CHALMERS
UNIVERSITY OF TECHNOLOGY

Automatisk styrning via GPS-rutt

Kandidatarbete inom elektroteknik
Av
Hampus Lidén Martinsson och Per Harring
TIELL 2020

Examinator: Tommy Svensson, Elektroteknik
Handledare: Fredrik Hellström, Elektroteknik

Förord

Detta projekt är ett avslutande moment för kandidatexamen på elektroingenjörsprogrammet på Chalmers Tekniska Högskola. Projektet utfördes på företaget Consat Engineering AB.

Vi vill tacka företaget Consat för all deras hjälp med utrustning och kunskap. Utan Consat och vår handledare Jonas Williamsson hade projektet aldrig varit möjligt. Consat har alltid varit tillmötesgående och villiga att hjälpa.

Vi vill också tacka alla medarbetare på kontoret som alltid gjorde vår miljö rolig, trivsam och lärorik. Trots att Coronakrisen gjorde denna tiden alldeles för kort var detta en mycket rolig upplevelse.

Sist men inte minst vill vi tacka vår handledare och vår examinator på Chalmers Fredrik Hellström och Tommy Svensson för all hjälp med att skriva denna rapport.

Hampus Lidén Martinsson och Per Harring, Göteborg, Maj 2020

Abstract

The projects goal was to record and then drive a route automatically using a radiocontrolled car as a prototype, by utilizing a RTK GPS System and Raspberry Pi. The record and play software are contained within the Raspberry in C code. The GPS-system samples positions that are sent to the Raspberry Serial port for processing. All of the settings and GUIs are taken into account within the report. The report also contains further information on the Raspberry libraries needed for the Serial interface and how to setup a cross-compilation workstation using Windows and Visual Studio. The RC car model is Traxxas E-maxx with a brushless electric engine and its standard radio controller. The control signal is divided into two different parts using a switch, where the controller is one and the autosignal is another. The autosignal is generated using an Adafruit PWM module with basic configuration. The data is generated using the car software within the Raspberry. The final result was a non-working prototype that needs more work. Since the the project did not make use of a compass, the car encountered issues because it does not know the way it was headed. If the project had more time this would be the easiest way to complete it.

Sammanfattning

Projektets mål är en prototyp i formen av en radiostyrda bil, spela in och spela upp olika GPS-rutter. Prototypen använder sig dels av ett RTK GPS-System men också en Raspberry. Raspberryn är huvudstationen där all mjukvara och information behandlas. GPS-systemet sänder sina koordinater genom USB-porten till Raspberryn för användning. GPS-systemet består av en bas vilket är en fast punkt på kartan och en rörlig modul som sitter på bilen. Inom rapporten visas och förklaras alla inställningar samt hur systemet ska ställas upp. Rapporten innehåller också mer information om Raspberryn, dess användning, bibliotek och vad som krävs för att skapa en arbetsstation för smidig mjukvaruutveckling. Projektet använder korskompilering på en Windows plattform i Visual Studio. Radiostyrda bilens modell är Traxxas E-maxx med dess orginalkontroller och en borstlös elektrisk motor. Styrsignalen till bilen är uppdelad mekaniskt genom en flipswitch där ena läget är automatisk styrning och det andra manuell styrning. Den automatiska signalen genereras av en Adafruit PWM-modul som i sin tur styrs direkt av Raspberryn. Raspberryn utgör centralenheten och används för att bearbeta koordinaterna i form av position och destination, samt att köra bilen åt riktingen den ska. Det slutgiltiga resultatet var en icke fungerande prototyp inom den existerande tidsramen. Mjukvaran stötte på stora problem när den kördes då den inte har någon sensor för vilken riktning bilen är påväg. Med ytterligare tid hade lösningarna som framtagits implementerats, så som en kompass för riktning eller andra metoder.

Innehållsförteckning

1 Inledning

1.1 Bakgrund

Examensarbetet utfördes på företaget Consat Engineering AB. Projektet togs fram av gruppen själva och är inte av företagets konstruktion. Consat tycker att detta är ett roligt projekt där kunskaper och arbetsvilja kan uppvisas. Företaget har tidigare jobbat med projekt som är likt detta och kan verkligen stötta oss i examensarbetet.

Idén grundas i att göra jordbruksarbete enklare och effektivare. Under arbete på en gård krävs det att man kör traktorer med olika verktyg, vilket gör att bönder antingen behöver byta traktor till en annan med rätt verktyg eller att en annan arbetare kör efter med den andra traktorn. Istället för detta är idén att andra traktorn kör automatiskt med en sparad GPS-rutt från den första traktorn.

1.2 Syfte

Rapporten avhandlar att få en radiostyrd bil att härma en redan körd rutt av en annan radiostyrd bil genom att spara ner en rutt från ett GPS-system som placerats på den radiostyrda bilen. Detta fungerar genom att använda en Raspberry Pi 3 model B+ (RPi) och en pulse width modulation (PWM)-modul utöver GPS-systemet, vilket har monterats på bilen för att kunna vara helt automatiskt styrd. Den krets som monteras på bilen är driven av en powerbank.

1.3 Avgränsningar

Arbetet infattar inte följande:

- Riktiga traktorer
- Fungerande gränssnitt
- Realtidsuppspelning/-inspelning
- Justering av rutter
- Hastighetsändringar

1.4 Precisering av frågeställning

- Hur ska bilen styras automatiskt?
- Vad kommer GPS-systemet ha för utsignal och hur ska denna signalen användas?
- Vad ska ingå i elkonstruktionen?
- Hur kan rutterna spelas in?
- Vilken minnesfunktion ska användas för rutten?

2 Teoretisk referensram

2.1 Signalbehandling

Signalbehandling är modifikation, manipulation och transformering av signaler. Detta används nästintill i all teknik nu för tiden på ett eller annat sätt utan att man vet om det. Det kan vara så enkelt att en lampa ska öka gradvis i styrka till att en radar ska fungera korrekt. I en radar sänds det ut en radiovåg som när den träffar ett föremål studsar tillbaka till en mottagare. När signalen är mottagen så behandlas den med matematiska uträkningar och då kan föremålets hastighet och position bestämmas.

2.1.1 Pulsbreddsmodulering

Pulsbreddsmodulering är en teknisk metod för att styra spänning med en följd av 1:or och 0:or i en viss frekvens[1]. Vid pulsbreddsmodulering betyder en 1:a in i systemet att spänning ska flöda i systemet och en 0:a att det är avstängt och detta sker periodiskt. Detta går att likna vid en switch som automatiskt slås på och av vid den frekvens som är inställd. En period för pulsbredden är tiden det tar från att den först varit 1 och 0 innan den blir 1 igen. Denna period är beroende av frekvensen som är vald men hur mycket av denna tid som är 1 och 0 bestäms av duty cycle. Duty cycle är en procentbaserad enhet som anger hur stor andel av en period som är 1. Detta betyder att vid en 100% duty cycle flödar spänning i systemet konstant och vid 50% duty cycle flödar spänning i systemet lika ofta som det inte gör.

När man använder en modul som skapar pulsbreddsmodulering använder den sig av en förutbestämd mängd av 1:or och 0:or som modulen behandlar. Om ett flöde av 1:or och 0:or sänds till modulen väntar modulen på denna mängden innan den behandlar signalen. Om en modul till exempel pulsbreddsmodulerar 8-bitar så väntar den alltså in totalt åtta 1:or och 0:or. Från den binära koden, som till exempel kan vara ”00100111”, skapas en 8-bitars period av en pulsbredd som utsignal. Denna pulsbredd kommer att ha ett heltalsvärde på 0-255 som styr enheten som PWM-modulen är kopplad till. En PWM-modul kan ha flera utsignaler och styra flera enheter samtidigt. Den använder dessa 8-bitars perioder och lägger dem på rätt pinnar ut från modulen. Allt detta är programmerbart genom addressbitar som kommer med insignaler till modulen.

3 Metod

3.1 Radiostyrda bilen

Bilen som användes i arbetet är en Traxxas E-maxx[2]. För arbetet så behövdes en tillräckligt stor bil för att kunna montera alla tekniska komponenter på. För att alla skulle rymmas behövdes en bil på ungefär en halvmeter. Apparaterna inkluderar: RPi, kretskort, powerbank, switchar och GPS-modul. Bilens mottagare som tar emot signaler från kontrollenheten är en Traxxas TQ och den kommunicerar med mottagaren på frekvensen 2.4 GHz. Bilen har två stycken 7-cell batteripaket av sorten NimH med 3700 mAh. Bilen har två servoenheter som styrs individuellt från mottagaren till vardera framhjul. En sista koppling från mottagaren finns till motorn VXL-6s brushless.

(a) Bil med skydd.

(b) Bil utan skydd.

Figur 1: Traxxas E-maxx

3.2 GPS-Systemet

Det GPS-system som användes består av två komponenter: basstationen Emlid Reach RS+ (RRS)[3] och RTK GNSS modulen Reach M+ (RM)[4]. Utöver dessa två enheter har utvecklarna Emlid en Graphical User Interface (GUI) (se figur 8) som kopplas till för de två enheterna. För att koppla sig till GUI så måste antingen enheterna vara kopplade till samma WiFi eller om de inte blir kopplade till något WiFi så startar enheterna sina egna hotspots. Man kan därefter antingen koppla en dator till detta hotspot genom att använda

en IP-adress eller genom att ladda ner en app på sin telefon från skaparna av enheterna. I den här GUI:n kan en rad inställningar göras för att konfigurera GPS-systemet som man själv vill ha det.

GPS-systemet använder sig av RTK-teknik, alltså Real-time kinematic (RTK) vilket betyder att systemet använder sig av korrigerande insignaler från en baspunkt till målet. Detta gör att den resulterande positionen kan variera med så lite som några millimeter. Systemet använder sig av satellitens egna bärvåg för att kalkylera positionen, eller närmare bestämt fasen på vågen, som jämförs med basen och modulens egna faser i deras individuella signaler. All information som finns inuti satellitens signal ignoreras och på detta sättet kan systemet ta fram avståndet mellan sig själv och satelliten. Med minst tre satelliter inom räckvidd så sker en triangulering för positionen. Efter detta sänder basstationen sina egna fasmätningar till den rörliga modulen som därefter jämför sitt resultat med basen [5]. Datalänken mellan basstationen och modulen kan vara genom Wifi eller radio m.m, där Wifi har den snabbaste länken fast med lägst räckvidd. I detta projektet används radiolänken eftersom det är den företaget rekommenderade ifall bilen inte ska köra snabbare än 50 km/h.

Det är viktigt att både modulen och basstationen har samma satelliter inom räckvidd för att mätningarna ska fungera. Ifall inte basstationen och modulen initieras korrekt sjunker precisionen markant och detta är för att klockcykel bias och atmosfärisk fördröjning m.m inte kan elimineras. Metoden som systemet använder kallas för ”triple difference”. En av de främsta okända variablerna i systemets algoritm, förutom positionen av själva mottagaren (basen eller modulen), är direkt översatt till ”heltals tvetydighet”, vilket är hur många hela cykler det har tagit för satellitens signal att nå mottagaren. För att eliminera detta fel används antingen dubbel eller trippel differens [6, 7].

3.2.1 Emlid Reach RS+

Figur 2: Bild på basstationen Emlid Reach RS+.

