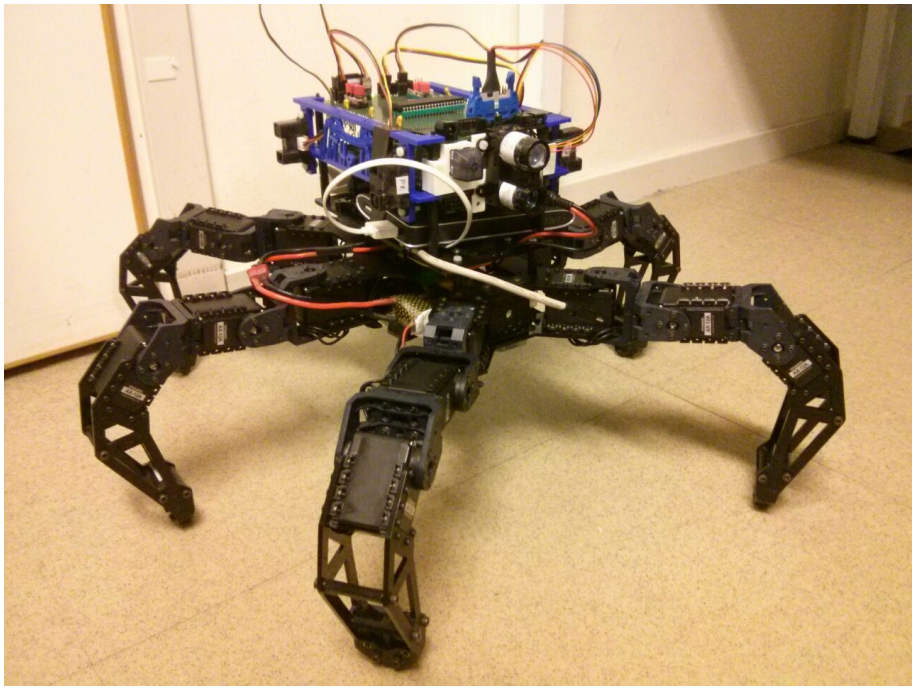


Teknisk dokumentation för sexbent robot

Noak Ringman
Emil Segerbäck
Frans Skarman
Robin Sliwa
Hannes Tuhkala
Malcolm Vigren
Olav Övrebö

Version 1.0

2017-01-27

Projektidentitet

Grupp 9, Ht 2016, LiTHe Hex

Linköpings Tekniska Högskola, ISY

Namn	**Ansvar**	**E-post**
Emil Segerbäck		emise935@student.liu.se
Frans Skarman	Dokumentansvarig	frask812@student.liu.se
Hannes Tuhkala		hantu447@student.liu.se
Malcolm Vigren	Projektledare	malvi108@student.liu.se
Noak Ringman		noari093@student.liu.se
Olav Övrebö		olaov121@student.liu.se
Robin Sliwa		robsl733@student.liu.se

Kursansvarig: Tomas Svensson Rum 3B:528 013–28 13 68 tomas.svensson@liu.se

Innehållsförteckning

Dokumenthistorik

Version	Datum	Utförda förändringar	Utförda av	Granskad
1.0	2016–12–14	Första version	Projektgruppen	Projektgruppen

1 Inledning

Detta dokument går i detalj in på konstruktionen och mjukvaran för en sexbent robot, med autonom och manuell styrning. Först presenteras en översiktlig beskrivning av robotens styrande processorer (”enheter”, vilka kopplas till ett färdigt PhantomX AX Metal Hexapod Mark III chassi från TrossenRobotics). Dessa enheter (central-, motorik- och sensorenhet), kommunikation mellan dessa, samt användargränssnitt specificeras i större noggrannhet under egna rubriker.

2 Komponenter

Tabell 1 visar komponenter som ingår i roboten, utöver chassiet, virkort och andra diverse mindre komponenter.

Komponent	**Antal**
Raspberry Pi 3	1
ATmega1284	2
GP2Y0A02YK	4
LIDAR Lite V3	1
74LS244	1
Extern 16 MHz-klocka EXO-3	1
TXB0104	2

Tabell 1: Komponenter som ingår i roboten

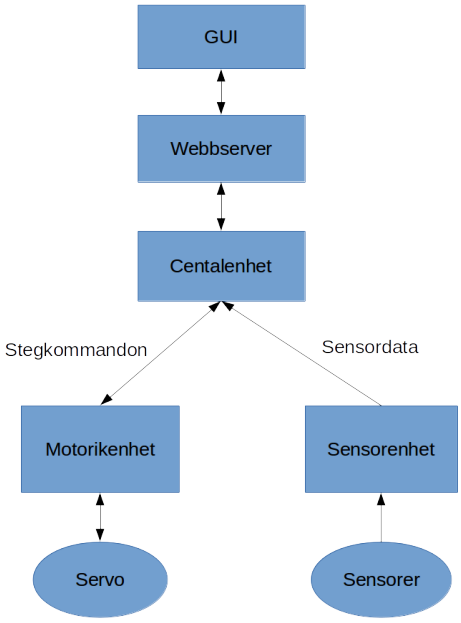
3 Systemöversikt

Systemet innehåller tre enheter. En centralenhet implementerad på en Raspberry Pi 3 agerar som master för de andra två (slav-)enheterna. Denna står för all kommunikation med omgivningen genom en webbserver som datorer kan koppla upp sig mot för att styra roboten eller få tillgång till debugdata. I autonomt läge är det även denna enhet som utifrån insamlad data fattar alla styrande beslut utifrån vilka de andra enheterna verkar. Vid manuell styrning vidarebefordras data från styrdatorn.

De två andra enheterna (slavarna till centralenheten) utgörs av AVR-processorer och styr mer specifika uppgifter. Den ena, motorikenheten, översätter generella indikationer om förflyttning från centralenheten till faktiska kommandon för robotens servon och exekverar dem.

Den andra AVR-processorn utgör en sensorenhet, vilken styr och läser robotens sensorsvit. Den svarar även för att brusreducera sensorernas råa indata.

