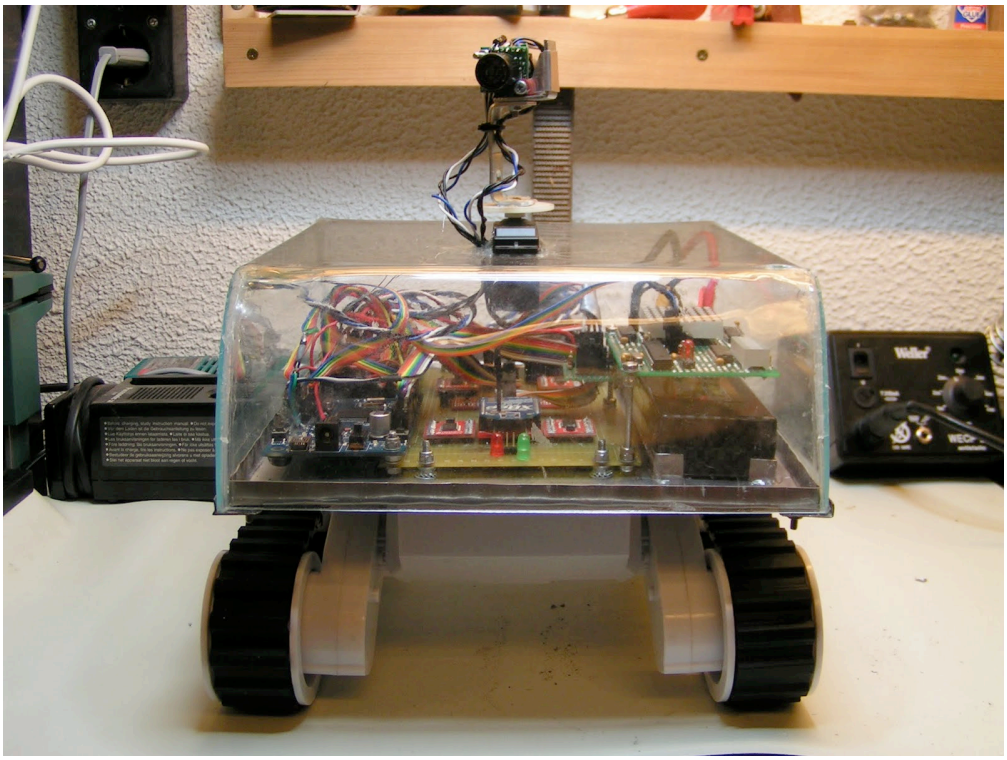
Projektarbete, 100p
Anton Olsson, Per Lindh Mångs och Erik Ihrén
Danderyds Gymnasium, NV09A
2011-2012

Kartobot

- Autonom kartritning i realtid



Handledare: Lars Gråsjö

Innehållsförteckning

1. Inledning 3
    1.1 Sammanfattning 3
    1.2 Bakgrund 3
    1.3 Syfte och mål 3
    1.4 Avgränsningar 3
2. Genomförande 4
    2.1 Hårdvara 4
        2.1.1 Strukturkarta 4
        2.1.2 Material 4
        2.1.3 Utförande 6
    2.2 Mjukvara 9
        2.2.1 Strukturkarta 9
        2.2.2 Verktyg 9
        2.2.3 Utförande 9
3. Resultat 12
4. Värdering av arbetet och resultatet 12
5. Referenser 14

1. Inledning

1.1 Sammanfattning

Målet med projektet var att bygga en robot som kan utforska okända omgivningar samtidigt som den kartlägger dem och utnyttjar kartan för att hitta i omgivningen. Under tiden som den utforskar så behöver roboten ha ett sätt att navigera runt omgivning medan den väljer nästa ställe att utforska på ett förnuftigt sätt.

Vi ville implementera en algoritm som navigerar mot den största outforskade regionen i omvigningen. Enligt tidigare foskning så ska det maximera effektiviteten [1]. Vi hann dock inte implementera det och det hade också tagit för mycket plats att spara all information vi behöver för det. Istället så skrev vi en algoritm som utforskar var femte ruta i rutnätet tills den åkt till alla av dem som den kommer fram till.

1.2 Bakgrund

Vi ville göra något med robotik och eftersom Anton och Erik redan hade en halvfärdig quadrocopter [2] så började vi med utgångspunkten att göra klart den. Byggandet av den gick framåt snabbt och när det var dags att välja projektarbete var den nästan klar. Vi förstod att trots att det var ett stort och avancerat projekt så var det inte något som kunde bli ett godkänt projektarbete med tanke på hur lite arbete som fanns kvar.

Vi behövde fortfarande ett projektarbete så vi tänkte vidare och bestämde oss för att göra ett tillägg till Quadrocoptern. Vi diskuterade eventuella tillägg fram och tillbaks tills vi beslutade oss för idén att kartlägga ett rum med hjälp av sensorer för att mäta avståndet.

Vi skrev en idéskiss som vi presenterade för Lars. Under diskussionen med Lars kom vi fram till att det vore bäst att förenkla projektet på två sätt. Först och främst tog vi helt och skippade idén med att ha den som en del av quadrocoptern utan tog istället och fokuserade på att göra en helt ny robot för projektarbetet. Vi tog också och begränsade oss till att bara kartlägga i 2d.

1.3 Syfte och mål

Målet med projektarbetet var att bygga en robot som själv kan skanna rummet den befinner sig i, generera en karta och sedan navigera optimalt (veta vägen och inte behöva leta sig fram) i rummet med hjälp av kartan.