När man sätter upp basstationen för användning behöver den räkna ut var den

är. Detta innebär att den ska hållas stationär efter konfigurering, eftersom en ny uträkning annars måste göras. Basstationen använder sig av sju olika satelliter för att få en så bra precision som möjligt på sin position. För uträkningen av positionen så beror precisionen mycket på hur lång tid basstationen kan jobba innan användning och denna tidsram kan väljas själv i konfigureringen och kan ställas in mellan 1-30 minuter. Precisionsäkerheten har tre olika grader:

- Single: precision på ca 2.5 m som kan då sättas upp på mindre än 5 minuter.
- Float: precision på ca 1 m som kan då sättas upp på ungefär 5 minuter.
- Fixed: precision på ca 7 mm som kan då sättas upp på ungefär 15 minuter.

Det är inte bara tiden man låter GPS-signalen stabilisera som kan försämra vilken grad man får av precision. Om det skulle vara något som blockerar signalen mot satelliter runt om så kan även detta göra att man får en lägre precision. Tid för konfigurering ovanför är under perfekta förhållanden. Parkopplingen från basstationen till en modul sker antingen över radiosignal eller mobilt nätverk. Basstationen sitter upphöjd 2 meter över marken på en tripod.

3.2.2 Reach M+

Reach M+ (RM) kopplas ihop med en antenn som har kontakt med basstationen och en powerbank. För att spara modulens position går det att logga under tiden den används och sedan få en fil som är nedladdningsbar. Denna fil kan vara i fyra olika format:

- LLH: latitud, longitud och höjd.
- XYZ: koordinater i XYZ format.
- ENU: koordinater i East, North och UP format.
- NMEA: NMEA är en specifik textfil för position som är standard för många kartor t.ex Google Maps.
- ERB: positionsystem för programmet Árdupilot.

Om man vill undvika att använda loggformatet kan man även överföra position seriellt direkt över USB.

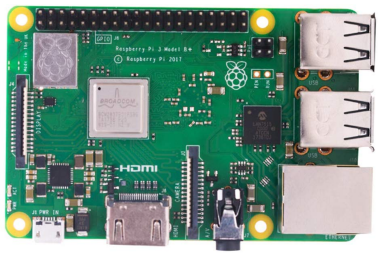
Figur 3: Bild på modulen Reach M+.

3.3 Radiostyrda bilens kontrollsystem och dess utveckling

I denna del förklaras kort om hur en RPi fungerar och vad det är för något. Därefter förklaras vilken mjukvara som använts för att korskompilera till RPi.

3.3.1 Raspberry Pi:n

För att utveckla radiostyrda bilens system som ska integrera GPS-systemets koordinater till en kontrollsignal krävdes först en arbetsstation. Raspberryn vi använder heter RPi och fungerar som centralenheten för alla beräkningar.



Figur 4: Raspberry Pi 3 Model B+

RPi:n är utrustad med många möjligheter för olika in- och ut-portar i olika former (se datablad). De portar som används i detta projekt är USB, GPIO pinnarna och WAN-porten. USB-portarna är självförklarande och berörs inte närmare i detta avsnitt. GPIO-pinnarna för Pi:n har en standard-layout som är i princip samma för alla modeller.

I figur 5 syns hela layouten för Pi:n som används. Lägg även märke till att

Figur 5: Raspberryns GPIO-Layout som har tagits fram i RPi:ns konsolfönster.

layouten hittas genom kommandot ”pinout” i Raspbians konsolfönster. Varje Raspberry har detta kommando som lätt kan visa vad layouten är och på bilden syns pinnarnas namn och numrering. Biblioteket till Raspberryn använder sig av antingen namnen eller numreringen, vilket syns nedan. Projektet använder sig av en PWM-modul trots att Pi:n har designerade pinnar för just den funktionen (t.ex GPIO13). Detta är för att PWM-pinnarna på Pi:n inte är lika flexibla som ett dedikerat kort är. De har dessutom svårt att tillföra tillräckligt med ström snabbt nog för att styra bilen på ett effektivt sätt. Det finns dessutom en risk att man bränner kortet ifall en kortslutning eller plötsliga störningar skulle ske på pinnarna. Då är det bättre ifall ett billigare kretskort går sönder istället för hela centralenheten.

Det smidigaste sättet att använda sig av Raspberryns olika pinnar och funktioner är genom wiringPi biblioteket[8]. Biblioteket kan lätt konfigurera alla pinnar och portar som ska användas. Den namnger dessutom alla pinnar antingen till GPIO-namnen i bilden eller enligt numreringen inom parenteser. Med hjälp av biblioteket krävs endast ett par enstaka kommandon för att sätta pinnar till ut- eller inportar. Det finns också funktioner för att lyssna på olika USB-portar och denna delen kallas för wiringSerial och är lite speciell. Eftersom kommunikation är seriell krävs lite annorlunda logik än vad de andra GPIO portarna har. Det krävs nämligen en ny iteration för varje byte av data som ska läsas in m.m. Hur detta arbete hanterade den seriella kopplingen förklaras närmare under kodkapitlet.

3.3.2 Visual Studio och VisualGDB

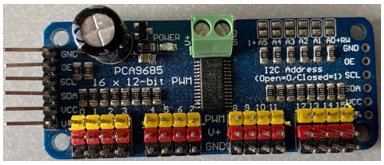
All kod skrivs i utvecklingsmiljön Visual Studio på Windows-datorer. Visual Studio är en omfattande arbetsmiljö som framförallt kommer med extra tillägg för att korskompilera till Raspberry.

VisualGDB är ett tillägg som används för att med hjälp av en inbyggd SSH-klient kunna debugga Raspberryn i realtid, annars hade debuggning ej varit möjligt. Tilläggets hemsida innehåller också omfattande guider om hur allt skall installeras för att det ska fungera felfritt. Guiderna innehåller bland annat hur man ställer in diverse inställningar till klienten så att den kan ansluta. Därefter laddar SSH-klienten i VisualGDB upp programmet på Raspberryn genom en nätverkskabel, vilket antingen går direkt mellan datorerna eller via en router. Detta sker automatiskt varje gång programmet kompileras och körs. Raspberryns konsolfönster går också att nå direkt ifrån klienten på arbetsstaionen så att en andra skärm till PI:n ej är nödvändig. Eftersom konsolen går att nå kan alla inställningar på RPi:n göras. Tillägget är förvånansvärt robust och lätt att ställa in för att börja programmera och feltesta.

3.3.3 PCA9685

Efter undersökning stod det klart att PWM-modulen som var det bästa alternativet för arbetet var PCA9685. Tanken var att bara beställa själva PWM-modulen direkt men detta var precis i början av Coronapandemin. Beställningstiden för denna modul var på över två månader redan då. Detta gjorde att vi valde att beställa ett färdigt kretskort (se figur 6) med modulen på eftersom detta fanns nära Göteborg och hade en leveranstid på en vecka. Detta gjorde att tester och mätningar kunde börja snabbt.

PCA9685 är ett kretskort med 16 kanaler som styrs via I2C med 12 bitar på varje pinne. Kortet kan använda en frekvens mellan 24 Hz och 1526 Hz med en programmerbar duty cycle. Frekvensen som programmeras är för alla pinnar och går ej att ställa in individuellt. Kortet har även adresser för att kunna parkoppla flera kretskort för mer styrning i samma krets.



Figur 6: PCA9685

Insignaler till kortet består av GND (jord), OE (om 0 så fungerar utsignaler och om 1 så fungerar ej utsignaler), SCL (klocka), SDA (data in), VCC (strömtillförsel)

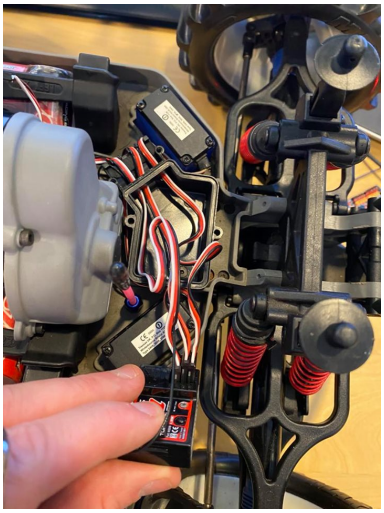
och V+ (om extra strömtillförsel behövs kopplas det här). I detta arbete används inte OE eller V+. För att programmera kortet i C-kod användes ett bibliotek som förklaras i senare kapitel[9].

4 Resultat och genomförande

Inledningsvis behandlar denna del en genomgång av mätningar och vidare följer även en sektion om hur programmeringen av RPi gick tillväga. I slutet av detta kapitel finns ett avsnitt som beskriver resultaten för slutprodukten.

4.1 Mätningar

I denna del visas de mätningarna som gjordes på bilen och hur resultaten från detta såg ut. Mätningarna som genomfördes är väldigt viktiga för arbetets gång då de visar vilka signaler som gör att bilen kan köra alls. För att förstå hur bilen ska köras utan kontroller så måste det finnas kunskap om hur den kör med kontroll.



Figur 7: Bild på Traxxas E-maxx mottagarenhet och kablar.

4.1.1 Kontrollenhet

Mätningar började på en kontrollenhet tillhörande en radiostyrd båt. Detta var då bilen ej hade levererats ännu och en kontrollenhet fanns på plats med snarlik teknik. Vid detta tillfälle fanns det även en tanke att modifiera kontrollenheten i arbetet. Mätningarna utfördes som ett test för djupare kunskap och ingen mätdata sparades.

4.1.2 Mottagare

När bilen anlände till kontoret så påbörjades mätningar direkt på de distribuerade signalerna till drivmotorn och de två servomotorerna. Det är tre kablar som går från varje port på mottagaren och det är tre portar där en port är till motorn och en port till vardera servo. De vita kablarna är utsignal från mottagaren som styr enheterna och de svarta kablarna är jord. Röda kabeln som är kopplad mellan motorn och mottagaren är strömtillförsel till mottagaren och servon genom resterande två röda kablar.

Hjulen fram är kopplade till de två servona och hjulen kan svänga max 45 grader åt vardera håll. Mätningar började på de vita kablarna till servona:

Sväng	Bitar
Rakt fram	300
Max höger	160
Max vänster	440

Tabell 1: Mätningsresultat av styrsignal till servona.

Mätningen visade även att spänningen till servona var stabilt på 3.3 V men att strömmen varierade när maximal svängning gjordes åt ett håll och frekvensen mättes till 50 Hz. Det visade då att en strömtransformator är kopplad i mottagaren som ändras beroende på styrsignal och denna strömtransformator valdes att användas istället för att omkonstruera bilen. Mätningar på motorn gjordes likadant och mättes på vita kabeln som är kopplad till motorn:

Rörelse	Bitar
Stillastående	300
Max bakåt	160
Max framåt	440

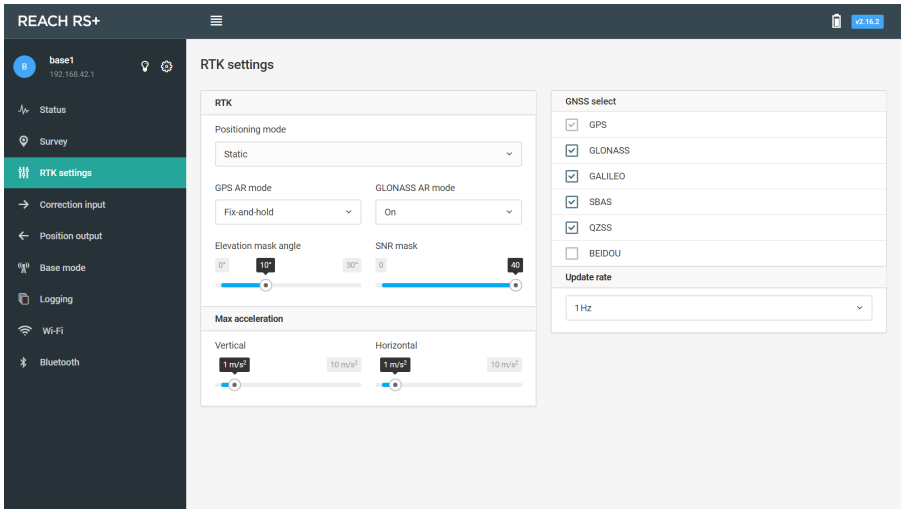
Tabell 2: Mätningsresultat av styrsignal till motor.