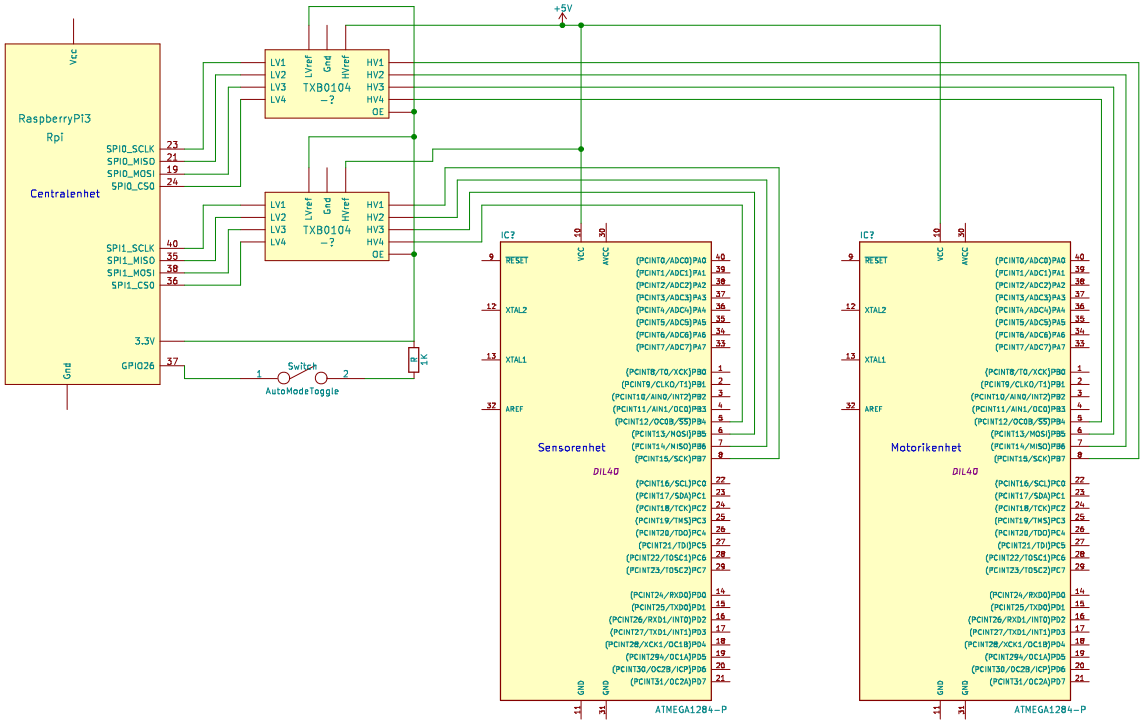
Figur 1 ger en översiktsbild av systemet.



Figur 1: Översikt av systemet

4 Kommunikation mellan enheterna

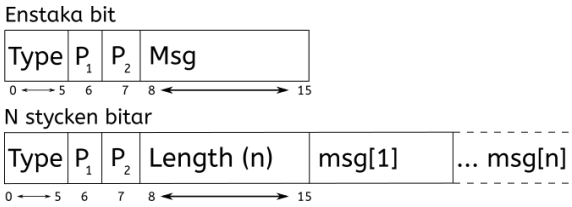
Kommunikation mellan centralenheten och de två AVR-processorerna sker med SPI. Sensor- och motorikenhetens SPI-portar är kopplade till varsin SPI-port på centralenheten. Detta illustreras i figur 2. All kommunikation inleds med att centralenheten skickar ett kommando eller en databegäran till slavenheterna. Slavenheterna utför sedan kommandot eller svarar med den begärda datan.



Figur 2: Kopplingsschema för centralenheten och SPI-kommunikationen

4.1 Kommunikationsprotokoll

Kommunikationen mellan centralenheten, motorikenheten och sensorenheten sker med protokollet som beskrivs i figur 3.



Figur 3: Översiktlig vy av kommunikationsprotokolllet

De första 6 bitarna av varje meddelande säger vilken typ av meddelande det är som skickas, dessa beskrivs i tabell 2. Den första biten i typparametern är 0 om meddelandets innehåll är en enstaka byte och 1 om meddelandet har dynamisk längd. Om längden är dynamisk är första byten i meddelandet längden av resten av meddelandet.

De två sista bitarna i första byten är paritetsbitar. Bit 7 är en paritetsbit för meddelandets innehåll, inklusive längd-byten, medan bit 8 är paritetsbit för de 7 tidigare bitarna.

Om någon av paritetsbitarna är fel så kommer enheten som tog emot meddelandet att svara med ett speciellt meddelande för att indikera sändningsfel. Annars kommer den att svara med ett ”acknowledgemeddelande”.

Syfte	ID	Data
Generella		
Send fail	1F	–
Acknowledge	0F	–
Datarequest	02	data
Centralenhet → motorikenhet		
Sätt hindergång	03	på/av
Sätt servohastighet	20	8 LSD, 8 MSD
Gåkommando	21	len, x- och y-hastighet, svänghastighet, autonom-läge
Return to neutral	05	–
Centralenhet ← motorikenhet		
Servostatus	22	len, ... data ...
Debugsträng	23	len, ”debugsträng i utf-8”
Upptagen med rotation?	03	ja: 0x01, nej: 0x00
Centralenhet ← sensorenhet		
Behandlad sensordata	24	len,ir-ner, ir-fv, ir-bv, ir-fh, ir-bh, lidar-lsd, lidar-msb
Korridordata	25	len, avst. fram, vänster, höger, ned, vinkel

Tabell 2: Meddelandespecifikation

4.2 Centralenheten

Centralenheten använder SpiDev-modulen för python för kommunikation med motorik- och sensorenheten över SPI. Den skapar en instans av SpiDev för varje enhet, då centralenheten använder två separata SPI-bussar för varje enhet. Alla funktioner i centralenheten som rör kommunikationen till AVR-processorerna tar alltså in instansen för den enhet som ska kommuniceras till.

Pseudokoden för läsning av spi-meddelanden beskrivs som följande:

```
def read_bytes(spidev):

    type_byte = spidev.read_single_byte()

    if is_multibyte_msg(type_byte):
        length_byte = spidev.read_single_byte()
        return spidev.read_several_bytes(length)
    else:
        return spidev.read_single_byte()
```

Algoritmen för att skicka data till AVR-processorerna ser lite annorlunda ut beroende på vad som ska skickas. Det som är gemensamt med alla transaktioner till enheterna är dock att först skickas en *garbage*-byte (definierad 0x00) som ska ignoreras, då AVR-processorerna inte alltid lyssnar på den första byten. Ifall AVR-processorerna märker att den första byten är en *garbage*, så struntar de alltså i den, annars behandlar den meddelandet som vanligt. Det betyder att inga data-ID:n kan ha samma värde som *garbage*-byten.

Efter att *garbage*-byten är skickad, skickar enheten hela meddelandet enligt protokollet, alltså genom att beräkna och lägga till paritetsbitar i byten för data-ID, skicka längden ifall det är flera bytes i meddelandet, och slutligen skicka de bytes som ska skickas.