Syftet med roboten är att den ska kunna underlätta räddningsarbeten i trånga utrymmen men det finns självklart andra användningsområden; avloppsunderhåll, generera planskisser etc. Genom expansionsportarna som sitter på experimentkortet så är det enkelt att sätta på olika moduler för olika användningsområden.

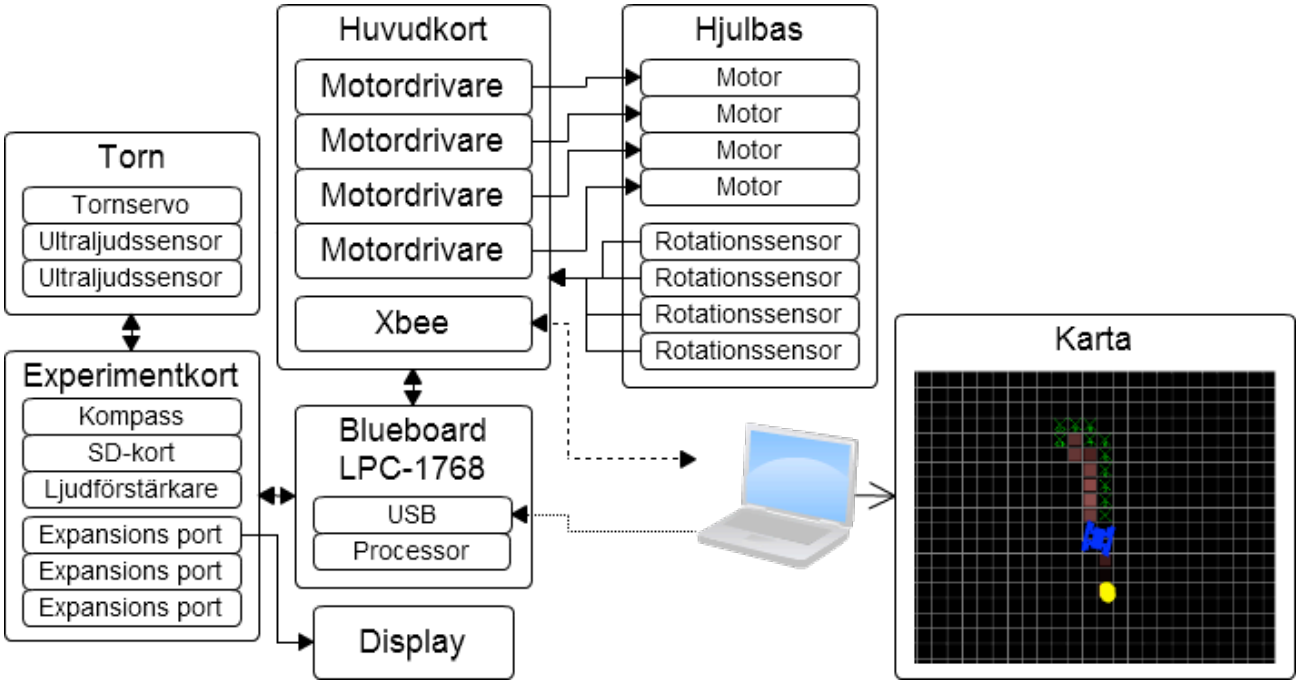
1.4 Avgränsningar

Vi begränsade oss till att bara kartlägga ett plan i 2d istället för ett rum i 3d som hade krävt en extra sensor och massa mer jobb. Vi tog också och begränsade vår design från en avancerad flygande robot till en markgående robot.

2. Genomförande

2.1 Hårdvara

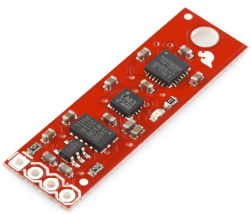
2.1.1 Strukturkarta



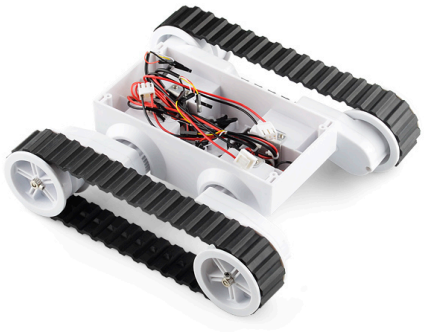
2.1.2 Material

Information får vi genom respektive komponents manual som tillhandahålls av tillverkaren och olika forum på internet.

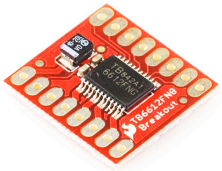
- 9 Degrees of Freedom - Sensor Stick [3] innehåller de flesta sensorerna som används för att lokalisera roboten.



- Rover 5 Robot Platform [4] hjulbasen för roboten.

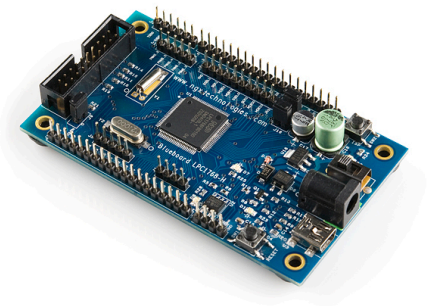


- 4st Motor Driver 1A [5] Styrelektronik för motorerna.

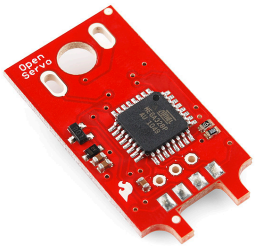


- 2st Ultrasonic Range Finder - XL-Maxsonar EZ3 [6] ultraljudssensorer för att mäta avstånd.