4.2 GPS-Systemet

Som det har nämnts ovan har GPS-systemet två delar, varav ena är en basstation och andra en rörlig modul. I denna del kommer inställningarna, en närmare förklaring för GPS-systemets GUI och hur man ställer upp systemet korrekt förklaras.

4.2.1 GUI:n och dess inställningar

Genom GUI:n ställs både modulen och basstationen in. Här finns dessutom statusfönster för att se vilka GPS-punkter som har funnits och vilka satelliter som är tillgängliga. GUI:n är tydligt uppdelad i flikar som syns på vänster sida där första intressanta fliken som används är ”RTK settings”.



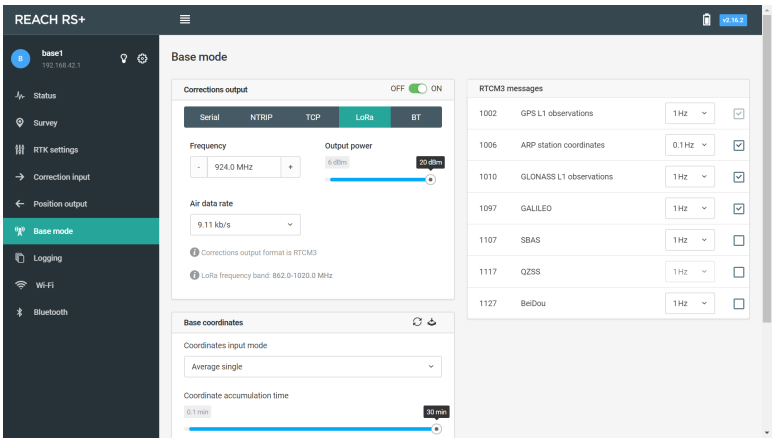
Figur 8: RTK-inställningar för basstationen.

Här sker de viktigaste inställningarna för GPS-systemet. Under RTK-rubriken ställs de olika parametrarna för mottagarens beteende in där ”Positioning mode” innebär ifall enheten ska mätas som en stillastående eller en rörlig enhet. ”GPS AR mode” är ifall GPS-systemet ska behålla en Fix lösning under en kort period (Fix-and-hold) eller ifall den ska använda en ny för varje epoch (Continuous). Fix-and-hold kan ses som att ifall mottagaren rör på sig åt något håll så läses positionen av med en viss ”tröghet” för att stabilisera koordinaterna. Emlid rekommenderar denna inställningen för rörliga enheter då koordinatpunkterna blir mer jämna. ”Elevation mask angle” är hur stor vinkel mottagaren får använda för att läsa av satelliter, eftersom ju närmare horisonten satelliten är vid tillfället desto sämre signal blir det. Utöver detta finns SNR mask vilket är

”Signal to noise Ratio” alltså hur mycket brus en signal som mest får ha för att den ska användas.

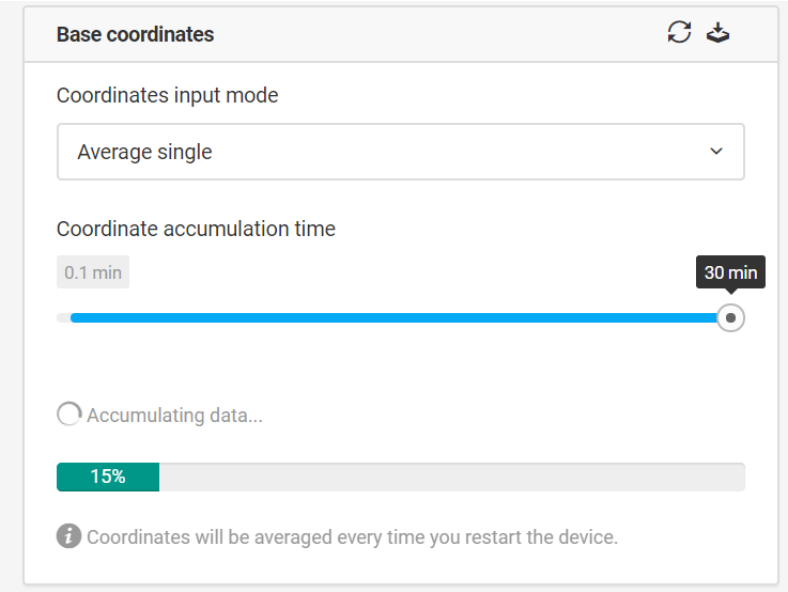
Under max acceleration anges mottagarens maximala accelerationen och eftersom bilen inte rör sig i någon hög hastighet åt något håll kan denna sättas väldigt låg. ”GNSS select” är vilka satelliter som mottagaren tillåts försöka använda. Här är det väldigt viktigt att både basstationen och modulen har samma satellitsystem valda. ”Update rate” är hur ofta mottagaren ska försöka kontakta satelliterna för en ny position, med högre frekvens fås mera data men kopplingen blir sämre (mer brus).

Därefter kommer nästa flik som är väldigt viktig att ställa in på basstationen, ”Base Mode”. Inställningar här bestämmer vilken slags koppling som skall ske mellan modulen och basen och det intressanta protokollet i detta projektet är Long Range (LoRa), alltså radio. Frekvensen som skall användas skall helst ligga på en nivå med lite störningar och det varierar lokalt vilka frekvenser som har minst brus. Med lite tester på olika frekvenser togs den optimala frekvensen fram, 924 MHz fungerar bäst i Göteborgsområdet. ”Air data rate” är antalet bitar som basstationen får lov att skicka till modulen, där ju högre data rate desto snabbare kan basstationen skicka sitt korrigerande data till modulen. Här användes Emlids rekommenderade inställningar.



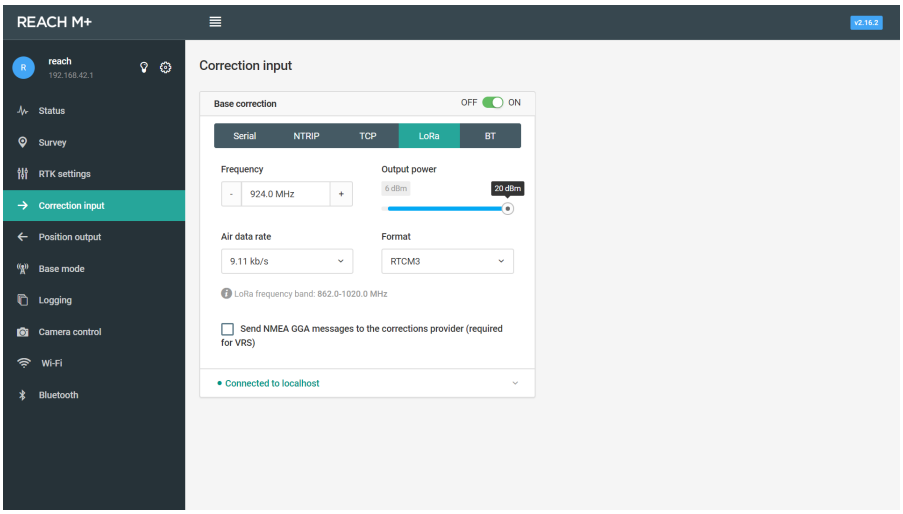
Figur 9: Basinställningar som kan göras på RRS.

När basen har erhållit sina inställningar korrekt ställs ”Base Coordinates” in. ”Coordinates input” mode anger vilken slags noggrannhet basen ska använda. Som tidigare nämnts så är Fix den mest noggranna, men eftersom bilen inte kräver så stor noggrannhet användes Single. Därefter ställs tidsramen in för hur länge basstationen ska kalkylera sin position, där ju längre tid desto mer noggrant blir det. Här användes den maximala tidsramen på 30 min för att få bästa möjliga värden. Det rekommenderas därför att ställa upp GPS-systemets basstation först för att vara mest effektiv.



Figur 10: Koordinat insamling för RRS position.

Modulens inställningar för korrigering görs under fönstret ”Correction input”. På Figur 11 syns de olika input-metoderna som modulen kan använda och i detta projekt användes LoRa vilket är radio signal. Det går även att använda TCP vilket är Wifi men eftersom det inte användes i projektet tas det inte med. Under LoRa-fliken ställs frekvens, datahastigheten, formatet och styrkan på signalen in. Den optimala frekvensen är beroende av det lokala områdets störningar och bör testas fram. Som nämndes tidigare så var 924 MHz bästa alternativet för frekvensen och datahastigheten är inställd på 9.11 kb/s. Högre datahastighet ger en känsligare signal medans positionen kan uppdateras snabbare och formatet är endast kommunikationsprotokollet som ska användas. För detta projektet var det enda viktiga att modulen och basen använde samma format (RTCM3). Signalstyrkan sattes till max för längsta räckvidd för att ökad signalstyrka ger mindre batteritid men eftersom bilens batteri bara håller i 5 timmar var detta aldrig ett problem.

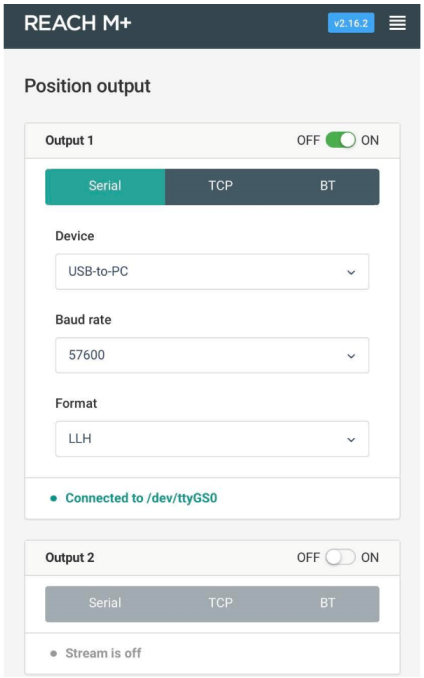


Figur 11: RRS inställningar för överföring mellan RRS och RM.