Algoritmen för mottagning och behandling av meddelanden på motorik- och sensorenheten körs i en avbrottsruting som beskrivs av följande pseudokod:

```
def on_spi_interrupt():

    received_frame = receive_frame()
    success = check_parity(received_frame)

    if success:
        send_acknowledge_message()
    else:
        send_fail_message()
        return

    if received_frame.type == DATA_REQUEST:
        # message requires reply
        reply_frame = build_reply_frame()
        send_frame(reply_frame)
    else:
        # message contains command to be processed
        handle_frame(received_frame)
```

Frame syftar på en datastruktur som innehåller all relevant information om meddelandet, alltså typ, längd och data.

5 Centralenheten

Centralenheten har två ansvarsområden: navigation/beslutsfattning samt kommunikation med omvärlden. Detta görs i två lägen, manuellt och autonomt läge. För att byta mellan dessa två lägen kan man antingen använda sig av den fysiska knappen eller knappen i användargränssnittet.

5.1 Navigation och beslutsfattning

Centralenheten sköter all beslutsfattning om hur roboten ska röra sig genom labyrinten. Den tar emot data från sensorenheten och GUI:t och skickar kommandon till motorikenheten.

5.1.1 Autonomt läge

I det autonoma läget frågar centralenheten om data från sensorenheten varje gång det ska fattas ett beslut. Detta gör att centralenheten själv bestämmer när den ska ha ny data för att den inte ska bli avbruten eller missa data för att den är mitt i utförandet av en annan funktion. Utifrån denna data fattar centralenheten beslut om hexapodens färdriktning.

I beslutsfattning ingår bland annat korridorreglering, svänga in i korridorer samt detektera och undvika återvändsgränder. Beslutsfattningen innehåller en lista med de tre senaste besluten och det är endast när alla dessa beslut är samma som beslutet uträttas.

Korridorregleringen sker med hjälp av de fyra IR-sensorerna som sitter på sidorna av roboten. En vinkel räknas ut med data från dessa som säger hur roboten är vinklad i en korridor. Regleringen sker samtidigt som när roboten går framåt för att få jämnare gång i korridoren. Detta uppnås genom att en offset från mitten av korridoren beräknas och används när roboten ska vrida sig tillbaka till mitten. Ju närmare mitten den är desto mindre kommer den att vrida sig mot mitten.

Förutom de fyra IR-sensorerna på sidorna används även LIDAR när beslut ska fattas. LIDAR-sensorn används för att undvika ingång i återvändsgränder framåt. För att detektera korridorer och återvändsgränderna på sidorna används sidosensorerna. Inget beslut fattas innan båda sidosensorerna på samma sida har detekterat samma sak, så att risken för felbeslut minskar.

När ett beslut om rotation har tagits väntar centralenheten tills att en hel rotation har skett. Detta görs genom att vänta tills roboten står rakt framåt m.h.a. den uträknade vinkeln men även genom att vänta tills motorikenheten svarar med att en hel rotation har gjorts klart.

Pseudokoden nedan beskriver hur beslutsfattning sker på centralenheten.

```
def get_decision(sensor_data, decision_packet):
    corridors_and_dead_angles =
        get_corridors_and_dead_angles(sensor_data)

    for value in corridors_and_dead_ends:
        if 3 corridors detected:
            decision = TURN_LEFT

        elif 2 corridors detected:
            if distance forward is greater than DEAD_END_DISTANCE:
                decision = GO_FORWARD
            else:
                if left corridor detected:
                    decision = TURN_LEFT
                else:
                    decision = TURN_RIGHT

        elif 1 corridor detected:
            if distance forward smaller than one tile
               and greater than LIDAR_STOP_DISTANCE:
               decision = GO_FORWARD
            elif left corridor detected:
                decision = TURN_LEFT
            elif right corridor detected:
                decision = TURN_RIGHT
            else:
                decision = GO_FORWARD

        elif 0 corridors detected:
            if distance forward is greater than LIDAR_STOP_DISTANCE:
                decision = GO_FORWARD
            else:
                decision = TURN_LEFT

if previous decision is TURN_LEFT or TURN_RIGHT:
    # A complete turn has been made
    if absolute value of the average angle is less than 10 degrees and
    turn has completed:
        decision = GO_FORWARD
        previous_decision = COMPLETE_TURN

elif previous decision is COMPLETE_TURN:
    decision = GO_FORWARD
    previous_decision = COMPLETE_TURN

    if robot is inside a corridor:
        previous_decision = GO_FORWARD
```

5.1.2 Manuell navigation

För den manuella navigationen används RabbitMQ, som är en kommunikationsserver mellan webbservern och centralenheten. Det kommer in ett nytt meddelande med data för varje knapptryck/händelse som sker på GUI:t. Detta tolkas av centralenheten och överförs till motorikenheten.

5.2 Kommunikation med webbserver

Centralenheten kommunicerar med en dator över WiFi. Datorns webbläsare ansluter till centralenhetens webbserver och då kan man via webbgränssnittet styra roboten samt läsa av robotens olika data.