- Blueboard LPC1768-H [7] Mikrokontroller vars uppgift är att styra alltihopa med hjälp av den kod vi utvecklat.



- OpenServo [8] Servots styrkrets där man kan läsa servots nuvarande position och hastighet.



- XBee Pro 50mW Series 2.5 Wire Antenna [9] för kommunikation med datorn

2.1.3 Utförande

Vi började med att köpa in alla delarna. När delarna kom började vi med den köpta hjulbasen som är basen för hela projektet.

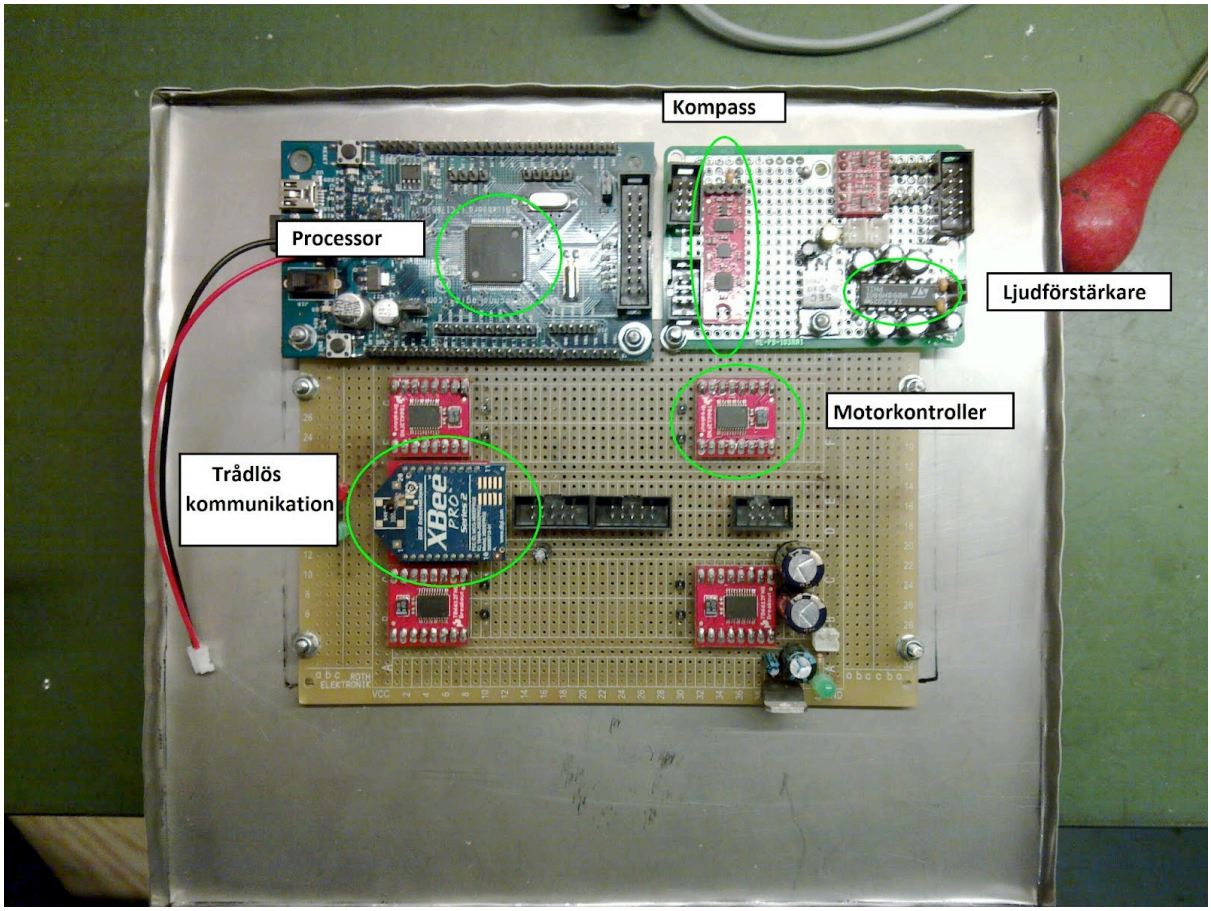
Hjulbasen har fyra skruvhål som man kan fästa saker i men eftersom den är gjord i plast så går de snabbt sönder om man skruvar ofta. Därför tog vi en plåtbit av rostfritt stål och sågade ut ett kvadratiskt hål i mitten så att kablarna från motorn kan nå kretskorten vi planerade att placera på plåtbiten. Vi fäste plåtbiten direkt på hjulbasen.

Huvudkretskortet fästes direkt på plåtbiten och på det kretskortet fäste vi två andra kretskort. Dels fäste vi processorkortet “Blueboard” i kanten. På “Blueboard” sitter processorn som gör alla beräkningar. Det är den som kör vår kod. Vi fäste också ett experimentkort på huvudkortet. Ett experimentkort är ett kretskort gjort för att man enkelt ska kunna löda fast komponenter på det.

Eftersom motorkablarna och rotationssensorerna kommer från hjulbasen så sitter deras kontakter på undersidan av huvudkortet. Huvudkortet har också hand om elförsörjningen till hela roboten samt den trådlösa kommunikation som fick sitta där på grund av att den var för stor för att passa på experimentkortet.