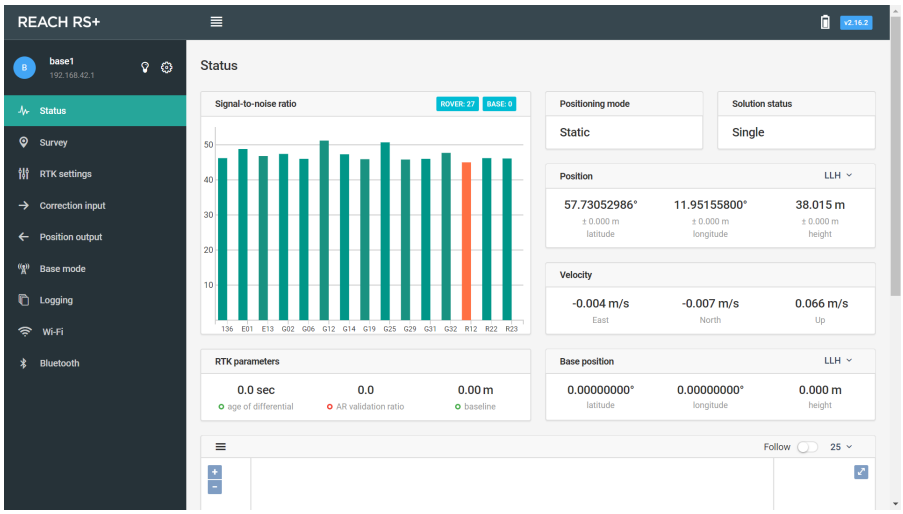
Under ”Position output” anges modulens val av output, där de olika metoderna är t.ex TCP, Serial och Bluetooth. Den som användes i projektet är Serial: USB to PC för överföringen mellan RM och bil. Under fliken väljs formatet på datan som ska skrivas till datorn. Eftersom mjukvaran är väldigt känslig för formatet på datan är det extremt viktigt att den ställs till LLH som är koordinater i longitud, latitud och höjden över havet. Mjukvaran kommer endast kunna läsa koordinaterna i just LLH-formatet. Här är också ”Baud raten” viktig eftersom den måste väljas till samma som mjukvaran använder sig av för att överföringen ska fungera korrekt. Mjukvaran använde sig av 57600 eftersom den rekommenderades i C-biblioteket för seriell kommunikation. Utöver LLH finns det många format att välja på varav NMEA är det vanligaste att använda för vanliga GPS-system. NMEA är bland annat systemet som Google maps använder sig av. NMEA var det formatet som skulle användas från början men eftersom omvandlingen ifrån detta till värden som kunde användas av vår mjukvara hade tagit för lång tid så blev det LLH. Fördelen med NMEA är då såklart att det går att rita upp rutten direkt i Google maps vilket ger en bättre överblick exakt vart punkten ligger.

Efter att systemets inställningar är klara och ifall allt fungerar ska det se ut som i figur 13 där bilden visar statusfönstret för basstationen. På diagrammet för ”signal to noise ratio” syns de användbara satelliterna och eftersom alla utom en stapel är grön har stationen god täckning och övervägande majoritet av satelliterna kan användas. Anledningen till att ”Base Position” inte är ifylld ännu är att stationen fortfarande bearbetade sin position medans bilden togs. I figur 12 syns sedan basstationens position och modulens olika positioner. Från det fönstret går det att få en klarhet om hur rutten som bilen får kommer att se ut och modulens status-fönster kommer vara näst intill identisk. Största

skillnaden kommer att vara noise to ratio grafen, eftersom den också kommer vara ifylld med basens staplar fast i grått.



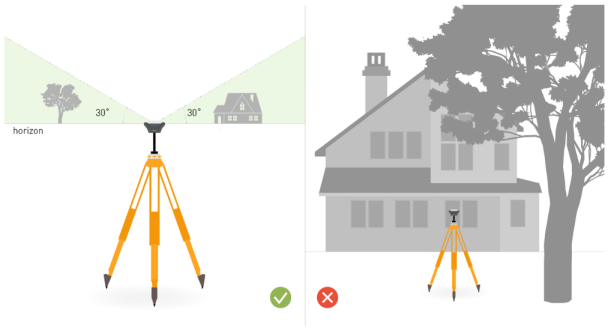
Figur 12: Inställningar för överföring mellan RM och bilen.



Figur 13: Status fönster för RRS.

4.2.2 Uppställning och placering av systemet

Systemets placering och uppställning är central för noggrannheten och funktionaliteten hos systemet. Basstationen ska ha en 30 graders vinkel från dess ovansida upp emot himlen helt fri. Detta är för att så många satelliter som möjligt ska fångas upp och användas. Annars blir den inräknade positionen för inexakt och kan inte användas. Basstationen måste också placeras så långt ifrån annan elektronisk utrustning som möjligt. Dess resulterande RF-strålning kan annars störa signalen mellan satelliterna och den själv.



Figur 14: Bild på rekommenderad placering för RRS till vänster och till höger är bild på dålig placering.

Som syns på bilden så ska basstationen levereras med en tripod som bör användas. Det rekommenderas starkt att ställa basstationen mitt ute på ett fält eller uppe

på en kulle för bästa resultat. Av erfarenhet så är skillnaden extremt stor från att använda basstastionen på ett bord mot att använda tripod. Det kan skilja mot att få 2-3 gröna staplar till att få 10.

4.3 Krets med RPi och PWM modul

För att koppla om från kontrollenheten till att använda RPi och PWM-modulen används tre switchar och de är monterade uppe på motorn så det går att ändra från manuell styrning till automatisk styrning enkelt och smidigt. Detta gjorde det även enklare att byta från manuell till automatisk styrning efter initieringssekvensen från kontrollenheten. Eftersom sekvensen behövs vid varje omstart av bilen för att använda den automatiska styrning var detta en bra lösning. Switcharna har två ingångar där ena är kopplad till mottagaren och andra till PWM-modulen. Utgången från en switch är antingen kopplad till motorn eller en servo. Jordkabel och strömkabel är kopplad enligt fabrikskopplade bilen.

Mellan RPi och PWM-modul kopplades fyra kablar för att styra bilen. Från RPi används statiska 3.3 V utgången för att PWM-modulen ska fungera och en kabel för jorden. De sista två kablarna är I2C-bussen för klockan (SCL) och datan (SDA) från RPi till PWM-modul. SCL:s funktion är att dela samma klocksignal som Raspberryn har med modulen och SDA att överföra RPi datan.

Från PWM-modul till bil kopplades fyra kablar in i systemet. En är jordkabeln som är kopplad in i systemet så jorden på RPi och PWM-modul har samma jord som bilen. De tre andra kablarna är överföring av data från 3.3 V pinnar på PWM-modulen som skickar ut pulsmodulering som är programmerad. De är kopplade till switcharna från pinnarna: 0 (servo), 3 (servo) och 7 (motor).

4.4 C-koden för RPi

Koden omfattar allt ifrån att läsa koordinaterna från modulen till att köra bilen mot dess destination. Målet var att få programmet att ta koordinaterna, spara dem i en fil som en rutt och sedan köra den. Detta krävs att koden hela tiden jämför dess nuvarande koordinater från USB-porten med sparade punkten den är på väg emot och sedan skall den svänga antalet grader som krävs för att nå punkten den är på väg emot. Koden behandlar alltså allt mellan USB-porten till bilens position.

Det första som genomfördes var att röra bilen i olika riktningar förutom baklänges, alltså att svänga bilen till höger, vänster och köra rakt fram. C-koden kan alltså inte behandla punkter som kräver att dess position körs baklänges. Detta är för att se hur koden bör utformas för att bilen ska konvergera mot destinationen på ett så bra sätt som möjligt.

4.4.1 Bibliotek

Två bibliotek används i koden för att styra PWM-modulen där ett av dem är för GPIO-pinnarna på RPi för att de ska fungera korrekt. Det är ett passivt bibliotek i detta arbete då den bara aktiverar pinnarna för användning och ej har några utsignaler på sig. Det andra biblioteket är specifikt för PWM-modulen som används, PCA9685. Biblioteket innehåller flera funktioner för att enkelt överföra data från RPi till PWM-modul till styrenheter[10]. För att initiera biblioteket krävs det att köra en setup-funktion där specifika variabler för kortet bestäms. Första är "PIN_BASE" som är vilken address pinnarna kommer starta på som sen kommer användas i varje funktion som styr PWM-modulen och den initieras här. Detta arbete använde 300 som bas då det rekommenderades av skaparna av biblioteket. Andra är vilken adress hela kortet har, i detta fall är det 0x40. Om man ska koppla ihop flera PCA9685-kort så använder man dessa adresser för att styra rätt signal till rätt kort. För att använda sig av de andra adresserna löder man direkt på kortet för att aktivera den specifika adressen. Sista variabeln är vilken frekvens pulsbredden ska använda och utifrån mätningar som gjorts tidigare var frekvensen i bilens system 50 Hz och då används det i koden som variabeln "HERTZ".

Användandet av biblioteket i koden görs mest genom funktionen "pwmWrite" som behöver två variabler när den blir kallad. Första är "PIN_BASE" adderat med vilken pinne som vill användas. Det finns 16 pinnar på PWM-modulen, och för att använda en specifik pinne skrivs "PIN_BASE + x" där x kan vara ett nummer mellan 0-15. Om man vill överföra samma data på alla pinnar kan x skrivas som 16. I detta arbete används pinnarna 0, 3 och 7 som förklarats tidigare. Andra variabeln är hur många bitar pulsbredden ska använda och eftersom PCA9685 är ett 12 bitars kort kan detta värde vara emellan 0-4096. Värden som används i arbetets kod är från mätningar och förklaras mer i senare kapitel hur de används.

4.4.2 Koordinater

Inläsningen av koordinater från RM till en USB-port på RPi sker genom programmeringen (se appendix 6.2). Eftersom GPS-signalen inte överförs kontinuerligt utan endast när RM är redo att överföra så måste koden vänta på att data är redo att överföras. Detta görs genom ett bibliotek för seriell överföring till RPi som kontrollerar om data är redo att överföras. För att använda detta programmeras det i en loop (se figur 15) som väntar på datan för nya koordinater till bilen. Denna överföring innehåller dock även mycket ointressant information för arbetet. Information som sorteras bort är: tid, datum, höjd, precisions säkerhet, signalstyrkor till satelliter och tid sedan RRS position blivit uppdaterad. Den enda information som inte sorteras bort är longitud och latitud.

```
while (1) {
    while (serialDataAvail(fd)) {
        i++;
        gotcords = TRUE;
        if (i > 26 && i < 39) {
            int temp = 0;
            temp = i - 27;
            lat[temp] = serialGetchar(fd);
        }
        else if (i > 41 && i < 54) {
            int temp = 0;
            temp = i - 42;
            lon[temp] = serialGetchar(fd);

            if (i == 53) {
                templat = strtod(lat, &lats);
                templon = strtod(lon, &lons);
                latlon[0] = templat;
                latlon[1] = templon;
                serialClose(fd);
                break;
            }
        }
        else {
            serialGetchar(fd);
        }
    }
    if (gotcords == TRUE) {
        break;
    }
}
```

Figur 15: Kod för loopen som hanterar inläsning och sortering av koordinater.

Funktionen ”serialDataAvail” registrerar om det finns data att överföra på USB-porten, och om detta finns så börjar loopen som ska sortera datan. Latitudens första bit i dataflödet är på plats 27 och sista biten på plats 38. Longitudens första bit är på plats 42 i data flödet och sista biten är på plats 53. Koordinaterna laddas i varsin char array. När de två är laddade och klara så omvandlas char arrayerna till double variabler som sparas i en global array. Om detta har fungerat sätts bool-variabeln ”gotcords” till true och loopen avslutas.

