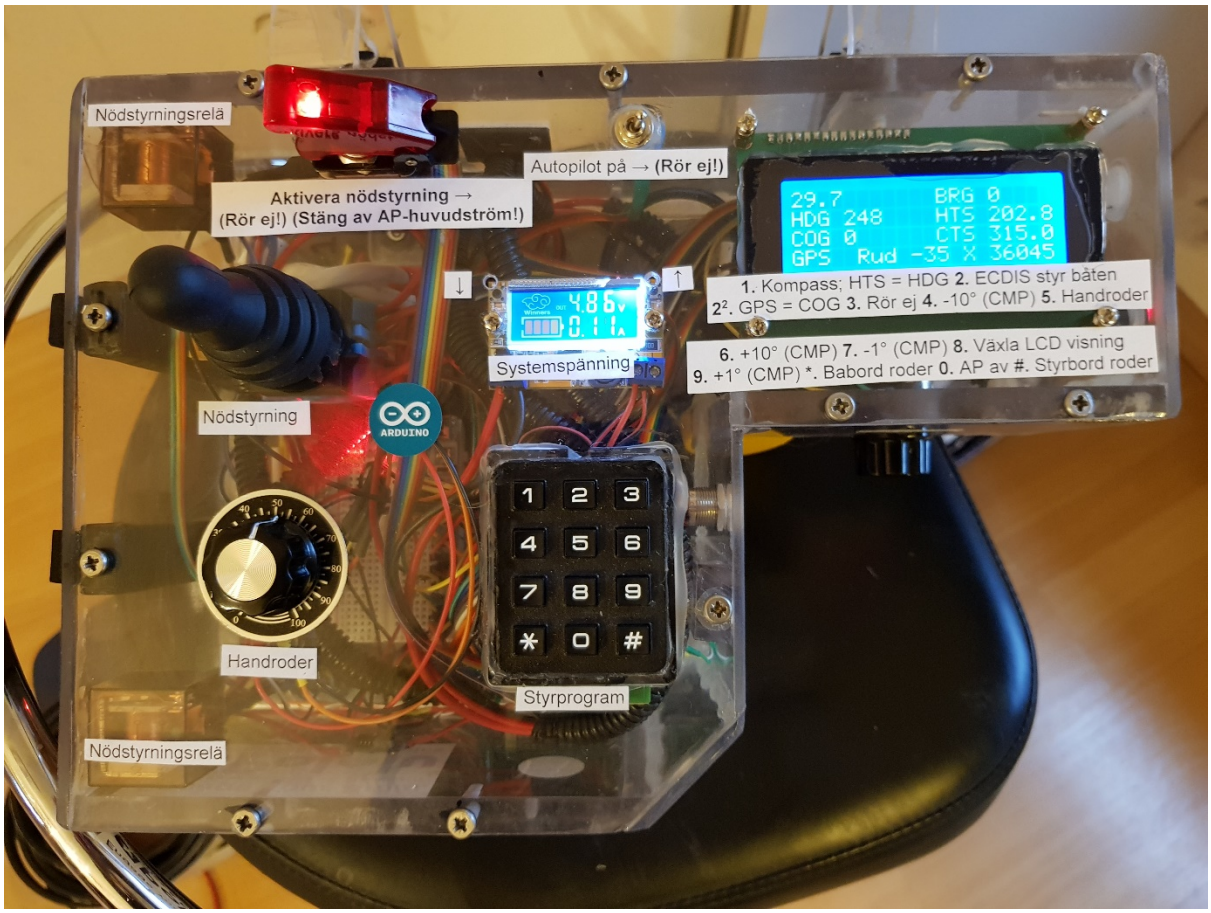
Detta problem kan lösas med hjälp av dioder men den möjligheten finns inte i autopiloten eftersom motorn snurrar både medsols och motsols. En slutgiltig lösning blev att förse roderstyrningen med egen strömförsörjning, direkt från båtens batteri, och inte via eltavlan.

Strömförsörjningen styrs istället via relän från eltavlan så att roderstyrningen inte är strömsatt samtidigt som nödstyrningen styr rodret. Denna lösning måste man installera eftersom om kretskortet (Pololu G2 18v15), som i vanliga fall styr rodret, parallellkopplas med nödstyrningen, kommer spänning att matas bakvägen in i kretskortet. Då leds 12 V via signalkabeln in till Arduinon och den stänger i bästa fall av sig, samtidigt som roderstyrningen kortsluts. Så är inte fallet om man använder en Pololu Qik 2s12v10 istället, men denna utvecklar hög värme om man matar den fel väg och brinner även den upp efter ca 30 minuters felaktigt användande.

4.2 Inbyggnad

Första versionen av autopiloten byggdes i en plastlåda tillsammans med nödstyrningen, vilket illustreras i figur 18, dels för att visuellt kunna felsöka lösa kablar och dels för att det gav ett häftigt intryck av elektroniken på insidan. Detta ledde till att autopiloten istället blev känslig för yttre störningar, framförallt från mobiltelefon, nödstyrningsrelä och VHF. Efter att 2018

ha installerat autopiloten i en separat metallåda enligt figur 19, och nödstyrningen i en annan separat metallåda, upphörde även dessa problem.



Figur 18. Autopiloten installerad i en plastlåda.

Figur 19. Den slutgiltiga installationen.

4.3 Temperaturkänslighet

Minusgrader är något som autopiloten inte gillar, men eftersom jag har värme i båten och enkelt även kan ta med mig autopiloten hem har inte detta varit något bekymmer jag försökt att åtgärda. Problem vid höga temperaturer på sommaren har jag inte upplevt trots att t.ex. gyrokompassen är installerad i direkt solljus ovanpå båtens tak.

4.4 Trasiga kablar

Kablarna mellan de olika komponenterna är inte fastlödda i min egen version av autopiloten, men de som hänger upp och ned i locket är fastlimmade med smältlim. Ingen kabel har ännu lossnat men det har skett vid flera tillfällen att en kabel gått av. Jag skulle hur som helst rekommendera att kablarna löds fast då man är nöjd med resultatet samt att förtennade kablar av god kvalitet används istället för de som rekommenderas i komponentlistan.

4.5 Problem med komponenter

Rent kvalitetsmässigt har det inte varit några problem med komponenterna som ingår i autopiloten, utan den har varit driftsäker över förväntan i det upplägget som detta arbete handlar om.

Det är viktigt att strömkällan som omvandlar 12 V till 5 V är av god kvalitet, som t.ex. den från Pololu i komponentlistan, eftersom komponenterna har visat sig vara känsliga för störningar orsakade av billigare strömkällor.

För att förhindra att autopiloten svänger fram och tillbaka vid vindstilla väder gör den inte korrektioner för kursavvikelser på mindre än två grader. Detta visade sig bli ett problem eftersom gyrokompassen genererar en girhastighet på mer än det trots att båten inte svänger. Lösningen blev att lägga till ” Bearingrate_smoothed = .95 * bearingrate_smoothed + .05 * bearingrate” i koden.

Autopiloten påbörjar också giren i den aktuella versionen innan den når fram till en waypoint, istället för att gira efter att den nått fram till en waypoint och komma långt bort från kursen ifall det handlar om en 90° kursändring.

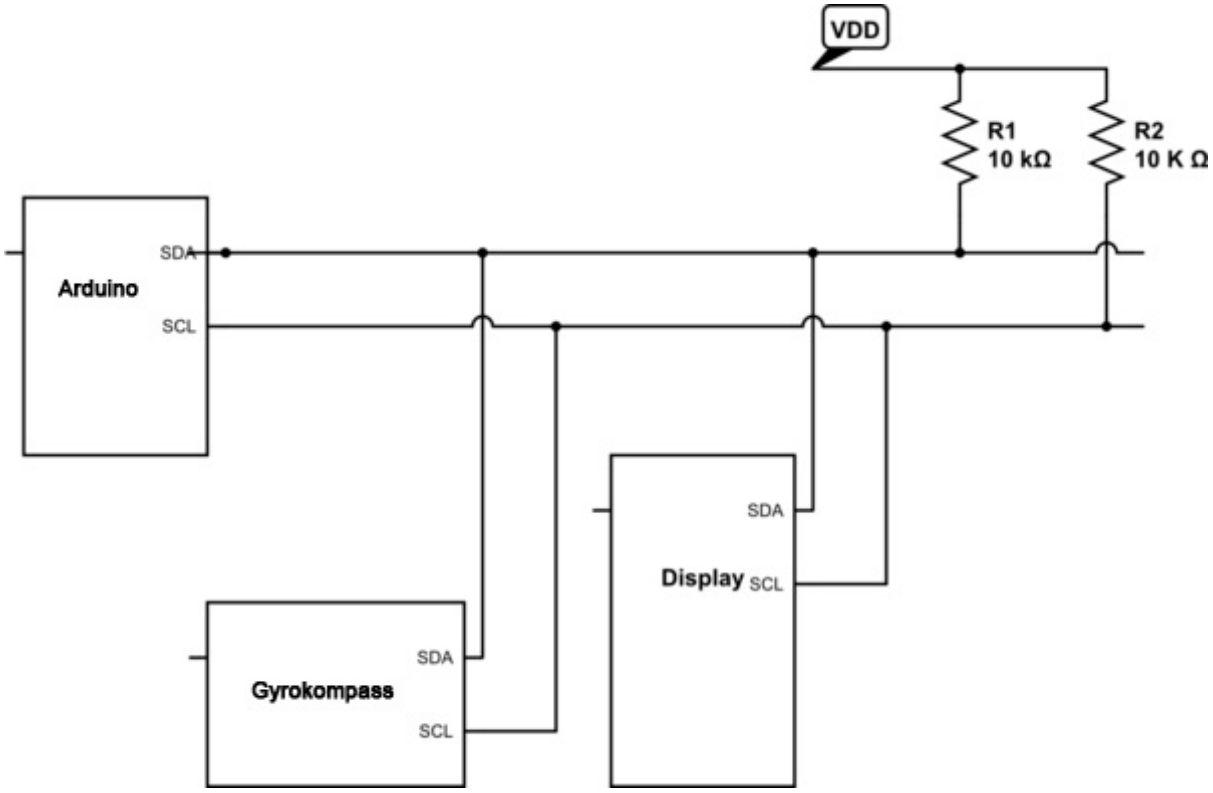
PWM-reglering av LCD-displayens kontrast orsakade mycket störningsproblem och ersattes istället med en potentiometer.

Vid enstaka tillfällen hakar gyrokompassen upp sig och svänger mellan +-45° från aktuell kurs, detta löses genom att antingen göra en 360° gir med båten eller genom att starta om autopiloten. Man kan pröva att förlänga ”delay” tiden runt rad 705 för att undvika detta problem, men det har inträffat så pass sällan att det är svårt att säga vad som orsakar problemet. Lämna som sagt var aldrig autopiloten oövervakad, och använd den inte i trånga farleder – precis som med alla andra autopiloter.

För att kunna använda autopiloten med polaroidiserande solglasögon har jag försökt att byta ut LCD-skärmen mot en motsvarande OLED-skärm men har efter många försök inte lyckats

med det eftersom hela koden måste skrivas om då LCD och OLED är två helt skilda tekniker. Det visade sig vara så mycket arbete att det inte var värt besväret att fortsätta då dokumentationen dessutom var bristfällig.

I samband med att detta arbete slutfördes våren 2021 har det dock kommit ut modernare alternativ till OLED-ersättningar av 20*4 LCD-skärmar som styrs via parallellkoppling in på I2C-protokollet tillsammans med gyrokompassen via SDA och SCL anslutningarna på Arduinon. Denna lösning kräver inte i lika mycket ändring av koden utan endast de LCD-relaterade delarna av koden måste modifieras, t.ex. ”lcd.print” måste bytas ut till ” display.println”. Jag rekommenderar att man undersöker detta vidare om man önskar att använda en OLED- istället före en LCD display. Exempel på inkoppling visas i figur 20.



Figur 20. Parallellkoppling av I2C för användning av OLED-display.

4.5.1 Fjärrstyrning

Jag hade också stora bekymmer med fjärrstyrningen av autopiloten som inte inkluderas i detta slutarbete och främst då på grund av piratkopior och strömförsörjning av nrf24l01-kortet – vilket ledde till störningar av annan elektronik.

4.6 Support och dokumentation

Bristfällig dokumentation av enskilda komponenter samt diskussioner i forum på internet av andra användare som inte nödvändigtvis har kunskapen de påstår sig ha om en komponent eller ett problem har också tagit mycket tid – främst för att jag själv bara har grundläggande programmeringskunskaper och inte alltid har möjlighet att avgöra vad som är rimligt eller inte.

Det har också varit svårt att få hjälp av Garmin med tolkning av deras NMEA-signal.

5 RESULTAT OCH SLUTSATS

Det var inte några större problem att bygga en första version av denna autopilot på skrivbordet, med endast kompasstyrning, utan problemen började när jag skulle försöka att få den att fungera tillsammans med annan elektronik i båten. Jag var för ett år sedan nära att ge upp hela projektet då jag trots att allting annat fungerade tillfredsställande inte blev av med omstarterna även fast jag installerat kondensatorer och optiska avskiljare enligt alla rekommendationer.

Bortsett från det problemet som nu äntligen är löst så är det en välfungerande autopilot som styr väldigt rakt oavsett väder, vilket figur 21 visar.

Figur 21. Autopiloten styr rakt över Bottenhavet.

Autopiloten har även varit tillförlitlig i skärgårdsnavigation och den har tagit mig flertalet gånger till Umeå genom den inre skärgården upp till Vasa och sedan norr om Björkö, genom världsnaturarvet och Valsörarna vidare till Umeå vilket visas i figur 22. De små kursavvikelserna har varit på grund av mötande trafik.

Figur 22. Spår efter båten genom Vasas inre skärgård.

Tidsåtgången innan jag var nöjd med resultatet blev mångfaldigt det jag hade förväntat mig men prismässigt blev det mycket billigare än vad en autopilot på marknaden kostar idag, speciellt med tanke på att jag kan använda denna på alla båtar resten av livet.

Kunskapsmässigt har detta projekt främst gett insikt i vilka problem man kan stöta på då man försöker installera utrustning i miljöer med mycket störningar trots att man har en färdigbyggd fungerande produkt på skrivbordet.

Avslutningsvis hoppas jag att de som läser detta slutarbete kommer att få nytta av autopiloten och all tid jag lagt ned på den.

KÄLL- OCH LITTERATURFÖRTECKNING

Bennet, P. (21, 04 23). *What is NMEA 0183?* From NMEA FAQ: http://www.vanhuystee.com/faq_nmea183.html

D'Epagnier, S. (2021, 03 18). *https://pypilot.org/*. From PyPilot.

Electronoobs. (2021, 04 15). *Arduino Keypad*. From Electronoobs: https://electronoobs.com/eng_arduino_tut124.php

Ellis, G. (n.d.). *Four types of Controllers*. From Science Direct: https://www.sciencedirect.com/topics/computer-science/ziegler-nichols-method

Florin, H. (2010). *Autopilot Prototype.* Stockholm: KTH, SE ISSN-1653-5715.

GL, D. (2017). Part 4 Systems and components. *RULES FOR CLASSIFICATION*, p. 18.

Häggblom, K. (2021, 03 15). *Åbo Akademi.* From http://users.abo.fi/khaggblo/RT/RTk7.pdf

Johnson, A. (2007, 07 16). *Installing a rudder*. From Boat Project: http://www.boat-project.com/electra/rudder.htm

Kansagara, R. (2021, 03 14). *Circuit Digest*. From https://circuitdigest.com/article/what-is-pid-controller-working-structure-applications

Minorsky, N. (1922). Directional stability of automatically steered bodies. *J. Amer. Soc of Naval Engineers*, 280-309.

S, O. (2021, 04 19). *Use potentiometer to control a servo*. From Osoyoo: https://osoyoo.com/2014/12/09/use-potentiometer-to-control-a-servo/

Sung-Soo Kim, S.-D. K. (2015, 7). Study on variation in ship's forward speed under regular waves depending on rudder controller. *Int. J. Nav. Archit. Ocean Eng*, pp. 364-374.

Ullah, H. (2021, 04 20). *Ewirediagrams*. From https://www.ewiringdiagrams.com/2020/11/14-pin-relay-wiring-diagram-base-wiring.html

BILAGOR

Bilaga 1: Google Drive-länk till bilder och dokumentation

Bilaga 2: Program för att testa knappsats och LCD

Bilaga 3: Autopilotens startsida i programmet

Bilaga 4: Programmet som styr hur ofta autopiloten uppdateras

Bilaga 5: GPS. Del 1 av 10

Bilaga 6: GPS. Del 2 av 10

Bilaga 7: GPS. Del 3 av 10, NMEA

Bilaga 8: GPS. Del 4 av 10, NMEA

Bilaga 9: GPS. Del 5 av 10, NMEA

Bilaga 10: GPS. Del 6 av 10, NMEA

Bilaga 11: GPS. Del 7 av 10, NMEA

Bilaga 12: GPS. Del 8 av 10, sammansättning

Bilaga 13: GPS. Del 9 av 10, gir innan WayPoint

Bilaga 14: GPS. Del 10 av 10, avstånd från kurslinje

Bilaga 15: BNO055 kompass

Bilaga 16: Pololu kompass, del 1 av 2

Bilaga 17: Pololu kompass, del 2 av 2

Bilaga 18: Pololu lufttryck

Bilaga 19: I2C

Bilaga 20: Knappsatsen

Bilaga 21: LCD

Bilaga 22: LCD, kompassvisning

Bilaga 23: Pololu output

Bilaga 24: PID

Bilaga 25: Framställning

Bilaga 26: RF24 fjärrstyrning

Bilaga 27: Kompasskompensering

Bilaga 28: Pololu vektor

Bilaga 29: Kompensation för vind

Bilaga 30: Pololu matris

Bilaga 31: Kalibrering av kompass

Bilaga 32: Knappsats till kompasskalibrering

Bilaga 1: Länk till Google Drive:

https://drive.google.com/drive/folders/1-ALKd8FNJvt4qAiHiePX7XyvRV398G_c?usp=sharing

För att läsa ritningarna behövs programmet Fritzing, som återfinns på webbplatsen: https://fritzing.org/download/

Programmet har tidigare varit gratis (de äldre versionerna finns att ladda ned gratis om man söker) men kostar idag 8 €.

Bilaga 2: Program för att testa knappsats och LCD

Namnge programmet till t.ex. ”Knappsats_test”, kopiera och klistra in i Arduino IDE. Spara som ”Knappsats_test”.

```
/*********************************
För att testa att LCD och knappsats fungerar
*************************************************/

/* @file EventSerialKeypad.pde
|| @version 1.0
|| @author Alexander Brevig
|| @contact alexanderbrevig@gmail.com
||
|| @description
|| | Demonstrates using the KeypadEvent.
|| #
*/

/* För testning av de första komponenterna installerade i autopiloten.
   0 tryck - Autopilot AV
   1 tryck - Båtens styrs med gyrokompassen
   2 tryck - Båtens styrs med GPS
   3 none
   4 tryck - minska kursen med 10 grader
   5 none
   6 tryck - öka med 10 grader
   7 tryck - minska med 1 grad
   8 none
   9 tryck - öka med 1 grad
   * tryck/släpp - Båten svänger babord
   # tryck/släpp - Båten svänger styrbord
*/

#include <Keypad.h>

// LCD Displayen:
#include <LiquidCrystal.h>
// Här skriver man in vilka portar man använder på Arduinon
LiquidCrystal lcd(39, 41, 43, 45, 47, 49);

// Knappsatsen
const byte ROWS = 4; //rader
const byte COLS = 3; //kolumner
byte rowPins[ROWS] = {25,35,33,29}; //anslut till raderna på knappsatsen
byte colPins[COLS] = {27,23,31}; //anslut till kolumnerna på knappsatsen

char keys[ROWS][COLS] = {
  {'1','2','3'},
  {'4','5','6'},
  {'7','8','9'},
  {'*','0','#'}
};

Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );
```

```
char input;
int mode = 0;
String str_mode = "OFF";
float heading = 15;
float heading_to_steer;

void setup(){
  Serial.begin(57600);
  keypad.addEventListener(keypadEvent); //add an event listener for this
keypad
  lcd.begin(20,4);  // initialize LCD for 4 row 20 characters
  lcd.setCursor(0,0);
}

void loop(){
  // put if (sw1) here then if(!sw1) get key
  char key = keypad.getKey();
  if (key != NO_KEY) {
    //lcd.print(key);
  }
lcd.setCursor (0,3);
lcd.print(str_mode);
}

//take care of some special events
void keypadEvent(KeypadEvent key){

  delay(20);
  switch (keypad.getState()){
    case PRESSED:
    Serial.print("Key = "); Serial.println(key);
      switch (key){
        case '0':
          mode = 0;
          str_mode = "OFF";
          lcd.begin(20,4);
          lcd.setCursor(0,0);
          lcd.print("Key = ");
          lcd.print(key);
          lcd.setCursor(0,1);
          lcd.print("MODE = OFF");
          break;

        case '1':
          mode = 1;
          str_mode = "COMPASS";
          lcd.begin(20,4);
          lcd.setCursor(0,0);
          lcd.print("Key = ");
          lcd.print(key);
          lcd.setCursor(0,1);
          lcd.print("mode = compass");
          heading_to_steer = heading;
          lcd.setCursor(0,2);
          lcd.print("HTS = ");
          lcd.print(heading_to_steer);
          break;

        case '2':
          mode = 2;
          str_mode = "GPS";
```

```
            lcd.begin(20,4);
            lcd.setCursor(0,0);
            lcd.print("Key = ");
            lcd.print(key);
            lcd.setCursor(0,1);
            lcd.print("mode = GPS");
            break;

            case '3':
            mode = 2;
            str_mode = "TACK";
            lcd.begin(20,4);
            lcd.setCursor(0,0);
            lcd.print("Key = ");
            lcd.print(key);
            lcd.setCursor(0,1);
            lcd.print("mode = TACK");
            break;


          case '4':
              str_mode = "Left 10";
              heading_to_steer = heading_to_steer -10;
              if (heading_to_steer < 0) heading_to_steer = heading_to_steer
+360;
              if (heading_to_steer > 360) heading_to_steer = heading_to_steer
-360;
              lcd.begin(20,4);
              lcd.setCursor(0,0);
              lcd.print("Key = ");
              lcd.print(key);
              lcd.setCursor(0,1);
              lcd.print("HTS = ");
              lcd.print( heading_to_steer);

              break;


              case '5':
                mode = 2;
                str_mode = "Knob Steering";
                lcd.begin(20,4);
                lcd.setCursor(0,0);
                lcd.print("Key = ");
                lcd.print(key);
                break;

           case '6':
              str_mode = "Right 10";
              heading_to_steer = heading_to_steer +10;
              if (heading_to_steer < 0) heading_to_steer = heading_to_steer
+360;
              if (heading_to_steer > 360) heading_to_steer = heading_to_steer
-360;
              lcd.begin(20,4);
              lcd.setCursor(0,0);
              lcd.print("Key = ");
              lcd.print(key);
              lcd.setCursor(0,1);
              lcd.print("HTS = ");
              lcd.print( heading_to_steer);
```

```
             break;

           case '7':
            str_mode = "Left 1";
            heading_to_steer = heading_to_steer -1;
            if (heading_to_steer < 0) heading_to_steer = heading_to_steer
+360;
            if (heading_to_steer > 360) heading_to_steer = heading_to_steer
-360;
            lcd.begin(20,4);
            lcd.setCursor(0,0);
            lcd.print("Key = ");
            lcd.print(key);
            lcd.setCursor(0,1);
            lcd.print("HTS = ");
            lcd.print( heading_to_steer);
            break;

         case '8':
            str_mode = "Switch Screens";
            lcd.begin(20,4);
            lcd.setCursor(0,0);
            lcd.print("Key = ");
            lcd.print(key);
            break;

        case '9':
            str_mode = "Right 1";
            heading_to_steer = heading_to_steer +1;
            if (heading_to_steer < 0) heading_to_steer = heading_to_steer
+360;
            if (heading_to_steer > 360) heading_to_steer = heading_to_steer
-360;
            lcd.begin(20,4);
            lcd.setCursor(0,0);
            lcd.print("Key = ");
            lcd.print(key);
            lcd.setCursor(0,1);
            lcd.print("HTS = ");
            lcd.print( heading_to_steer);
            break;
       } // end key pressed
    break;

    case RELEASED:
    Serial.print("Key "); Serial.print(key); Serial.println(" Released");
      switch (key){
        case '*':
          str_mode = "COMPASS";
          lcd.begin(20,4);
          lcd.setCursor(0,0);
          lcd.print("Key * Released");
          lcd.setCursor(0,1);
          lcd.print("RUDDER OFF");
          break;

         case '#':
          str_mode = "Rudder Stop";
          lcd.begin(20,4);
          lcd.setCursor(0,0);
          lcd.print("Key # Released");
```

```
        lcd.setCursor(0,1);
        lcd.print("RUDDER OFF");
        break;
    } // end key released
 break;

 case HOLD:
 Serial.print("Key "); Serial.print(key); Serial.println(" Held");
   switch (key){
     case '*':
       str_mode = "Dodge Left";
       lcd.begin(20,4);
       lcd.setCursor(0,0);
       lcd.print("Key = ");
       lcd.print(key);
       lcd.setCursor(0,1);
       lcd.print("LEFT RUDDER ON");
       break;

     case '#':
       str_mode = "Dodge Right";
       lcd.begin(20,4);
       lcd.setCursor(0,0);
       lcd.print("Key = ");
       lcd.print(key);
       lcd.setCursor(0,1);
       lcd.print("RIGHT RUDDER ON");
       break;

   }  // end key HOLD
   break;
 }  // end get keypad state (Pressed, Released or Hold)
}  // end keypad event
```