Det är enklare att löda små komponenter på experimentkortet än på huvudkortet så där placerade vi kompassen, ultraljudskontakterna samt kontakten för ultraljudtornets servo. Vi passade på att sätta dit en ljudförstärkare som vi plockade från ett ljudkort från en gammal dator. Att lägga till support för ljud var bara några timmars jobb. Processorkortet har en analog signal ut som lämpar sig för att spela upp ljud och eftersom vi ansåg att ljudet kan hjälpa felsökningsprocessen så tyckte vi att det var något som var värt att spendera några timmar på.

Per började jobba på att montera hjulbasen vilket innebar mycket lödande.



Här är en bild på robotens elektronik i ett tidigt skede. De viktiga delarna är markerade.

Erik började jobba på att bygga tornet när ultraljudssensorerna kom fram. För att skanna rummet och få en bild av avståndet till närliggande hinder använder vi oss av ultraljudssensorer.

Valet var mellan ultraljud och infraröd. IR-sensorn vi kollade på gav resultatet analogt med ett olinjärt samband mellan spänning och avstånd. Ytan har en större påverkan på IR-sensorn än på ultraljudet och IR-sensorn har problem med objekt som är närmare än 15cm och ger då samma värde som om de var på 25cm avstånd. Ultraljudet har digital utgång som ger resultat direkt i centimeter. Analoga signaler är jobbigare att läsa. Ultraljuden vi kollade på hade mycket dokumentation i jämförelse med IR-sensorn som knappt hade någon. Detta kombinerat med de andra nackdelarna med IR ledde till att vi valde det simplare alternativet, ultraljud. Den enda riktiga nackdelen med att välja ultraljud är att de kostar ungefär tre gånger så mycket som IR-sensorerna.

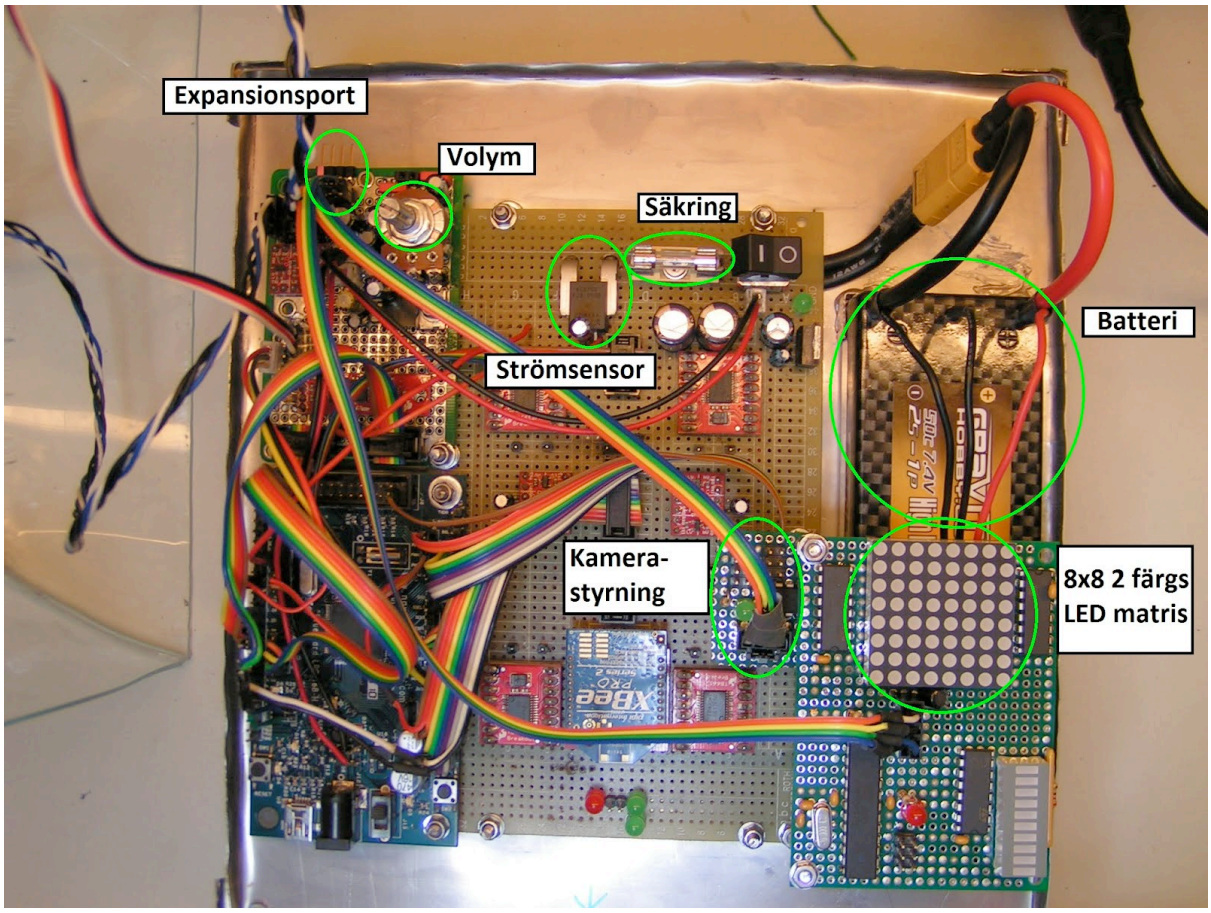
För att snurra tornet använder vi ett servo. Servot vi använde var ett digitalt servo av typen OpenServo V2. För att få tornet att snurra överhuvudtaget behövdes fungerande kommunikation till servots mikrokontroller.