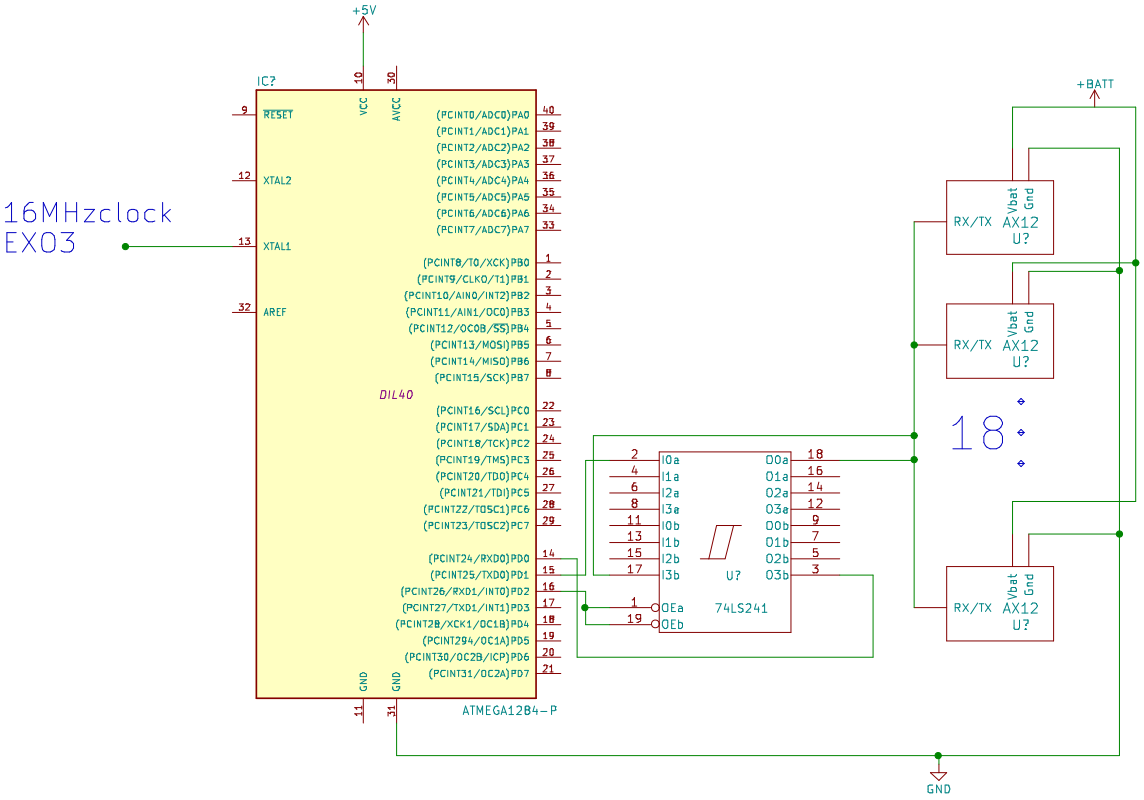
Centralenheten skickar och hämtar kontinuerligt meddelanden i JSON-format från RabbitMQ-servern. Centralenheten skickar den data som GUI:t behöver till RabbitMQ-servern och webbservern läser sedan in den data som finns där. Den data som skickas mellan centralenheten till webbservern kan ses i tabell 3 och den data som skickas från webbservern till centralenheten kan ses i tabell 4 i avsnitt 8.4.

För att resten av centralenheten ska ha så enkel interaktion med webbservern som möjligt, skapas två stycken trådsäkra köar, en för mottagning av meddelanden och en för att skicka meddelanden till servern. Dessa köar skickas till en separat tråd som i sin tur skapar en tråd som kontinuerligt lägger meddelanden från servern i mottagningskön, och en tråd som skickar meddelanden som lagts i sändningskön. Detta gör att resten av centralenheten endast behöver lägga paket i sändningskön för att skicka data, och hämta ett datapaket ur mottagningskön för att ta emot data, då alla detaljer med kommunikation till servern sker i bakgrunden.

6 Motorikenheten

Motorikenheten översätter kommandon från centralenheten till servokommandon. Den tar emot instruktioner från centralenheten som anger steglängd, fart, rotation och riktning för förflyttnnig. Motorikenheten behandlar detta, räknar ut en lämplig gångstil och signalerar nödvändiga vinklar till de sex benen. Vissa delmoment i förflyttningen görs hårdkodade. Ett kretschema för motorikenheten finns i figur 4.

För att kunna kommunicera med servona i 1 MBAUD klockas motorikenheten med en 16 MHz EXO3 klocka.



Figur 4: Kretsschema för motorikenheten

6.1 Kommandotolkning

Motorikenheten tar in ett flyttalspar med koordinater för önskad målposition (önskat center för roboten efter förflyttning), med x-koordinat fram och y-koordinat åt vänster om roboten, och med origo i robotens mittpunkt. Om totalavståndet origo till målposition är mindre än 1 enhet skalas rörelsen ned utifrån detta avstånd. Det vill säga, om ett kommando om en enhetsdistans (maximal steglängd) leder till ett 1,6 decimeters kliv, leder halva enhetsdistansen till ett ca: 0.8 decimeters kliv givet samma startposition och riktning.

Vidare ges den rotation som roboten skall rotera, angiven positiv vänster med 0 framåt, och en flagga för om roboten är i autonomt eller manuellt styrt läge. Om roboten är i autonomt läge, tolkas rotation utan koordinater som att en strikt rotation skall utföras (det vill säga, att roboten inte skall ändra rörelse förrän den utfört rotationen i fullo). I alla andra fall tar roboten in ett nytt kommando efter att ett enda steg utförts i indikerad riktning.

6.2 Planläggning och utförande

Första steget i motorikenheten för en anpassad gångstil är att utifrån efterfrågad förflyttning avgöra en önskad slutposition för respektive ben, som för roboten närmare det läge indikerat av centralenheten. Sedan avgörs den nedskalning av rörelsen som för varje set om tre ben (vänster fram-, höger mitt-, vänster bak-ben alternativt höger fram-, vänster mitt-, höger bak-ben) är nödvändig för att robotbenen skall kunna nå positionen utan att ben skär varandra, och utan att gå utanför benens räckvidd eller in under roboten. Skalningen görs först utan rotation, så att den skalade målförflyttningen är precis uppnåbar, varefter omberäkning och skalning görs med rotation och ny, nedskalad målposition. Detta gör att rotationen inte skalas ner till nära nollvärd, bara för att förflyttningen behöver skalas kraftigt. Denna dubbla skalning görs två gånger, en med vardera uppsättning ben stegande frammåt, och roboten använder sig av det alternativ där markfästa bens förflyttning är störst för att ta fram benrörelsen, då dessa ger den faktiska förflyttningen av chassit. En direkt nedskalning av förflyttningsvektor från nuvarande till målposition för fötter görs inte, då detta vid upprepad rotation (där benen rör sig i en ark runt roboten) skulle leda till successiv nedkortning av fötternas avstånd till kroppen.

Beräkning av nedskalning sker i tre avseenden. För det första skalas benförflyttningarna ner så att de inte tar något ben utanför dess räckvidd. För det andra skalas rörelserna så att benen inte skär en uppsättning hårdkodade gränser mellan närliggande ben eller mellan fötterna och robotchassit. Avslutningsvis görs en nedskalning utifrån angivet kommando, så att steglängd kan justeras av användaren. Om ingen rörelse anges, kommmer roboten att förbli stillastående.

Beräkningen av benposition sker endast i planet; höjdled tas i beräkning endast inför åberopning av inverterad kinematik, för bestämning av servovinklar. Själva klivet sker genom en sekvens av positionsinställningar med höjning och sänkning av ben, och stegvis övergång från nuvarande till önskad position. För en översiktlig inblick i gångstilen, se figur 5. Figuren visar beteendet specifikt i manuellt läge; i autonomt läge tillkommer möjligheten att med ett funktionsanrop rotera en strikt vinkel på plats. Koden för ett enda steg, illustrerat i figuren, returnerar andelen av den begärda rörelsen som slutförts, så för en strikt rotation beräknas kvarvarande vinkel att rotera. Vid strikt rotation gör även roboten en återgång till en neutral position före och efter själva rotationen.