Bilaga 3: Autopilotens startsida i programmet

Alla programdelar till autopiloten, inklusive denna, sparas i en gemensam mapp som har samma namn som autopilotens första sida, t.ex. AP_HA. Kopiera och klistra in i Arduino IDE. Spara som AP_HA.

```
/*************************************************************************
***
Användarvillkor:

Användaren av denna autopilot är själv ansvarig för att den används på ett
säkert sätt.

Lämna aldrig autopiloten oövervakad och se alltid till att alternativ
styrning till fartyget finns tillgänglig.

Utvecklad och färdigställd av André Haglund efter Henrik Florins
ursprungliga idé, baserat på
Jack Edwards mjukvara. Anpassad för användning i finländska (kalibrerad för
Vasa skärgård) och svenska farvatten.

Fri att användas och modifieras.


*************************************************************************
*********************************************/
/*
   30.9.2020. Projekt avslutat efter 10 års arbete. Kod version 172. Hög
stabilitet i allt förutom den trådlösa fjärrkontrollen. Rekommenderas att
användaren
   använder IMU10v3 och Pololu Simple motor kontroll. Nödstyrning via
relän. Alla gamla idéer finns kvar i koden. Fjärrstyrning direkt till
rodermotorn via relä rekommenderas också
   för enklast och billigast möjliga lösning. Sladdansluten fjärrstyrning
raderad från koden.

   20.04.2021. 500 rader gammal kod borttagen för att göra det lättare för
läsaren till detta slutarbete vid Högskolan på Åland.
*/
#define Arduino 0
#define Board Arduino//  0 = Arduino eller  1 = Teensy (borttaget, gammal
kod)

#include <Keypad.h>
#include <LiquidCrystal.h>
#include <Wire.h>

/******       Väljs av användaren       ********/

#define Compass 0 //  0 = Pololu, 1 = BNO055
#define IMU 103 // För att välja Pololu IMUs versioner och kalibrering.
//Tillåtna värden: 2 IMU9 V2; 93 (IMU9V3); 103 (IMU10V3; 51 (IMU9V5 #1); 52
(IMU9 V5 #2)
//Väljer kalibrering, återfinns på rad ca 500
#define GPS_Used 1 // 1 för GPS, 0 utan
#define Motor_Controller 3  // 1 = Pololu Qik dual controller, 2 = Pololu
Trex dual controller, 3 = Pololu Simple Controller
#define Clutch_Solenoid 0 // 1 för alternativ 1 och 2 ovan
```

```
#define RUDDER_MODE 0 // 0 = rodervinkelssensor installerad, 1 =
rodervinkelssensor saknas
#define RF24_Attached 0 // 0 = RF 24 radiomodul icke installerad , 1 =
installerad
#define Wind_Input 0 // 1 = NMEA vinddata. 0 = icke installerad
#define RUDDER_OFFSET 1 // 1 = rudder offset, 0 = icke tillgänglig
#define BEARINGRATE_OFFSET 1 // 1 = aktiverad, 0 = icke tillgänglig
// float PID_Ks[4] = {.75,.4,.01,0}; // om rudder_command = rudder_command
+ PID_output;
float PID_Ks[4] = {2, .4, 2, .0005};  // [ K(overall), K(heading error),
K(differential error), K(integral error)] // om rudder_command = PID_output
// Utan rodervinkelssensor kan man testa: 2, .4, 4, 0.  Eller {2, .4, 1,
.000}. Eller {1, .4, 2, .000}
#define PID_MODE 3 // Se PID-fliken
boolean Accept_Terms = 0; //  0 = ja. 1 = nej.
boolean just_started = 0; // för att få en stabilare start av gyrot.
//float Kxte[3] = {0, 0, 0}; // för XTE PID, för att ta bort XTE följning
//float Kxte[3] = {.2, 0, 0}; // {.2, 4, .0004} baseline; {.05, .5, .0005}
senast använd;  0 = proportional, 1 = differential, 2 = integral error, se
GPS_Steer() i PID
// .36 = 45 deg korrektion 30 meter XTE för att passa Garmin GPSMAP 740s se
PID tab, voidActual_Gps_Steering()
float K_course_avg = .999; //för att göra GPS-kursen jämnare
float Maximum_Rudder = 45; // rudder in degrees
// Modifiera motorspeed, crtl-F för att hitta motorspeedMIN - efter
användarens smak och tycke
float Tack_Angle = 100;  // För segelbåtar, kryssning. Knapp 4 och 6 i
program 3.
int Tack_rudder_MAX = 32;// Begränsar rodret så att det inte bromsar båten.
float Tack_rudder_speed = .5; // roderhastighet vid kryss. Multiplicerar
med förvald hastighet.
float Rudder_Offset = 0;
float bearingrate_Offset = 0;
float MagVar_default = 9;// För Vasa skärgård. Skall uppdateras då man
byter position. Pgm använder GPS information om tillgängligt, + = öst, - =
väst
boolean GPS_Auto_Steer = 1; // 0 = håller kurs efter att WP nåtts.
//  1 = styr mot följande fördefinierade WP. Var försiktig....
boolean Use_CTS = 1  ;// 0 = nej, 1 = ja;  om man vilja följa GPS:en kurs
(om den tillhandahåller någon) istället för att använda sig av autopilotens
egen uträkning av BOD och XTE
//float bearingrate_correction = 0.0; //om man vill ha avg stationary
bearing rate = 0 på skärm 2

#if Board == Arduino
#define LCD_Contrast 10 // 0 till 255, ersätter potentiometern för
kontrastkontrollen på LCD-skärmen, V0. FUNGERAR INTE. Leder till
störningar. Inte värt besväret.
#endif
#define SW2_Used 0 // 0 = inte i bruk. 1 = SW2 installerad, om man
tillexempel vill ha en fjärrkontroll och kunna välja mellan den och den
stationära autopiloten.
#define UseBarometer 1 // 1 = ja.
#define Wind_Steer_Direct 0 /* 0 = wind-error eller GPS_error som underlag
för heading_error -> kompasstyrning används.
                               1 = PID använder Wind error eller GPS course
error direkt utan kompassen, men kompassen måste oavsett fungera för
girhastigheten
*/
#define GPS_Steer_Direct 0 // samma som ovan
```

```
#define TFT_Used 0 //  1 för att använda TFT. Gör det inte, inte värt
besväret.
int print_level_max = 3;  //  0 till 4, för analys av serial monitor detail
// 0 = ingen, 1=PID Output, 2 omvandlad GPS data, 3 rå GPS input, 4 lägger
till checksum resultat
int print_time = 5; // intervall i sekunder, för analys av data
boolean Print_ETdata = 0; //visar GPS incoming data. Stäng av för loop tid
boolean Print_heading  = 0 ; // kurs, intervall justeras i A_P_loop
boolean Print_LCD_IMU9 = 0;  //visar Head, Pitch, Roll, Yaw på LCD, för att
se om det täcker över annan text
boolean Print_LCD_AP = 1; // visar huvud A/P LCD output data och menyer,
använd bara ett av alternativen i listan här åt gången....
boolean Print_Gyro = 0; //  visar LCD och Serial scaled X(roll), Y(pitch),
Z(Yaw) deg/sekund
boolean Print_PID = 0;
boolean Print_UTC = 0;
boolean print_Nav_Data = 0; // Print_1 Tab
boolean Print_Motor_Commands = 0;  // visar kommandon i PID fliken
boolean Print_Anticpate_Turn = 0;  // visar data från Actual_GPS_Steering
för att se om Anticipate turn fungerar
int print_level = print_level_max;
//  print modes for MinIMU9
/*För felsökning*/
//OUTPUTMODE=1 korrigerad data,
//OUTPUTMODE=0 okorrigerad data
#define OUTPUTMODE 1
#define PRINT_DATA 0 // 1 för att visa seriell data
#define PRINT_EULER 0  //Visar Eulers vinklar; Roll, Pitch och Yaw
//#define PRINT_DCM 0     //direction cosine matrix
#define PRINT_ANALOGS 0 //analog raw data

//*************  GPS Användarinställningar  ******************

#if GPS_Used ==1
#define GPS_source 1  //  1 = GPS på huvud AP, 2 = på separat Arduino,
används inte längre....med Easy Transfer, borttaget
#define Serial_GPS Serial3
int Input_Source = 1; // 1 = Garmin GPS60CSX(har $GPRMC & $GPAPB //
// 2= Nobeltec (saknar sista delen av $GPRMC och $GPAPB för GPS status.
//3 = samma som 1 tills vidare, problem med 198 c, saknar APB men har BOD
#define LatLon_decimals 0 // Använd print_NMEA för att få reda på hur många
decimaler. 0 = auto detect. Använd 0 med försiktighet....
float  Turn_distance = 35;  //  Svängradie i hastigheten som båten färdas.
I meter. Alltså ett helt varv, 360 grader.
boolean Anticipate_Turns = 1; // 1 = ja, 0 = nej.
//   boolean Use_CTS = 0;  //  1 för att använda CTS (course to steer) från
GPS strängen $GPAPB. Stäng av Anticipate Turns om GPS:en har denna funktion
boolean print_NMEA = 0;  //1 = ja, 0 = nej
boolean print_GPS_buffer = 0;  //1 = ja, 0 = nej
boolean print_RMC = 0;  //1 = ja, 0 = nej
boolean print_APB = 0; //1 = ja, 0 = nej
boolean print_RMB = 0; //1 = ja, 0 = nej
boolean print_BOD = 0; //1 = ja, 0 = nej
boolean print_WPL = 0;  //1 = ja, 0 = nej
boolean print_RTE = 0;  //1 = ja, 0 = nej
boolean print_timing = 0;
boolean print_checksum = 0;

//**********  GPS VARIABLER  ************
int oldtime = 0;
int newtime = 0;
```

```
int deltatime = 0;
int byteGPS = -1;
const int buffer_length = 84;
char gps_buffer[buffer_length] = "";
int bufpos = 0;
String data_APB[17];
String data_RMB[17];
String data_RMC[17];
String data_BOD[7];
String data_WPL[6];
String data_RTE[17];
int GPS_WPT_index = 0;
boolean Anticipated_Turn_Active = 0;
String Route[20];
double Waypoint_Lat [20];
double Waypoint_Lon [20];
double Waypoint_Range_From[20]; // avstånd mellan WP
long Number_of_sentences = 0;
int Current_sentence_number = 0;
int Number_of_waypoints;
int word_count; //  A_GPS
int word_count_temp;
int Word_count = 0; // RTE
boolean Route_Completeness = 0;
unsigned long RTE_timer;
boolean RTE_Active = 0;
unsigned  long temp = 0;
unsigned long temp2 = 0;  ;
String temp4 = "";
String temp5 = "";
char char1;
char char2;
char char3;
String string3;
unsigned long long2;
unsigned long long3;
int int1;
boolean GPS_available = false;
String GPScase = "";
String GPSheader = "";
boolean NMEA_sentence = false;
String GPRMC_fix_status = "";
boolean GPAPB_fix = false;
String GPAPB_fix_status = "";
boolean GPRMB_fix = false;
String GPRMB_fix_status = "";
boolean NewData = false;
unsigned  long Waypoint_Age;
String Lat_Lon;
double Lat_Lon_Deg;
String CTS_MorT;
float SOG;
String XTE_LR;
String XTE_unit;
String Nullstring;
String Waypoint_next;
double Lat_current;
double Lon_current;
double Lat_destination;
double Lon_destination;
String Origin_Waypoint;
```

```
String Waypoint_name;
double Lat_Waypoint;
double Lon_Waypoint;
double Current_Waypoint_Lat;
double Current_Waypoint_Lon;
long Waypoint = 0;
long Latitude = 0;
String BOD_MorT = "";
double Active_Bearing_to_destination;
String BTD_MorT = "";
double Range_Destination_by_LatLon;
double Active_Range_Destination;
float Velocity_towards_destination;
String UTC_string;
long Date;
double Range_and_Bearing[2];
boolean Detect_LatLon_decimals = 0;
float Time_decimal;
float Time_decimal_old;
float Time_decimal_delta;
const float degrees_to_meters = 111462; // meter per grad av latitud (lon
vid ekvatorn), 111462 (Pörtmossen, i meter, 365688 i fot) 6.5.2019
#endif  // endif GPS_USed == 1

// GPS parameters för vind och LCD
int PT_old = 0;
String GPS_status = "NO GPS";
float Avg_course;
float CTS_GPS2;
float GPS_course_to_steer;
float course_error;
float AVG_tracking_error;
float XTE_integral_error;
float XTE_course_correction;
unsigned long UTC_timer;
unsigned long UTC_timer_old;
boolean GPS_Available = 0;
boolean GPS_Was_Available = 0;
int MSG = 0; // för sladdansluten fjärrkontroll
int j_MAX = 0;
double float1;
double float2;
double float3;
char *brkb, *pEnd;
String string1;
String string2;
unsigned  long long1;
char char_buf[16];
float course;
long checksum_received = 0;
int checksum = 0;
boolean checksum_status = false;
String data_IN[17];
String Active_waypoint;
float XTE;
boolean GPRMC_fix = false;
double Bearing_to_destination = 0;
float course_to_steer;
double Bearing_origin_to_destination = 0;
unsigned long UTC;
float NEXT_TURN;
```

```
float Next_Turn[20];
double Range_Destination;
double Bearing_to_destination_by_LatLon = 0;
int WPT_index = 0;
double Waypoint_Bearing_From[20];
float MagVar; //Magnetisk Variation E +, W -


//******  KOMPASS  ***************
#if Compass == 0
#include <L3G.h>  // Bibliotek från Pololu
http://www.pololu.com/product/1268
#include <LSM303.h>  // Använd rätt version på biblioteket....
#endif

#if Compass == 1
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>
#include <EEPROM.h>
Adafruit_BNO055 bno = Adafruit_BNO055(55);
#define BNO055_SAMPLERATE_DELAY_MS (100)
int eeAddress = 0;
long bnoID;
bool foundCalib = false;
bool DataStored = false;
#endif
int bnoCAL_status = 0; //RF fjärrkontroll

#if UseBarometer
#include <LPS.h>
LPS ps;
float pressure;
float altitude;
float temperature;
#endif

/** RADIO RF24 Förklaringar ***/

#if RF24_Attached == 1
//#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

const uint64_t pipes[2] = { 0xF0F0F0F0E1LL, 0xF0F0F0F0D2LL };
#if Board == Arduino //  Fungerar inte
// RF info
int serial_putc( char c, FILE * )
{
  Serial.write( c );
  return c;
}
//  radio CE,CS utgångar
RF24 radio(9, 10);
#endif



int RFdata_set;
```

```
char KeyIn2;


struct RF_DATA
{

  char RFD_text[8];
  int16_t RFD_int1;
  int16_t RFD_int2;
  int16_t RFD_int3;
  int16_t RFD_int4;
  int8_t  RFD_int5;
  int16_t RFD_int6;
  int16_t RFD_int7;
  int16_t RFD_int8;
  int8_t  RFD_int9;
  int16_t RFD_int10; // BNO Cal Status
  int16_t RFD_int11;
  int16_t RFD_int12;
  int16_t RFD_int13;
  // #endif
  /*
    #if Board == Arduino
        char RFD_text[8];
        int RFD_int1;
        int RFD_int2;
        int RFD_int3;
        int RFD_int4;
        byte RFD_int5;
        int RFD_int6;
        int RFD_int7;
        int RFD_int8;
        byte RFD_int9;
        int RFD_int10;
        int RFD_int11;
        int RFD_int12;
        int RFD_int13;
    #endif
  */

};
RF_DATA RFdata;
#endif

#if Board == Arduino
// aktivering av LCD (RS,E,D4,D5,D6,D7)
LiquidCrystal lcd(39, 41, 43, 45, 47, 49);
long lcdtimer = 0;
#define LCD_Contrast_Pin 7 // LCD kontrast, använd med försiktighet, läs på
om vilken kondensator som behövs för att minska störningar
#endif


//  KEYPAD SETUP
const byte ROWS = 4;
const byte COLS = 3;
char keys[ROWS][COLS] = {
  {'1', '2', '3'},
  {'4', '5', '6'},
  {'7', '8', '9'},
  {'*', '0', '#'}
```

```
};

#if Board == Arduino
byte rowPins[ROWS] = {25, 35, 33, 29};
byte colPins[COLS] = {27, 23, 31};
#endif
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );
char key = 0;
boolean toggle = false;

//const int motorspeed = 32; // 0 till 127  // använd med pololu motor
kontroller


float Magnetic_Variation;
float heading_error = 0;
float differential_error = 0;
float integral_error = 0;
float deadband;
float rudder_position = 0;
float rudder_command = 0;
float rudder_error;
int motorspeed;
int rudder_MAX;
boolean TACK_ON = false;
// float rudder_total_time;
float bearingrate = 0;
float bearingrate_smoothed;
float bearingrate2;
unsigned long bearingrate2_time;
float heading_old;
float delta_heading;
long delta_compass_time;
float PID_output = 0;
// float GPS_PID = 0;
boolean GPS_Steering = false;
int Steering_Mode = 0;
String Mode = "OFF";
String Previous_Mode;
boolean Steering = false;
boolean sw1_turned_on = false;
boolean sw1 = false;
boolean sw2 = false;
boolean sw1Status = 1;
boolean DODGE_MODE = false;
int Screen = 0;
boolean rudder_on;
boolean rudder_was_off;
unsigned long rudder_time_old;

#if Motor_Controller == 1  //1 för Pololu Qik Dual controller
int Motor_0_fwd = 137;
int Motor_0_rev = 139;
int Motor_1_fwd = 141; //byt plats på dessa två för att ändra
styrbord/babord roder.
int Motor_1_rev = 143;
int motorspeedMIN = 75; // Lägsta värdet för att få rodret att röra på sig.
Orig 555, 5.5.2019 1600 30.7.2020 50
//  högre värden ger autopiloten bättre respons.
int  motorspeedMAX = 127;
#endif
```

```
#if Motor_Controller == 2  //2 för Pololu Trex Dual controller
int Motor_0_fwd = 202;
int Motor_0_rev = 201;
int Motor_1_fwd = 194; //byt plats på dessa två för att ändra
styrbord/babord roder.
int Motor_1_rev = 193;
int motorspeedMIN = 30; // Lägsta värdet för att få rodret att röra på sig.
Orig 555, 5.5.2019 1600 30.7.2020 50
//  högre värden ger autopiloten bättre respons.
int motorspeedMAX = 127;
#endif

#if Motor_Controller == 3 //3 för Pololu Simple controller
int Motor_1_fwd = 0x85; // byt plats på dessa två för att ändra
styrbord/babord roder.
int Motor_1_rev = 0x86;
int motorspeedMIN = 1600; // Lägsta värdet för att få rodret att röra på
sig. Orig 555, 5.5.2019 1600 30.7.2020 50
//  högre värden ger autopiloten bättre respons.
int motorspeedMAX = 3199;
#endif

#if Wind_Input == 1
#include <SoftwareSerial.h>
SoftwareSerial SoftSerial1 =  SoftwareSerial(11, 12); // RX, TX
int SoftSerial1_Bytes;
byte byteWind;
int count_b;
String Windheader;
const int wind_buffer_length = 32;
char Wind_buffer[wind_buffer_length];
String data_MWV[6];
#endif

float Wind_Dir;
float Wind_Speed;
float wind_to_steer;
float wind_error;
float Wind_Avg;
float Wind_MAX;
float Depth = 0;

/***********  COMPASS  SET UP  ************************************/
float heading;
float heading_to_steer = 0;
float MAG_Heading_Degrees;


#define ToRad(x) ((x)*0.01745329252)  // *pi/180
#define ToDeg(x) ((x)*57.2957795131)  // *180/pi
long timer = 0;
long timer_old;
long timer24 = 0;
unsigned int counter = 0;
unsigned int counter2 = 0;
unsigned long counter3 = 0;
float roll;
float pitch;
float yaw;
```

```
#if Compass == 0
int SENSOR_SIGN[9] = {1, -1, -1, -1, 1, 1, 1, -1, -1}; //x,y,z - gyro,
accelerometer, magnetometer
#define GRAVITY 256  //256 = 1G
#define Gyro_Gain_X .00875 //X axel Gyro gain .07 för FS 2000 DPS, .00875
för full skala, 245 grader per sekund
#define Gyro_Gain_Y .00875 //Y axel Gyro gain
#define Gyro_Gain_Z .00875 //Z axel Gyro gain
#define Gyro_Scaled_X(x) ((x)*ToRad(Gyro_Gain_X))
#define Gyro_Scaled_Y(x) ((x)*ToRad(Gyro_Gain_Y))
#define Gyro_Scaled_Z(x) ((x)*ToRad(Gyro_Gain_Z))



#if IMU == 2 //IMU9V2
#define M_X_MIN -663
#define M_Y_MIN -683
#define M_Z_MIN -611
#define M_X_MAX 453
#define M_Y_MAX 427
#define M_Z_MAX 460
#endif

#if IMU == 93 //IMU9V3
#define M_X_MIN -3525
#define M_Y_MIN -3473
#define M_Z_MIN -3398
#define M_X_MAX 3600
#define M_Y_MAX 3226
#define M_Z_MAX 2802
#endif

#if IMU == 103 // IMU-10 V3
#define M_X_MIN -3290
#define M_Y_MIN -3288
#define M_Z_MIN -2840
#define M_X_MAX 3611
#define M_Y_MAX 3553
#define M_Z_MAX 4127
#endif

#if IMU == 51 // IMU V5 #1
#define IMU_V5 //Pololu IMU V5-specifik rad
#define M_X_MIN -1955
#define M_Y_MIN -4857
#define M_Z_MIN 728
#define M_X_MAX 5459
#define M_Y_MAX 5518
#define M_Z_MAX 7758
#endif

#if IMU == 52 //IMU V5 #2
#define IMU_V5 //Pololu IMU V5-specifik rad
#define M_X_MIN -2582
#define M_Y_MIN -6418
#define M_Z_MIN 194
#define M_X_MAX 4837
#define M_Y_MAX 1381
#define M_Z_MAX 7236
#endif
```

```
#define Kp_ROLLPITCH 0.02
#define Ki_ROLLPITCH 0.00002
#define Kp_YAW 1.2
#define Ki_YAW 0.00002

float G_Dt = 0.02;
int AN[6];
int AN_OFFSET[6] = {0, 0, 0, 0, 0, 0};
int gyro_x;
int gyro_y;
int gyro_z;
int accel_x;
int accel_y;
int accel_z;
int magnetom_x;
int magnetom_y;
int magnetom_z;
float c_magnetom_x;
float c_magnetom_y;
float c_magnetom_z;
float MAG_Heading;

float Accel_Vector[3] = {0, 0, 0};
float Gyro_Vector[3] = {0, 0, 0};
float Omega_Vector[3] = {0, 0, 0};
float Omega_P[3] = {0, 0, 0};
float Omega_I[3] = {0, 0, 0};
float Omega[3] = {0, 0, 0};

// Euler angles

float errorRollPitch[3] = {0, 0, 0};
float errorYaw[3] = {0, 0, 0};

byte gyro_sat = 0;

float DCM_Matrix[3][3] = {
  {
    1, 0, 0
  }
  , {
    0, 1, 0
  }
  , {
    0, 0, 1
  }
};
float Update_Matrix[3][3] = {{0, 1, 2}, {3, 4, 5}, {6, 7, 8}}; //För gyro
float Temporary_Matrix[3][3] = {
  {
    0, 0, 0
  }
  , {
    0, 0, 0
  }
  , {
    0, 0, 0
  }
};
// slut på IMU9 kod
#endif // end if kompass == 0
```

```
/************************************/


//*  SETUP    SETUP   SETUP   *****

void setup() {
#if Board == Arduino
  pinMode(48, INPUT); //On knapp 1
  pinMode(46, INPUT); //On knapp 2
#define Serial_MotorControl Serial2
#endif
  delay(1000); // så kortet hinner komma igång
  Serial.begin(57600); // Serial conection to Serial Monitor

#if GPS_Used == 1
  Serial_GPS.begin(4800); // input data GPS
#endif
  Serial.print("Setup stated and Serial Opened");

  Serial_MotorControl.begin(19200);
  delay(1000);

#if Wind_Input == 1
#if Board == Arduino
  pinMode(11, INPUT);
  SoftSerial1.begin(4800);
#endif

#endif

#if Board == Arduino
  pinMode (LCD_Contrast_Pin, OUTPUT);
  analogWrite(LCD_Contrast_Pin, LCD_Contrast); // PWM kontroll av LCD
kontrast
  lcd.begin(20, 4);
  lcd.setCursor(0, 0);
#endif


  keypad.addEventListener(keypadEvent);

#if RF24_Attached == 1
  // Radio Setup

#if Board  == Arduino
  fdevopen( &serial_putc, 0 );
  attachInterrupt(digitalPinToInterrupt(2), Recv_Data, CHANGE);
#endif

  radio.begin();
  radio.setChannel(108); // förvald kanal är 108
  Serial.print("Radio Channel "); Serial.println(radio.getChannel());

  radio.openWritingPipe(pipes[0]);
  radio.openReadingPipe(1, pipes[1]);
  int dataSize = sizeof(RF_DATA);
  Serial.print("Size of RF_DATA1: "); Serial.println(dataSize);
  if ( dataSize > 32 )
    Serial.println("***  RF_DATA1 struct is too large ***");
  radio.maskIRQ(1, 1, 0); // http://tmrh20.github.io/RF24/
```

```
  radio.setPALevel(RF24_PA_MAX);
  radio.setDataRate(RF24_250KBPS); // långsammare data för bättre räckvidd
och stabilitet
  radio.setChannel(108);  // högre ska ge mindre störningar
  radio.printDetails();
  radio.startListening();
  //  End Radio Setup
#endif

#if Compass == 0
  //SETUP för MinIMU9
  lcd.print("M/Y Regina - SE-9283");
  I2C_Init();
  //Serial.println("Pololu MinIMU-9");
  // digitalWrite(STATUS_LED,LOW);
  delay(1500);
  Accel_Init();
  Compass_Init();
  Gyro_Init();
  delay(20);
  for (int i = 0; i < 32; i++)
  {
    Read_Gyro();
    Read_Accel();
    for (int y = 0; y < 6; y++)
      AN_OFFSET[y] += AN[y];
    delay(20);
  }
  for (int y = 0; y < 6; y++)
    AN_OFFSET[y] = AN_OFFSET[y] / 32;
  AN_OFFSET[5] -= GRAVITY * SENSOR_SIGN[5];
  //Serial.println("Offset:");
  for (int y = 0; y < 6; y++)
    Serial.println(AN_OFFSET[y]);

  delay(2000);
  // digitalWrite(STATUS_LED,HIGH);

  timer = millis();
  delay(20);
  counter = 0;

#endif

  if (Motor_Controller == 1) Serial_MotorControl.write(170);
  if (Motor_Controller == 3)
  {
    Serial_MotorControl.write(170);
    Serial_MotorControl.write(131);
  }
#if Compass == 1
  if (!bno.begin())
  {
    /* There was a problem detecting the BNO055 ... check your connections
*/
    //Serial.print("Ooops, no BNO055 detected ... Check your wiring or I2C
ADDR!");
    lcd.setCursor(0, 0);
    lcd.print(" No BNO055");
    while (1);
  }
```

```
  //  Hämta och återställ kalibrering
  BNO_RestoreCal();
#endif  // Compass == 1

#if UseBarometer
  Init_Barometer();
#endif
}

/*********************************************/

void loop()
{
#if Compass == 0
  if (just_started)
  { setup();  //kör uppstart flera gånger för att undvika
kalibreringsproblem, tveksam om det fungerar.
    just_started = 0;
  }
#endif
#if Board == Arduino
  sw1 = digitalRead(48);
  sw2 = digitalRead(46);
  if (SW2_Used == 0) sw2 = true; // true = bara en knapp kontrollerar
autopiloten
  if (!sw1 || !sw2)  // om man vill ha två knappar för att aktivera
autopiloten, båda måste vara ON
  {
    Rudder_Stop();
#if Clutch_Solenoid == 1 // om dual motor contrpller används, för manuell
hydraulstyrning
    Open_Solenoid();   // öppna solenoid för manuell hydraulstyrning
#endif
    Steering = false;
    Steering_Mode = 0;
    Mode = "OFF";

  }
  //Serial.print(" Use_CTS = "); Serial.println(Use_CTS);
#endif // end if Board == Arduino
  KEYPAD();
  if (Accept_Terms) Terms_and_Conditions();


  A_P_Loop(); // Autopilot Loop
}

/*********************************************/
void Terms_and_Conditions()
{ lcd.setCursor(0, 0);
  lcd.print("Jag accepterar anvan  ");
  lcd.setCursor(0, 1);
  lcd.print("dar villkoren      ");
  lcd.setCursor(0, 2);
  lcd.print("Tryck 0             ");
  lcd.setCursor(0, 4);
  lcd.print("for OK           ");
  delay (100);
}
```