Det finns två funktioner som hanterar koordinaterna innan de används i koden. Dessa finns där för att eliminera felen som kan uppstå om en koordinatpunkt hamnar mer än en mil bort eller om koordinaterna bara är 0:or. Båda felen uppstår om RM tappar eller har dålig kontakt med satelliter vilket skapar problem i både inspelning och uppspelning av rutt. För att undvika att dessa koordinatpunkter används i programmet skapades dessa funktioner.

Varje gång bilens koordinat uppdateras sparas koordinaterna i en global array kallad ”latlon”. Denna array används i koden konstant då den alltid innehåller bilens nuvarande position. Första latituden och longituden sparas i varsin variabel kallad ”startlat” och ”startlon”. Dessa variabler används i funktionerna som behandlar om koordinaterna är godkända koordinater för koden eller om de ska avfärdas för nya koordinater. Vid inspelning av rutt så kontrolleras varje ny koordinatpunkt gentemot startvariablerna om den är på samma heltal som

start. Detta eliminerar alla punkter som inte hamnar inom en mil ifrån riktiga positionen på bilen. När det väl blev fel i koordinaterna så blev det grova fel som gjorde att punkten ej var inom Sveriges gränser längre och detta sker i funktionen "firstcords" (se appendix 6.2). Om koordinaterna som överförs bara är 0:or så används olika funktioner vid inspelning och uppspelning, "checkposrecord" och "checkposdrive". I "checkposrecord" (se appendix 6.2) körs funktionen för inläsning av koordinater kontinuerligt tills att koordinaterna inte är 0:or längre, vilket betyder att signalen har blivit stabil igen. I "checkposdrive" så händer samma sak men i denna funktionen så stannar även bilen upp tills signalen är bra igen. Detta behövs så bilen inte kör förbi punkter som egentligen skulle registreras som avklarade vid en bra signalstyrka.

Funktionen som avslutar programmet är "finish" (se appendix 6.2). Om bilens position är inom 6 meter från startkoordinaterna åt varje håll så är rutten färdig. Det görs genom att öka startkoordinaterna med 0.00006, där en ökning med 1 i femte decimalen är 1.1132 meter, och minska den med 0.00006. Funktionen undersöker om bilens position är inom detta spann i longitud och latitud. Under både inspelning och uppspelning används funktionen för att avsluta programmet. Utöver att den avslutar programmet så stänger den även filerna vi använder och återför bilen till stopp-position.

4.4.3 Uträkningar för bil

Eftersom GPS-systemet ej vet vilken riktning den är vänd mot så används enhetscirkeln för att bestämma åt vilket håll den ska svänga. Funktionen som gör uträkningen mot koordinatpunkten bilen ska köra till använder fyra variabler: "carcordslat" som är bilens latitud, "carcordslon" som är bilens longitud, "targetcordslat" som är koordinatpunktens latitud och "targetcordslon" som är koordinatpunktens longitud. Funktionen som utför beräkningarna för graderna är kallad "calcs" (se appendix 6.2). För att beräkna graderna mot nästa punkt bilen ska köra till används uträkningen nedan:

$$\Delta x = x_{\text{bil}} - x_{\text{mål}}$$

$$\Delta y = y_{\text{bil}} - y_{\text{mål}}$$

$$\theta = \arctan\left(\frac{\Delta x}{\Delta y}\right)$$

Eftersom C-biblioteket "math.h" beräknar arctan till radianer omvandlas det till grader:

$$\theta = \left(\frac{\theta * 180}{\pi}\right)$$

För att sedan använda dessa grader runt hela rutten behövs de anpassas i enhetscirkeln beroende på var bilen befinner sig.

Difflat	Difflon	Vinkel	Sväng
$l>0$	$l>0$	$\theta>0$	Vänster
$l>0$	$l>0$	$\theta<0$	Höger
$l>0$	$l<0$	$\theta>0$	Vänster
$l>0$	$l<0$	$\theta<0$	Höger
$l<0$	$l>0$	$\theta>0$	Höger
$l<0$	$l>0$	$\theta<0$	Vänster
$l<0$	$l<0$	$\theta>0$	Höger
$l<0$	$l<0$	$\theta<0$	Vänster

Tabell 3: Beräkning av riktning på hjulen.

Eftersom PWM-signalen ej kan vara negativ så multipliceras graderna med -1 innan den returneras vid tillfällena den är negativ. Då det här måste göras så sätts en variabel till 0 om bilen ska svänga vänster och till 1 om bilen ska svänga höger. Denna variabel är en global array med en plats kallad "turning".

4.4.4 Körande av bil

Funktionen som styr hur bilen ska köras kallas "driving" (se appendix 6.2). I grunden är det en while-loop som körs tills nästa koordinatpunkt är nådd. För att koordinatpunkten ska vara nådd används samma spann som används i funktionen "finish" dvs inom 6 meter. Innan funktionen "driving" kallas så skickas en 12 bitars kod till PWM-modulen som överför en pulsbredd till motorn som får en hastighet framåt. Vid tidigare mätningar mättes stillastående bil till 300 bitar (se tabell 2) och för vanlig gas framåt används 345 bitar under automatisk körning. Det är en lagom hastighet för detta ändamål då det är enklare att hantera en något långsammare hastighet. Det första som händer i while-loopen är att bilens position uppdateras. Detta görs för att position ska vara så uppdaterad som möjligt innan "calcs" kallas. Graderna som returneras från "calcs" skickas vidare in i funktionen för att bilen ska svänga kallad "turn" (se appendix 6.2).

"Turn" styr hjulen i de grader som nästa punkt är emot. Eftersom graderna alltid är positiva när funktionen kallas så måste variabeln "turning" följa med som visar om svängen skall vara åt vänster eller om den ska vara åt höger. Hjulen har en maxsväng på 45 grader och maxsvängen i bitar enligt tabell 1. För att beräkna vad för pulsbredd som behövs delas skillnaden i bitar från hjul rakt fram med antalet grader hjulen kan svänga. I detta fall är det:

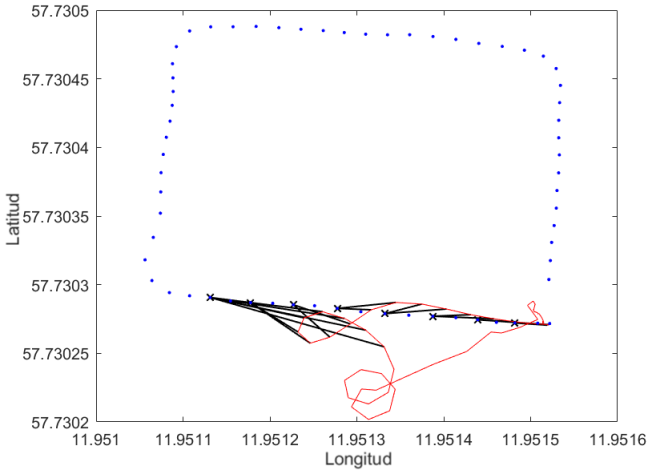
$$Högersväng : 300 - 160 = 140$$

$$Vänstersväng : 440 - 300 = 140$$

$$Pulsbredd\ per\ grad : \frac{140}{45} = 3.11$$

”Turn” innehåller två if-satser där båda är för vilket håll hjulen ska svänga. När variabeln ”turning” är 1 så svänger bilen höger. Här multipliceras graderna från ”calcs” med 3.11. Resultatet av detta är hur många bitar hjulen ska svänga. För en högersväng subtraheras bitarna för hjul rakt fram (300) med resultatet. Detta rundas av till ett heltal då pulsbredden inte kan hantera decimaltal, som sedan överförs på pinnarna till servona (pin 0 och 3). För en vänstersväng fungerar det likadant förutom att resultatet från grader multiplicerat 3.11 adderas med bitarna för hjul rakt fram. Om graderna åt något av hållen är över maxsvängen på 45 grader ändras det till pulsbredd för 45 grader för att undvika fel.

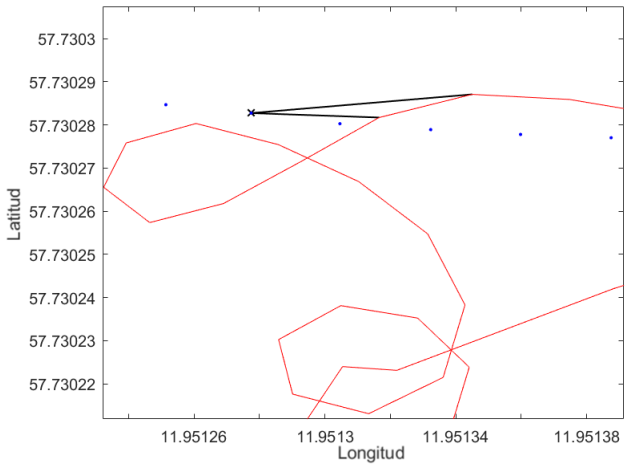
4.5 Matlab för felsökning



Figur 16: Ett exempel på en inspelad (blåa prickar) och sedan uppspelad rutt. Röd linje är hur bilen kört och svarta linjer är vilken koordinatpunkt bilen siktade mot.

Efter att bilen har körts är loggfilerna fyllda med positionerna bilen passerade och koordinatpunkterna som bilen körde mot. Genom att logga i filen vid namn CarPosition.txt plottades rutten i Matlab som den borde köra jämfört med dit den egentligen körde och detta för att lättare se vad det är som händer med bilens position jämfört med vart den skulle svänga. Ett exempel på bilens rutt jämfört med där den egentligen körde syns i figur 16. På bilden syns den inspelade rutten med de blåa prickarna och där bilen egentligen körde i det

röda strecket. På nästan varje datapunkt på det röda strecket har punktens mål ritats ut. Målets rikting markeras med ett svart streck från bilen position till målet med ett svart kryss. Det som syns väldigt tydligt här är att bilen bara siktar på varannan punkt i rutten. Detta är för att sample-hastigheten ska bli tillräcklig. Bilen måste hinna uppdatera sin position minst två gånger innan den kan byta mål.



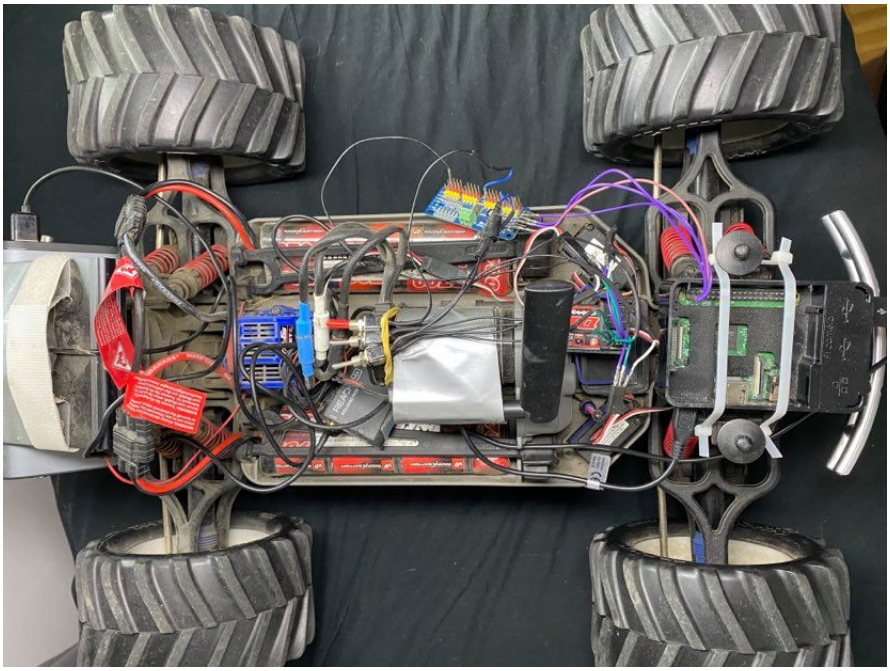
Figur 17: Matlabs resultat med variablarna Samplesize = 11, Samplestart = 10.