Nedan finns pseudokod för den mainloop som körs på motorikenheten. Besluten

fattas utifrån en struct som ändras av SPI via avbrott.

```
def motorik_mainloop():
     stand_robot

     while true:
         if should move or rotate:
             if autonomous and only rotation:
                 rotate_until_done
             else
                 take_one_step_in_requested_direction
```

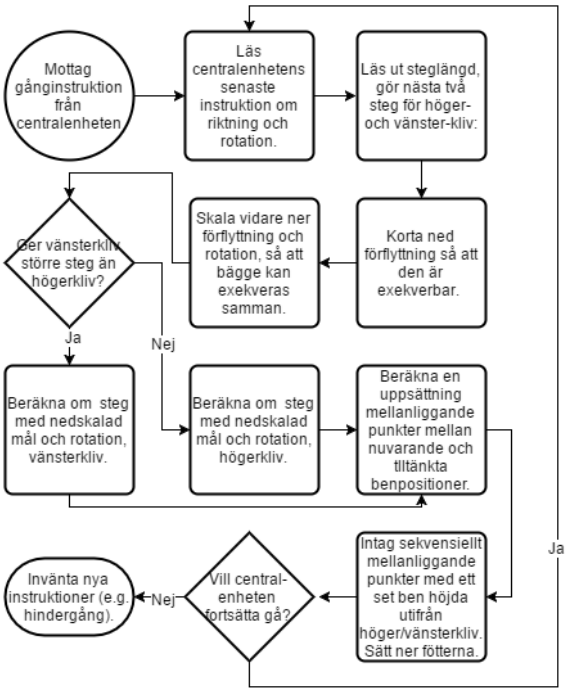
Vidare beskrivs nedan förenklat funktionaliteten för *work_towards_goal* i gangstil.c, i pseudokoden skriven som *take_one_step_in_requested_direction* för tydlighet:

```
def take_one_step_in_requested_direction(rotation, goal_position,
    current_leg_placement):
   scale_down_leg_movement_to_executable(no_rotation,
       leg_set_1_raised, goal_position)
   scale_down_leg_movement_to_executable(rotation, leg_set_1_raised,
       downscaled_goal_position)

   scale_down_leg_movement_to_executable(no_rotation,
       leg_set_2_raised, goal_position)
   scale_down_leg_movement_to_executable(rotation, leg_set_2_raised,
       downscaled_goal_position)

   if(leg_set_1_raised_movement > leg_set_2_raised_movement):
      execute_scaled_movement(leg_set_1_raised)
   else:
      execute_scaled_movement(leg_set_1_raised)
   return downscale_of_movement
```

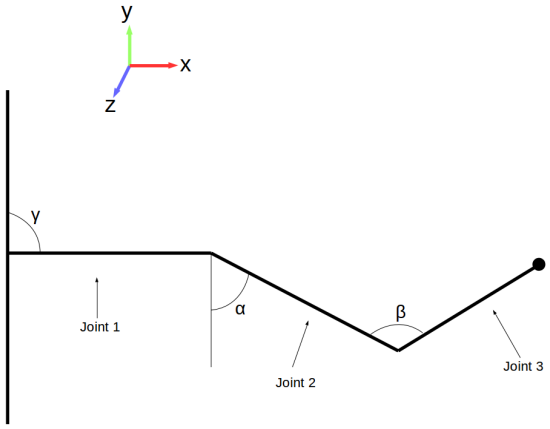
6.3 Inverterad kinematik

Då planläggningsdelen av motorikenheten beslutat om var ett ben skall flyttas beräknas lämpliga vinklar för benets alla servon genom inverterad kinematik. Utifrån kända längder på benen och positioner där man vill att fötterna ska placeras används trigonometri för att avgöra slutgiltiga servovinklar.

Inverterad kinematik är ett 3D-problem men vi har delat upp det i två stycken 2D-problem istället. Y-axeln går rakt upp, x-axeln går rakt ut från roboten i benfästets riktning och z-axeln går i riktningen $x \times y$ längs med roboten. Det första 2D-problemet beräknar vinkeln mellan roboten och den innersta leden γ, det andra 2D-problemet beräknar vinkeln α mellan den första och andra leden samt vinkeln β mellan den andra och tredje leden. Se figur 6.



Figur 5: Flödesschema för normal gångstil hos roboten



Figur 6: 3D-problemet för IK med ett ben

6.4 Kommunikation med servon

Kommunikation med servona sker via en half-duplex UART-länk.
AVR-processorns RX och TX pinnar är kopplade till var sin port på en 74LS241 tri-state-grind. Portens riktning styrs med en vanlig IO pinne på processorn som sätts till HIGH när data ska skickas och LOW när data tas emot.

6.4.1 Skrivning av data

För att undvika onödiga svar från servona skrivs data till dem med hjälp av REG_WRITE instruktionen som sparar data i en buffer på servot tills en ACTION instruktion skickas. Actioninstruktionen skickas med funktionen send_servo_action.

6.4.2 Läsning av data

För att läsa data från servona används funktionen read_servo_data som skickar en databegäran till servona och läser resultatet. Begäran görs med hjälp av en READ_DATA instruktion där addres och mängd bitar som ska läsa specificeras.

Datan som returneras sparas i en struct som innehåller fält för metadata (ID, errorflaggor, längd etc.) samt en array som innehåller den lästa datan.