Bilaga 4: Programmet som styr hur ofta autopiloten uppdateras

Kopiera och klistra in i Arduino IDE. Spara som 2_AP_Loop.

```
 /***********************
Integration av gyro och GPS med gyro och PID 50 gånger per sekund
 ************************/

   void A_P_Loop()
  {
    static int DT_test;

   static int Icount;
  #if GPS_Used == 1 && GPS_source == 1
    GET_sentence();
  #endif
  #if Wind_Input == 1
   if (Steering_Mode != 2 || Steering_Mode != 22) //Förhindrar krascher (?)
   {
    get_Wind();
   }
  #endif
    if((millis()-timer)>=20)  //Uppdateras med 50Hz
  {
    counter++;
    counter2++;
    counter3++;
    timer_old = timer;
    timer=millis();
  #if Compass == 0
    if (timer>timer_old)
      G_Dt = (timer-timer_old)/1000.0;
    else
      G_Dt = 0;


    Read_Gyro();
    Read_Accel();
    Matrix_update();
    Normalize();
    Drift_correction();
    Euler_angles();
    Bearing_Rate();

 #endif


    if (counter > 1)  // om man vill köra uppdateringen av autopiloten med
10Hz
     {


      counter=0;
   #if Compass == 0
      Read_Compass();
      Compass_Heading();
      AP_Compass_Correction();
```

```
       //Serial.print(heading,1); Serial.print("   ");
Serial.println(bearingrate); // Minns inte vad problemet var här
   #endif

  #if Compass == 1
    BNO055();
    AP_Compass_Correction();  // Beräkning av sann kurs
    if(counter2 >40)BNO055_Get_Cal(); //Behövs inte i 50Hz
  #endif


 #if GPS_Used == 1
  #if GPS_source == 1  //  Inte längre aktuellt, använd 1.
  #endif

#if CTS == 0
  Waypoint_Current();
  if(NewData)
  {
    for (int i = 0; i< Number_of_waypoints; i++)
     {
       if( Waypoint_next == Route[i])
         { GPS_WPT_index = i;
                              // Serial.print("GPS waypoint index ");
Serial.println(GPS_WPT_index);
         break;
         }
     }
    if (millis() - RTE_timer < 60000) RTE_Active = true;
    if(Anticipate_Turns && RTE_Active) ANTICIPATE_TURN();

    if (Anticipated_Turn_Active == false)
      {
        WPT_index = GPS_WPT_index;
      }
      Active_waypoint = Route[WPT_index];
    Get_Cross_Track_Error();
    //if(Use_CTS == 0) course_to_steer = -1;
  }  // end if new data
#endif
 if (Use_CTS == 1)Active_waypoint = Waypoint_next;
 /*****   End GPS BLOCK A    ******/


 #endif // end if GPS_Used

 #if RF24_Attached == 1
     sendData1();
 #endif

    Steer_PID();

   }

 #if Compass == 0
   if (PRINT_DATA == 1) {
   printdata();
   }

   if(Print_Gyro)
    {
```

```
    Serial.print("Gyros ");
    Serial.print(ToDeg( Gyro_Vector[0])); Serial.print("' ");
    Serial.print( ToDeg(Gyro_Vector[1])); Serial.print("' ");
    Serial.println( ToDeg(Gyro_Vector[2]));
      lcd.setCursor(0,0);
      lcd.print ("GYROS");
      lcd.setCursor(0,1);
      lcd.print(ToDeg(Gyro_Vector[0]));
       lcd.setCursor(0,2);
      lcd.print(ToDeg(Gyro_Vector[1]));
       lcd.setCursor(0,3);
      lcd.print(ToDeg(Gyro_Vector[2]));
    }
 #endif

  }


  if (counter2 >47)
  {

   counter2 = 0;

   if(Print_LCD_AP) LCD(); //GPS & komapss data på LCD:n, kan stängas av om
man t.ex. vill visa annan info.
  #if Compass == 0
   if(Print_LCD_IMU9) LCDprint();
  #endif
  #if GPS_Used
  if (print_Nav_Data) NAV_DATA_PRINT();
  #endif
  // DT_test = millis() - DT_test;
  // Serial.println(DT_test);
  // DT_test = millis();
  //Serial.println(heading);
  #if UseBarometer
  if (Screen == 4)
    Read_Barometer();
  #endif
  }

  }
```

Bilaga 5: GPS. Del 1 av 10

Kopiera och klistra in i Arduino IDE. Spara som 3_GPS.

```
#if GPS_Used == 1
/*  GPS TAB INCLUDES
    void GET_sentence() it reads the NMEA sentence.
    void Checksum_calc() computes a check sum on NMEA sentence to verify
data integrity.
    void Get_GPRMC() parses the NEMA sentence and looks for the GPRMC
sentence.
    void Get_GPAPB() parses the NEMA sentence and looks for the GPRMC
sentence.
    Get_sentence is called from Get_GPRMC and Get_GPAPB so program can
alternate between the two GPS sentences.
    GPS Reading based on code by Igor Gonzalez Martin. 05-04-2007
igor.gonzalez.martin@gmail.com
    English translation by djmatic 19-05-2007

    Modifierad för att passa autopiloten och nordens farvatten av A.
Haglund
 */

  void GET_sentence() {
if (bufpos > buffer_length -1) Reset_buffer();
if (Serial_GPS.available() > 0){
    byteGPS = Serial_GPS.read();
    if(byteGPS !=13){
      gps_buffer[bufpos] = byteGPS;
      if(byteGPS == ',') word_count_temp = word_count_temp+1;
      if (print_NMEA)
Serial.write(byteGPS);
      bufpos++;
    }
    else { Process_GPS_Data();
    }
  }
 }

void Process_GPS_Data(){
     checksum_status= false;
     NMEA_sentence=false;
     gps_buffer[bufpos] = '\0';
     GPSheader = "";
     for (int i=4;i<7;i++){
       GPSheader = GPSheader + gps_buffer[i];
     }


       for (int i=bufpos+1;i<buffer_length;i++){
         gps_buffer[i]=' ';
          bufpos = 0;
         }
      word_count = word_count_temp;
      word_count_temp = 1;
         if(print_GPS_buffer) Serial.print(gps_buffer);
       //Serial.println(); Serial.print(GPSheader);

      if (GPSheader == "RMC"){
```

```
        Get_GPRMC();
        return;
     }

     if (GPSheader == "APB") {
        Get_GPAPB();
        return;
     }

     if (GPSheader == "RMB"){
        Get_GPRMB();
        return;
     }


     if(Use_CTS == 0)
   {
      if (GPSheader == "BOD"){
          Get_GPBOD();
          return;
      }

      if (GPSheader == "WPL"){
          Get_GPWPL();
          return;
      }

        if (GPSheader == "RTE"){
          Get_GPRTE();
          return;
      }
   }  // end if(Use_CTS ==0 )
       NewData=false;
       Reset_buffer();
  }

void Reset_buffer(){
  for (int i=0;i<buffer_length;i++){
     gps_buffer[i]=' ';
     bufpos = 0;
  }
}

    void Checksum_calc(){

       /*********************************************
        From garmin.com search support for "how is checksum calculated in
NMEA 0183
        The checksum is the 8-bit exclusive OR (no start or stop bits) of
all characters in the sentence, including the "," delimiters,
        between -- but not including -- the "$" and "*" delimiters. The
hexadecimal value of the most significant and least significant
        4 bits of the result are converted to two ASCII characters (0-9, A-
F) for transmission. The most significant character is transmitted first.
        Therefore in the routine below the counter starts at 1 to skip "$"
in  checksum routine and ends at index of "*".
      *********************************************/
            int index=0;
                           //  Serial.println("");
```

```
                              //  Serial.print("DATA TO BE CHECK
SUMMED  ");
                                  checksum=0;
            for(int x=1; x<100; x++){  // you have to skip the $ sign and
it works if x starts at 1
                if (gps_buffer[x] == '$'){
                                     index=x;}
                                   break;}
                                  //   Serial.print("index = ");
                                   //  Serial.println(index);
                                 for(int x=index+1; x<100; x++){
                                     if(gps_buffer[x]=='*'){
          checksum_received = strtol(&gps_buffer[x + 1], NULL, 16);

                                       break;
          }else{
          checksum ^= gps_buffer[x];

          }}

          checksum_status = false;
          if(checksum_received == checksum){
                                     checksum_status=true;
                                       }


        }    // End of Checksum subroutine
#endif // end if GPS_Used == 1
```

Bilaga 6: GPS. Del 2 av 10

Kopiera och klistra in i Arduino IDE. Spara som 4_GPS.

```
/*************************************************************************
GPS data is processed on separate Arduino UNO and GPS2 receives the data.
This saves processing time.  The MinIMU9 could not process data fast enough
with
the time out it took the GPS to get data. With separate processing data is
transfered
using Easy Transfer by Bill Porter in one millisecond.

*************************************************************************/
#if GPS_Used == 1

//borttagen överföringsdata, används inte längre

/************************ IS GPS AVAILABLE ***************/

   void Is_GPS_Available()
    //sets waypoint to NO GPS if no update for more than x seconds
    {
      int GPS_max_time_lost = 6;  // maximum time (seconds) gps UTC is not
updated until GPS is NOT avilable
        Date_Time();  // gets UTC converted to decimal
        Time_decimal_delta = Time_decimal - Time_decimal_old;
        if(GPS_Available) GPS_Was_Available = true; // used to capture Dead
reckon heading only once then set false in PID
        if(Time_decimal > Time_decimal_old)
          {
            Time_decimal_old = Time_decimal;
            UTC_timer_old = millis();
            GPS_Available = true;
          }

        if( Time_decimal_delta >= 0)
          {
          UTC_timer = millis();
          }

        if (UTC_timer - UTC_timer_old >1000 * GPS_max_time_lost)
          {
          GPS_Available = false;
          }
    } // end Is GPS Available

    /************************ DATE TIME CALCULATOR
***************************/

    void Date_Time()
    {
      char char1[7];
      String year;
      String month;
      String day;
      String hour;
      String minute;
      String second;
      float fsecond, fminute, fhour, fday;
```

```
      /*
      string1 = String(Date);
      day = string1.substring(0,2);
      month = string1.substring(2,4);
      year = string1.substring(4,6);
      */

      string1 = String(UTC);
      string1 = "0000" + string1;
      string1 = string1.substring(string1.length() -6);
      hour = string1.substring(string1.length() -6, string1.length() -4);
      minute = string1.substring(string1.length() -4, string1.length() -2);
      second = string1.substring(string1.length() -2, string1.length());

      strtoflo(second);
      fsecond = float1;
     // Serial.print("sec float: "); Serial.println(fsecond);

       strtoflo(minute);
      fminute = float1;
     // Serial.print("min float: "); Serial.println(fminute);

      strtoflo(hour);
      fhour = float1;
     // Serial.print("hour float: "); Serial.println(fhour);

   //    strtoflo(day);
    //  fday = float1;
     //Serial.print("Day float: "); Serial.println(fday);

     Time_decimal = fhour*3600 + fminute*60 + fsecond;
    // Serial.println(Time_decimal,0);
    /*
      Serial.print("year: "); Serial.println(year);
      Serial.print("month: "); Serial.println(month);
      Serial.print("day: "); Serial.println(day);
      Serial.print("hour: "); Serial.println(hour);
      Serial.print("minute: "); Serial.println(minute);
      Serial.print("second: "); Serial.println(second);
      Serial.print("Decimal time: "); Serial.println(Time_decimal,4);
      Serial.println();
    */
    }  // End void data time

    void strtoflo(String string1)
    {
      char char_buf[16];
      string1.toCharArray(char_buf,16);
      long1 = strtol(char_buf,&pEnd,10);
      float1 = float(long1);
    } // end String_to_float
#endif // Endif GPS_Used ==1
```

Bilaga 7: GPS. Del 3 av 10, NMEA

Kopiera och klistra in i Arduino IDE. Spara som GPAPB.

```
#if GPS_Used == 1
 void Get_GPAPB()
   {
     //Serial.println("GPAPB");
    j_MAX = 16;  // Antal ord i NMEA-strängen
    Parse_Sentence();
    for(int j = 0; j<j_MAX; j++)
     {
      data_APB[j] = data_IN[j];
      if(print_APB)Serial.println(data_IN[j]);
     }
   //    *************  PROCESS DATA  **********************


        // data_APB[3], Cross Track Error
         // string1 = data_APB[3];
         // NMEA_TO_FLOAT(2); //string1 är input, 2 antal decimaler, ger
float3
        //  XTE = float3;

        // data_APB[4], XTE L/R
         // if( data_APB[4] == "L"){ XTE=-XTE;}  //R styr styrbord, L
babord och således negativt ->> PID/GPS
        // XTE_LR = data_APB[4];

        // data_APB[5], XTE Units
        // XTE_unit = data_APB[5];


        // data_APB[6], Arrival Alarm Circle
        // data_APB[7], Arrival Alarm Perpendicular

        // data_APB[8], Bearing Origin to Destination
          string1 = data_APB[8];
          NMEA_TO_FLOAT(1);
          Bearing_origin_to_destination = float3;


        // data_APB[9], Mag or True
         BOD_MorT = data_APB[9];

        // data_APB[10], Destination Waypoint ID
         Waypoint_next = data_APB[10];
         if(Waypoint_next == "") { Active_waypoint = "NO WPT"; }
       //  Waypoint_next =Waypoint_next.substring(10);


         string1 = data_APB[13];
         NMEA_TO_FLOAT(1);
         //course_to_steer = float3;
         GPS_course_to_steer = float3;
```

```
         if (Word_count == 15)  data_APB[15] = "D"; // NOBELTEC sänder inte
data_APB[15]

         if(data_APB[15]=="A" || data_APB[15]=="D" && checksum_status){
              GPAPB_fix = true;
              Waypoint_Age = millis();
          }
         else{
              GPAPB_fix= false;}
        // data_APB[16], Fix Validity
         if(GPAPB_fix){
              GPAPB_fix_status = "GPAPB OK";}
         else{
              GPAPB_fix_status= "NO GPAPB";}

         if(GPAPB_fix) NewData = true;

      if(print_APB) {PRINT_APB();}

   }  //end of void GPAPB() case


  /*************************  PRINT APB  ******************************/

   void PRINT_APB()
{
     Serial.println();
    Serial.println("---------------");

            Serial.print("Header: ");
                 Serial.println(data_APB[0]);

            Serial.print("Loran Data: ");
                 Serial.println(data_APB[1]);

            Serial.print("Loran Data: ");
                 Serial.println(data_APB[2]);

            Serial.print("Cross Track Error: ");
                 Serial.println(XTE);

             Serial.print("Error L or R: ");
                 Serial.println(XTE_LR);

             Serial.print("Cross track error units: ");
                 Serial.println(XTE_unit);

             Serial.print("Arival Alarm Circle: ");
                 Serial.println(data_APB[6]);

             Serial.print("Arrival Alarm Perpendicular: ");
                 Serial.println(data_APB[7]);

             Serial.print("Bearing Origin to Destination: ");
                 Serial.println(Bearing_origin_to_destination);

             Serial.print("Mag or True: ");
                 Serial.println(BOD_MorT);

             Serial.print("Destination Waypoint ID: ");
                 Serial.println(Waypoint_next);
```

```
            Serial.print("Bearing Present Position to Destination: ");
                Serial.println(Bearing_to_destination);

            Serial.print("Mag or True: ");
                Serial.println(BTD_MorT);

            Serial.print("Course to Steer: ");
                Serial.println(course_to_steer,1);

          Serial.print("Mag or True: ");
                Serial.println(CTS_MorT);

          Serial.print("Type of Fix: ");
                Serial.println(data_APB[15]);

            Serial.print("GPAPB Fix Validity; ");
                Serial.println(GPAPB_fix_status);

            Serial.println();

   }

#endif
```

Bilaga 8: GPS. Del 4 av 10, NMEA

Kopiera och klistra in i Arduino IDE. Spara som GPBOD.

```
#if GPS_Used ==1
 void Get_GPBOD()
   {
     //Serial.println("GPBOD");
    j_MAX = 7;  // antal ord i NMEA-strängen
    Parse_Sentence();
    for(int j = 0; j<j_MAX; j++)
     {
      data_BOD[j] = data_IN[j];
      if(print_BOD)Serial.println(data_IN[j]);
     }
   //    ************  PROCESS DATA  *********************

         // data_BOD[0], kurs, beräknad i Get_Sentence()
         // data_BOD[1], Sann kurs från start till WP
          string1 = data_BOD[1];
           NMEA_TO_FLOAT(1); //string1 är input, 2 antal decimaler, ger
float3
          Bearing_origin_to_destination = float3;

         // data_BOD[2], M or T True for BOD[1}
          BOD_MorT = data_BOD[2];

         // data_BOD[5], Destination Waypoint ID
          Waypoint_next = data_BOD[5];
          if(Waypoint_next == "") {Active_waypoint = "NO WPT"; }
          Waypoint_Age = millis();
       // data_BOD[6], Origin Waypoint ID
          Origin_Waypoint = data_BOD[6];
          if(Origin_Waypoint == "") { Origin_Waypoint = "NONE"; }
          NewData = true;
       if(print_BOD) {PRINT_BOD();}

   }  //end of void GPBOD() case


  /*************************  PRINT BOD  ******************************/

   void PRINT_BOD()
 {
    Serial.println();
    Serial.println("---------------");

            Serial.print("Header: ");
                 Serial.println(data_BOD[0]);

            Serial.print("True Bearing Origin to Destination: ");
                 Serial.println(Bearing_origin_to_destination);

            Serial.print("Mag or True: ");
                 Serial.println(data_BOD[2]);

            Serial.print("Destination Waypoint: ");
                 Serial.println(Waypoint_next);
                // Serial.println(Active_waypoint);
```

```
            Serial.print("Origin Waypoint: ");
                Serial.println(Origin_Waypoint);

    }

#endif
```