Redan i figur 16 börjar bilden bli väldigt kladdig trots att bara hälften av alla datapunkter är helt utritade. Detta löstes genom att introducera ”SampleSize” och ”SampleStart”. Genom dessa två variabler är det möjligt att välja från och till vilken punkt målet skall ritas upp. Intressanta datapunkter i figur 16 kan vara till exempel det femte målet. Där svängde bilen nedåt vid andra punkten istället för att räta ut. Med variablerna inställda korrekt syns dessa lättare i figur 17. Detta gjorde det mycket enklare att urskilja när bilen gjorde fel/rätt under tester vid exakta punkter.

4.6 Prototyp

I figur 18 syns bilen med alla komponenter som behövdes för arbetet. Längst till höger på bilden sitter RPi som är kopplad till PWM-modulen med de lila och rosa kablarna. PWM-modulen är kopplad med de blå kablarna till switcharna och den svarta till en av jordkablarna på bilen. Switcharna är sedan inkopplade med utsignalen till de olika styrenheterna och den andra insignalen från mottagaren. Till RPi sitter även strömtillförseln som är den silvriga powerbanken längst till vänster. USB-porten på RPi är kopplad till RM som är i mitten av bilden. Antennen i mitten är för kopplingen till RRS och antennen till vänster

som sitter på en metallplatta är för kopplingen till satelliterna.



Figur 18: Prototyp med all utrustning.

5 Teknisk utveckling

Eftersom allt blir mer och mer automatiserat i dagens samhälle skulle den här produkten vara en bra addition till jordbrukets automatisering. Om det endast arbetar en person på gården så skulle detta bara underlätta arbetets gång, för då kan en uppgift på gården sköta sig själv. Problemet är att den skulle ersätta en arbetsuppgift om det är flera som arbetar på gården. Att köra den andra traktorn kan vara en persons heltidsarbete. Vi tycker det är värt att fortsätta utveckla alla delar i samhället men att arbetare blir uppsagda från jobb är inget vi arbetar för. Det är viktigt för arbetsgivaren att ge arbetaren, vars heltidsarbete blir automatiserat, en möjlighet att antingen arbeta med andra arbetsuppgifter eller utbilda dem för ett nytt arbete som kan uppstå med en automatisering. Detta kan innebära i detta fall att utbilda en tekniker för den automatiska styrningen som har uppstått. Åtskillig mängd anställda på gårdarna är redan tekniskt kunniga inom alla verktyg som används, därför är utbildningen till tekniker inte nödvändigtvis en stor investering.

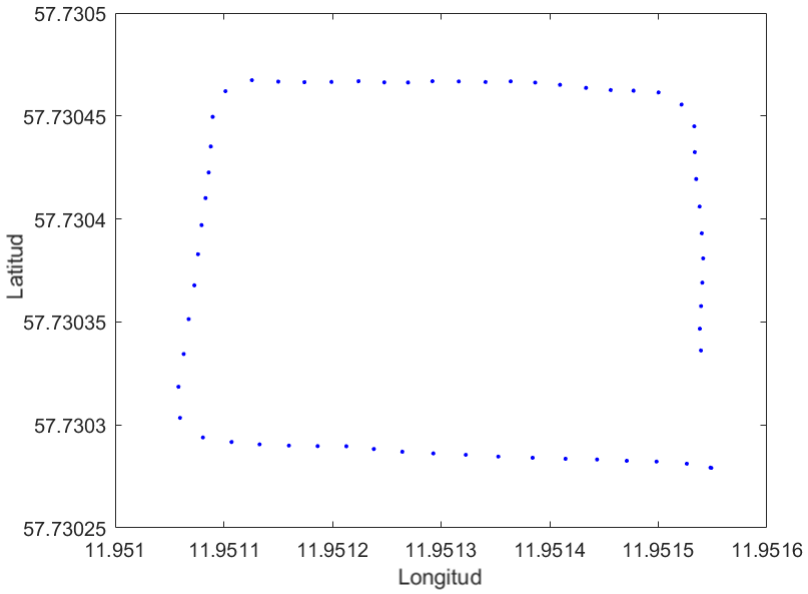
Mot hållbar utveckling så skulle detta göra att varje traktor med detta system skulle köra så optimalt som möjligt om den som spelade in första rutten gjorde detta. Som exempel kommer det inte att vara några utsvängningar som kan uppstå vid manuell körning. Den har även en inbyggd farthållare som gör att människors oregelbundna tryck på gaspedalen försvinner.

6 Slutsats och diskussion

Här presenteras bilens nuvarande status och diskuteras mer ingående. Vad som fungerar, vad som är fel och varför. Flera exempel på förbättringar till bilen kommer också att presenteras. Utöver bilen kommer rekommendationer för felsökning, planering och programmering.

6.1 Bilens status

Bilens nuvarande mjukvara kan svänga och köra rakt fram beroende på vinkeln mellan dess position och destination. Programmet är fullständigt kapabelt till att spela in en rutt utan större problem. Det enda problemet med inspelningen har till större del varit att starten har antingen skett utan täckning eller att bilen har markerat tre-fyra punkter på exakt samma position. Lösningen till detta var endast att göra så bilen hoppar över koordinater som ligger för nära den i starten. De inspelade rutterna blev till exempel som i figur 20. Där

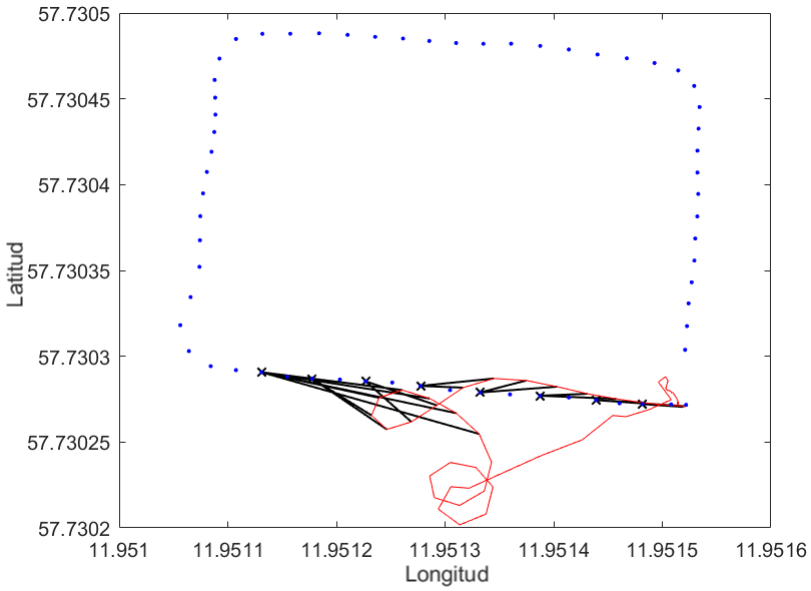


Figur 19: Resultatet av inspelningen, rutten är alltid representerad med blåa prickar.

syns problemet som nämndes ovan genom att titta på punkten nere i högra hörnet. Denna punkt ligger alltså precis där bilen startar. Detta kan resultera i att bilen kör förbi den utan att registrera det. Men med vår lösning blev detta inte längre ett problem. Den inspelade rutten ligger sedan sparad i ett text-dokument "target.txt". Denna fil kan sedan flyttas och kopieras hur som helst. För att använda den på en annan enhet räcker det med en ssh-klient och internetsladd för att ladda ned filen för att sedan ladda upp den. På detta sätt behövs inget separat minne mer än det Raspberry enheten redan har.

Med inspelningen avklarad fokuserades projektet större delen av tiden på att få automatisk styrning till att fungera. Denna delen är väldigt mjukvaruberoende och det stöttes på stora problem i olika funktioner. Det bästa exemplet som kan ges är på figur 20 vilket är samma figur som syntes i Matlab-delen i avsnitt 4 (figur 16). I avsnittet nämndes det att bilen väljer att öka svängen ytterligare trots att den bör räta upp.

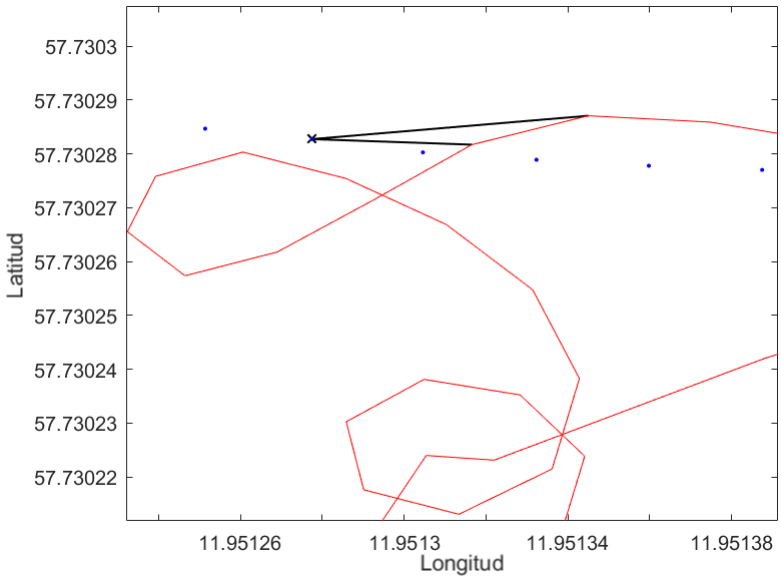
Genom att återanvända figur 16 i Matlab-avsnittet och förstora den (se figur 21) syns detta ännu tydligare. Bilen ökar sin sväng vid andra punkten där den egentligen skulle rätats upp. Efter extremt många tester och mycket diskussion har vi kommit fram till att detta problemet har uppstått från en fundamental brist i algoritmens logik. Genom att titta närmare på funktionen "Turn" (se figur 22) blir detta mera självklart. En parameter för funktionen är wheelturn, denna variabel är resulterande vinkeln mellan punkten och positionen. Denna



Figur 20: Ett exempel på en inspelad (blåa prickar) och sedan uppspelad rutt.

variabel sätts in i en omvandling grader till pulsbredd och denna pulsbredd är graderna representerade i svängen på däcken. Alltså ifall wheelturn är 20 grader kommer däcken att vrida sig 20 grader till höger eller vänster beroende på turning flaggan. Resultatet av denna funktion är att bilen kommer addera sin sväng med mängden grader från sin position till destination vid varje kontrollpunkt, trots att första svängen redan är i linje med målet. Grundtanken är alltså inkorrekt eftersom graderna på däcket inte är lika med resulterande svängens grader. Däcken kan svänga i 20 grader under två sekunder eller under tre sekunder, vilket resulterar i två olika svängar. Det hade fungerat i teorin om bilen alltid höll sig perfekt efter rutten men då bilen inte gör det så går den från små fel till större fel.