Vi hade mycket svårt med att få tornet att snurra mjukt. I februari bytte vi ut det mot ett vanligt analogt servo. Nackdelen med ett analogt servo är att vi inte kan läsa av var den är men eftersom vi roterar tornet sakta så borde vinkeln den är på vara väldigt nära vinkeln vi sagt åt den att vara på.

Det första batteriet vi hade råkade vi dra ner till så låg spänning att det tappade mycket av sin kapacitet. För att motverka det i framtiden kopllade vi batterispänningen till en av processors analoga mätkontakter så att vi kan se batteriets spänning och varna när det blir

för lågt. Det gjorde det även möjligt att se om roboten körs från batteri eller USB. Detta är användbart eftersom man kan stänga av motorerna om den körs från USB för att förebygga att man skadar datorn. Man får inte dra mer än 500 mA från USB vilket inte räcker till motorerna. Vissa datorer kan ge mer och då funkar det, men eftersom vi inte skrivit någon USB kod så anger roboten inte att den behöver ström så som USB protokollet kräver. Det går inte bara att kolla om USB-kabeln är i eftersom både batteriet och USB kan vara inkopplat samtidigt. En diod ser till att elen då tas från batteriet.

Vid ett försök att minska mätfelet på sensorkortet så gick accelerometern, gyroskopet och magnetometern sönder. Detta skedde troligtvis på grund av ESD (Electrostatic discharge). Vi var därför tvungna att köpa nya sensorer och ESD-utrustning för att motverka problem i framtiden.

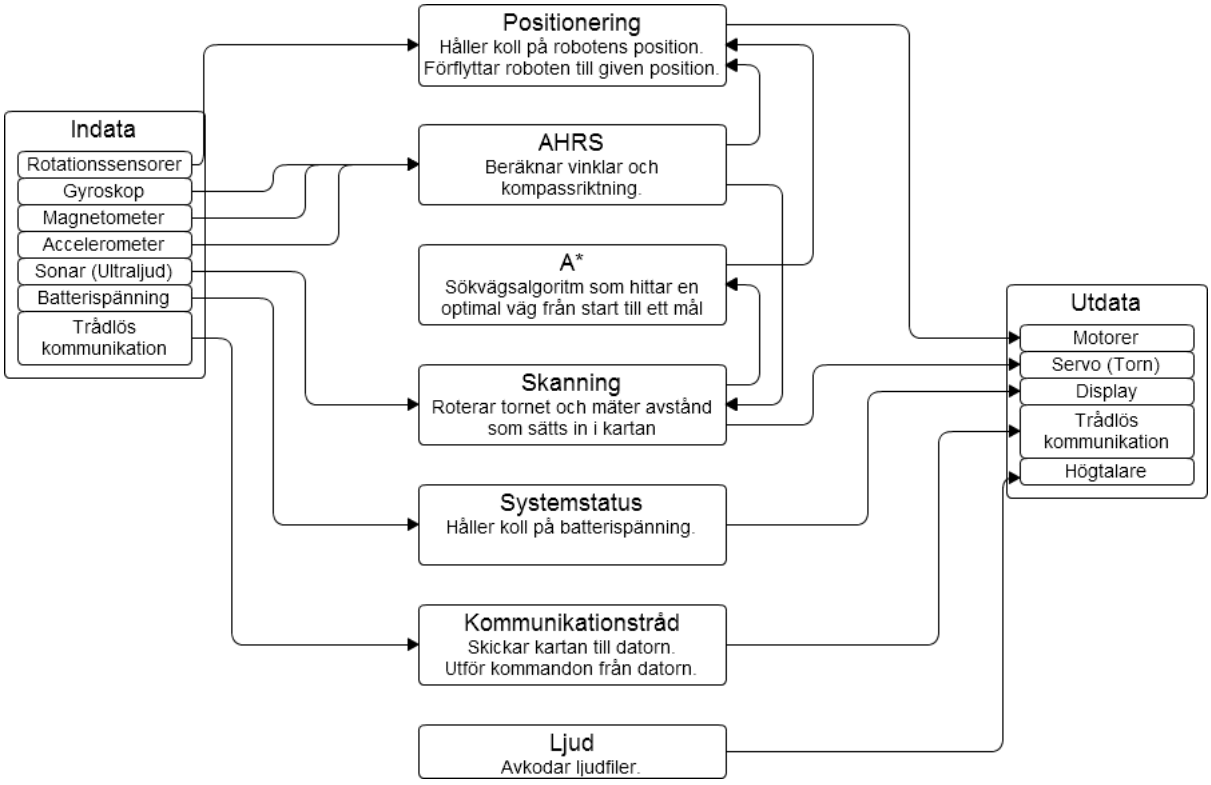


Här är en nyare bild på robotens elektronik. Intressanta saker som inte fanns i förra bilden är markerade.

Displaykortet (kortet längst ner till höger) sitter fast på två skruvar som höjer upp den över batteriet. Det sitter även ett litet kretskort för att styra kameran som ska monteras vid senare tillfälle. Det finns en expansionsport som ger el och kommunikation med processorn där man kan koppla in ytterligare moduler för nya funktioner. I tidigare versioner så halkade batteriet runt fritt när roboten körde. Vi limmade fast fyra L-formade metalbågar runt batteriet för att hålla det på plats.

2.2 Mjukvara

2.2.1 Strukturkarta



2.2.2 Verktyg

Vi har använt en hel del olika program och datorverktyg under projektets gång. Här är en del av dem.