Bilaga 9: GPS. Del 5 av 10, NMEA

Kopiera och klistra in i Arduino IDE. Spara som GPRMB.

```
#if GPS_Used ==1
   void Get_GPRMB()
   {
     //Serial.println("GPRMB");
    j_MAX = 15;  // antal ord i NMEA-strängen
    Parse_Sentence();
    for(int j = 0; j<j_MAX; j++)
     {
      data_RMB[j] = data_IN[j];
      if(print_RMB)Serial.println(data_IN[j]);
     }

    // data_RMB[4] Waypoint Origin
      Origin_Waypoint = data_RMB[4];
      if(Origin_Waypoint == "") { Origin_Waypoint = "NO Origin"; }

    // data_RMB[5] Destination Waypoint
      Waypoint_next = data_RMB[5];
         if(Waypoint_next == "") { Active_waypoint = "NO WPT"; }
     Waypoint_Age = millis();
   if(Anticipate_Turns == 0 && Number_of_waypoints ==
0)
   {
    // data_RMB[6] Destination Latitude
     To_Degrees(data_RMB[6]);
     Lat_destination = Lat_Lon_Deg;
     if(data_RMB[7] == "S") Lat_Waypoint = - Lat_Waypoint;
     //data_RMB[7]; // Lat N/S

    // data_RMB[8] Destination Logitude
     To_Degrees(data_RMB[8]);
     Lon_destination = Lat_Lon_Deg;
     if(data_RMB[9] == "W")Lon_destination = -Lon_destination;
   }
    // data_RMB[10] Destination Range
      string1 = data_RMB[10];
      NMEA_TO_FLOAT(3);
      Range_Destination = float3;

   //  data_RMB[11] Destination Bearing
      string1 = data_RMB[11];
      NMEA_TO_FLOAT(1);
      Bearing_to_destination = float3;



/*
     Serial.println();
     Serial.print("Lat Destination = ");
     Serial.println(Lat_destination,4);
     Serial.print("Lon Destination = ");
     Serial.println(Lon_destination,4);
*/
      if(print_RMB) {PRINT_RMB();}
```

```
   }  //end of void GPRMB() case


/********************* PRINT RMB
************************************************/

   void PRINT_RMB()
     {
                 Serial.println();
                 Serial.println("---------------");

                 Serial.print("Header: ");
                 Serial.println(data_RMB[0]);

                 Serial.print("Data Status: ");
                 Serial.println(data_RMB[1]);

                 Serial.print("Cross Track Error: ");
                // Serial.println(XTE);
                Serial.println(data_RMB[2]);

                 Serial.print("Error L or R: ");
                // Serial.println(XTE_LR);
                Serial.println(data_RMB[3]);

                 Serial.print("Origin Waypoint: ");
                // Serial.println(Origin_Waypoint);
                Serial.println(Origin_Waypoint);

                 Serial.print("Destination Waypoint: ");
                // Serial.println(Waypoint_next);
                Serial.println(Waypoint_next);

                 Serial.print("Destination Latitude: ");
                 Serial.println(Lat_destination,4);

                 Serial.print("Latitude N/S: ");
                 Serial.println(data_RMB[7]);

                 Serial.print("Destination Logitude: ");
                 Serial.println(Lon_destination,4);

                 Serial.print("Longitude E/W: ");
                 Serial.println(data_RMB[9]);

                 Serial.print("Range to Destination: ");
                 Serial.println(Range_Destination,3);

                 Serial.print("True Bearing to Destination: ");
                // Serial.println(Bearing_to_destination);
                 Serial.println(Bearing_to_destination);

                 Serial.print("Velocity towards Destination: ");
                 Serial.println(Velocity_towards_destination);

                 Serial.print("Arrival Alarm, A = Arrived, V= Not Arrived:
");
                 Serial.println(data_RMB[13]);

                 Serial.print("Fix Status A, D or V: ");
                 Serial.println(data_RMB[14]);
```

```
            Serial.println("---------------");
        }  // end void PRINT_RMB
#endif
```

Bilaga 10: GPS. Del 6 av 10, NMEA

Kopiera och klistra in i Arduino IDE. Spara som GPRMC.

```
#if GPS_Used ==1
 void Get_GPRMC()
   {
   int N_magvar; // antal decimaler för magvar. Garmin har 1, nobeltec 2
      //Serial.println("GPRMC");
    j_MAX = 13;  // antal ord i NMEA-strängen
    Parse_Sentence();
    //Serial.println("done Parsing");
    for(int j = 0; j<j_MAX; j++)
     {
      data_RMC[j] = data_IN[j];
      if(print_RMC)Serial.println(data_IN[j]);
     }

 /******************** CONVERT DATA  *************************/
     // data_RMC[0] = header no processing

     UTC_string = data_RMC[1];
    // Date_Time();
   // long1 = strtol(char_buf,&pEnd,10);
    // temp = strtol(char_buf,&pEnd,10);
     UTC = UTC_string.toInt();

    // Serial.println(UTC);

     // data_RMC[2] = UTC Status, no processing

     To_Degrees(data_RMC[3]);
     Lat_current = Lat_Lon_Deg;
     if(data_RMC[4] == "S") Lat_current = - Lat_current;
    // Serial.println(Lat_current);
     To_Degrees(data_RMC[5]);
     Lon_current = Lat_Lon_Deg;
     if(data_RMC[6] == "W")Lon_current = -Lon_current;
    //  = Lon E/W, no processing

    // data_RMC[7] = hastighet över grund

    string1 = data_RMC[7];
    NMEA_TO_FLOAT(1);
    SOG = float3;

    // data_RMC[8] = kurs. prövat med 234.6
    string1 = data_RMC[8];
    NMEA_TO_FLOAT(1);
    course = float3;


    //   data_RMC[9]Date
   /*  data_RMC[9].toCharArray(char_buf,16);
    long1 = strtol(char_buf,&pEnd,10);
    // temp = strtol(char_buf,&pEnd,10);
     Date = long1;
   */
```

```
    //  data_RMC[10] and data_RMC[11] Magnetic Variation E/W,
    string1 = data_RMC[10];
    N_magvar = data_RMC[10].length() - data_RMC[10].indexOf('.') -1; //
antal decimaler
    NMEA_TO_FLOAT(N_magvar); // 2 för Nobeltec, 1 för GPSMAP 60CSX
    MagVar = float3;
    if(data_RMC[11] == "W") MagVar = -MagVar;



    // data_RMC[12] Fix Status
     if (Word_count == 12)  data_RMC[12] = "D"; // NOBELTEC sänder inte
data_APB[15] or RMC[12]
           if(data_RMC[12]=="A" || data_RMC[12]=="D" &&
checksum_status){GPRMC_fix = true;}
           else{GPRMC_fix= false;}
           if(GPRMC_fix){GPRMC_fix_status = "VALID FIX";}
           else{ GPRMC_fix_status= "BAD FIX";}

      if(GPRMC_fix) NewData = true;

      if(print_RMC) {PRINT_RMC();}

   }  //end of void GPRMC() case


/********************* PRINT RMC
**********************************************/

     void PRINT_RMC()
   {
                Serial.println();
                Serial.println("---------");

                Serial.print("Header: ");
                Serial.println(data_RMC[0]);

                Serial.print("UTC: ");
                Serial.println(UTC_string);

                Serial.print("UTC Status: ");
                Serial.println(data_RMC[2]);
                Serial.print("Latitude: ");
                Serial.println(Lat_current,8);

                Serial.print("Lat N/S: ");
                Serial.println(data_RMC[4]);

                Serial.print("Longitude: ");
                Serial.println(Lon_current,8);

                Serial.print("Lat E/W: ");
                Serial.println(data_RMC[6]);

                Serial.print("Speed: ");
                Serial.println(SOG,1);

                Serial.print("Course: ");
                Serial.println(course,1);

                Serial.print("Date: ");
```

```
                Serial.println(data_RMC[9]);

                Serial.print("Mag Variation: ");
                Serial.println(MagVar);

                Serial.print("variation E/W: ");
                Serial.println(data_RMC[11]);

                Serial.print("Type of Fix: ");
                Serial.println(data_RMC[12]);

                Serial.print("GPRMC Fix Status: ");
                Serial.println(GPRMC_fix_status);

                Serial.println();

        }  // End void PRINT_RMC
#endif
```

Bilaga 11: GPS. Del 7 av 10, NMEA

Kopiera och klistra in i Arduino IDE. Spara som GPRTE.

```
 #if GPS_Used == 1
  void Get_GPRTE()

 {
     static int klast;
     static int kstart;
     RTE_timer = millis();
     int waypoint_count;

    j_MAX = word_count ;
    Parse_Sentence();


    for(int j = 0; j<j_MAX; j++)
     {
      data_RTE[j] = data_IN[j];
      if(print_RTE)Serial.println(data_IN[j]);
     }

     Number_of_sentences = To_Integer(data_RTE[1]);

     //Current_sentence_number = strtol(data_RTE[2],&pEnd,10)
     Current_sentence_number = To_Integer(data_RTE[2]);
     waypoint_count = Word_count-5;
     if(Current_sentence_number == 1)
       {
         kstart = 0;
         klast = waypoint_count;
       }
     else
       {
         kstart = klast ;
         klast = klast + waypoint_count;
       } // end else

    if(Current_sentence_number == Number_of_sentences) Number_of_waypoints
= klast;

     for(int j = 5; j<Word_count; j++)
       {
        Route [kstart + j-5 ] = data_RTE[j];
       }

      if(print_RTE) {PRINT_RTE();}
 }

 /*****************************************************************/

    void PRINT_RTE()
 {
    Serial.println();
    Serial.println("---------------");
    Serial.println("RTE DATA ");
    Serial.print("Number of sentences ");
Serial.println(Number_of_sentences);
```

```
    Serial.print("Current Sentence ");
Serial.println(Current_sentence_number);
    Serial.print("Word Count "); Serial.println(Word_count);
     for(int j = 0; j<Word_count; j++)
      {
         Serial.println(data_RTE[j]);
      }
    Serial.print("Number of Waypoints, ");
Serial.println(Number_of_waypoints);
    Serial.print("Route Data Array  ");
    for(int j = 0; j< Number_of_waypoints; j++)
    {
      Serial.print(j);
      Serial.print(Route[j]);
      Serial.print(", ");
    }
    if(Current_sentence_number == Number_of_sentences) Serial.println();


 }
 #endif
```

Bilaga 12: GPS. Del 8 av 10, sammansättning

Kopiera och klistra in i Arduino IDE. Spara som GPS_samman.

```
 #if GPS_Used == 1
/******************* PARSE SENTENCE ************************/

void Parse_Sentence()
{
  //Serial.println("Parse Sentence");
  char comma = ',';
     char star = '*';
     int last_k = 0;
     int k_start = 0;
     Word_count = 0;

   //  j_Max = 14; // antal ord i NMEA-strängen

        Checksum_calc();
         if (checksum_status)
         {
           NMEA_sentence = true;

      for (int j=0; j<j_MAX + 1; j++)
      {data_IN[j] = "";
      }

                for (int j=0; j<j_MAX; j++)
                {
                       for(int k = k_start; k < buffer_length; k++)
                       {
                              data_IN[j] = data_IN[j] + gps_buffer[k];
                              last_k = k;
                              if (gps_buffer[k] == comma || gps_buffer[k]
== star )
                               {
                                 data_IN[j] =
data_IN[j].substring(0,data_IN[j].length() - 1); //tar bort sista tecknet,
som är ett kommatecken
                                 Word_count = Word_count
+1;
                                 break;
                               }

                       }
                           //Serial.println(data_IN[j]);
                        k_start = last_k +1;
                }// end for j
      } // end for checksumi
}  // end Parse_Sentence
/******************  CONVERT LAT/LON TO DEGREES  ************************/

 void To_Degrees(String Lat_Lon)
{

  // Nordlig latitud och västlig longitud är positiva. Sydlig latitud och
ostlig longitud har problem...

double Degrees;
```

```
double Minutes;
double Decimal;

int N1;

if(LatLon_decimals == 0 && Detect_LatLon_decimals == 0){
 N1 = Lat_Lon.length() - Lat_Lon.indexOf('.') -1;
 //Detect_LatLon_decimals = 1; Används för att spara tid
}
else N1 = LatLon_decimals; //
 //Serial.print("number of decimals = "); Serial.println(N1);
 //int N1 = 4; // antal tecken
 int N2 = N1+1; // längd, 0 = N1 + 1
 int N3 = N1+3; // längd - n3 startposition för minuter = N1 + 3
 int N4 = pow(10,N1); // konverterar heltal till decimal = 10^N1

  //Serial.print("Lat_Lon "); Serial.println(Lat_Lon);
string1 = Lat_Lon.substring(Lat_Lon.length() - N1);
string1.toCharArray(char_buf,16);
long1 = strtol(char_buf,&pEnd,10);
Decimal = double(long1)/N4; // heltal till decimal
 //Serial.print("Decimal "); Serial.println(Decimal,6);
string1 = Lat_Lon.substring(Lat_Lon.length()-N3, Lat_Lon.length()-N2);
string1.toCharArray(char_buf,16);
long1 = strtol(char_buf,&pEnd,10);
Minutes = double(long1);
  //Serial.print("minutes "); Serial.println(Minutes,3);
string1 = Lat_Lon.substring(0, Lat_Lon.length()-N3); // läser ut grader
string1.toCharArray(char_buf,16);
long1 = strtol(char_buf,&pEnd,10);
Degrees = double(long1);
  //Serial.print("Degrees "); Serial.println(Degrees,8);
Lat_Lon_Deg = Degrees + Minutes/60.0 + Decimal/60.0;
  //Serial.print("Lat_Lon_Deg "); Serial.println(Lat_Lon_Deg,8);
} // end To_Degrees

/*********************************************************************/

    void Range_Bearing_Between_Points(int j, float Lat_from, float
Lon_from, float Lat_to, float Lon_to)
    {
      float Dx1, Dy1;
      float bearing;
      float range;


      Dx1 = cos(Lat_from*PI/180)*degrees_to_meters*(Lon_to - Lon_from);
//tidigare degrees_to_feet 6.5.2019
      Dy1 = degrees_to_meters*(Lat_to - Lat_from); //tidigare
degrees_to_feet 6.5.2019
      bearing = (180/PI) * atan2(Dx1 , Dy1); // 180/PI*atan2 ger -180 till
+180 vilket är grund för Dx och Dy
      if(bearing < 0 ) bearing = 360 + bearing;
      range = sqrt(Dx1*Dx1 + Dy1*Dy1);
      Range_and_Bearing[0] = range;
      Range_and_Bearing[1] = bearing;
      if(j>=0)
      {
        Waypoint_Range_From[j] = range;
        Waypoint_Bearing_From[j] = bearing;
      }
```

```
    }  // end void Range and Bearing


/*********************  WAYPOINT  AGE  ***************************/
void Waypoint_Current()
{
  int Waypoint_Age_Max = 6000; //milli seconds
   if (millis() - Waypoint_Age > Waypoint_Age_Max)
   {
    Active_waypoint = "NO WPT";
    //Serial.println("Waypoint_Age > 6");
   }
  // else Waypoint_Age_Old = millis();
}  // end waypoint current

#endif

/*****************************/

 int To_Integer(String string1)
{
  int int1;
  string1.toCharArray(char_buf,16);
  int1 = strtol(char_buf,&pEnd,10);
  return int1;
}  // end To_Integer

/****************** CONVERT NMEA INPUT TO FLOATING VARIABLE
*****************/

void NMEA_TO_FLOAT(int N)
{  // kan förbättras så att man inte manuellt måste ange hur många
decimaler som behövs, vet inte hur man ska lösa detta...
 //string1 input, float3 output
    string2 = string1.substring(string1.length()- N);
    string2.toCharArray(char_buf,16);
    long1 = strtol(char_buf,&pEnd,10);
    float1 = float(long1)/pow(10,N);
    //Serial.println();
   // Serial.println(float1);
    string2 = string1.substring(0, string1.length() - N+1);
   // Serial.println(string2);
    string2.toCharArray(char_buf,16);
    long1 = strtol(char_buf,&pEnd,10);
    float2 = float(long1);
    //Serial.println(float2);
    float3 = float1 + float2;
}  // END NMEA TO FLOAT
```

Bilaga 13: GPS. Del 9 av 10, gir innan WayPoint

Kopiera och klistra in i Arduino IDE. Spara som GPWPL.

```
. #if GPS_Used == 1
 void Get_GPWPL()
 {

    j_MAX = 6;  // antal ord i NMEA-strängen
    Parse_Sentence();
    for(int j = 0; j<j_MAX; j++)
     {
      data_WPL[j] = data_IN[j];
      if(print_WPL)Serial.println(data_IN[j]);
     }
   //    *************  PROCESS DATA  **********************

        // data_BOD[0], header, Processed in Get_Sentence()
        // data_BOD[1],Waypoint Lattitude

     To_Degrees(data_WPL[1]);
     Lat_Waypoint = Lat_Lon_Deg;
     if(data_WPL[2] == "S") Lat_Waypoint = - Lat_Waypoint;
     // data_WPL[2] = Lat N/S, ingen beräkning

     To_Degrees(data_WPL[3]);
     Lon_Waypoint = Lat_Lon_Deg;
     if(data_WPL[4] == "W") Lon_Waypoint = - Lon_Waypoint;
    // data_WPL[5] = Waypoint namn
       Waypoint_name = data_WPL[5];

      //  STORE DATA IN ARRAY USING ROUTE INDEX
     for (int i = 0; i< Number_of_waypoints; i++)
     {
       if (data_WPL[5] == Route[i])
       {

         Waypoint_Lat[i] = Lat_Waypoint;
         Waypoint_Lon[i] = Lon_Waypoint;
         if(i>0)
           {
              Range_Bearing_Between_Points(i, Waypoint_Lat[i-1],
Waypoint_Lon[i-1], Waypoint_Lat[i], Waypoint_Lon[i]);

              Next_Turn[i-1] = Waypoint_Bearing_From[i] -
Waypoint_Bearing_From[i-1];
                   if(Next_Turn[i-1] > 180) Next_Turn[i-1] = 360 -
Next_Turn[i-1]; // mindre än 180 grader, medsols positiv
                   if(Next_Turn[i-1] < -180) Next_Turn[i-1] = -360 -
Next_Turn[i-1];
           }

        int1 = i;  //used to print stored matrix data

        break;
       }

       if( Waypoint_next == Route[0])
       {
```

```
          Waypoint_Bearing_From[0] = Bearing_origin_to_destination;
         // Waypoint_Range_From[0] =
        }  // end if waypoint next == first waypoint (Route[0]

        if (Waypoint_next == Route[i]) NEXT_TURN = Next_Turn[i];
        if (Active_waypoint == "NO WPT") NEXT_TURN = 0.0;
//        ETdata.SD_NEXT_TURN = NEXT_TURN;
     } // end for

       if(int1 == Number_of_waypoints -1){
        if(print_WPL) PRINT_WPL();
       }

 }  // END GET GPWPL


/*********************************************************************************
*/

   void PRINT_WPL()
 {
    Serial.println();
    Serial.println("---------------");

        Serial.print("Waypoint, ");  Serial.print("WPL Lat, ");
Serial.print("WPL Lon, ");
        Serial.print("Bearing From Previous, "); Serial.print("Range From
Previous, "); Serial.println("Next Turn, ");

        for (int i = 0; i< Number_of_waypoints; i++)
         {
          Serial.print(Route[i]); Serial.print(",  ");
          Serial.print( Waypoint_Lat[i],8); Serial.print(",  ");
          Serial.print( Waypoint_Lon[i],8); Serial.print(",  ");
          Serial.print(Waypoint_Bearing_From[i],2);  Serial.print(",  ");
          Serial.print(Waypoint_Range_From[i],1); Serial.print(",  ");
          Serial.println(Next_Turn[i],1);
         }
       Serial.print("Next Turn, "); Serial.println(NEXT_TURN);

           // Serial.print("WPL Lat ");  Serial.print(Lat_Waypoint,4);

            // Serial.print("WPL Lat  N/S ");
                // Serial.println(data_WPL[2]);
           //   if (data_WPL[2] == "N") Serial.println ( "   NORTH");

          //  Serial.print("WPL Lon ");   Serial.print(Lon_Waypoint,4);

           // Serial.print("WPL Lon  E/W ");
               //  Serial.println(data_WPL[4]);
            //  if (data_WPL[4] == "W") Serial.println ( "  WEST");
   }
#endif
```

Bilaga 14: GPS. Del 10 av 10, avstånd från kurslinje

Kopiera och klistra in i Arduino IDE. Spara som G_XTE.

```
#if GPS_Used ==1
#if Use_CTS == 0
/****************** CROSS TRACK ERROR CALC ********************/

void Get_Cross_Track_Error()
{
     float angle1;
     //float angle2;  // vinkeln mellan BOD och kursen

     Destination_Bearing ();// Använd detta om man vill ta
Bearing_to_Destination från GPRMB (mindre exakt men mera stabil autopilot)
     angle1 = Waypoint_Bearing_From[WPT_index] -
Bearing_to_destination_by_LatLon; // ändra för att använda WPT_index
     //angle1 = Waypoint_Bearing_From[GPS_WPT_index] -
Bearing_to_destination_by_LatLon;
     //angle1 = Waypoint_Bearing_From[GPS_WPT_index] -
Bearing_to_destination;
     if(angle1 > 180) angle1 = 360 - angle1; // mindre än 180, medsols som
positiv
     if(angle1 < -180) angle1 = 360 + angle1;
     XTE = Range_Destination_by_LatLon * sin(PI*angle1/180.0);// XTE i
meter, om XTE är posotiv, gira babord
     //XTE = Range_Destination * sin(PI*angle1/180.0);// XTE i meter, om
XTE är posotiv, gira babord
     // Serial.print("angle1 "); Serial.print(angle1);
Serial.print("  Range destination, ");
Serial.print(Range_Destination_by_LatLon); Serial.print(" XTE,
");Serial.println(XTE);
     //angle2 = course - Bearing_origin_to_destination; // positiv om Dxte
är positiv
     //if(angle2 > 180) angle2 = 360 - angle2; // håll vinkeln < 180 med
medsols som positiv
     //if(angle2 < -180) angle2 = 360 + angle2;
     //XTE_differential_error = SOG* sin(PI*angle2/180.0); // XT hastighet
i knop

     if(print_Nav_Data)
     {
      Serial.println();
      Serial.print("Waypoint Next ");
      Serial.println(Waypoint_next);
      Serial.print("WPT_Index "); Serial.println(WPT_index);
      Serial.print("BOD: ");
      //Serial.println(Bearing_origin_to_destination);
      Serial.println(Waypoint_Bearing_From[WPT_index]);
      Serial.print("Angle1 "); Serial.println(angle1);
      Serial.println("WPT Lat Lon followed by current Lat Lon ");
      Serial.print(Current_Waypoint_Lat,5); Serial.print("  ");
Serial.println(Current_Waypoint_Lon,5);
      Serial.print(Lat_current,5); Serial.print("  ");
Serial.println(Lon_current,5);
      Serial.print("Bearing to Destination ");
      Serial.print(Bearing_to_destination,3);Serial.print("  ");
//Serial.println(Bearing_to_destination,3);
      Serial.print("Range: ");
```

```
      //Serial.println(Range_Destination,3);
      Serial.println(Range_Destination_by_LatLon,0);
      Serial.print("XTE: ");
      Serial.println( XTE,2);
      //Serial.print ("XTE rate ");
      //Serial.println(XTE_differential_error,1);
     }  // end print_Nav_Data

}// End Get cross track error

/***************  DESTINATION BEARING  *******************************/
  void Destination_Bearing ()
  {
      float Dx1, Dy1;
      float bearing;


/*     for (int i = 0; i< Number_of_waypoints; i++)
     {
         //if (data_WPL[5] == Waypoint_next)
         if(Route[i]== Waypoint_next)
        {
          //Serial.print(Waypoint_next); Serial.print("
Current_Waypoint_Lat Index "); Serial.println(i);
         Current_Waypoint_Lat = Waypoint_Lat[i];
         Current_Waypoint_Lon = Waypoint_Lon[i];
         break;
        }
     }
*/
     Current_Waypoint_Lat = Waypoint_Lat[WPT_index];
     Current_Waypoint_Lon = Waypoint_Lon[WPT_index];

      Dx1 = cos(Lat_current*PI/180)*degrees_to_meters*(Current_Waypoint_Lon
- Lon_current); //tidigare feet 6.5.2019
      Dy1 = degrees_to_meters*(Current_Waypoint_Lat - Lat_current);
//tidigare feet 6.5.2019
      bearing = (180/PI) * atan2(Dx1 , Dy1);
      if(bearing < 0 ) bearing = 360 + bearing;
      Bearing_to_destination_by_LatLon = bearing;
      Range_Destination_by_LatLon = sqrt(Dx1*Dx1+Dy1*Dy1);  //avstånd i
meter 6.5.2019


  }  // End Destination Bearing
  /*****************************************************************/

    void ANTICIPATE_TURN()
   {
      // if(Number_of_waypoints == 0) return; // Använd denna rad om $GPWPL
saknas från plottern
     if( 1852 * Range_Destination < Turn_distance)
      {
        if(Anticipated_Turn_Active == false)
        {
        WPT_index = GPS_WPT_index + 1;
        Anticipated_Turn_Active = true;
        }
      }
      else  // if range not < turn distance
       {
```

```
         if( WPT_index == GPS_WPT_index)
          {
            Anticipated_Turn_Active = false;
          }
        }
    }  //  End ANTICIPATE_TURN
    /*************************************************************************
**************************/
#endif
#endif
```