I figur 21 på det andra svarta strecket syns det att bilen har svängt däcken de grader som den skulle enligt funktionen fast åt felhåll. Detta innebär att ”calcs” funktionen också är fel, eller åtminstone vänd på felhåll. ”Calcs” funktionen bygger på en grundläggande idé att enhetscirkeln är dels förskjuten 90 grader, spegelvänd och att den inte vrider sig med bilens riktning. Detta för att bilens riktning startar rakt västerut. Detta är dock inkorrekt eftersom bilen i nuläget inte har någon riktning överhuvudtaget vilket är där både calcs-problemet och turn-problemet uppstår. Bilen vet aldrig åt vilket håll den egentligen är påväg.



Figur 21: Den förstorade versionen av den körda rutten med fokus på där bilen körde fel (andra svarta strecket).

6.2 Förbättringar

I den här delen förklaras vad som kan undersökas för att projektet ska fungera. Det finns två olika tillvägagångssätt som har utforskats för att den automatiska styrningen ska fungera korrekt. Det ena fallet är beroende av en magnetometer medans den andra använder sig av C-kod. Dessa diskuteras mer ingående nedan samt vad som hade kunnat förbättras mera allmänt innan och under projektets gång.

6.2.1 För att bilen ska fungera

För att lösa bilens grundläggande problem med riktningen har projektet tagit fram två olika förslag, båda kräver nästan en total omarbetning av ”turn”, ”calcs” och ”main” funktionerna. Det första förslaget som togs upp var den första mest logiska lösningen på problemet och det är att skaffa en modul som känner av riktningen i realtid, alltså en kompass/magnetometer. Med magnetometern går det att implementera fasta svängar som pågår tills bilens riktning representerar punktens grad relativt positionen. Ifall detta är lättaste lösningen är debatterbart eftersom detta kräver ytterligare arbete av USB-inläsning och att den behöver köpas. Logiskt sett är detta dock den lättaste eftersom den hade krävt minimal uträkning för att bilen ska svänga korrekt. Den kan också

```
void turn(double wheelturn, double turning[], FILE *test) {
     //Omvandlar grader till hjulens sväng höger eller vänster
    int x;
    if (turning[0] == 1) {
         //Hjulen ska vridas höger
        wheelturn = WHEELFORWARD - (wheelturn * 3.11);    //3.11 inställda
          pulsbredden ger den konstanten.
        x = round(wheelturn);   //PWM registret kan inte läsa decimaler
        fprintf(test, "HOGER \n");
        if (wheelturn < MAX_TURN_RIGHT)
        {
            wheelturn = MAX_TURN_RIGHT;
        }
        pwmWrite(PIN_BASE + 0, x);
        pwmWrite(PIN_BASE + 3, x);

    }
    else if (turning[0] == 0) {
         //Hjulen ska vridas vänster
        fprintf(test, "VANSTER \n");
        wheelturn = WHEELFORWARD + (wheelturn * 3.11);
        if (wheelturn > MAX_TURN_LEFT)
        {
            wheelturn = MAX_TURN_LEFT;
        }
        x = round(wheelturn);       //PWM registret kan inte läsa decimaler
        pwmWrite(PIN_BASE + 0, x);
        pwmWrite(PIN_BASE + 3, x);


    }
}
```

Figur 22: Här är en bild på funktionen ”turn” som användes för att bilen skulle svänga.

lättare förbättras genom att lägga till en procentuell sväng som inkrementeras desto större skillnad det är i grader mellan bilens rikting och mål. Detta bör eliminera ett framtida problem som annars kan uppstå där bilen istället börjar ”slingra” sig fram i små svängar istället för att bara köra rakt fram.

Den andra lösningen kräver inte en magnetometer överhuvudtaget. Lösningen innebär att lägga in en virtuell riktning genom att lägga in för hand vilken riktning bilen startar i. Med det antagandet är det möjligt att följa bilens riktning relativt hur den svänger. Lösningen innebär fyra fasta svängar som är två till höger och två till vänster. Den ena svängen i en riktning är så nära 90 grader som möjligt medans den andra ska vara lite mindre, ca 30 grader. På detta sättet är det möjligt att alltid veta hållet som bilen kör mot genom att addera och subtrahera svängens grader från den ursprungliga startens vinkel. Den fundamentala skillnaden är synsättet på däckens grader. Även ifall däcken är vridna tio grader kommer svängen att utföras tills den resulterande riktningen har ändrat sig 30/90 grader beroende på hur stor skillnaden är. De fasta svängarna kommer dock att resultera i att bilen ”slingrar” sig fram eftersom skillnaden i grader mellan position och destination aldrig kommer vara noll. Detta skapar en minimering med en viss tolerans i grader men aldrig en total eliminering.

6.2.2 Övrigt

För att nämna det självklara så har såklart Coronapandemin skapat problem för arbetets gång. Den första månaden spenderades på Consats kontor och det var en väldigt gynnsam och harmonisk arbetsplats. Allt som då behövdes i arbetet fanns på kontoret och var tillgängligt för användning. Handledaren på kontoret var väldigt närvarande i arbetets gång och alla kollegor var behjälpliga vid behov. När kontoret stängde så blev arbetsgången jobbigare eftersom det behövdes någon sorts lokal för allt material som används i arbetet. Det var inte möjligt att åka buss med allt material för arbetet och tyvärr kunde ej skolan lösa en lokal heller.

Det var även arbetets upplägg som förvärrades av Coronakrisen. När vi skapade planeringen i januari så räknades såklart inte detta med. Rapportskrivningen fick starta väldigt mycket tidigare än planerat då inte bil fanns tillgänglig varje dag. Detta gjorde att tester inte kunde fortsätta kontinuerligt. Det kunde vara uppehåll på en vecka mellan två testtillfällen vilket förvärrade testerna. Nu i efterhand tror vi att det skulle behövts mer handledning i form av vad som var viktigt att fokusera på när arbetet pågick. Detta var pågående under tiden inne på kontoret men inte efter. Handledaren på Consat var alltid kontaktbar på telefon och mail men vi använde inte detta tillräckligt.

I framtiden bör vi också trots pandemin arbeta mera med planeringen och hur upplägget bör vara. Ett exempel på detta är vad som krävs för att på ett effektivt sätt debugga och felsöka bilens kod. Ett av de största problemen var att vi inte hade lagt tillräckligt med betänketid på hur detta ska genomföras på ett bra sätt. Matlab-programmet växte fram utefter projektetsgång men hade egentligen kunnat tas fram mycket snabbare ifall mera tid hade lagts på just frågan runt felsökning. Större tid bör läggas på all infrastruktur runt projektet i framtiden istället för nästan bara på just prototypen, vilket hade effektiviserat tiden bättre, samt bör möjligheter för avgränsningar diskuterats mera innan projektets start. Dessa avgränsningar hade kunnat användas beroende på vår planering, vart vi bör ligga under veckornas gång och projektets aktuella situation. Istället så skedde alla avgränsningar mot slutet av projektet eftersom vi märkte att bilens mjukvara inte skulle fungera i tid. Målstolpar sattes upp i början av projektet men vi insåg inte vikten av vad det innebar ifall de inte nåddes när de skulle eller tvärtom allt låg i fas. Projektets gång hade blivit mer dynamiskt och bilen hade kanske fungerat till viss del fast t.ex bara till en viss rutt.

Referenser

[1] B. K. Lars Bengtsson, *Transformer och filter*, 2nd ed. Malmö, Sweden, 2016.

[2] Traxxas, *Xmaxx traxxas owner's manual*, ed. by Traxxas, 2015. [Online]. Available: https://traxxas.com/sites/default/files/77076-4-OM-EN-R00.pdf.

[3] Emlid, *Reach rs/rs+ docs*, ed. by Emlid, May 25, 2020. [Online]. Available: https://docs.emlid.com/reachrs/ (visited on 05/25/2020).

[4] ——, *Reach m2 docs*, ed. by Emlid, May 25, 2020. [Online]. Available: https://docs.emlid.com/reachm2/ (visited on 05/25/2020).

[5] R. Langley, "Rtk gps", *GPS WORLD*, Sep. 1, 1998. [Online]. Available: https://pdfs.semanticscholar.org/df71/4604a0d0fa323dacd7a2804da89df8f56209.pdf.

[6] J. V. Sickle. (May 25, 2020). The integer ambiguity. J. V. Sickle, Ed., College of Earth and Mineral Sciences, [Online]. Available: https://www.e-education.psu.edu/geog862/node/1787 (visited on 05/25/2020).

[7] ——, (2020). Differencing. J. V. Sickle, Ed., College of Earth and Mineral Sciences, [Online]. Available: https://www.e-education.psu.edu/geog862/node/1727 (visited on 05/25/2020).

[8] Drogon. (Jan. 1, 2020). Wiring pi gpio interface library for the raspberry pi. Drogon, Ed., [Online]. Available: http://wiringpi.com/ (visited on 05/25/2020).

[9] N. Semiconductors, *Data sheet pca9685*, ed. by N. Semiconductors, Apr. 16, 2015. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/PCA9685.pdf.

[10] R. Sprung. (Mar. 19, 2019). Pca9685 readme. R. Sprung, Ed., [Online]. Available: https://github.com/Reinbert/pca9685.

7 Appendix

7.1 Matlab-kod

2020-05-12 14:07 C:\Users\hampus.l.mar...\PrintRoute.m 1 of 1

```
%% Read car and target routes from raspberry files
clc
clear all
fileID1 = fopen('C:\Users\hampus.l.martinsson\Desktop\target.txt','r');
formatSpec = '%f';
A = fscanf(fileID1,formatSpec);
fclose(fileID1);
fileID2 = 'C:\Users\hampus.l.martinsson\Desktop\CarPosition.txt';
B = dlmread(fileID2);
lastn = size(B);
NewSize=lastn(1)-1;
tmp = B(lastn(1),1);
if tmp < 50 % lon must be above fifty for this testing location
    B = B(1:NewSize,:); %remove last row since it has a high risk of being filled↙
with gibberish
    disp("Warning last row error");
end

%% Sorting Carposition and its aim
SampleStart=1; %Draw from
SampleSize=20; %Draw to
arrSize=size(B);
CarPos = B(1:2:arrSize(1)-1,:);
CarTarget = B(2:2:arrSize(1),:);
for n=SampleStart:SampleSize
    if SampleSize == 0
        break
    end
    plot([CarPos(n,2) CarTarget(n,2)], [CarPos(n,1) CarTarget(n,1)], 'k-↙
x','MarkerIndices',2,'LineWidth',1)
    hold on
end
tmp=size(CarPos);
Carlat = CarPos(:,1);
Carlon = CarPos(:,2);
plot(Carlon,Carlat,'r')
hold on

%% plots recorded route
targetlat = A(1:2:size(A)-1);
targetlon = A(2:2:size(A));
plot(targetlon,targetlat, 'b.')
```