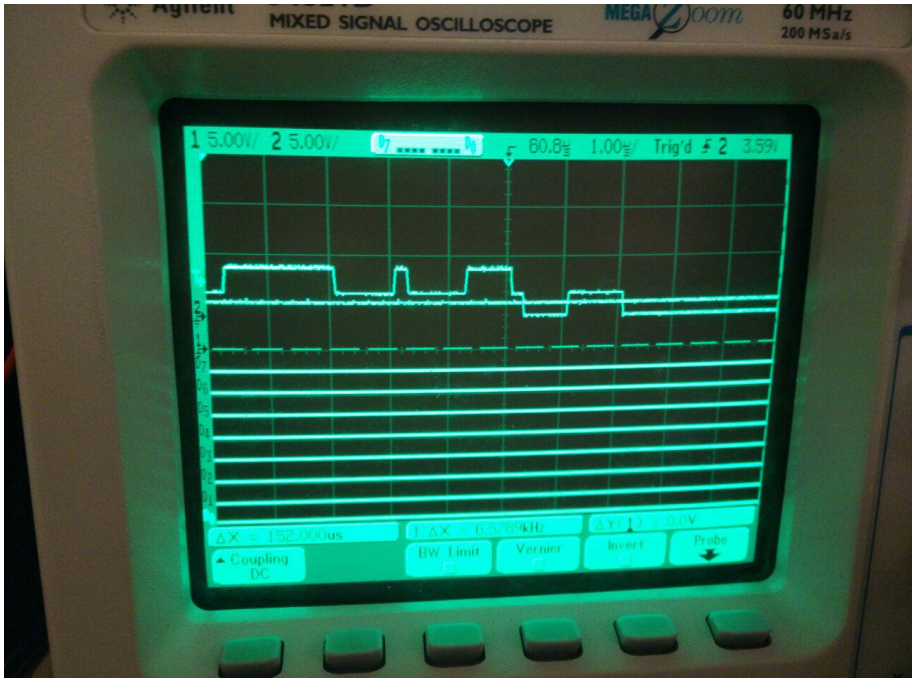
6.4.3 Väntning på servorotation

För att inte behöva gissa hur lång tid ett steg kommer att ta så returnerar inte send_servo_action fören alla servon är nära sin målvinkel. Vad som räknas som ”nära” beror på vilken del av steget som utförs och skickas som parameter till send_servo_action.

För att se när rotationen är klar läses först varje servos nuvarande målvinkel in och sparas i en array. Varje servos nuvarande vinkel läses sedan in och jämförs med målet. Om skillnaden mellan nuvarande vinkel och målvinkel är under gränsen för ”närhet” för varje servo så räknas rotationen som klar och funktionen returnerar.

6.4.4 Undantag för kommunikation

På grund av ett okänt fel i hård- eller mjukvaran blir UART-signalen fel om data begärs från de 3 första servona på roboten. En bild på fenomenet kan ses i figur 7. Eftersom att flera ben kommer att göra ungefär samma rörelser samtidigt ignoreras därför de tre första servona vid beräkningar där data från servona behövs.



Figur 7: Felaktig servokontrollsignal

7 Sensorenheten

Sensorenheten är den enhet som ger roboten sinnen för omvärlden, så att den kan navigera autonomt genom labyrinten. Sensorenheten sköter allting som rör läsning och brusreducering av sensorerna och skickar dem till centralenheten på begäran.

7.1 Processorn i sensorenheten

Sensorenheten innehållr en ATmega1284-processor, som läser av och behandlar informationen från robotens sensorer. Sensorenheten sköter även kommunikationen med centralenheten. Den tar emot förfrågningar om data och skickar behandlad data på begäran.

Sensorenheten behandlar den råa sensordatan till den grad att centralenheten inte behöver göra allt för många egna beräkningar på sensorinformationen. Den sköter därför bland annat brusreducering av data. ATmega1284 har tillräckligt med beräkningskraft för att ta hand om den data som sensorerna genererar. Den kritiska delen är istället A/D-omvandling. Eftersom vi har många IR-sensorer tar det tid att mäta och omvandla de analoga signalerna till digitala värden. Att starta en IR-sensor tar 52 900 mikrosekunder och en A/D omvandling kan ta upp till 260 mikrosekunder.

7.1.1 Huvudloop

Sensorenheten kör en huvudloop som hanterar en kö som varje ir-sensor är placerad i. Den första ir-sensorn i kön är den första sensorn som kommer att ha ett nytt värde. Tiden sedan förra mätningen av IR-sensorn beräknas och det är inte möjligt att läsa en IR-sensor innan tiden passerat. Om det det finns ett nytt värde görs en A/D-omvandling. Data från omvandlingen sparas för den ir-sensorn tillsammans med äldre värden och sen utförs en brusreducering. När det inte finns någon IR-sensor som har ett nytt värde läses det värde som LIDAR skickar. Sedan uppdateras den tabell som innehåller den data som är klar att skickas till centralenheten. Pseudo-kod för huvudloopen kan ses nedan.

```
 # Main loop
def main():
    port_init() # Setup hardware ports on AVR
    timer_init(timer8)
    timer_init(timer16)
    spi_init()
    ir_init()
    ir_queue_init(ir_queue) # Setup queue for scheduling of ir sensors
    adc_init() # Setup interna A/D-converter on avr
    lidar_init(lidar)

    for ir in NUM_SENSORS:
        schedule(ir)

    # Contains data ready to be sent to central unit
    table_init(table)

    while(true):

        if (has_new_value(ir_queue)):
            ir = dequeue(ir_queue)
            ir_add_data(ir, adc_read(ir))
            ir_reduce_noise(ir)
            schedule(ir_queue, ir)

            lidar_measure(lidar)

        # Update table with new data values
        update(table, lidar)

# Schedule ir sensor to read it when has new value
def schedule(ir_queue, ir):
    e = ir
    e.last_time_measured = time(ir_queue.timer)

    ir_queue.elements[ir_queue.current_size] = e
    ir_queue.current_size++
```

```
# Calculate if ir sensor has new value
def has_new_value(ir_queue):
    if (ir_queue.current_size == 0)
        return false
    else:
        return time(ir_queue.timer) -
            ir_queue.elements[0].last_time_measured >= IR_UPDATE_TIME
```

7.1.2 Avbrott

Följade avbrott är aktiva:

- Avbrott 20 genereras när någon bit i registret för SPI-kommunikationen ändras. Det används för kommunikationen över SPI med centralenheten. Avbrottsrutinen för SPI genereras vid första byte från centralenheten sedan sköts läsning av resterande bytes samt evenetuellt svar i samma avbrott.
- Avbrott 16/19 genereras när Counter1/0 når sitt maxvärde (overflow). Det användas för att räkna upp antal overflows vilket gör att vi hela tiden vet hur lång tid det passerat sedan start.

7.2 Avståndssensorer fram

Sensorenheten använder en avståndssensor på framsidan för att upptäcka väggar.