- En IDE (Integrated Development Environment) som heter Eclipse. En IDE är en programmeringsmiljö som underlättar arbetet för programmeraren genom att förenkla eller helt automatisera vissa delar av utvecklingsprocessen.
- C kompilatorn “arm-none-eabi-gcc” som är gjord av Codesourcery för att kompilera C/C++-koden.
- Adobe Photoshop CS5 för att göra hjulbasens silhuett.
- Cacoo för att skapa strukturkartor till hårdvaran och mjukvaran.
- EAGLE för att skapa elektronikritning.

2.2.3 Utförande

Vi visste att programmeringen skulle vara det största jobbet så vi bestämde oss för att skapa en ordentlig arbetsmiljö där allting funkar från början istället för att börja halvdant och få problem på vägen. Vi började från början utan någon tidigare kod och det tog ett tag innan vi faktiskt skrev någon kod.

För att synkronisera projektet mellan medlemmarna använde vi versionshanteringssystemet SVN. Detta gjorde det enkelt för oss att programmera samtidigt eftersom SVN underlättar synkroniseringen av koden. SVN hanterade också skapandet av regelbundna säkerhetskopior. Detta är otroligt användbart när man jobbar i ett projekt med flera personer

och det har förmodligen sparat oss många problem som hade uppkommit annars. I många situationer kan det vara väldigt skönt att kunna gå tillbaka till en äldre version av koden.

Om läsaren vill kolla igenom så finns allting på svn://exuvo.se/kartobot . Den innehåller all kod till hela projektet samt all dokumentation till alla delar vi använt. Det finns också bilder vi tagit på roboten under projektets gång. Bilderna på roboten går också att hitta på http://imgur.com/a/tiBZL.

Programmeringsdelen kan delas upp i två separata delar. Först har vi koden som körs på roboten. Det är koden som hanterar allting som roboten gör. Det är här det största jobbet ligger. Denna kod är skriven i C [10]. För oss var C ett självklart val. Vi visste att vår mikrokontroller inte skulle ha någon imponerande prestanda och vi ville kunna skriva kod som är låg nivå (nära datorns inre) för att det skulle bli så effektivt som möjligt. C uppfyller alla dessa kraven.

Att få in koden i processorn är inte helt elementärt. Vi hade en arduino (en mikrokontroller) till hands som vi använde för att programmera in en USB-bootloader i processorn som gör det lätt att programmera den eftersom man då kan använda en simpel USB-kabel för att föra över koden.

Anton började programmera kommunikationen mellan datorn och roboten. Det var mer komplicerat än vi först hade tänkt oss. På grund av ett fel i standardbiblioteket från tillverkaren så tog det extra lång tid.

Vi delade upp arbetet mellan oss men eftersom Anton har mycket mer erfarenhet inom både hårdvara och mjukvara så har han gjort en del mer arbete än de andra. Med det extra arbetet som Anton lagt ner så finns det nu en fungerande LED-matris och ljud. Detta har varit användbart vid felsökning.

Per har skrivit koden som får robotens fyra motorer att röra sig samt koden som kontrollerar dessa så att roboten kan styra sig mot ett givet mål.

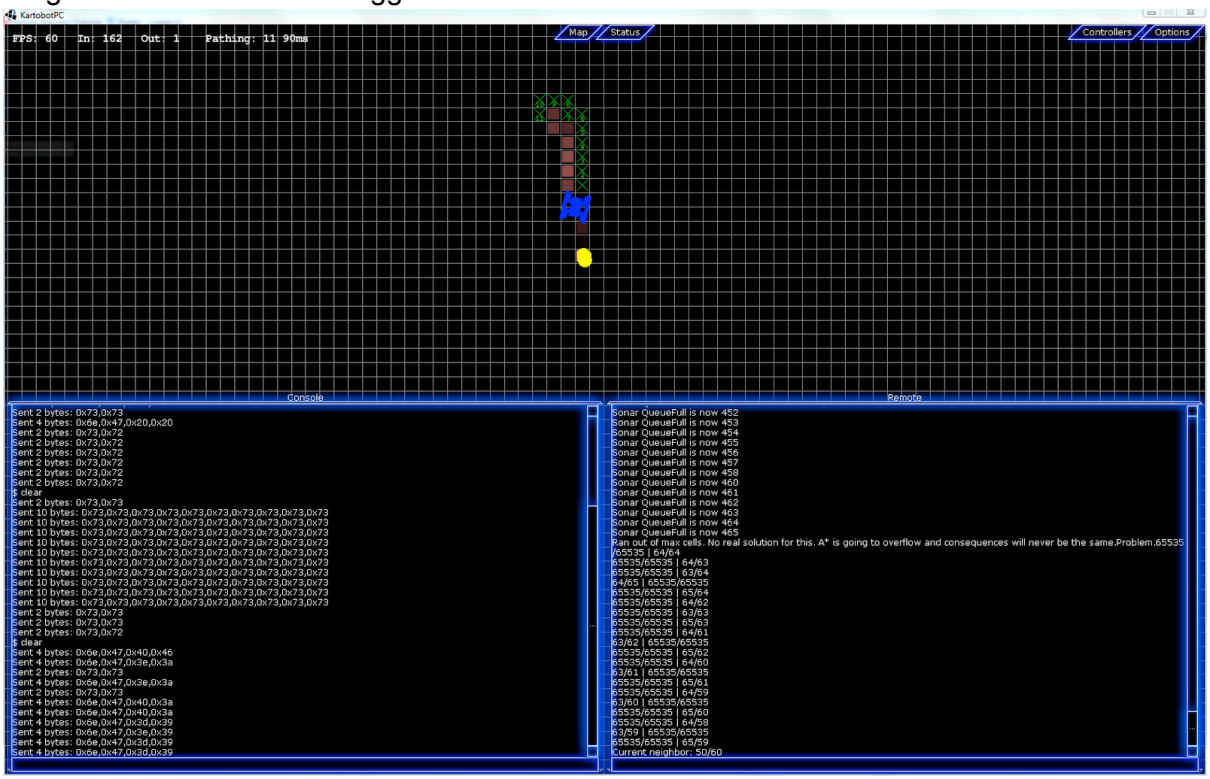
Erik har skrivit koden som finner den optimala vägen till ett givet mål. Vi använde en algoritm som heter A* (uttalas A star) [11]. Anledningen till att vi valde A* är för att den är en av de bästa och mest kända sökvägsalgoritmerna. Det finns alternativ som till exempel Djikstras algoritm. Dock tar den mer minne (vilket vi har väldigt ont om på grund av begränsningar i hårdvaran) och ofta mer tid. Dock är den enklare att skriva. Vi tog den svårare vägen för att vinna prestanda.