7.2 Raspberry-kod

...ource\repos\LinuxProject1\LinuxProject1\LinuxProject1.c 1

```c
#include <stdio.h>
#include <stdlib.h>
#include <wiringPi.h>
#include <string.h>
#include <math.h>
#include <pca9685.h>
#include "pca9685.c"
#include <stdio.h>
#include <stdbool.h>
#include <wiringSerial.h>
#define PIN_BASE 300 //basnummer för registret i pwm modulen 301 = pinne 1 osv
// Pinne 7 = motor (307), Pinne 0 och 3 = Servo
// Pulsbredder för servo och motor
#define MAX_TURN_LEFT 440
#define MAX_TURN_RIGHT 160
#define WHEELFORWARD 300
#define FORWARD 345
#define STOP 300

#define HERTZ 50 // PWM frekvens
#define Pi 3.141592654
#define dTol 0.00006 //distance tolerance, femte decimalen är en meter

#define RESET FALSE // True ifall PWM modulen inte har återställts korrekt
#define RECORD FALSE // True ifall en rutt ska spelas in istället för att köras



void position(double latlon[]) {
     // Väntar på data från gps och sparar den
    int fd;
    bool gotcords = FALSE;
    if (wiringPiSetup() < 0) {
        printf("Wiring-setup error");
        exit(0);
    }

    if ((fd = serialOpen("/dev/ttyACM0", 57600)) < 0) {
        exit(0);
    }
    printf("serial test start ...\n");
    int i = 0;
    char lat[12];
    char *lats;
    char lon[12];
    char *lons;
    double templat, templon;
    while (1) {
        while (serialDataAvail(fd)) {
            i++;
            gotcords = TRUE;
            if (i > 26 && i < 39) {
                int temp = 0;
                temp = i - 27;
                lat[temp] = serialGetchar(fd);
            }
```

```
            else if (i > 41 && i < 54) {
                int temp = 0;
                temp = i - 42;
                lon[temp] = serialGetchar(fd);

                if (i == 53) {
                    templat = strtod(lat, &lats);
                    templon = strtod(lon, &lons);
                    latlon[0] = templat;
                    latlon[1] = templon;
                    serialClose(fd);
                    break;
                }
            }
            else {
                serialGetchar(fd);
            }
        }
        if (gotcords == TRUE) {
            break;
        }
    }
}

void checkposrecord(double latlon[], double startlat, double startlon, FILE *target) {
    // Kollar så att koordinaterna är "rimliga"
    double templat, templon;
    templat = startlat - latlon[0];                  // Kollar ifall koordinaterna ligger inom heltal, annars har något blivit fel
    templon = startlon - latlon[1];
    templat = round(templat);
    templon = round(templon);
    if ((templat == 0 && templon == 0)) {
        fprintf(target, "%.9lf %.9lf ", latlon[0], latlon[1]);
    }
}

void checkposdrive(double latlon[]) {
    // Kollar så att positionerna är "rimliga"
    double templat, templon;
    while (1) {
        if ((latlon[0] && latlon[1]) == 0) {
                    // GPS har ingen täckning, stannar och väntar in.
            pwmWrite(PIN_BASE + 7, STOP);
            delay(1000);
            position(latlon);
        }
        else {
            break;
        }
    }
}

void finish(double latlon[], double startlat, double startlon, FILE *target, FILE *test, FILE *CarPosition) {
```

```
        // Ifall positionen är samma som den startade så är den framme.
    if((latlon[0] < startlat + dTol) && (latlon[0] > startlat - dTol)) {
                                                    // Används så att alla filer stängs korrekt.
        if((latlon[1] < startlon + dTol) && (latlon[1] > startlon - dTol)) {
            pwmWrite(PIN_BASE + 7, STOP);
            fclose(target);
            fclose(test);
            if (RECORD == 0)
            {
                fclose(CarPosition);

            }
            exit(1);
        }
    }
}

int firstcords(double latlon[], FILE *target, double startlat, double startlon) {
        //Första koordinaten är extra känslig för störningar,

    //eftersom någon måste dra ut sladden mellan dator och raspberry
    int badcords = 0;
    if ((startlat && startlon) == 0)
    {
        badcords = 1;
        return badcords;
    }
    else
    {
        fprintf(target, "%.9lf %.9lf ", latlon[0], latlon[1]);
        return badcords;
    }
}


void turn(double wheelturn, double turning[], FILE *test) {
     //Omvandlar grader till hjulens sväng höger eller vänster
    int x;
    if (turning[0] == 1) {
         //Hjulen ska vridas höger
        wheelturn = WHEELFORWARD - (wheelturn * 3.11);    //3.11 inställda pulsbredden ger den konstanten.
        x = round(wheelturn);   //PWM registret kan inte läsa decimaler
        fprintf(test, "HOGER \n");
        if (wheelturn < MAX_TURN_RIGHT)
        {
            wheelturn = MAX_TURN_RIGHT;
        }
        pwmWrite(PIN_BASE + 0, x);
        pwmWrite(PIN_BASE + 3, x);

    }
    else if (turning[0] == 0) {
         //Hjulen ska vridas vänster
        fprintf(test, "VANSTER \n");
```

```
        wheelturn = WHEELFORWARD + (wheelturn * 3.11);
        if (wheelturn > MAX_TURN_LEFT)
        {
            wheelturn = MAX_TURN_LEFT;
        }
        x = round(wheelturn);       //PWM registret kan inte läsa decimaler
        pwmWrite(PIN_BASE + 0, x);
        pwmWrite(PIN_BASE + 3, x);


    }
}

double calcs(double carcordslat, double carcordslon, double targetcordslat, double
  targetcordslon, FILE *test, double turning[]) {
        //Räknar ut grader på svängen och åt vilket håll den skall svänga
    double angle, difflat, difflon;
    double wheelturn;
    difflat = carcordslat - targetcordslat;
    difflon = carcordslon - targetcordslon;
    fprintf(test, "difflat: %lf difflon: %lf \n", difflat, difflon);

    angle = atan(difflat / difflon);    //retunerar i radianer
    angle = angle * 180 / Pi;

    //Beroende på katetrarnas tecken skall hjulen vridas åt höger eller vänster

    if(difflat < 0 && difflon > 0) {
        if (angle < 0)
        {
            angle = angle *(-1);
            turning[0] = 1;
        }
        else
        {
            turning[0] = 0;
        }
    }
    else if(difflat < 0 && difflon < 0) {
        angle = 90 - angle;
        if (angle > 0) {
            turning[0] = 1;
        }
        else {
            angle = angle * -1;
            turning[0] = 0;

        }
    }
    else if(difflat > 0 && difflon < 0) {
        if (angle < 0)
        {
            angle = angle *(-1);
            turning[0] = 1;
        }
        else
```

```
        {
            turning[0] = 0;

        }
    }
    else if(difflat > 0 && difflon > 0) {
        angle = angle;
        if (angle > 0) {
            turning[0] = 0;
        }
        else {
            angle = angle * -1;
            turning[0] = 1;

        }
    }
    fprintf(test, "angle: %lf \n", angle);
    return angle;
}
void driving(int iterations, double latlon[], double targetcordslat, double
  targetcordslon, FILE *test, FILE *CarPosition, double turning[]) {
    while (1)
    {
        //fprintf loggar för felsökning, CarPos för matlab, test för vanlig logg
        fprintf(test, "iterations = %d \n", iterations);
        position(latlon);   // Hämtar en ny position
        fprintf(test, "%.9lf %.9lf \n", latlon[0], latlon[1]);
        fprintf(CarPosition, "%.9lf,%.9lf \n", latlon[0], latlon[1]);
        fprintf(CarPosition, "%.9lf,%.9lf \n", targetcordslat, targetcordslon);
        checkposdrive(latlon);
        double angle = calcs(latlon[0], latlon[1], targetcordslat, targetcordslon,
          test, turning);
        turn(angle, turning, test);
        pwmWrite(PIN_BASE + 7, FORWARD);
        if ((latlon[0] < targetcordslat + dTol) && (latlon[0] > targetcordslat -
          dTol)) {
            if ((latlon[1] < targetcordslon + dTol) && (latlon[1] > targetcordslon
              - dTol)) {
                break;
            }
        }
    }
}

int main() {
    wiringPiSetup();                                //Ställer in Raspberryns GPIO
    int fd = pca9685Setup(PIN_BASE, 0x40, HERTZ);   //Ställer in PWM modulen
    if(fd < 0)
    {
        printf("Error in setup\n");
        return fd;
    }
    pca9685PWMReset(fd);
    FILE *target;
    FILE *test;
    FILE *CarPosition;
```

```
    int iterations, i = 1;
    int badcords = 0;
    double startlat, startlon;
    double latlon[2];
    double turning[1];  //1 om bilen ska svänga höger, 0 om vänster
    double targetcordslat, targetcordslon;
    printf("Running..\n");
    if (RESET == TRUE)
    {
        pwmWrite(PIN_BASE + 7, STOP);  // Ställer motorn till korrekt "idle" nivå
        pwmWrite(PIN_BASE + 0, 340);  // Kollar så att servon är omställd
        pwmWrite(PIN_BASE + 3, 340);
        delay(5000);
        pwmWrite(PIN_BASE + 0, WHEELFORWARD);
        pwmWrite(PIN_BASE + 3, WHEELFORWARD);
        exit(1);
    }
    if (RECORD == TRUE) //Initierar filer för läsning
        {
            target = fopen("/home/pi/target.txt", "w");
            test = fopen("/home/pi/test.txt", "w");
        }
    else if (RECORD == FALSE) //Initierar filer för läsning "r" och skrivning "w"
        {
            target = fopen("/home/pi/target.txt", "r");
            test = fopen("/home/pi/test.txt", "w");
            CarPosition = fopen("/home/pi/CarPosition.txt", "w");

        }
    while (1) {
        iterations++;
        i = i * (-1);
        if (RECORD == TRUE) {
             //Spela in

            position(latlon);
            if (iterations == 1) {
                 // Spara de första koordinaterna för finish funktionen.
                startlat = latlon[0];
                startlon = latlon[1];
                badcords = firstcords(latlon, target, startlat, startlon);
                fprintf(test, "%d \n", badcords);
            }
            else if (iterations > 1) {
                checkposrecord(latlon, startlat, startlon, target);
            }

            if (iterations > 10) {
                 //Ifall minst tio iterationer, kolla ifall bilen är framme
                printf("iterations= %d\n", iterations);
                finish(latlon, startlat, startlon, target, test, CarPosition);
            }
        }
        else if (RECORD == FALSE) //Spela upp
            {
```

```
                position(latlon);
                checkposdrive(latlon);
                if (iterations == 1) // laddar in första koordinaten
                    {
                        printf("Running..\n");
                        delay(10000);           // Behövs så att gpsen återfår
                     kontakt efter att internetsladden har dragits ur raspberryn
                        fscanf(target, "%lf %lf", &targetcordslat,
                     &targetcordslon);
                        startlat = latlon[0];
                        startlon = latlon[1];
                    }
                if (iterations > 10) {
                     // Om minst 10 punkter har träffats kollar den ifall den är
                     framme
                     finish(latlon, startlat, startlon, target, test, CarPosition);
                }
                //fscanf två gånger för att köra efter varannan koordinat i rutten.
                   Annars blir sampling raten på bilens position för låg.
                fscanf(target, "%lf %lf", &targetcordslat, &targetcordslon);
                fscanf(target, "%lf %lf", &targetcordslat, &targetcordslon);
                pwmWrite(PIN_BASE + 7, FORWARD);
                //Loggar för felsökning
                fprintf(test, "target: %.9lf %.9lf \n", targetcordslat,
                  targetcordslon);
                fprintf(test, "%.9lf,%.9lf \n", latlon[0], latlon[1]);
                fprintf(CarPosition, "%.9lf,%.9lf \n", latlon[0], latlon[1]);
                fprintf(CarPosition, "%.9lf,%.9lf \n", targetcordslat,
                  targetcordslon);
                //Börja köra mot punkten
                driving(iterations, latlon, targetcordslat, targetcordslon, test,
                  CarPosition, turning);
            }
    }
}
```