Bilaga 15: BNO055 kompass

Kopiera och klistra in i Arduino IDE. Spara som BNO055.

```
#if GPS_Used ==1
#if Use_CTS == 0
/***************** CROSS TRACK ERROR CALC ********************/

void Get_Cross_Track_Error()
{
     float angle1;
     //float angle2;  // vinkeln mellan BOD och kursen

     Destination_Bearing ();// Använd detta om man vill ta
Bearing_to_Destination från GPRMB (mindre exakt men mera stabil autopilot)
     angle1 = Waypoint_Bearing_From[WPT_index] -
Bearing_to_destination_by_LatLon; // ändra för att använda WPT_index
     //angle1 = Waypoint_Bearing_From[GPS_WPT_index] -
Bearing_to_destination_by_LatLon;
     //angle1 = Waypoint_Bearing_From[GPS_WPT_index] -
Bearing_to_destination;
     if(angle1 > 180) angle1 = 360 - angle1; // mindre än 180, medsols som
positiv
     if(angle1 < -180) angle1 = 360 + angle1;
     XTE = Range_Destination_by_LatLon * sin(PI*angle1/180.0);// XTE i
meter, om XTE är posotiv, gira babord
     //XTE = Range_Destination * sin(PI*angle1/180.0);// XTE i meter, om
XTE är posotiv, gira babord
     // Serial.print("angle1 "); Serial.print(angle1);
Serial.print("  Range destination, ");
Serial.print(Range_Destination_by_LatLon); Serial.print(" XTE,
");Serial.println(XTE);
     //angle2 = course - Bearing_origin_to_destination; // positiv om Dxte
är positiv
     //if(angle2 > 180) angle2 = 360 - angle2; // håll vinkeln < 180 med
medsols som positiv
     //if(angle2 < -180) angle2 = 360 + angle2;
     //XTE_differential_error = SOG* sin(PI*angle2/180.0); // XT hastighet
i knop

     if(print_Nav_Data)
     {
      Serial.println();
      Serial.print("Waypoint Next ");
      Serial.println(Waypoint_next);
      Serial.print("WPT_Index "); Serial.println(WPT_index);
      Serial.print("BOD: ");
      //Serial.println(Bearing_origin_to_destination);
      Serial.println(Waypoint_Bearing_From[WPT_index]);
      Serial.print("Angle1 "); Serial.println(angle1);
      Serial.println("WPT Lat Lon followed by current Lat Lon ");
      Serial.print(Current_Waypoint_Lat,5); Serial.print("  ");
Serial.println(Current_Waypoint_Lon,5);
      Serial.print(Lat_current,5); Serial.print("  ");
Serial.println(Lon_current,5);
      Serial.print("Bearing to Destination ");
      Serial.print(Bearing_to_destination,3);Serial.print("  ");
//Serial.println(Bearing_to_destination,3);
      Serial.print("Range: ");
```

```
      //Serial.println(Range_Destination,3);
      Serial.println(Range_Destination_by_LatLon,0);
      Serial.print("XTE: ");
      Serial.println( XTE,2);
      //Serial.print ("XTE rate ");
      //Serial.println(XTE_differential_error,1);
     }  // end print_Nav_Data

}// End Get cross track error

/***************  DESTINATION BEARING  *******************************/
  void Destination_Bearing ()
  {
      float Dx1, Dy1;
      float bearing;


/*     for (int i = 0; i< Number_of_waypoints; i++)
     {
         //if (data_WPL[5] == Waypoint_next)
         if(Route[i]== Waypoint_next)
        {
          //Serial.print(Waypoint_next); Serial.print("
Current_Waypoint_Lat Index "); Serial.println(i);
         Current_Waypoint_Lat = Waypoint_Lat[i];
         Current_Waypoint_Lon = Waypoint_Lon[i];
         break;
        }
     }
*/
     Current_Waypoint_Lat = Waypoint_Lat[WPT_index];
     Current_Waypoint_Lon = Waypoint_Lon[WPT_index];

      Dx1 = cos(Lat_current*PI/180)*degrees_to_meters*(Current_Waypoint_Lon
- Lon_current); //tidigare feet 6.5.2019
      Dy1 = degrees_to_meters*(Current_Waypoint_Lat - Lat_current);
//tidigare feet 6.5.2019
      bearing = (180/PI) * atan2(Dx1 , Dy1);
      if(bearing < 0 ) bearing = 360 + bearing;
      Bearing_to_destination_by_LatLon = bearing;
      Range_Destination_by_LatLon = sqrt(Dx1*Dx1+Dy1*Dy1);  //avstånd i
meter 6.5.2019


  }  // End Destination Bearing
  /*****************************************************************/

    void ANTICIPATE_TURN()
   {
      // if(Number_of_waypoints == 0) return; // Använd denna rad om $GPWPL
saknas från plottern
     if( 1852 * Range_Destination < Turn_distance)
      {
        if(Anticipated_Turn_Active == false)
        {
        WPT_index = GPS_WPT_index + 1;
        Anticipated_Turn_Active = true;
        }
      }
      else  // if range not < turn distance
       {
```

```
         if( WPT_index == GPS_WPT_index)
          {
            Anticipated_Turn_Active = false;
          }
        }
    }  //  End ANTICIPATE_TURN
    /*************************************************************************
**************************/
#endif
#endif
```

Bilaga 16: Pololu kompass, del 1 av 2

Kopiera och klistra in i Arduino IDE. Spara som Compass.

```
/*

MinIMU-9-Arduino-AHRS
Pololu MinIMU-9 + Arduino AHRS (Attitude and Heading Reference System)

Copyright (c) 2011 Pololu Corporation.
http://www.pololu.com/

MinIMU-9-Arduino-AHRS is based on sf9domahrs by Doug Weibel and Jose Julio:
http://code.google.com/p/sf9domahrs/

sf9domahrs is based on ArduIMU v1.5 by Jordi Munoz and William Premerlani,
Jose
Julio and Doug Weibel:
http://code.google.com/p/ardu-imu/

MinIMU-9-Arduino-AHRS is free software: you can redistribute it and/or
modify it
under the terms of the GNU Lesser General Public License as published by
the
Free Software Foundation, either version 3 of the License, or (at your
option)
any later version.

MinIMU-9-Arduino-AHRS is distributed in the hope that it will be useful,
but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License
for
more details.

You should have received a copy of the GNU Lesser General Public License
along
with MinIMU-9-Arduino-AHRS. If not, see <http://www.gnu.org/licenses/>.

*/
#if Compass == 0
void Compass_Heading()
{
  float MAG_X;
  float MAG_Y;
  float cos_roll;
  float sin_roll;
  float cos_pitch;
  float sin_pitch;

  cos_roll = cos(roll);
  sin_roll = sin(roll);
  cos_pitch = cos(pitch);
  sin_pitch = sin(pitch);

  // adjust for LSM303 compass axis offsets/sensitivity differences by
scaling to +/-0.5 range
```

```
  c_magnetom_x = (float)(magnetom_x - SENSOR_SIGN[6]*M_X_MIN) / (M_X_MAX -
M_X_MIN) - SENSOR_SIGN[6]*0.5;
  c_magnetom_y = (float)(magnetom_y - SENSOR_SIGN[7]*M_Y_MIN) / (M_Y_MAX -
M_Y_MIN) - SENSOR_SIGN[7]*0.5;
  c_magnetom_z = (float)(magnetom_z - SENSOR_SIGN[8]*M_Z_MIN) / (M_Z_MAX -
M_Z_MIN) - SENSOR_SIGN[8]*0.5;

  // Tilt compensated Magnetic filed X:
  MAG_X =
c_magnetom_x*cos_pitch+c_magnetom_y*sin_roll*sin_pitch+c_magnetom_z*cos_rol
l*sin_pitch;
  // Tilt compensated Magnetic filed Y:
  MAG_Y = c_magnetom_y*cos_roll-c_magnetom_z*sin_roll;
  // Magnetic Heading
  MAG_Heading = atan2(-MAG_Y,MAG_X);
}
#endif
```

Bilaga 17: Pololu kompass, del 2 av 2

Kopiera och klistra in i Arduino IDE. Spara som DCM.

```
/*

MinIMU-9-Arduino-AHRS
Pololu MinIMU-9 + Arduino AHRS (Attitude and Heading Reference System)

Copyright (c) 2011 Pololu Corporation.
http://www.pololu.com/

MinIMU-9-Arduino-AHRS is based on sf9domahrs by Doug Weibel and Jose Julio:
http://code.google.com/p/sf9domahrs/

sf9domahrs is based on ArduIMU v1.5 by Jordi Munoz and William Premerlani,
Jose
Julio and Doug Weibel:
http://code.google.com/p/ardu-imu/

MinIMU-9-Arduino-AHRS is free software: you can redistribute it and/or
modify it
under the terms of the GNU Lesser General Public License as published by
the
Free Software Foundation, either version 3 of the License, or (at your
option)
any later version.

MinIMU-9-Arduino-AHRS is distributed in the hope that it will be useful,
but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License
for
more details.

You should have received a copy of the GNU Lesser General Public License
along
with MinIMU-9-Arduino-AHRS. If not, see <http://www.gnu.org/licenses/>.

*/

/*************************************************/
#if Compass == 0
void Normalize(void)
{
  float error=0;
  float temporary[3][3];
  float renorm=0;

  error= -Vector_Dot_Product(&DCM_Matrix[0][0],&DCM_Matrix[1][0])*.5;
//eq.19

  Vector_Scale(&temporary[0][0], &DCM_Matrix[1][0], error); //eq.19
  Vector_Scale(&temporary[1][0], &DCM_Matrix[0][0], error); //eq.19

  Vector_Add(&temporary[0][0], &temporary[0][0], &DCM_Matrix[0][0]);//eq.19
  Vector_Add(&temporary[1][0], &temporary[1][0], &DCM_Matrix[1][0]);//eq.19
```

```
  Vector_Cross_Product(&temporary[2][0],&temporary[0][0],&temporary[1][0]);
// c= a x b //eq.20

  renorm= .5 *(3 - Vector_Dot_Product(&temporary[0][0],&temporary[0][0]));
//eq.21
  Vector_Scale(&DCM_Matrix[0][0], &temporary[0][0], renorm);

  renorm= .5 *(3 - Vector_Dot_Product(&temporary[1][0],&temporary[1][0]));
//eq.21
  Vector_Scale(&DCM_Matrix[1][0], &temporary[1][0], renorm);

  renorm= .5 *(3 - Vector_Dot_Product(&temporary[2][0],&temporary[2][0]));
//eq.21
  Vector_Scale(&DCM_Matrix[2][0], &temporary[2][0], renorm);
}

/**************************************************/
void Drift_correction(void)
{
  float mag_heading_x;
  float mag_heading_y;
  float errorCourse;
  //Compensation the Roll, Pitch and Yaw drift.
  static float Scaled_Omega_P[3];
  static float Scaled_Omega_I[3];
  float Accel_magnitude;
  float Accel_weight;


  //*****Roll and Pitch***************

  // Calculate the magnitude of the accelerometer vector
  Accel_magnitude = sqrt(Accel_Vector[0]*Accel_Vector[0] +
Accel_Vector[1]*Accel_Vector[1] + Accel_Vector[2]*Accel_Vector[2]);
  Accel_magnitude = Accel_magnitude / GRAVITY; // Scale to gravity.
  // Dynamic weighting of accelerometer info (reliability filter)
  // Weight for accelerometer info (<0.5G = 0.0, 1G = 1.0 , >1.5G = 0.0)
  Accel_weight = constrain(1 - 2*abs(1 - Accel_magnitude),0,1);  //

  Vector_Cross_Product(&errorRollPitch[0],&Accel_Vector[0],&DCM_Matrix[2][0
]); //adjust the ground of reference
  Vector_Scale(&Omega_P[0],&errorRollPitch[0],Kp_ROLLPITCH*Accel_weight);

  Vector_Scale(&Scaled_Omega_I[0],&errorRollPitch[0],Ki_ROLLPITCH*Accel_wei
ght);
  Vector_Add(Omega_I,Omega_I,Scaled_Omega_I);

  //*****YAW***************
  // We make the gyro YAW drift correction based on compass magnetic
heading

  mag_heading_x = cos(MAG_Heading);
  mag_heading_y = sin(MAG_Heading);
  errorCourse=(DCM_Matrix[0][0]*mag_heading_y) -
(DCM_Matrix[1][0]*mag_heading_x);  //Calculating YAW error
  Vector_Scale(errorYaw,&DCM_Matrix[2][0],errorCourse); //Applys the yaw
correction to the XYZ rotation of the aircraft, depeding the position.

  Vector_Scale(&Scaled_Omega_P[0],&errorYaw[0],Kp_YAW);//.01proportional of
YAW.
  Vector_Add(Omega_P,Omega_P,Scaled_Omega_P);//Adding  Proportional.
```

```
  Vector_Scale(&Scaled_Omega_I[0],&errorYaw[0],Ki_YAW);//.00001Integrator
  Vector_Add(Omega_I,Omega_I,Scaled_Omega_I);//adding integrator to the
Omega_I
}
/**************************************************/
/*
void Accel_adjust(void)
{
 Accel_Vector[1] += Accel_Scale(speed_3d*Omega[2]);  // Centrifugal force
on Acc_y = GPS_speed*GyroZ
 Accel_Vector[2] -= Accel_Scale(speed_3d*Omega[1]);  // Centrifugal force
on Acc_z = GPS_speed*GyroY
}
*/
/**************************************************/

void Matrix_update(void)
{
  Gyro_Vector[0]=Gyro_Scaled_X(gyro_x); //gyro x roll
  Gyro_Vector[1]=Gyro_Scaled_Y(gyro_y); //gyro y pitch
  Gyro_Vector[2]=Gyro_Scaled_Z(gyro_z); //gyro Z yaw

  Accel_Vector[0]=accel_x;
  Accel_Vector[1]=accel_y;
  Accel_Vector[2]=accel_z;

  Vector_Add(&Omega[0], &Gyro_Vector[0], &Omega_I[0]);  //adding
proportional term
  Vector_Add(&Omega_Vector[0], &Omega[0], &Omega_P[0]); //adding Integrator
term

  //Accel_adjust();    //Remove centrifugal acceleration.   We are not
using this function in this version - we have no speed measurement

 #if OUTPUTMODE==1
  Update_Matrix[0][0]=0;
  Update_Matrix[0][1]=-G_Dt*Omega_Vector[2];//-z
  Update_Matrix[0][2]=G_Dt*Omega_Vector[1];//y
  Update_Matrix[1][0]=G_Dt*Omega_Vector[2];//z
  Update_Matrix[1][1]=0;
  Update_Matrix[1][2]=-G_Dt*Omega_Vector[0];//-x
  Update_Matrix[2][0]=-G_Dt*Omega_Vector[1];//-y
  Update_Matrix[2][1]=G_Dt*Omega_Vector[0];//x
  Update_Matrix[2][2]=0;
 #else
  Update_Matrix[0][0]=0;
  Update_Matrix[0][1]=-G_Dt*Gyro_Vector[2];//-z
  Update_Matrix[0][2]=G_Dt*Gyro_Vector[1];//y
  Update_Matrix[1][0]=G_Dt*Gyro_Vector[2];//z
  Update_Matrix[1][1]=0;
  Update_Matrix[1][2]=-G_Dt*Gyro_Vector[0];
  Update_Matrix[2][0]=-G_Dt*Gyro_Vector[1];
  Update_Matrix[2][1]=G_Dt*Gyro_Vector[0];
  Update_Matrix[2][2]=0;
 #endif

  Matrix_Multiply(DCM_Matrix,Update_Matrix,Temporary_Matrix); //a*b=c

  for(int x=0; x<3; x++) //Matrix Addition (update)
  {
```

```
    for(int y=0; y<3; y++)
    {
      DCM_Matrix[x][y]+=Temporary_Matrix[x][y];
    }
  }
}

void Euler_angles(void)
{
  pitch = -asin(DCM_Matrix[2][0]);
  roll = atan2(DCM_Matrix[2][1],DCM_Matrix[2][2]);
  yaw = atan2(DCM_Matrix[1][0],DCM_Matrix[0][0]);
}
#endif
```

Bilaga 18: Pololu lufttryck

Kopiera och klistra in i Arduino IDE. Spara som Hg.

```
#if UseBarometer

void Init_Barometer(){
  if (!ps.init())
  {
    Serial.println("Failed to autodetect pressure sensor!");
    while (1);
  }
  ps.enableDefault();
}

void Read_Barometer(){
  pressure = ps.readPressureMillibars();
  altitude = ps.pressureToAltitudeFeet(pressure);
  temperature = ps.readTemperatureC();

  Serial.print("p: ");
  Serial.print(pressure);
  Serial.print(" mbar\ta: ");
  Serial.print(altitude);
  Serial.print(" ft\tt: ");
  Serial.print(temperature);
  Serial.println(" deg C");
}
#endif
```

Bilaga 19: I2C

Kopiera och klistra in i Arduino IDE. Spara som I2C.

```
#if Compass == 0
/*

MinIMU-9-Arduino-AHRS
Pololu MinIMU-9 + Arduino AHRS (Attitude and Heading Reference System)

Copyright (c) 2011-2016 Pololu Corporation.
http://www.pololu.com/

MinIMU-9-Arduino-AHRS is based on sf9domahrs by Doug Weibel and Jose Julio:
http://code.google.com/p/sf9domahrs/

sf9domahrs is based on ArduIMU v1.5 by Jordi Munoz and William Premerlani,
Jose
Julio and Doug Weibel:
http://code.google.com/p/ardu-imu/

MinIMU-9-Arduino-AHRS is free software: you can redistribute it and/or
modify it
under the terms of the GNU Lesser General Public License as published by
the
Free Software Foundation, either version 3 of the License, or (at your
option)
any later version.

MinIMU-9-Arduino-AHRS is distributed in the hope that it will be useful,
but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License
for
more details.

You should have received a copy of the GNU Lesser General Public License
along
with MinIMU-9-Arduino-AHRS. If not, see <http://www.gnu.org/licenses/>.

*/

#ifdef IMU_V5

//#include <LSM6.h>
//#include <LIS3MDL.h>

LSM6 gyro_acc;
LIS3MDL mag;

#else // older IMUs through v4

//#include <L3G.h>
//#include <LSM303.h>

L3G gyro;
LSM303 compass;
```

```
#endif


void I2C_Init()
{
  Wire.begin();
}

void Gyro_Init()
{
#ifdef IMU_V5
  // Accel_Init() should have already called gyro_acc.init() and
enableDefault()
  gyro_acc.writeReg(LSM6::CTRL2_G, 0x4C); // 104 Hz, 2000 dps full scale
#else
  gyro.init();
  gyro.enableDefault();
  //gyro.writeReg(L3G::CTRL_REG4, 0x20); // 2000 dps full scale
   //commenting the above line out results in default value of 245 dps full
scale.
  gyro.writeReg(L3G::CTRL_REG1, 0x0F); // normal power mode, all axes
enabled, 100 Hz
#endif
}

void Read_Gyro()
{
#ifdef IMU_V5
  gyro_acc.readGyro();

  AN[0] = gyro_acc.g.x;
  AN[1] = gyro_acc.g.y;
  AN[2] = gyro_acc.g.z;
#else
  gyro.read();

  AN[0] = gyro.g.x;
  AN[1] = gyro.g.y;
  AN[2] = gyro.g.z;
#endif

  gyro_x = SENSOR_SIGN[0] * (AN[0] - AN_OFFSET[0]);
  gyro_y = SENSOR_SIGN[1] * (AN[1] - AN_OFFSET[1]);
  gyro_z = SENSOR_SIGN[2] * (AN[2] - AN_OFFSET[2]);
}

void Accel_Init()
{
#ifdef IMU_V5
  gyro_acc.init();
  gyro_acc.enableDefault();
  gyro_acc.writeReg(LSM6::CTRL1_XL, 0x3C); // 52 Hz, 8 g full scale
#else
  compass.init();
  compass.enableDefault();
  switch (compass.getDeviceType())
  {
    case LSM303::device_D: // Pololu IMU9 V3
      compass.writeReg(LSM303::CTRL2, 0x18); // 8 g full scale: AFS = 011
      break;
    case LSM303::device_DLHC:
```

```
      compass.writeReg(LSM303::CTRL_REG4_A, 0x28); // 8 g full scale: FS =
10; high resolution output mode
      break;
    default: // DLM, DLH
      compass.writeReg(LSM303::CTRL_REG4_A, 0x30); // 8 g full scale: FS =
11
  }
#endif
}

// Reads x,y and z accelerometer registers
void Read_Accel()
{
#ifdef IMU_V5
  gyro_acc.readAcc();

  AN[3] = gyro_acc.a.x >> 4; // shift left 4 bits to use 12-bit
representation (1 g = 256)
  AN[4] = gyro_acc.a.y >> 4;
  AN[5] = gyro_acc.a.z >> 4;
#else
  compass.readAcc();

  AN[3] = compass.a.x >> 4; // shift left 4 bits to use 12-bit
representation (1 g = 256)
  AN[4] = compass.a.y >> 4;
  AN[5] = compass.a.z >> 4;
#endif
  accel_x = SENSOR_SIGN[3] * (AN[3] - AN_OFFSET[3]);
  accel_y = SENSOR_SIGN[4] * (AN[4] - AN_OFFSET[4]);
  accel_z = SENSOR_SIGN[5] * (AN[5] - AN_OFFSET[5]);
}

void Compass_Init()
{
#ifdef IMU_V5
  mag.init();
  mag.enableDefault();
#else
  // LSM303: doesn't need to do anything because Accel_Init() should have
already called compass.enableDefault()
#endif
}

void Read_Compass()
{
#ifdef IMU_V5
  mag.read();

  magnetom_x = SENSOR_SIGN[6] * mag.m.x;
  magnetom_y = SENSOR_SIGN[7] * mag.m.y;
  magnetom_z = SENSOR_SIGN[8] * mag.m.z;
#else
  compass.readMag();

  magnetom_x = SENSOR_SIGN[6] * compass.m.x;
  magnetom_y = SENSOR_SIGN[7] * compass.m.y;
  magnetom_z = SENSOR_SIGN[8] * compass.m.z;
#endif
}
#endif
```