När sökvägsalgoritmen var klar märkte vi att den blev mycket långsam när den körde på roboten. Vissa sökningar kunde ta upp mot 10 sekunder. Erik letade runt lite efter optimeringar som kunde göra det snabbare. Den bästa lösningen som han hittade verkade vara att implementera “Binary heaps” [12] som är ett alternativt sätt att lagra datat. Det gör det snabbare att hitta den bästa noden vid ett givet tillfälle.

Ljudet spelar just nu ett eget format som liknar det som finns på gamla spelkonsoler. Den kan även spela WAVE-filer men eftersom de är okomprimerade så får de inte plats i processorns minne. Man skulle kunna lagra dem på roboten om man sätter dit ett sd-kort men det har ingen funktion för projektet så det har vi inte lagt ner tid på.
LED-matrisen skulle gå att använda för att visa de närmaste rutorna i rutnätet, men det har vi inte tid att implementera just nu. Det finns även en stapel med lysdioder som skulle kunna användas till att tydligt och lätt visa batteriets spänning.

Den andra delen av programmeringen är programmet som körs på datorn och är skriven av Anton och Per. Programmet består av ett fönster som har två stycken konsoler, en lokal och

en extern. Den lokala konsolen skickar kommandon till själva programmet och man kan genom denna t.ex. säga åt programmet att rensa kartan. Den externa konsolen skickar kommandon till roboten och man kan genom denna ge order till roboten som att t.ex. åka framåt. Vi bestämde oss snabbt att ha två skilda konsoler eftersom det annars blir väldigt rörigt om all data som ska loggas skrivs ut i en enda konsol.



Här är en bild som demonstrerar datorprogrammets användargränssnitt.

I samma fönster som de två terminalerna finns också en karta som representerar ett plan i rummet som roboten befinner sig i. Planet är ett rutnät som representerar hur roboten ser på det hela. Varje ruta representerar en kvadratdecimeter i det geografiska planet som går igenom tornet och är parallellt med golvet. Varje ruta har ett värde som är ett mått på hur svårgenomtränglig punkten bedöms vara för roboten. Varje gång ultraljudssensorerna känner av en punkt i en ruta, så ökas rutans värde. Om ultraljudet får flera mätpunkter i en ruta så ökar värdet ännu mer vilket leder till att den är ännu ovilligare att åka på den rutan och hellre försöker hitta en väg runt.

Det finns en anledning till att vi inte bara stänger av en ruta direkt om vi ser en punkt i den. Vi kan inte lita blint på värdena från ultraljudssensorerna eftersom mätfel inte är så ovanliga. Genom att istället öka hur svårt roboten tycker det är att ta sig igenom rutan allt eftersom fler punkter landar i rutan så påverkas roboten inte lika mycket av mätfelen.

Kartan är ett kraftfullt verktyg. Man kan snabbt navigera runt på kartan och man kan till och med zooma in och ut. Robotens nuvarande position syns på kartan och man kan manuellt sätta ett nytt mål för den att åka till. Kartan visar vägen till målet och den tar hänsyn till hindrena den upptäckt. Kartan uppdateras i realtid för att visa hur roboten uppfattar världen för tillfället och var den har hittat hinder.

3. Resultat

Vi lyckades inte uppnå alla de mål vi satt.

Ultraljudet för att mäta avstånd funkar ganska bra vid korta avstånd men det blir lite för mycket mätfel speciellt vid längre avstånd. Vi har motverkat mätfelen en del genom att ignorera punkter som är mer än två meter bort då vi ändå kommer åka närmare dom senare och få en bättre mätning. Mätfelen ger dock fortfarande stora problem. Den stora vinkeln där de känner av hinder leder till att smala hinder kan se väldigt breda ut på kartan.

Navigationskoden har lite problem som gör att den krashar ibland, oftast är det på grund av de minnesbegränsningar som finns. Vi har försökt motverka detta genom att minska rutnätets storlek från 128x128 till 64x64 dm men det är fortfarande svårt att navigera eftersom de flesta gånger som man söker efter en ordentlig väg till ett mål så kommer det ta för mycket minne.

Datorprogrammet funkar utmärkt och är väldigt stabilt. Det går bra att både styra och övervaka roboten. Det går att logga mätvärden för att senare generera grafer om så skulle behövas.