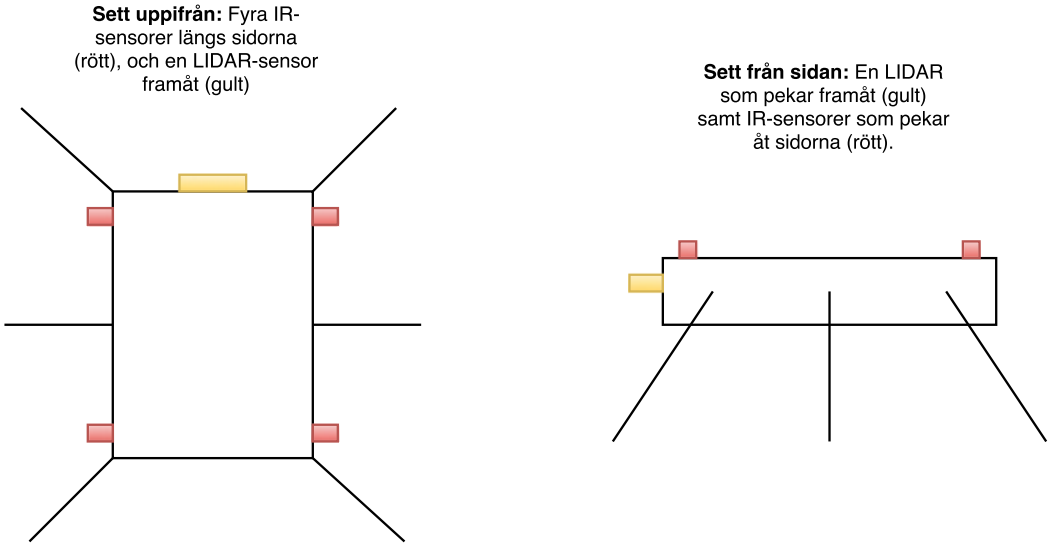
Avståndssensorn som mäter framåt ska utgöras av en LIDAR-sensor, som har ett mycket brett mätspann (0-40 m) med 1 centimeters upplösning. LIDAR-sensorn är ansluten med PWM och längden av varje puls motsvarar den uppmätta distansen. Alltså för att beräkna värdet från LIDAR mäter vi tiden mellan att signalen går hög och att den går låg så att vi vet hur lång pulsen var.

Monteringen av dessa sensorer illustreras i Figur 8.

7.3 Avståndssensorer åt sidorna

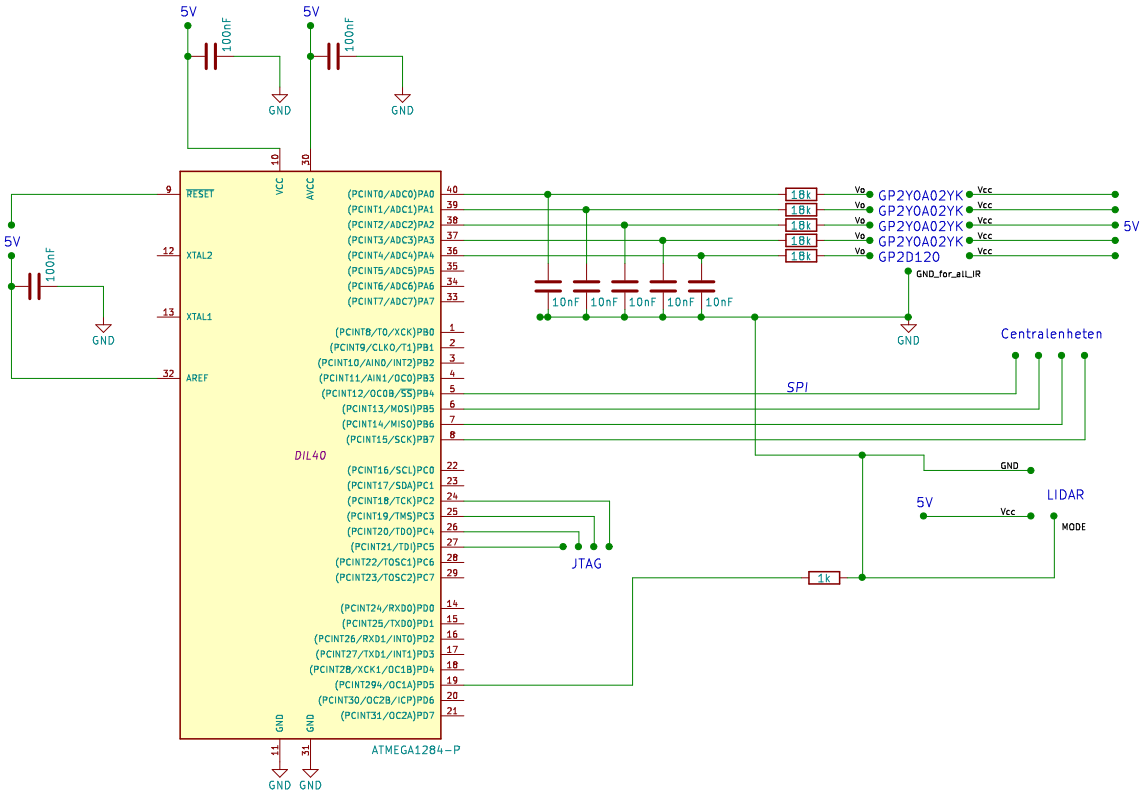
Roboten har avståndssensorer på vardera långsida av roboten, för detektion av väggar och återvändsgränder. Dessa sensorer utgörs av två IR-sensorer på varje sida där båda pekar ut från samma sida och är placerade på varsin ända av plattformen. IR-sensorerna som är placerade på sidan är typen GP2Y0A02YK som kan mäta avtånd mellan 20 och 150 cm. Detta spann gör det möjligt att navigera i korridorer och upptäcka återvändsgränder. Med två IR-sensorer på varje sida möjliggörs mätning av hur roboten är vriden i en rak korridor, vinkeln roboten har jämfört med korridoren används av regleralgoritmen i centralenheten.

Monteringen av dessa sensorer illustreras i Figur 8.



Figur 8: Montering av avståndssensorer

7.4 Krets



Figur 9: Kretschema över sensorenheten

Ett antal lågpass-filter används för filtrering av brus från IR-sensorerna och utgörs av en kondenstator på 10nF och en 18kOhm-resistor för varje sensor. Filtret är kopplat mellan sensorerna och de analoga PA-ingångarna på processorn.

LIDAR-sensorn är ansluten via PWM till PD5 på processorn.

PC2 upp till PC5 används som anslutning till JTAG, för programmering av processorn, och PB4 upp till PB7 är anslutna via SPI till centralenheten.