Bilaga 20: Knappsatsen

Kopiera och klistra in i Arduino IDE. Spara som Keypad.

```
  /*********************************

  KEYPAD TAB

  *************************************/
 // based on eventkeypad by Alexander Brevig

/* Förklaring av knappar:

  KEY TYPE  ACTION
   0 press - OFF
   1 press - COMPASS
   2 press - Följer en kurs, eller om man håller den intryckt styr den i
nuvarande GPS-riktning
   3 press - Kryss
   4 press - -10 deg, -100 deg i kryss
   5  press - rattstyrning
   6 press - +10 deg, +100 deg i kryss
   7 press - minska kursen med 1 deg
   8 press - Byter visning på displayen
   9 press - öka kursen med 1 deg
   * press/release - girar babord så länge knappen är intryckt, sedan
återgår rodret till 0 grader
   # press/release - girar styrbord så länge knappen är intryckt, sedan
återgår rodret till 0 grader
*/
void KEYPAD()
{
  char key = keypad.getKey();

  /*   if (key != NO_KEY) {
      //delay(100);
       Serial.print(" getket = "); Serial.println(key);
     }
  */
}

//tar hand om problem
void keypadEvent(KeypadEvent key){
  switch (keypad.getState()){
    case PRESSED:
      KeyPressed(key);
    break;

    case RELEASED:
     KeyReleased(key);
    break;

    case HOLD:
     KeyHeld(key);
    break;
    } // end switch(keypad.getstate)
  }  //  end void keypadEvent(KeypadEvent key)
```

```
 /*****************************************************/

void KeyPressed(char keyin)
{
     //Serial.print("keyin = "); Serial.println(keyin);
     switch (keyin){
        case '0':
          Key0_Pressed();
        break; // end case 0

        case '1':
        Key1_Pressed();
        break;

        #if GPS_Used == 1
        case '2':
        if(!GPS_Available) break;
         Steering_Mode = 2;
          Steering = true;
          Mode = "GPS";
          UTC_timer_old = millis();
          Date_Time();
          Time_decimal_old = Time_decimal;
          GPS_Steering = true;
          XTE_course_correction = 0;
          XTE_integral_error = 0;
          Avg_course = course;
          lcd.setCursor(0,3);
          lcd.print("          ");
          lcd.setCursor(0,3);
          lcd.print(Mode);
        break;
       #endif

        case '3':



          toggle = !toggle;
          if (toggle)
            {
            if (Steering_Mode != 1)
            {
                  heading_to_steer = heading;
                  integral_error = 0; // reset av integrerande felet
               #if RUDDER_OFFSET == 1
                  Rudder_Offset = rudder_position;
               #if PID_MODE == 3
                  integral_error = Rudder_Offset/PID_Ks[0];
               #endif
               #endif
               #if BEARINGRATE_OFFSET == 1
                 bearingrate_Offset = - bearingrate;
               #endif
            }
            Steering_Mode = 3;
            Steering = true;
            Mode = "TACK";
            } // end if toggle is true
```

```
          if (!Wind_Input) // om vindmätare inte finns tillgängligt
            { toggle = false;
             break;
            }
            if(!toggle)
              {
                Steering_Mode = 4;
                wind_to_steer = Wind_Dir ;
                Steering = true;
                Mode = "WIND";
              }

          lcd.setCursor(0,3);
          lcd.print("          ");
          lcd.setCursor(0,3);
          lcd.print(Mode);
        break;

        case '*':
          if(Steering_Mode == 0 || Steering_Mode ==5) break;
          {
            DODGE_MODE = true;
            Previous_Mode = Mode;
            Mode = "PORT";
           motorspeed = motorspeedMAX;
            Left_Rudder();
          }
         break;

        case '#':
         if(Steering_Mode == 0 || Steering_Mode ==5) break;
          {
            DODGE_MODE = true;
            Previous_Mode = Mode;
            Mode = "STBD";
           motorspeed = motorspeedMAX;
            Right_Rudder();
          }
        break;

        case '4':
             if (Steering_Mode==1) heading_to_steer = heading_to_steer -
10;
             if(Steering_Mode ==3)
            {
              heading_to_steer = heading_to_steer - Tack_Angle;
              TACK_ON = true;
            }
           if(Steering_Mode !=4){
             if (heading_to_steer < 0) heading_to_steer = heading_to_steer
+360;
             if (heading_to_steer > 360) heading_to_steer =
heading_to_steer -360;
             lcd.setCursor(15, 1);
             lcd.print(heading_to_steer,1);
           }  // end if steering mode != 4

 #if Wind_Input == 1
           if(Steering_Mode == 4)
            {
```

```
              wind_to_steer = wind_to_steer + 10;
              if (wind_to_steer < 0) wind_to_steer = wind_to_steer +360;
              if (wind_to_steer > 360) wind_to_steer = wind_to_steer -360;
              lcd.setCursor(15, 1);
              lcd.print(wind_to_steer,1);
            }
#endif
             if (Steering_Mode == 22)
            {
              CTS_GPS2 = CTS_GPS2 - 10;
              if (CTS_GPS2 < 0) course = CTS_GPS2 +360;
               if (CTS_GPS2 > 360) CTS_GPS2 = CTS_GPS2 -360;
               lcd.setCursor(15, 2);
               lcd.print(CTS_GPS2,1);
            } // end if steering mode == 22

            if(Steering_Mode != 1 && Steering_Mode != 3 && Steering_Mode
!= 22 && Steering_Mode !=4)
            {
               lcd.begin(20,4);
               lcd.setCursor(6,2);
               lcd.print("WRONG MODE");  // här kan man t.ex. sätta en
summer för ljudsignal
              // delay(250);
            }
         break;


         case '5':
          if(Screen !=4)
          {
            Steering_Mode = 5;
            Steering = true;
            Mode = "KNOB";
            lcd.setCursor(0,3);
            lcd.print("          ");
            lcd.setCursor(0,3);
            lcd.print(Mode);
          }
        #if Compass == 1
          if(Screen == 4)
          {
          DataStored = false;
          BNO_SaveCal();
          }
        #endif
         break;  // case5

        case '6':
             if(Steering_Mode==1) heading_to_steer = heading_to_steer + 10;
             if(Steering_Mode ==3)
            {
              heading_to_steer = heading_to_steer + Tack_Angle;
              TACK_ON = true;
            }

            if(Steering_Mode!=4)
            {
              if (heading_to_steer < 0) heading_to_steer = heading_to_steer
+360;
```

```
              if (heading_to_steer > 360) heading_to_steer =
heading_to_steer -360;
               lcd.setCursor(15, 1);
               lcd.print(heading_to_steer,1);
             }
 #if Wind_Input == 1
             if(Steering_Mode == 4){
               wind_to_steer = wind_to_steer - 10;
               if (wind_to_steer < 0) wind_to_steer = wind_to_steer +360;
               if (wind_to_steer > 360) wind_to_steer = wind_to_steer -360;
               lcd.setCursor(15, 1);
               lcd.print(wind_to_steer,1);
             }
#endif
             if (Steering_Mode == 22)
              {
                CTS_GPS2 = CTS_GPS2 + 10;
                if (CTS_GPS2 < 0) course = CTS_GPS2 +360;
                 if (CTS_GPS2 > 360) CTS_GPS2 = CTS_GPS2 -360;
                 lcd.setCursor(15, 2);
                 lcd.print(CTS_GPS2,1);
              } // end if steering mode == 22

              if(Steering_Mode != 1 && Steering_Mode != 3 && Steering_Mode
!= 22 && Steering_Mode !=4)
             {
                lcd.begin(20,4);
                lcd.setCursor(6,2);
                lcd.print("WRONG MODE");  // Installation av summer här?
                delay(250);
             }
          break;


          case '7':
             if (Steering_Mode==1 || Steering_Mode ==3)
             {
                heading_to_steer = heading_to_steer -1;
                if (heading_to_steer < 0) heading_to_steer =
heading_to_steer +360;
                 if (heading_to_steer > 360) heading_to_steer =
heading_to_steer -360;
                lcd.setCursor(15, 1);
                lcd.print(heading_to_steer,1);
             }
 #if Wind_Input == 1
             if(Steering_Mode == 4){
               wind_to_steer = wind_to_steer + 1; // motsats till HTS
               if (wind_to_steer < 0) wind_to_steer = wind_to_steer +360;
               if (wind_to_steer > 360) wind_to_steer = wind_to_steer -360;
               lcd.setCursor(15, 1);
               lcd.print(wind_to_steer,1);
              }
#endif
              if (Steering_Mode == 22)
             {
               CTS_GPS2 = CTS_GPS2 -1;
               if (CTS_GPS2 < 0) course = CTS_GPS2 +360;
                if (CTS_GPS2 > 360) CTS_GPS2 = CTS_GPS2 -360;
                lcd.setCursor(15, 2);
                lcd.print(CTS_GPS2,1);
```

```
            } // end if steering mode == 22

            if(Steering_Mode != 1 && Steering_Mode != 3 && Steering_Mode
!= 22 && Steering_Mode != 4)
            {
              lcd.begin(20,4);
               lcd.setCursor(6,2);
               lcd.print("WRONG MODE");
               delay(250);
            }
          break;

         case '8':
              Screen = Screen +1;
            #if Compass == 1
            if( Screen > 5) Screen = 0;
            #endif
            #if Compass == 0
            if( Screen > 4) Screen = 0;
            #endif
            lcd.clear();
         break;

        case '9':
            if (Steering_Mode == 1 || Steering_Mode == 3)
            {
               heading_to_steer = heading_to_steer +1;
               if (heading_to_steer < 0) heading_to_steer =
heading_to_steer +360;
               if (heading_to_steer > 360) heading_to_steer =
heading_to_steer -360;
               lcd.setCursor(15, 1);
               lcd.print(heading_to_steer,1);
            }
 #if Wind_Input == 1
           if(Steering_Mode == 4){
              wind_to_steer = wind_to_steer - 1;
              if (wind_to_steer < 0) wind_to_steer = wind_to_steer +360;
              if (wind_to_steer > 360) wind_to_steer = wind_to_steer -360;
              lcd.setCursor(15, 1);
              lcd.print(wind_to_steer,1);
           }
#endif
              if (Steering_Mode == 22)
            {
              CTS_GPS2 = CTS_GPS2 + 1;
              if (CTS_GPS2 < 0) course = CTS_GPS2 +360;
               if (CTS_GPS2 > 360) CTS_GPS2 = CTS_GPS2 -360;
               lcd.setCursor(15, 2);
               lcd.print(CTS_GPS2,1);
            } // end if steering mode == 22

            if(Steering_Mode != 1 && Steering_Mode != 3 && Steering_Mode
!= 22 & Steering_Mode !=4)
            {
              lcd.begin(20,4);
              lcd.setCursor(6,2);
              lcd.print("WRONG MODE");
              delay(250);
            }
```

```
            break;
        // these next two cases come from the remote keypad where A or B
sent when * or # released
        case'A':
         Star_Released(key);
        break;

        case'B':
         Pound_Released(key);
        break;

        case'C':
        #if GPS_Used == 1
         GPS2_mode();
        #endif
        break;

      }   // end swtich key
} // End void KeyPressed

/*************************************************************************/
void KeyReleased (char keyin)
{
   switch (keyin){

        case '*':
         Star_Released(key);
        break;

        case '#':
         Pound_Released(key);
        break;
      }  // end swtich key
} // end key released

/*************************************************/
void KeyHeld(char keyin){
  switch (keyin){
      case '2':
      #if GPS_Used ==1
      GPS2_mode();
      #endif
      break;
  } // end switch key
} // end void KeyHeld


/*************************************************/

        void Key0_Pressed()
        {
          #if Clutch_Solenoid  == 1
          Open_Solenoid();
          #endif
          Steering_Mode = 0;
          Mode = "OFF";
          Steering = false;
          GPS_Was_Available = false;
          Accept_Terms = 0;  // Tar bort användarvillkoren vid start, om
man vill ha de sätt en 1:a här
          Screen = 0;
```

```
          toggle = false; // om man vill ändra knapp 3 från kryss till
vind-programmet

        #if Board == Arduino
          lcd.begin(20,4);
        #endif
        #if BEARINGRATE_OFFSET == 1
          bearingrate_Offset = 0;

        #endif
          LCD();
         // lcd.setCursor(0,3);
         // lcd.print("          ");
          lcd.setCursor(0,3);
          lcd.print(Mode);
        }  // end Key0 pressed

/*****************************************************/
void Key1_Pressed()
{
          Steering_Mode = 1;
          Steering = true;
          Mode = "COMP";
          heading_to_steer = heading;
          integral_error = 0;
          #if RUDDER_OFFSET == 1
            Rudder_Offset = rudder_position;
            #if PID_MODE == 3
              integral_error = Rudder_Offset/PID_Ks[0] +
bearingrate_Offset/PID_Ks[0];
            #endif
           #endif
           #if BEARINGRATE_OFFSET == 1
             bearingrate_Offset = - bearingrate;
           #endif

          lcd.setCursor(0,3);
          lcd.print("          ");
          lcd.setCursor(0,3);
          lcd.print(Mode);
          lcd.setCursor(15, 1);
          lcd.print(heading_to_steer,1);
}  // end key1 pressed

void Key2_Pressed(){
  #if GPS_Used
          Steering_Mode = 2;
          Steering = true;
          Mode = "GPS";
          UTC_timer_old = millis();
          Date_Time();
          Time_decimal_old = Time_decimal;
          GPS_Steering = true;
          XTE_course_correction = 0;
          XTE_integral_error = 0; // tar bort integreringsfelet genom att
trycka in knapp 2
          Avg_course = course;
          #if BEARINGRATE_OFFSET == 1
             bearingrate_Offset = - bearingrate;
          #endif
          lcd.setCursor(0,3);
```

```
          lcd.print("          ");
          lcd.setCursor(0,3);
          lcd.print(Mode);
 #endif
}  // end Key2 pressed


/*********************************************************/
void Star_Released(char keyin)
  {
          if(Steering_Mode == 0 || Steering_Mode ==5) return;
          DODGE_MODE = false;
          Rudder_Stop();
          Mode = Previous_Mode;
   }  // end Star_Released

/*********************************************************/
void Pound_Released(char keyin)
  {
          if(Steering_Mode == 0 || Steering_Mode ==5) return;
          DODGE_MODE = false;
          Rudder_Stop();
          Mode = Previous_Mode;

   }  // end Star_Released
/*********************************************************/
#if GPS_Used == 1
void GPS2_mode()
 {
     if(!GPS_Available)return;
          //Mode = "NA"; // avstängd, minns inte varför
          Steering_Mode = 22;
          Steering = true;
          Mode = "GPS2";
          CTS_GPS2 = course; // styr GPS-kurs istället för till följande WP
          UTC_timer_old = millis();
          Date_Time();
          Time_decimal_old = Time_decimal;
          GPS_Steering = true;
          XTE_course_correction = 0;
          XTE_integral_error = 0;
          Avg_course = course;
          #if BEARINGRATE_OFFSET == 1
             bearingrate_Offset = - bearingrate;
          #endif
          lcd.setCursor(0,3);
          lcd.print("          ");
          lcd.setCursor(0,3);
          lcd.print(Mode);
 } // end GPS2_mode
 #endif
```

Bilaga 21: LCD

Kopiera och klistra in i Arduino IDE. Spara som LCD.

```
 /********************** PRINT LCD ******************************/
void LCD(){
    // börjar på kolumn 0, rad 1
    // (note: man börjar räkna från 0):
    String RP;
    int UTC_seconds;
    //lcd.clear();

    if(Screen == 0)
    {
     lcd.setCursor(0,0);
     lcd.print("        "); //14.4.2019 4 st extra mellanslag
     lcd.setCursor(0,0);
     //if(Use_CTS)lcd.print(Waypoint_next);
     //else
     lcd.print(Active_waypoint);

    // lcd.print(HDG)
   /*  lcd.setCursor(0, 1);
     lcd.print("HDG        ");
     lcd.setCursor(4, 1);
     lcd.print(heading,0);
  */

     lcd.setCursor(0,3);
     lcd.print("          "); //14.4.2019 added 4 extra spaces
     lcd.setCursor(0,3);
     lcd.print(Mode);

     if(Steering_Mode != 4)
     {
       lcd.setCursor(11, 1);
       lcd.print("HTS     "); //14.4.2019 added 4 extra spaces
       lcd.setCursor(15, 1);
       lcd.print(heading_to_steer,1);

       if(!Use_CTS) {
       lcd.setCursor(13,3);
       lcd.print("X      "); //14.4.2019 added 4 extra spaces
       lcd.setCursor(15,3);
       lcd.print(XTE,1);
      }
     }

     if(Steering_Mode == 4)
     {
       lcd.setCursor(11, 0);
       lcd.print("WTS      "); //14.4.2019 added 4 extra spaces
       lcd.setCursor(15, 0);
       lcd.print(wind_to_steer,1);
     }

     if( RUDDER_MODE == 0)  // Om det finns en rodervinkelssensor
installerad
    {
```

```
     lcd.setCursor(5,3);
     lcd.print("Rud    "); // extra spaces clear old data 14.4.2019 added 4
extra spaces
     lcd.setCursor(9,3);
     lcd.print(rudder_position,0);
    }

 //   if(MSG >0)
 //   {
      lcd.setCursor(0,2);
     switch (MSG)
      {
       // case 0:
       // Message = "                    ";
       // break;

        case 1:
         lcd.print( " NO GPS, 0 to Clear");
        break;

        case 2:
         lcd.print("NO GPS Steer HTS");
        break;

        case 3:
         lcd.print( "New WPT Press 2 Turn");
        break;

        case 4:
         lcd.print("Turning to New WPT");
        break;
     } // end switch
   // } // end if MSG > 0
    /*
     // TEMORARY
     lcd.setCursor(5,3);
     lcd.print("CMD    "); // extra spaces clear old data
     lcd.setCursor(9,3);
     lcd.print(rudder_command,0);
*/

  if (GPS_Available  && GPRMC_fix)
  {
     if(Steering_Mode != 4 && Steering_Mode != 3)
     {
        if(MSG == 0) // if MSG not zero, i. e. null, print the current
message here
         {
           lcd.setCursor(11, 0);
           lcd.print("BRG      "); //14.4.2019 added 4 extra spaces
           lcd.setCursor(15, 0);
           lcd.print(Bearing_to_destination,0);

           lcd.setCursor(0, 2); //
           lcd.print("COG        "); //14.4.2019 added 4 extra spaces
           lcd.setCursor(4, 2);
           lcd.print(course,0);

           lcd.setCursor(11, 2);
           lcd.print("CTS      ");  //14.4.2019 added 4 extra spaces
           lcd.setCursor(15, 2);
```

```
            //if(Use_CTS == 1) course_to_steer = GPS_course_to_steer;
            lcd.print(course_to_steer,1);
          }
      }  // end if steering mode != 4 or 3
  } // End if GPRMC and GPAPB true

  //if(GPRMC_fix && !GPAPB_fix) {
   //  lcd.setCursor(0, 0);
    // lcd.print("No WPT");
    // delay(LCD_delay);
   //}
} // END IF SCREEN = 0

    /************** SCREEN = 1 ****************/
      if(Screen == 1)
     {
      lcd.setCursor(0,0);
      lcd.print("BRG ");
      lcd.print(Bearing_to_destination,3);

      //lcd.setCursor(11,0);
      //lcd.print("XDE ");
      //lcd.print(XTE_differential_error,2);

      lcd.setCursor(0,1);
      lcd.print("XTE ");
      lcd.print(XTE,1);

      lcd.setCursor(11,1);
      lcd.print("Xint");
      lcd.print(XTE_integral_error,0);

      lcd.setCursor(0,2);
      lcd.print("BOD ");
      lcd.print(Bearing_origin_to_destination,1);

     if(Steering_Mode !=4){
      lcd.setCursor(11,2);
      lcd.print("BRT ");
      lcd.print(bearingrate);
     }

      lcd.setCursor(0,3);
      lcd.print("XCR ");
      lcd.print(XTE_course_correction);

     /* lcd.setCursor(11,3);
      lcd.print("Rud      "); // extra spaces clear old data
      lcd.setCursor(15,3);
      lcd.print(rudder_position,0);
    */

     lcd.setCursor(11,3);
     lcd.print("CMD    "); // extra spaces clear old data 14.4.2019 added 4
extra spaces
     lcd.setCursor(15,3);
     lcd.print(rudder_command,0);

      /*
      lcd.setCursor(0,1);
      lcd.print("DXTE ");
```

```
   lcd.print(XTE_differential_error);
   */
  }// end screen 1


 /************* SCREEN = 2 ***************/


  if(Screen == 2)
  {

   lcd.setCursor(0, 0);
   lcd.print("                    ");
   lcd.setCursor(0, 0);
  lcd.print(GPS_status); //no gps, no waypoint, or waypoint
 // lcd.print("Lt "); lcd.print(Lat_current);
  // lcd.setCursor(10,0);
  // lcd.print("Ln "); lcd.print(Lon_current);

//  This is a diagnostic it it prints seconds if GPS is processing
   lcd.setCursor(0,1);
   lcd.print("UTC ");
   lcd.print(UTC);
   /*
   lcd.setCursor(0,2);
   lcd.print("UTC Start ");
   lcd.print(UTC_start);

   lcd.setCursor(0,3);
   lcd.print("Max DT ");
   lcd.print(Time_decimal_MAX_seconds,0);
   */
   lcd.setCursor(0,3);
   lcd.print("MAGV ");
   lcd.print(Magnetic_Variation,1);

   lcd.setCursor(0,2);
   lcd.print("Rudder SPD       ");
   lcd.setCursor(15,2);
   lcd.print(motorspeed);

  } // End if screen = 2

if(Screen == 3)
{
  lcd.setCursor(0,0);
  if(Next_Turn >0) lcd.print("Next Turn R ");
   else lcd.print("Next Turn L ");
  // lcd.print(abs(Next_Turn), 1);

  lcd.setCursor(0,1);
  lcd.print("Dist To Wpt = ");
//  float Range_meters;
 //Range_meters = Range_Destination x 1852;
  lcd.print(Range_Destination,2);

  lcd.setCursor(0,2);
  lcd.print("Range m =      ");
  lcd.setCursor(11,2);
```

```
      lcd.print(Range_Destination *1852,0);

      lcd.setCursor(0,3);
      lcd.print(Active_waypoint);
    }  // End screen = 3

       if(Screen == 4)
    {
       lcd.setCursor(0,0);
       lcd.print("HDG     "); //14.4.2019 added 4 extra spaces
       lcd.setCursor(4,0);
       lcd.print(heading,0);
       lcd.setCursor(0,1);
       lcd.setCursor(10,0);
       lcd.print("DPT      "); //14.4.2019 added 4 extra spaces
       lcd.setCursor(14,0);
       lcd.print(Depth,1);
       lcd.setCursor(0,1);
       lcd.print("Wind     "); //14.4.2019 added 4 extra spaces
       lcd.setCursor(5,1);
       lcd.print(Wind_Speed,0);
       lcd.setCursor(10,1);
       lcd.print("MAX     "); //14.4.2019 added 4 extra spaces
       lcd.setCursor(14,1);
       lcd.print(Wind_MAX,0);
       lcd.setCursor(0,2);
       lcd.print("Wind Angle      "); //14.4.2019 added 4 extra spaces
       lcd.setCursor(11,2);
       lcd.print(Wind_Dir,0);
       #if UseBarometer
       lcd.setCursor(0,3); lcd.print("Pa     "); //14.4.2019 added 4 extra
spaces
       lcd.setCursor(3,3); lcd.print(pressure);
       lcd.setCursor(10,3); lcd.print("Temp     ");   //14.4.2019 added 4
extra spaces
       lcd.setCursor(15,3); lcd.print(temperature,1);
       #endif

    }

 #if Compass == 1
      if(Screen == 5)  // Sparar kompasskalibrering
    {

      lcd.setCursor(0,0);
      lcd.print("BNO055 Compass Cal");
      lcd.setCursor(0,1);
      lcd.print("To save current Cal");
      lcd.setCursor(0,2);
      lcd.print("Press Key 5");
      lcd.setCursor(0,3);
      if(DataStored)
       lcd.print("DATA STORED");
    }  // End screen = 4
 #endif

}  // END Void LCD()
```

Bilaga 22: LCD, kompassvisning

Kopiera och klistra in i Arduino IDE. Spara som LCD_comp.

```
#if Compass == 0
  void LCDprint()
  {
      lcd.setCursor(0,0);
      lcd.print ("HEAD");
      lcd.setCursor(7,0);
      lcd.print(MAG_Heading_Degrees);

      lcd.setCursor(0,1);
      lcd.print ("PITCH");
      lcd.setCursor(7,1);
      lcd.print(ToDeg(pitch));
      //lcd.print(Head_count);

      lcd.setCursor(0,2);
      lcd.print ("ROLL");
      lcd.setCursor(7,2);
      lcd.print(ToDeg(roll));

      /*
      lcd.setCursor(0,1);
      lcd.print ("mag X       ");
      lcd.setCursor(7,1);
      lcd.print(magnetom_x);

      lcd.setCursor(0,2);
      lcd.print ("mag Y        ");
      lcd.setCursor(7,2);
      lcd.print(magnetom_y);
      */

      lcd.setCursor(0,3);
      lcd.print ("YAW");
      lcd.setCursor(7,3);
      lcd.print(ToDeg(yaw));

  }  // end void LCD print
#endif
```