Positioneringskoden funkar ganska bra men har några problem. Största problemet är att kompasssensorn inte verkar vara helt linjär under ett varv. Ett varv är alltid 360 grader men en rotation på 90 grader uppmäts inte alltid som 90. Endast en av fyra hjulsensorer som ska känna av hur långt den åker fungerar. Signalerna verkar vara för svaga. Vi har öppnat upp dom och försökt laga dom men inte riktigt lyckats. Det finns ett hjul i varje sensor med svarta och vita cirkelsegment varav några vita har blivit smutsiga, men försök att göra rent dom misslyckades.

Vi hade en annan idé som kanske skulle funka men som vi inte hann inte implementera den. Det sitter ett par små kretskort som konverterar 5V signaler till 3V just nu, men vi har i efterhand upptäckt att dessa inte behövs och om vi tog bort dem skulle processorn få lättare att känna av rotationssensorerna.

Om man säger åt roboten att åka till en given ruta så tycker den att den kom fram dit men om man mäter avståndet den åkt så ser man att den inte alltid hamnar i rutan man valt utan ibland rutorna brevid. Detta beror på de två problemen som angetts ovan.

En av de största svårigheterna med att koda för små processorer är att man inte riktigt ser var eller varför det blir fel. Det blir mer ungefärligt som t.ex nånstans bland de här fem raderna. Ibland blir felen tydliga som t.ex ogiltig pekare på den här raden men inte så ofta som man skulle vilja. Oftast är det mer missledande en hjälpande eftersom en bit av kod kan ha fått en annan bit av kod att krasha genom att skriva över den. Då ser det ut som den andra biten kod krashat när det egentligen är i den första kodbiten som problemet ligget. Om man jämför med att koda för en vanlig dator där man ser exakt var och varför programmet krashat, så är det en ny utmaning.

4. Värdering av arbetet och resultatet

Under projektets gång så har vi naturligtvis gjort en hel del saker som vi utan tvekan skulle ändrat om vi gjort om allt från början. Vi startade med en alldeles för optimistisk tidsplan som vi var tvungna att korrigera väldigt tidigt under projektet. Det var inte sista gången vi behövde ändra tidsplanen.

Vad vi lärde oss från detta var att det är guld värt att ha planerat projektet så mycket som möjligt innan man börjar. Att behöva byta arbetsuppgifter mellan medlemmarna kan inte alltid undvikas men att planera ordentligt minskar risken för att det behöver bytas arbetsuppgifter senare.

Det hade varit en stor fördel om vi hade satt upp ordentliga delmål från början som vi sedan följt. Relevanta delmål är en otrolig hjälp för att få projektet att flyta på ordentligt och den gör det också lättar att fokusera på de relevanta delarna. Vi hade bara ett delmål som vi satte upp mot slutet av projektet. På grund av detta så hade vi väldigt länge en robot som bara kunde stå stilla och göra ingenting.

En viktig sak hade varit att alltid sträva efter simplicitet då det blir mindre saker kan gå fel. Det aningen komplexa digitala servot gav oss massor med problem till den punkt att vi fick byta ut det mot ett mycket simplare analogt servo. Det gav oss en sämre teoretisk precision men i praktiken en ökning då servot nu faktiskt funkade.

Vi gjorde dock en hel del bra saker i vårt projekt. Vi satte upp en SVN-server som tillät oss att programmera på projektet samtidigt. Den skötte också automatisk säkerhetskopiering. Det faktum att vi inte behövde skicka koden fram och tillbaks mellan varann eller oroa oss för att få tag i rätt version betydde att vi säkert undvikt en hel del problem som vi skulle stött på annars.

All kod funkar inte och det finns saker som vi skulle kunna implementera för ett förbättrat resultat. Men i grund och botten så finns det ett budgetproblem. Vi fick valet att få 500 kronor i sponsring av skolan men då behövde vi ge projektet till skolan när vi var klara. Med tanke på att projektet kostat oss 5,000 kronor som vi själva betalat så vill vi självklart inte ge bort den bara för att skolan betalar en bråkdel.

Med en liten budget får man nöja sig med lite sämre sensorer med mycket mätfel. Vi hade kunnat fixa alla buggar och implementerat alla möjliga fixar men i slutändan skulle den ändå inte bli mycket bättre än vad den är.

Det finns ju dock en hel del som vi inte han med men med tanke på budgeten och tidsgränsen så känner vi att vi har gjort ett bra jobb. Det är ett relativt avancerat projekt och vi har lärt oss mycket och kommit långt sedan vi började i höstas.

5. Referenser

[1] - http://www.araa.asn.au/acra/acra2000/papers/paper02.pdf 2012-03-30
[2] - http://en.wikipedia.org/wiki/Quadrotor 2012-03-30
[3] - http://www.sparkfun.com/products/10724 2012-03-30
[4] - http://www.sparkfun.com/products/10336 2012-03-30
[5] - http://www.sparkfun.com/products/9457 2012-03-30
[6] - http://www.sparkfun.com/products/9494 2012-03-30
[7] - http://www.sparkfun.com/products/9931 2012-03-30
[8] - http://www.sparkfun.com/products/9014 2012-03-30
[9] - http://www.sparkfun.com/products/8876 2012-03-30
[10] - http://en.wikipedia.org/wiki/C_(programming_language) 2012-03-30
[11] - http://en.wikipedia.org/wiki/A* 2012-03-30
[12] - http://www.policyalmanac.org/games/binaryHeaps.htm 2012-03-30