Bilaga 23: Pololu output

Kopiera och klistra in i Arduino IDE. Spara som Output.

```
/*

MinIMU-9-Arduino-AHRS
Pololu MinIMU-9 + Arduino AHRS (Attitude and Heading Reference System)

Copyright (c) 2011 Pololu Corporation.
http://www.pololu.com/

MinIMU-9-Arduino-AHRS is based on sf9domahrs by Doug Weibel and Jose Julio:
http://code.google.com/p/sf9domahrs/

sf9domahrs is based on ArduIMU v1.5 by Jordi Munoz and William Premerlani,
Jose
Julio and Doug Weibel:
http://code.google.com/p/ardu-imu/

MinIMU-9-Arduino-AHRS is free software: you can redistribute it and/or
modify it
under the terms of the GNU Lesser General Public License as published by
the
Free Software Foundation, either version 3 of the License, or (at your
option)
any later version.

MinIMU-9-Arduino-AHRS is distributed in the hope that it will be useful,
but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License
for
more details.

You should have received a copy of the GNU Lesser General Public License
along
with MinIMU-9-Arduino-AHRS. If not, see <http://www.gnu.org/licenses/>.

*/
#if Compass == 0
void printdata(void)
{
      Serial.print("!");

      #if PRINT_EULER == 1
      Serial.print("ANG:");
      Serial.print(ToDeg(roll));
      Serial.print(",");
      Serial.print(ToDeg(pitch));
      Serial.print(",");
      Serial.print(ToDeg(yaw));
      #endif
      #if PRINT_ANALOGS==1
      Serial.print(",AN:");
      Serial.print(AN[0]);  //(int)read_adc(0)
      Serial.print(",");
      Serial.print(AN[1]);
```

```
      Serial.print(",");
      Serial.print(AN[2]);
      Serial.print(",");
      Serial.print(AN[3]);
      Serial.print (",");
      Serial.print(AN[4]);
      Serial.print (",");
      Serial.print(AN[5]);
      Serial.print(",");
      Serial.print(c_magnetom_x);
      Serial.print (",");
      Serial.print(c_magnetom_y);
      Serial.print (",");
      Serial.print(c_magnetom_z);
      #endif
      /*#if PRINT_DCM == 1
      Serial.print (",DCM:");
      Serial.print(convert_to_dec(DCM_Matrix[0][0]));
      Serial.print (",");
      Serial.print(convert_to_dec(DCM_Matrix[0][1]));
      Serial.print (",");
      Serial.print(convert_to_dec(DCM_Matrix[0][2]));
      Serial.print (",");
      Serial.print(convert_to_dec(DCM_Matrix[1][0]));
      Serial.print (",");
      Serial.print(convert_to_dec(DCM_Matrix[1][1]));
      Serial.print (",");
      Serial.print(convert_to_dec(DCM_Matrix[1][2]));
      Serial.print (",");
      Serial.print(convert_to_dec(DCM_Matrix[2][0]));
      Serial.print (",");
      Serial.print(convert_to_dec(DCM_Matrix[2][1]));
      Serial.print (",");
      Serial.print(convert_to_dec(DCM_Matrix[2][2]));
      #endif*/
      Serial.println();

}

long convert_to_dec(float x)
{
  return x*10000000;
}
#endif
```

Bilaga 24: PID

Kopiera och klistra in i Arduino IDE. Spara som PID.

```
/*******************************

PID fliken - beräknar och sänder roderkontrollssignalen

*******************************/
/*******************************
PID_MODE
 MODE 0: rudder_command = PID_output utan I-delen
 MODE 1: rudder_command = rudder_command + PID_output.  På sätt och vis
integrerad I-del, fungerar tillfredsställande
 MODE 3: rudder_command = PID_output med I-delen
RUDDER_MODE
  Avgör vilket styrprogram av PID-kontroll och roderkontroll som används
  MODE 0: Rodret har en specigik position rudder_position
  MODE 1: Rodret rör på sig tills rudder_command < deadband


*********************************/


        void Steer_PID()
        {
          deadband = 2.0;

         RUDDER_POSITION();
         if(abs(rudder_position) > Maximum_Rudder)  Rudder_Stop();

        if(!DODGE_MODE)
        {

           MSG = 0; // null
     #if GPS_Used
           if (Steering_Mode == 2 || Steering_Mode == 22) GPS_Steer();
     #endif

 #if Wind_Input == 1
     if(Steering_Mode == 4){  // heading error = wind error
        wind_error = wind_to_steer - Wind_Avg;
        if (abs(wind_error) > 180)
          {
             if(wind_to_steer > Wind_Avg)  wind_error = wind_error - 360;
             if(wind_to_steer < Wind_Avg) wind_error = 360 + wind_error;
          }
       heading_to_steer = -wind_error + heading;
       } // end Steering_Mode == 4
#endif  // Wind_Input

         heading_error = heading_to_steer - heading;
           if (Wind_Steer_Direct == 1 && Steering_Mode == 4) heading_error =
-wind_error;
           if (GPS_Steer_Direct == 1 && (Steering_Mode == 2 || Steering_Mode
== 22)) heading_error = course_error;
```

```
           // Serial.print("Wind Error "); Serial.print(wind_error);
Serial.print("; heading to steer ");
Serial.println(heading_to_steer);
           if (abs(heading_error) > 180)
            {
               if(heading_to_steer > heading)  heading_error =
heading_error - 360;
               if(heading_to_steer < heading) heading_error = 360 +
heading_error;
            }

          /*
          lcd.setCursor(0,1);
          lcd.print("BRT       ");
          lcd.setCursor(5,1);
          lcd.print(bearingrate);
          */
          //if(abs(bearingrate) < 0.5 ) bearingrate = 0; // eliminering av
störningar


          #if Compass == 0
           differential_error = bearingrate;
          #endif

          // Serial.print(heading_error);
          // Serial.print("  ");
          // Serial.println(differential_error);

          #if PID_MODE == 0
            PID_output = PID_Ks[0] * (PID_Ks[1] * heading_error  -
PID_Ks[2] * differential_error);
            rudder_command = PID_output + Rudder_Offset;
          #endif

          #if PID_MODE == 1
           PID_output = PID_Ks[0] * (PID_Ks[1] * heading_error  - PID_Ks[2]
* differential_error);
           rudder_command = rudder_command + PID_output + Rudder_Offset;
          #endif

          #if PID_MODE == 3
           if(abs(heading_error)> deadband) // om heading error < deadband
integral error förhindras att öka
           {integral_error = integral_error + PID_Ks[3] * heading_error;
           }
           if (!Steering) integral_error = 0;
           integral_error = constrain(integral_error,-10,10);  //
förhindrar att I stiger till mer än +-10 grader
           PID_output = PID_Ks[0] * (PID_Ks[1] * heading_error  - PID_Ks[2]
* differential_error + integral_error);
           rudder_command = PID_output;
          /* // Felsökning
          lcd.setCursor(0,0);
          lcd.print("CMD ");
          lcd.print(rudder_command,1);// diagnostic
          lcd.setCursor(11,0);
          lcd.print("IE ");
          lcd.print(integral_error *PID_Ks[0],1);
          */
           // if(abs(heading_error) < 10) TACK_ON = false;
```

```
          //  if(TACK_ON) rudder_MAX = Tack_rudder_MAX; //
          #endif

          if (Steering_Mode == 5)
         {
           Knob_Steering();
         }  // end if(Steering_Mode == 5)

          rudder_MAX = Maximum_Rudder;
          if(TACK_ON)   rudder_MAX = Tack_rudder_MAX;
  //        if(new_waypoint) rudder_MAX = new_waypoint_rudder_MAX;
          rudder_command=constrain(rudder_command,-rudder_MAX,rudder_MAX);
        }   // end if  not DODGE_MODE

        if(!Steering)
           {
             rudder_command = 0;
             heading_to_steer = 0;
           }


         Rudder_Control();

         if(Print_PID)
         {
           Serial.print("course : "); Serial.println(course,1);
           //Serial.print("Heading Average: "); Serial.println(Cavg,1);
           Serial.print("heading to steer: ");
Serial.println(heading_to_steer,1);
           Serial.print("Heading Error: ");
Serial.println(heading_error,1);
         //  Serial.print("compass delta T in sec: ");
Serial.println(compass_delta_T);
          // Serial.print("delta heading: ");
Serial.println(delta_heading);
           Serial.print("Bearing Rate: ");Serial.println(bearingrate);
           Serial.print("Integral error: ");
Serial.println(integral_error);
           Serial.print("PID Output: "); Serial.println(PID_output);
          // Serial.print("Rudder: "); Serial.println(rudder_change);
           Serial.println("**********************");
         }   // end if Print PID
       }  // End Void Steer_PID()

/*********************************************************************/
        void Rudder_Control()
        {
         // int motorspeed_min = 30;
         float Rudder_Power_coeff = .5;  //  Använd 0 om man inte vill ha
denna funktion, ökar roderhastigheten vid högre gradtal på
rodervinkelssensorn för att få bättre respons i dåligt väder.

          RUDDER_POSITION();
          if (RUDDER_MODE ==1) rudder_position = 0;
          rudder_error = rudder_command - rudder_position;

          if(Steering_Mode == 0 || !sw1 || !sw2)  // knapp 1 och knapp 2
måste vara ON för att aktivera automatisk styrning
         {
            Steering = false;
            Rudder_Stop();
```

```
           #if Clutch_Solenoid  == 1
            Open_Solenoid();  // öppnar solenoid för att möjliggöra
automatisk styrning
          #endif
         }
        #if Clutch_Solenoid  == 1
        if(Steering_Mode != 0 && sw1 && sw2) Close_Solenoid();   // stänger
solenoiden
        #endif

       // if(Steering_Mode == 1 || Steering_Mode == 2 || Steering_Mode ==3
|| Steering_Mode == 5) Steering = true;
       if(Steering_Mode >0) Steering = true;
       // if(DODGE_MODE) Steering = false;

        if(!DODGE_MODE)
        {
       //   if(rudder_on)  RUDDER_POSITION();
          if(Steering)
          {
        motorspeed = motorspeedMAX/ 30 *rudder_error; // vid 30 grader:
rudded speed = MAX
        motorspeed = abs(motorspeed);
         #if Rudder_Power_coeff > 0
          if(abs(rudder_command) > abs(rudder_position))
          {
            motorspeed = motorspeedMIN + float(motorspeedMAX-motorspeedMIN)
* ((abs(rudder_position)/Maximum_Rudder) * Rudder_Power_coeff);
          }
         #endif
        motorspeed = constrain(motorspeed, motorspeedMIN, motorspeedMAX);
       // Serial.print("motor speed ");Serial.println(motorspeed);

        if(TACK_ON || Steering_Mode == 2 || Steering_Mode == 22)
          {
             motorspeed = min(motorspeed, Tack_rudder_speed*motorspeedMAX);
            if (abs(rudder_command) < deadband+1) TACK_ON =
false;
          }  // end if tack on


                        if(abs(rudder_error) < deadband)
                          {
                           Rudder_Stop();
                          }

                       if(rudder_error > deadband)
                          {
                            Right_Rudder();
                          }

                       if(rudder_error < - deadband)
                          {
                            Left_Rudder();
                          }

            } // end  if Steering
        }  // end if(!DODGE_MODE)
        }  // void rudder control
```

```
  //------------------------  RUDDER POSITION  ----------------------

  void RUDDER_POSITION()
  {
     float rudder_position_max = 45;
     float rudder_position_min = -45;
     float counts_max = 900;
     float counts_at_zero = 492;
     float counts_min = 195;
     float counts;

     counts = analogRead(4);
     //Serial.print("Rudder = ");
     //Serial.println(counts);
      if(counts >= counts_at_zero)
      {
          rudder_position = rudder_position_max *(counts - counts_at_zero)
/ (counts_max - counts_at_zero);
      }
      else
      {
          rudder_position = rudder_position_min * (counts - counts_at_zero)
/ (counts_min - counts_at_zero);
      }
      rudder_position = - rudder_position;  // tillfällig reverse direction
of positive rudder position 10.5.2019

    // rudder_position =map(rudder_position, 187,910,-45,45);

   //  Serial.print("rudder, "); Serial.println(rudder_position);

  }  // END VOID RUDDER POSITION


  // ----------------------  END RUDDER POSITION --------------------


 //------------- RUDDER CONTROLS ----------------------------------------
--------
#if Clutch_Solenoid  == 1
 void Open_Solenoid()
   {
        Serial_MotorControl.write(Motor_0_fwd);  // set motor 0 framåt, 137
för Qik, 201 för Trex
        Serial_MotorControl.write(0);  // speed = 0
          if(Print_Motor_Commands)
     {  Serial.print ("Motor Code, motorspeed ");
        Serial.print(Motor_0_fwd);Serial.print(", "); Serial.println(0);
     }

   }  // End Void Open Solenoid


 void Close_Solenoid()
   {
       Serial_MotorControl.write(Motor_0_fwd); // 137 för Qik, 201 för Trex
       Serial_MotorControl.write(127); // speed = full
         if(Print_Motor_Commands)
     {  Serial.print ("Motor Code, motorspeed ");
        Serial.print(Motor_0_fwd);Serial.print(", "); Serial.println(127);
```

```
     }
     //digitalWrite(10, LOW);   // stänger solenoiden för manuell styrning
   }  // end void Close-Solenoid
# endif

  void Rudder_Stop()
 {
    #if Motor_Controller == 3
      Serial_MotorControl.write(Motor_1_fwd); //  för Qik 141.  Motor 1
framåt för Trex(193)
      Serial_MotorControl.write(0);
   #endif

   #if Motor_Controller == 3
       motorspeed = 0;
       Serial_MotorControl.write(Motor_1_fwd);
       Serial_MotorControl.write(motorspeed & 0x1F);
       Serial_MotorControl.write(motorspeed >> 5);
   #endif
  //  rudder_stop_time = millis();
      rudder_on = false;
      rudder_was_off = true;
     if(Print_Motor_Commands)
     {  Serial.print ("Motor Code, motorspeed ");
        Serial.print(Motor_1_fwd);Serial.print(", "); Serial.println(0);
     }
 }  // end Rudder_Stop


  void Left_Rudder()
  {
     #if Motor_Controller == 3
      Serial_MotorControl.write(Motor_1_rev);  //   set motor 1 i back, 143
för Qik, 194 för TREX
      Serial_MotorControl.write(motorspeed); // set speed = motorspeed 0
till 127, 0% till 100%
     #endif

     #if Motor_Controller == 3
       Serial_MotorControl.write(Motor_1_rev);
       Serial_MotorControl.write(motorspeed & 0x1F);
       Serial_MotorControl.write(motorspeed >> 5);
     #endif

    if(Print_Motor_Commands)
     {  Serial.print ("Rudder Command Motor Code, motorspeed ");
        Serial.print(rudder_command); Serial.print(", ");
Serial.print(Motor_1_rev);Serial.print(", "); Serial.println(motorspeed);
     }
      rudder_on = true; //used in rudder position
      if (rudder_was_off)
          {
            rudder_time_old = millis();
            rudder_was_off = false;
          }
   // }
  } // end Left_Rudder()
// ---------------------------------------

  void Right_Rudder()
  {
```

```
     #if Motor_Controller == 3
      Serial_MotorControl.write(Motor_1_fwd);  //   set motor 1 i back, 143
för Qik, 194 för TREX
      Serial_MotorControl.write(motorspeed); // set speed = motorspeed 0
till 127, 0% till 100%
     #endif

     #if Motor_Controller == 3
       Serial_MotorControl.write(Motor_1_fwd);
       Serial_MotorControl.write(motorspeed & 0x1F);
       Serial_MotorControl.write(motorspeed >> 5);
     #endif

        rudder_on = true; //used in rudder position
        if (rudder_was_off)
          {
            rudder_time_old = millis();
            rudder_was_off = false;
          }
        if(Print_Motor_Commands)
         {   Serial.print ("Rudder Command Motor Code, motorspeed ");
        Serial.print(rudder_command); Serial.print(", ");
Serial.print(Motor_1_fwd);Serial.print(", "); Serial.println(motorspeed);
         }
    //}  // end if rudder < rudder MAX
  } // end Right_Rudder


    /*********************************************************************
*/
 #if GPS_Used
    void GPS_Steer()
    {
      static int waypoint_error_count;

      if(!GPS_Available)
        {
          if(GPS_Was_Available) // GPS_Was_Available sätt "true" i båda
fallen nedan om GPS:en fungerar, om inte sätt "false"
            {
              heading_to_steer = heading;
              GPS_Was_Available = true;
            }  // end If(GPS_Was_Available
         GPS_Steering = true;
         MSG = 2;
        }  // end if GPS not available

     if(GPS_Available)
     {
      MSG = 0; // null message
      GPS_Steering = true;
       if(Mode == "GPS2") Actual_GPS_Steering();
        if ( Active_waypoint != "NO WPT")
       {
           Actual_GPS_Steering();
           if (MSG != 4) MSG = 0;
       }  // end if Active_waypoint != "NO WPT"
     }  // End if GPS available
    } // end GPS_Steer()
```

```
/*************************************************************************
***********/
void Actual_GPS_Steering()
{  const float XTE_to_45_correction = 60; //XTE i meter, snligt Garmin GPS
map 720 sätt att beräkna en kurs på, i meter 6.5.2019
   static long CTStime;
  //CORRECTION FOR CROSS TRACK ERROR
 /* Course correction using Cross Track Error  */
    if(!Use_CTS){
    // #if GPS_source != 1
    //   Get_Cross_Track_Error(); // behövs inte
    // #endif
       XTE_course_correction = 45.0 * XTE /
XTE_to_45_correction;
       XTE_course_correction = constrain(XTE_course_correction,-45.0,45.0);
       //course_to_steer = Bearing_origin_to_destination -
XTE_course_correction;
       course_to_steer = Waypoint_Bearing_From[WPT_index] -
XTE_course_correction;
    }   // end if(!USe_CTS)
   /* This is couse corretion using chart plotter Course to steer  */
       if(Use_CTS) course_to_steer = GPS_course_to_steer;  //Använder CTS
from NMEA GPAPB word 13
        if (course_to_steer > 360) course_to_steer = course_to_steer - 360;
        if (course_to_steer < 0) course_to_steer = course_to_steer + 360;
        course_error = course_to_steer - course;

        heading_to_steer = heading + course_error;

        if (heading_to_steer > 360) heading_to_steer = heading_to_steer -
360;
        if (heading_to_steer < 0) heading_to_steer = heading_to_steer +
360;

     if(Print_Anticpate_Turn == 0) // slå på för felsökning
     {
       if (millis() - CTStime >1000){  // n/1000 sek
         CTStime= millis();
       Serial.println();
       Serial.println("WPT,     WPTix, BOD,   BRG, XTE,
RNG,   COG,   CTS,   HDG,    CE,   HTS ");
       Serial.print(Active_waypoint); Serial.print(", ");
       Serial.print(WPT_index); Serial.print(",   ");
       Serial.print(Waypoint_Bearing_From[WPT_index]); Serial.print(", ");
       Serial.print(Bearing_to_destination_by_LatLon,0); Serial.print(",
");
       Serial.print(XTE,0); Serial.print(", ");
       Serial.print(Range_Destination_by_LatLon,0); Serial.print(", ");
       lcd.setCursor(0,3);
       lcd.print("RNG         ");
       lcd.setCursor(4,3);
       lcd.print(Range_Destination_by_LatLon,0);
       Serial.print(course); Serial.print(", ");
       Serial.print(course_to_steer,0); Serial.print(", ");
      // Serial.print(Avg_course); Serial.print(", ");
       Serial.print(heading); Serial.print(", ");
       Serial.print(course_error); Serial.print(", ");
       Serial.print(heading_to_steer); Serial.print(", ");
      // Serial.print(heading_error); Serial.print(", ");
       }
```

```
     }

}// end actual gps steering
#endif


/*************************************************************************
***********/
 void Knob_Steering()
 {
   float Knob;
    Knob =   analogRead(2);
   //Serial.println(Knob);
   rudder_command =  82 * float(Knob/1000) -42 ; // rudder command +/-45


   lcd.setCursor(0, 0);
  // lcd.print("           ");
  // lcd.setCursor(0, 0);
   lcd.print("CMD ");
   lcd.print(rudder_command);

     lcd.setCursor(5,3);
     lcd.print("Rud    "); // mellanslag tar bort gammal data
     lcd.setCursor(9,3);
     lcd.print(rudder_position,0);



   /*
   lcd.setCursor(0,1);
   lcd.print("speed      ");
   lcd.setCursor(7,1);
   lcd.print("     ");
   lcd.setCursor(7,1);
   lcd.print(motorspeed);
   */


 } // end void knob steering
```

Bilaga 25: Framställning

Kopiera och klistra in i Arduino IDE. Spara som Print.

```
#if GPS_Used
        void Print_interval(){

       int  print_timer = millis()-PT_old;

        if (print_timer  > print_time*1000){
          print_level = print_level_max;
          PT_old = millis();
        }
        else{ print_level=0;}
       /************************* // diagnostic
        print_level=0;
        Serial.print("millis(): "); Serial.println(millis());
        Serial.print("print_time: "); Serial.println(print_time);
        Serial.print("PT_old: "); Serial.println(PT_old);
        Serial.print("Print Interval: "); Serial.println(print_time*1000);
        ************************************************/
        }  // end of  void Print_interval()



/*****************************************************/

    void NAV_DATA_PRINT()
    {

     // Serial.println(" UTC, waypoint, BRG, RNG, BOD, COG, XTE, XTE_Corr,
CTS,HDG,  HTS, CMD, RUD);
      Serial.print(UTC_string); Serial.print(", ");
      Serial.print(Waypoint_next); Serial.print(", ");
      Serial.print(Bearing_to_destination); Serial.print(", ");
      Serial.print(Range_Destination,3); Serial.print(", ");
      Serial.print(Bearing_origin_to_destination ); Serial.print(", ");
      Serial.print(course ); Serial.print(", ");
      Serial.print(XTE); Serial.print(", ");
      Serial.print(XTE_course_correction ); Serial.print(", ");
      Serial.print(course_to_steer); Serial.print(", ");
      Serial.print(heading); Serial.print(", ");
      Serial.print(heading_to_steer); Serial.print(", ");
      Serial.print(bearingrate); Serial.print(", ");
      Serial.print(rudder_command ); Serial.print(", ");
      Serial.print(rudder_position);Serial.print(", ");
      Serial.println();
    }  // end NAV_DATA_PRINT
#endif

/*************************************************************************
*********/
```

Bilaga 26: RF24 fjärrstyrning

Kopiera och klistra in i Arduino IDE. Spara som RF24.

```
#if RF24_Attached == 1
 /*  RADIO RF24 */
 // based on formatted data sketch
 /*
Sending formatted data packet with nRF24L01.
Maximum size of data struct is 32 bytes.
 contributted by iforce2d

The wire numbers listed are using a ribbon wire and a two row ribbon wire
connector
1 - GND, wire 2
2 - VCC 3.3V !!! NOT 5V, wire 1
3 - CE to Arduino pin 9, wire 4
4 - CSN to Arduino pin 10, wire 3
5 - SCK to Arduino pin 13 for Uno, 52 on Mega, wire 6
6 - MOSI to Arduino pin 11 for Uno, 51  on Mega, wire 5
7 - MISO to Arduino pin 12 for Uno,  50 on Mega, wire 8
8 - UNUSED, wire 7
*/

void sendData1()
{
  //int tmp1 = int(Wind_Dir);
  //int tmp2 = int(Wind_Speed);
// Serial.println(sizeof(RF_DATA));
 Active_waypoint.toCharArray(RFdata.RFD_text,8);
 RFdata.RFD_int1 = int(heading);
 RFdata.RFD_int2 = int(heading_to_steer);
// if(Steering_Mode == 4) RFdata.RFD_int2 = int(wind_to_steer);
 RFdata.RFD_int3 = int(course);
 RFdata.RFD_int4 = int(course_to_steer);
 RFdata.RFD_int5 = byte(Steering_Mode);
 RFdata.RFD_int6 = int(bearingrate);
 RFdata.RFD_int7 = int(Waypoint_Bearing_From[WPT_index]); //BOD
 RFdata.RFD_int8 = int(Bearing_to_destination_by_LatLon);
 RFdata.RFD_int9 = byte(MSG);
 RFdata.RFD_int10 = bnoCAL_status;
 RFdata.RFD_int11 = int(Wind_Dir);
 RFdata.RFD_int12 = int(Wind_Speed);
 if (Use_CTS == 1) RFdata.RFD_int13 = 0;
 else RFdata.RFD_int13 = int(XTE);

 radio.stopListening();
 radio.powerUp();
 delay(3);
 radio.write(&RFdata, sizeof(RF_DATA));
 radio.startListening();
/*
 RFdata.RFD_set = 2;
 Mode.toCharArray(RFdata.RFD_text,8);
// RFdata.RFD_float1 = XTE;
 RFdata.RFD_float1 = bearingrate;  // temporary displays bearing rate on RF
remote where XTE is programed
 RFdata.RFD_float2 = Bearing_origin_to_destination;
 RFdata.RFD_float3 = Bearing_to_destination;
```

```
 RFdata.RFD_float4 = MSG;
 #if Compass == 1
 RFdata.RFD_float5 = float(bnoCAL_status);
 #endif
 radio.stopListening(); // stop listening so we can send data
 radio.write(&RFdata, sizeof(RF_DATA));
 radio.startListening(); // resume listening
 */
}
/*******************************/
/*
void sendData2()
{
// test data comment out
SOG = 6.2;
XTE = 123;
Bearing_origin_to_destination = 234;
Bearing_to_destination = 245;
Range_Destination = 2.3;
course_to_steer = 241;
// end test data
 RFdata2.RFdata_set = 2;
 RFdata2.SOG = SOG;
 RFdata2.XTE = XTE;
 RFdata2.Bearing_origin_to_destination = Bearing_origin_to_destination;
 RFdata2.Bearing_to_destination = Bearing_to_destination;
 RFdata2.Range_Destination = Range_Destination;
 RFdata2.course_to_steer = course_to_steer;


 radio.stopListening(); // stop listening so we can send data
 radio.write(&RFdata2, sizeof(RF_DATA2));

 radio.startListening(); // resume listening
}
*/
/*********************************************/

void Recv_Data()
{  //Serial.println(" test Radio rec interrupt  ");
  //delay(5);
       if ( radio.available())
  {
   // Serial.println ("radio Available");
    radio.read( &KeyIn2, sizeof(KeyIn2) );
    //Serial.print("KeyIn2 = ");  Serial.println(KeyIn2);
    KeyPressed(KeyIn2);
  }
}
/*******************/

#endif
```

Bilaga 27: Kompasskompensering

Kopiera och klistra in i Arduino IDE. Spara som Subs.

```
#if Compass == 0
void Bearing_Rate()
{
// JNE coding bearing rate som används i PID
 //float Kbr = .05; // 1/50 = 1 sekunds utjämning


   bearingrate = ToDeg((Gyro_Vector[2]*cos(roll) +
Gyro_Vector[1]*sin(roll))*cos(pitch) - Gyro_Vector[0]*sin(pitch));
   #if BEARINGRATE_OFFSET == 1
     bearingrate = bearingrate + bearingrate_Offset;
   #endif

   // Serial.print("bearing rate 1 and BR Gyros ");
   //Serial.print(bearingrate);
   //Serial.print(" ");
   // Serial.print("Bearing Rate smoothed, ");
   // Serial.println(bearingrate_smoothed);

}  // void Bearing_Rate
#endif
/*********************  COMPASS CORRECTION
*********************************/

void AP_Compass_Correction()
{
#if Compass == 0
  MAG_Heading_Degrees = ToDeg(MAG_Heading);  // nedan denna rad kopierat av
JNE
  // AVG_Heading = 0.9*AVG_Heading + .1*MAG_Heading ; // low pass filter
   heading = ToDeg(yaw);
   // heading = MAG_Heading_Degrees;  using direct compass heading
   //heading = ToDeg(AVG_Heading);  // using low pass filtered compass
heading
#endif

   Magnetic_Variation = MagVar_default;
   if (GPRMC_fix) Magnetic_Variation = MagVar;

  // Magnetic_Variation = 0;  // use this to read magnetic heading for
calibration etc.

   heading = heading + Magnetic_Variation;
   if(heading < 0) heading = 360 + heading;
   if(heading > 360) heading = heading -360;

  //  Compass_Calibration();
     if(Screen == 0){
       lcd.setCursor(0, 1);
       lcd.print("HDG    ");
       lcd.setCursor(4, 1);
       lcd.print(heading,0);
     } // end if screen = 0
```

```
/*
 void Compass_Calibration() //Compass Calibration added by Jack Edwards
{
  float compass_correction;
//two approaches interpolation of error and curve fit to error

//INTERPOLATION
  float Compass_CAL[25] = {.1,-.6,-1.2,-1.6,-2,-2.3,-2.3,-2.7,-2.0,-2.0,-
1.1,-.5,-.3,-.1,0,.6,-.2,.2,-.3,-.5,-.4,.1,-.1,-.4,-.4};
 index = (unsigned int)heading/15;  //integer part of heading/15
   compass_correction = Compass_CAL[index] + (heading/15 -
index)*(Compass_CAL[index +1] -Compass_CAL[index]);

 // CURVE FIT TO ERROR curent best fit is two intersecting straight lines
use excel to plot error and get curve fits

       /*if (heading <=79.2)
       {
        compass_correction =-.054*heading - .3298;
       }
        else
       {
         compass_correction =.0177*heading - 6.0007;
       }
       */

 /*
  heading= heading + compass_correction;
  if(heading < 0) heading = 360 + heading; //already a minus, convert to 0-
360
  if(heading>360) heading = heading -360;  //these corrections need if
calibration result runs over or under 360
  */
          /*
          lcd.setCursor(0,2);
          lcd.print("Correction");
          lcd.setCursor(12,2);
          lcd.print(compass_correction);
          */
  //}  // End Compass Calibration

}  // End AP_Compass_Correction()
  /*****************************/
```

Bilaga 28: Pololu vektor

Kopiera och klistra in i Arduino IDE. Spara som Vector.

```
/*

MinIMU-9-Arduino-AHRS
Pololu MinIMU-9 + Arduino AHRS (Attitude and Heading Reference System)

Copyright (c) 2011 Pololu Corporation.
http://www.pololu.com/

MinIMU-9-Arduino-AHRS is based on sf9domahrs by Doug Weibel and Jose Julio:
http://code.google.com/p/sf9domahrs/

sf9domahrs is based on ArduIMU v1.5 by Jordi Munoz and William Premerlani,
Jose
Julio and Doug Weibel:
http://code.google.com/p/ardu-imu/

MinIMU-9-Arduino-AHRS is free software: you can redistribute it and/or
modify it
under the terms of the GNU Lesser General Public License as published by
the
Free Software Foundation, either version 3 of the License, or (at your
option)
any later version.

MinIMU-9-Arduino-AHRS is distributed in the hope that it will be useful,
but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License
for
more details.

You should have received a copy of the GNU Lesser General Public License
along
with MinIMU-9-Arduino-AHRS. If not, see <http://www.gnu.org/licenses/>.

*/

//Computes the dot product of two vectors
float Vector_Dot_Product(float vector1[3],float vector2[3])
{
  float op=0;

  for(int c=0; c<3; c++)
  {
  op+=vector1[c]*vector2[c];
  }

  return op;
}

//Computes the cross product of two vectors
void Vector_Cross_Product(float vectorOut[3], float v1[3],float v2[3])
{
  vectorOut[0]= (v1[1]*v2[2]) - (v1[2]*v2[1]);
```

```
  vectorOut[1]= (v1[2]*v2[0]) - (v1[0]*v2[2]);
  vectorOut[2]= (v1[0]*v2[1]) - (v1[1]*v2[0]);
}

//Multiply the vector by a scalar.
void Vector_Scale(float vectorOut[3],float vectorIn[3], float scale2)
{
  for(int c=0; c<3; c++)
  {
   vectorOut[c]=vectorIn[c]*scale2;
  }
}

void Vector_Add(float vectorOut[3],float vectorIn1[3], float vectorIn2[3])
{
  for(int c=0; c<3; c++)
  {
     vectorOut[c]=vectorIn1[c]+vectorIn2[c];
  }
}
```

Bilaga 29: Kompensation för vind

Kopiera och klistra in i Arduino IDE. Spara som Wind.

```
#if Wind_Input == 1

void get_Wind()
{
const int print_wind = 0;
float Kwind = .1; // 1 all new data (unfiltered) .2 (5 samples smoothing)
Look into Kalman filter
if (count_b > wind_buffer_length -1) Reset_wind_buffer();
if (SoftSerial1.available() > 0){
    byteWind = SoftSerial1.read();
    if(byteWind !=13)
    {
      Wind_buffer[count_b] = byteWind;
      if (print_wind)Serial.write(byteWind);
      if(byteWind>127)
        {
          Reset_wind_buffer();
          return;

        }
      count_b++;
    }   // if byteGPS != 13
    else
    {
       Wind_buffer[count_b] = '\0';
       // Clear rest of buffer
       for (int i=count_b+1;i<wind_buffer_length;i++)
       {
         Wind_buffer[i]=' ';
         count_b = 0;
       }
       Windheader = "";
       for (int i=2;i<7;i++)
       {
         Windheader = Windheader + Wind_buffer[i];
       } // end for i= 2,7
         // Serial.print("Header = ");
Serial.println(Windheader);
       if (Windheader == "WIMWV")
       {
         Get_WIMWV();
         Wind_MAX = max(Wind_Speed,Wind_MAX);
         if(!sw2) Wind_MAX = 0;
         Wind_Avg = (1-Kwind) * Wind_Avg + Kwind * Wind_Dir; //vind
uppdatering 1 Hz Kwind = .2 borde vara ca 5 sek i medeltal
           //  Serial.print(Wind_Dir); Serial.print(" ");
Serial.println(Wind_Avg);
           //  Wind_Differential();
         Reset_wind_buffer();
         return;
       }
       if (Windheader == "SDDBT")
       {
         Get_SDDBT();
         Reset_wind_buffer();
```

```
       }
    } // end else
 }  // if (SoftSerial1.available()
} // void get_Wind2

void Reset_wind_buffer()
{
 for (int i=0;i<wind_buffer_length;i++)
 {
   Wind_buffer[i]=' ';
 }
 count_b = 0;
}

/*************************************************/

void Get_WIMWV()
   {
    const int print_MWV = 0;
    j_MAX = 5;
    Parse_Wind ();
    //Serial.print("Checksum Status "); Serial.println(checksum_status);
    if(checksum_status){
    for(int j = 0; j<j_MAX; j++)
     {
      data_MWV[j] = data_IN[j];
     }
    string1 = data_MWV[1];
    NMEA_TO_FLOAT(1);
    Wind_Dir = float3;
     //Serial.print("wind Dir in WIMWV ");
Serial.println(Wind_Dir,0);
        // data_MWV[2], R or T Relative or True
        // data_MWV[3], Wind Speed
    string1 = data_MWV[3];
    NMEA_TO_FLOAT(1);
    Wind_Speed = float3;
        // data_MWV[4], Wind_Speed Units K/M/N N = knots
      if(print_MWV) {PRINT_MWV();}
    } // end if checksum_status true
    //PRINT_MWV();
   }  //end of void WIMWV() case

  /*********************************************************************/

void Get_SDDBT() //Depth
{
   j_MAX = 2;
   Parse_Wind();
   if(checksum_status){
     string1 = data_IN[1];
      NMEA_TO_FLOAT(1);
      Depth = float3;
      //Serial.print("Depth "); Serial.print(Depth,1);
   }
} // End get SDDBT
  /*********************************************************************/

void Parse_Wind()
{
  char comma = ',';
```

```
      char star = '*';
      int last_k = 0;
      int k_start = 0;
      //Word_count = 0;
        Checksum_wind();
        //Serial.print("checksum status "); Serial.println(checksum_status);
          if (checksum_status)
         {

       for (int j=0; j<j_MAX + 1; j++)
       {data_IN[j] = "";
       }
                 for (int j=0; j<j_MAX; j++)
                 {
                        for(int k = k_start; k < 80; k++)
                        {
                               data_IN[j] = data_IN[j] + Wind_buffer[k];
                               last_k = k;
                               if (Wind_buffer[k] == comma || Wind_buffer[k]
== star )
                                {
                                  data_IN[j] =
data_IN[j].substring(0,data_IN[j].length() - 1);
                                 // Word_count = Word_count +1;
                                  break;
                                }
                        }  // end for k

                         k_start = last_k +1;
                 }// end for j
       } // end for i
}  // end Parse_Wind


  /*************************  PRINT MWV  ******************************/

   void PRINT_MWV()
 {
    // Serial.println();
    Serial.println("---------------");

            Serial.print("Header: ");
                 Serial.println(data_MWV[0]);

            Serial.print("Wind Bearing: ");
                 Serial.println(Wind_Dir);

            Serial.print("Wind R or T ");
                 Serial.println(data_MWV[2]);

            Serial.print("Wind Speed: ");
                 Serial.println(Wind_Speed);

             Serial.print("Speed Units ");
                 Serial.println(data_MWV[4]);

             Serial.print("Depth ");
                 Serial.println(Depth);

             Serial.println();
   }
```

```
/*************************************************************/

  void Checksum_wind(){
       /*******************************************
        From garmin.com search support for "how is checksum calculated in
NMEA 0183
        The checksum is the 8-bit exclusive OR (no start or stop bits) of
all characters in the sentence, including the "," delimiters,
        between -- but not including -- the "$" and "*" delimiters. The
hexadecimal value of the most significant and least significant
        4 bits of the result are converted to two ASCII characters (0-9, A-
F) for transmission. The most significant character is transmitted first.
        Therefore in the routine below the counter starts at 1 to skip "$"
in  checksum routine and ends at index of "*".
      *******************************************/
            int index=0;
                                //  Serial.println("");
    // diagnostic may be commented out
                                //  Serial.print("DATA TO BE CHECK
SUMMED  ");   // diagnostic may be commented out
            checksum=0;
            for(int x=1; x<100; x++){  // you have to skip the $ sign and
it works if x starts at 1
                if (Wind_buffer[x] == '$'){
                                     index=x;}
                                   break;}
                                  //   Serial.print("index = ");  //
diagnostic may be commented out
                                    //  Serial.println(index);  //
diagnostic may be commented out
                                 for(int x=index+1; x<100; x++){
                                     if(Wind_buffer[x]=='*'){
          checksum_received = strtol(&Wind_buffer[x + 1], NULL,
16);//Parsing received checksum...
    // Serial.print("checksum_received =
");Serial.println(checksum_received);
                                       break;
          }else{
          checksum ^= Wind_buffer[x]; //XOR the received data...
                                      // Serial.print(gps_buffer[x]);  //
this should = gps_buffer less $ in and * at end which is data to checksum
          }}
   //Serial.print("checksum = "); Serial.println(checksum);
          checksum_status = false;
          if(checksum_received == checksum){
             checksum_status = true;
   //Serial.print("checksum status ");
Serial.println(checksum_status);

        }
  }// End of Checksum subroutine
 /*
  void Wind_Differential()
  {
   static float delta_Wind, Wind_DeltaT, windrate_smoothed;
    static float Wind_old, windtime_old;
    const float Kwr = .3;

    delta_Wind = Wind_Dir - Wind_old;
```

```
    if (abs(delta_Wind) > 180)
      {
         if(Wind_old < Wind_Dir) delta_Wind = delta_Wind - 360;
         if(Wind_old > Wind_Dir) delta_Wind = 360 + delta_Wind;
      }
  Wind_DeltaT = float((millis()-windtime_old)/1000.0);
  windrate = delta_Wind /Wind_DeltaT;
  windrate_smoothed = (1-Kwr) * windrate_smoothed + Kwr * windrate; //
updates 50/sec, 3000/min, 10,000 = 3.3 min avg.
  windrate =  windrate_smoothed;
  if (Screen == 1 && Steering_Mode == 4)
  {
    lcd.setCursor(11, 2);
    lcd.print("WRT      ");
    lcd.setCursor(15, 2);
    lcd.print(windrate);
  }
   windtime_old = millis();
   Wind_old = Wind_Dir;
 if (counter2 >5){ Serial.println(); Serial.print("Wind rate and bearing
rate, "); Serial.print(windrate); Serial.print(" ");
Serial.println(bearingrate);}
  } // end wind differential
*/
  #endif
```

Bilaga 30: Pololu matris

Kopiera och klistra in i Arduino IDE. Spara som Matrix.

```
/*

MinIMU-9-Arduino-AHRS
Pololu MinIMU-9 + Arduino AHRS (Attitude and Heading Reference System)

Copyright (c) 2011 Pololu Corporation.
http://www.pololu.com/

MinIMU-9-Arduino-AHRS is based on sf9domahrs by Doug Weibel and Jose Julio:
http://code.google.com/p/sf9domahrs/

sf9domahrs is based on ArduIMU v1.5 by Jordi Munoz and William Premerlani,
Jose
Julio and Doug Weibel:
http://code.google.com/p/ardu-imu/

MinIMU-9-Arduino-AHRS is free software: you can redistribute it and/or
modify it
under the terms of the GNU Lesser General Public License as published by
the
Free Software Foundation, either version 3 of the License, or (at your
option)
any later version.

MinIMU-9-Arduino-AHRS is distributed in the hope that it will be useful,
but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License
for
more details.

You should have received a copy of the GNU Lesser General Public License
along
with MinIMU-9-Arduino-AHRS. If not, see <http://www.gnu.org/licenses/>.

*/

/*************************************************/
//Multiply two 3x3 matrixs. This function developed by Jordi can be easily
adapted to multiple n*n matrix's. (Pero me da flojera!).
void Matrix_Multiply(float a[3][3], float b[3][3],float mat[3][3])
{
  float op[3];
  for(int x=0; x<3; x++)
  {
    for(int y=0; y<3; y++)
    {
      for(int w=0; w<3; w++)
      {
       op[w]=a[x][w]*b[w][y];
      }
      mat[x][y]=0;
      mat[x][y]=op[0]+op[1]+op[2];
```

```
      float test=mat[x][y];
    }
  }
}
```

Bilaga 31: Kalibrering av kompass

Kopiera och klistra in i Arduino IDE. Spara som Kompass i en egen mapp med samma namn tillsammans med bilaga 32.

```
/*  COMMENTS
För kalibrering av kompass
 */
#include <Wire.h>
#include <LSM303.h>
#include <LiquidCrystal.h>
LiquidCrystal lcd(41,43,45,47,49,39);

//#include <Key.h>
#include <Keypad.h>
 //  KEYPAD SETUP
const byte ROWS = 4;
const byte COLS = 3;
char keys[ROWS][COLS] = {
  {'1','2','3'},
  {'4','5','6'},
  {'7','8','9'},
  {'*','0','#'}
};
byte rowPins[ROWS] = {25, 35, 33, 29};
byte colPins[COLS] = {27, 23, 31};
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

LSM303 compass;
LSM303::vector<int16_t> running_min = {32767, 32767, 32767}, running_max =
{-32768, -32768, -32768};
boolean serial_Print_rawdata=0;
boolean LCD_Print_rawdata = 0;
boolean serial_Print_maxmin=0;
boolean LCD_Print_maxmin = 0; // this prints the maximum and minimums
captured
boolean serial_Print_drift =0;
boolean CompassCalibration = 1; // captures avg compass.m .x .y .z and
prints to serial monitor along with reference heading
float smooth_fraction = .01; // .01 means 100 point low pass filter
float Xsmooth;
float Ysmooth;
float Zsmooth;
long timer;
long timer1;
long timer2;
long timer3;
long timer4;
long counter;
String inHeading = "";
long Heading;
boolean newHeading = 0;
boolean juststarted = 1;
float XmTotal = 0;
float YmTotal = 0;
float ZmTotal = 0;
float XmAvg = 0;
float YmAvg = 0;
```

```
float ZmAvg = 0;
int n = 0;
void setup() {
  Serial.begin(57600);
  Wire.begin();
  compass.init();
  compass.enableDefault();
  lcd.begin(20,4);
//  keypad.addEventListener(keypadEvent); //add an event listener for this
keypad

}
char key = 0;

void loop() {

  compass.read();

  running_min.x = min(running_min.x, compass.m.x);
  running_min.y = min(running_min.y, compass.m.y);
  running_min.z = min(running_min.z, compass.m.z);

  running_max.x = max(running_max.x, compass.m.x);
  running_max.y = max(running_max.y, compass.m.y);
  running_max.z = max(running_max.z, compass.m.z);

  Xsmooth =(1-smooth_fraction) * Xsmooth + smooth_fraction * compass.m.x;
  Ysmooth =(1-smooth_fraction) * Ysmooth + smooth_fraction * compass.m.y;
  Zsmooth =(1-smooth_fraction) * Zsmooth + smooth_fraction * compass.m.z;



  if(CompassCalibration){

         if(juststarted){
         Serial.println("Average Magnetometer Readings");
         Serial.println("Heading,  XmAvg,  YmAvg,  ZmAvg");
         juststarted = 0;
        }

      newHeading = 0;
      lcd.begin(20,4);  // clears the screen
      lcd.setCursor(0,0);
      lcd.print("Enter Heading or #");
      lcd.setCursor(0,1);
        while(newHeading !=1){
          KEYPAD();
        }
      Serial.print(Heading);
      for(int avgCounter =0; avgCounter<1000; avgCounter++){
        compass.read();
        XmTotal = XmTotal + compass.m.x;
        YmTotal = YmTotal + compass.m.y;
        ZmTotal = ZmTotal + compass.m.z;
        n = avgCounter;
      }
     XmAvg = XmTotal/n;
     YmAvg = YmTotal/n;
     ZmAvg = ZmTotal/n;
```

```
     Serial.print(", "); Serial.print(XmAvg,1); Serial.print(", ");
Serial.print(YmAvg,1); Serial.print(", "); Serial.println(ZmAvg,1);
     XmTotal = YmTotal = ZmTotal = 0;
     XmAvg = YmAvg = ZmAvg = 0;
      }

  if (serial_Print_maxmin)
  {
   // if(millis() - timer1 > 100) {

      timer1 = millis();
  Serial.print("M min ");
  Serial.print("X: ");
  Serial.print((int)running_min.x);
  Serial.print(" Y: ");
  Serial.print((int)running_min.y);
  Serial.print(" Z: ");
  Serial.print((int)running_min.z);

  Serial.print(" M max ");
  Serial.print("X: ");
  Serial.print((int)running_max.x);
  Serial.print(" Y: ");
  Serial.print((int)running_max.y);
  Serial.print(" Z: ");
  Serial.println((int)running_max.z);
 // }
  }

  if( serial_Print_rawdata)
  {
    if(millis() - timer2 >500)
  {
    timer2=millis();
  Serial.print(compass.m.x); Serial.print(", ");
  Serial.print(compass.m.y); Serial.print(", ");
  Serial.print(compass.m.z); Serial.println();
  }
  }
  //This block prints Max and Mins
  if (LCD_Print_maxmin)
  {
      lcd.setCursor(0,0);
      lcd.print ("XMIN   YMIN   ZMIN  ");
      lcd.setCursor(0,1); lcd.print("                    ");
      lcd.setCursor(0,1);  lcd.print((int)running_min.x);
      lcd.setCursor(7,1); lcd.print((int)running_min.y);
      lcd.setCursor(14,1); lcd.print((int)running_min.z);
      lcd.setCursor(0,2);

      lcd.print ("XMAX   YMAX   ZMAX  ");
      lcd.setCursor(0,3); lcd.print("                    ");
      lcd.setCursor(0,3);  lcd.print((int)running_max.x);
      lcd.setCursor(7,3); lcd.print((int)running_max.y);
      lcd.setCursor(14,3); lcd.print((int)running_max.z);
  }


     //This block prints raw compass readings
      if(LCD_Print_rawdata)
     {
```

```
     if(millis() - timer >1000)
    {
      timer = millis();
     lcd.setCursor(0,0);
     lcd.print(Xsmooth);
     lcd.setCursor(0,1);
     lcd.print(Ysmooth);
     lcd.setCursor(0,2);
     lcd.print(Zsmooth);
    }
     }

  // print every 10 minutes to watch drift

 if (serial_Print_drift)
 {
  if (millis()/1000 - timer3 >= 600) //1 min = 60
 {
   timer3 =  millis()/1000;
   counter += 1;
   Serial.print(counter);
   Serial.print(", ");
   Serial.print(Xsmooth);
   Serial.print (", ");
   Serial.print(Ysmooth);
   Serial.print (", ");
   Serial.print(Zsmooth);
   Serial.println();
 }
 }
}
```

Bilaga 32: Knappsats till kompasskalibrering

Kopiera och klistra in i Arduino IDE. Spara som Knappsats i en egen mapp tillsammans med bilaga 31.

```
void KEYPAD(){
 char key = keypad.getKey();
   if (key != NO_KEY){
   //Serial.println(key);
   lcd.print(key);

     if (key == '#'){
      newHeading = 1;
      Heading = inHeading.toInt();
     // Heading = 2 * Heading;
      // Serial.print(Heading);
      inHeading = "";
      return;
     }

    inHeading = inHeading + key;
   // Serial.print(Heading);
   // Heading = inHeading.toInt();
   // Heading = 2 * Heading;
   // Serial.println (Heading);
   // lcd.print(key);
  } // end if (key != NO_KEY)

}  // end KEYPAD()

/*******************************************/
/*
 void KEYPAD()
{
  char key = keypad.getKey();

           // if (key != NO_KEY) {   }
           //delay(100);
     //  Serial.print(" getket = "); Serial.println(key);
}

/*********************************************/

/*void keypadEvent(KeypadEvent key){
  switch (keypad.getState()){

    case PRESSED:
     KeyPressed(key);
    break;

    case RELEASED:
     KeyReleased(key);
    break;

    case HOLD:
     KeyHeld(key);
    break;
```

```
    // }  // end switch (key)
    } // end switch(keypad.getstate)
  }  //  end void keypadEvent(KeypadEvent key)

/*******************************************************/

  /*void KeyPressed(char keyin)
{


} // End void KeyPressed

/*****************************************/

/*void KeyReleased (char keyin)
{
   switch (keyin){

        case '*':

        break;

        case '#':

        break;

      }  // end swtich key

} // end key released

/***********************************************/
/*
void KeyHeld(char keyin){
 switch (keyin){
  case '#':
  break;
   }  // end switch (keyin)
}  // end KeyHeld
*/
```