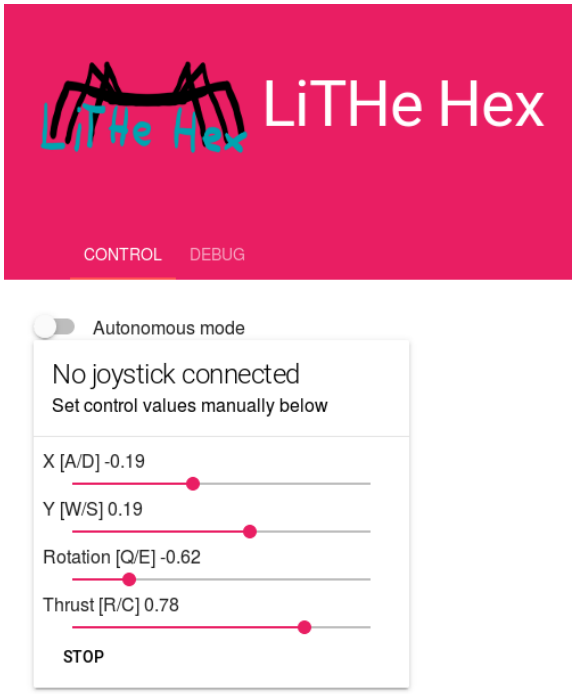
Sensorenheten är även förberedd för montering av en IR-avståndssensor av typen GP2D120, med mätspann 4-30 cm, som skulle kunna användas för detektion av hinder. Denna sensor är dock för närvarande inte ansluten, då roboten inte är gjord för hindergång, men sensorenheten mäter ändå det analoga värdet från porten där sensorn skulle kopplats in och skickar det till centralenheten, som skickar värdet vidare till GUI:t. Det enda som behöver göras för mätning och användningen av den sensorn är således att koppla in den enligt kretsschemat i figur 9 samt behandla värdet i centralenheten.

Figur 9 visar ett kretschema över sensorenheten.

8 Grafiskt användargränssnitt

På centralenheten körs en webbserver som tillhandahåller ett gränssnitt för användaren där man kan läsa sensordata eller kontrollera roboten i manuellt läge. Front-end-delen är skriven i Elm och back-end-delen i Elixir med hjälp av webbramverket Phoenix. På webbservern körs också RabbitMQ som används som en kommunikationsserver mellan centralenheten och webbservern/gui:t. Se 8.4 Kommunikation nedan.

I det manuella läget styrs roboten med en joystick som är kopplad till användarens dator eller med knappar på gränssnittet. Figur 10 visar en illustration av det grafiska gränssnittet.



Figur 10: Det grafiska gränssnittet utan joystick

8.1 Webbserver

Webbservern är skriven i Elixir och använder webbramverket Phoenix. Kommunikationen mellan front-end och back-end sker via en channel. En channel är en socket som tillåter kommunikation mellan en klient och en server. I channeln så skickas det från front-end till back-end styrdata, d.v.s, joystickdata och om det autonoma läget ska köras eller inte. Från back-end till front-end skickas debugdata som IR-sensorer, LIDAR samt vinklar.

Back-end skickar styrdata till RabbitMQ servern som centralenheten sedan läser av. Det skickas två gånger per sekund till RabbitMQ servern.

8.2 Styrning

Under styrning kan man välja om roboten ska köras i autonomt eller manuellt läge. För att ändra vilket läge roboten ska köras i finns en knapp som växlar dess läge.

8.2.1 Autonomt läge

Användaren klickar på en knapp som då sätter roboten i autonomt läge. Detta beskrivs i sektion 5.1.1.

8.2.2 Manuell Styrning

Under styrning kan man styra roboten med en joystick. Det finns även möjlighet att styra roboten med knappar/reglage på skärmen. Med dessa kan roboten styras i en viss riktning, rotera eller stanna.

8.3 Debug

Under debug visas robotens olika inställningar samt sensordata vid tillfället. Sidan uppdateras 10 gånger per sekund och klienten får den uppdaterade datan med hjälp av JSON. Det finns också flera grafer som beskriver några av de olika sensorernas data. Man kan också ändra styrparametrarna till regleralgoritmen.

8.4 Kommunikation

Data	**Nyckel**	**Typ**
IR-sensor ned	”ir_down”	float
IR-sensor fram vänster	”ir_fl”	float
IR-sensor fram höger	”ir_fr”	float
IR-sensor bak vänster	”ir_bl”	float
IR-sensor bak höger	”ir_br”	float
LIDAR	”lidar”	float
Vinkel till vänster vägg	”angle_l”	float
Vinkel till höger vägg	”angle_r”	float
Genomsnittlig vinkel	”angle_avg”	float
Självgående läge	”auto”	bool
Debugsträng	”debug”	string

Tabell 3: Format på JSON-strängen från centralenheten

Data	**Nyckel**	**Typ**

X-hastighet	”x”	float
Y-hastighet	”y”	float
Rotation	”rotation”	float
Thrust	”thrust”	float
Självgående läge	”auto”	bool
Återgå till neutralläge	”reset”	bool

Tabell 4: Format på JSON-strängen till centralenheten

9 Simuleringar

För att underlätta utvecklingen av vissa delar av projektet har vi att skrivit kod för att simulera dessa delar. Simuleringarna kör samma kod som ska köras på roboten men visualisera resultatet utan resten av roboten. De delar där simulering är mest aktuellt är inverterad kinematik, korridorföljning och gångstil.

9.1 Simulering för inverterad kinematik

Simuleringen för inverterad kinematik användes för att implementera algoritmen för 2D som beskrivs i avsnitt 6.3. Simuleringen ritas sedan ut i webbläsaren för att visualisera den inverterade kinematiken. Den här simuleringen är skriven i Elm.

9.2 Simulator för korridorföljning

Simulering för korridorföljning kan testa olika regleralgoritmer utan att behöva testa dem på en gående robot. Simuleringen ritar ut en korridor och en robot som går framåt. Simulatorn skickar ut sensorvärden till en fil, regleralgoritmen läser in sensorvärden och tar sedan beslut. Roboten i simulatorn använder två IR-sensorer på varje sida som pekar rakt ut från roboten, det går att välja var på roboten som sensorerna ska sitta. Beslutet av regleralgoritmen skrivs till en fil som simulatorn läser in och styr roboten efter det. Det finns möjlighet att lägga till en störningsfunktion som till exempel kan vara att roboten alltid går med en liten rotation.

9.3 Simulator för gångstil

Simulatorn för gångstil fungerar på samma sätt som korridorföljningssimulatorn. Motorikenhetens kod skriver informationen som på roboten skulle skickas till servona till en fil som simulatorn läser. Simulatorn visualiserar resultatet av kommandona med hjälp av en 3d-rendering av benens lägen och skriver data som servona skulle svara med till en annan fil som motorikkoden kan läsa. Simulatorn är skriven i Rust och använder biblioteket Kiss3D för rendering.

För att simulera motorikenheten körs kommandot

```
make test
```

i motorikmappen. Det kompilerar motorikkoden med avr-specifik kod utbytt mot kod som går att köra på en vanlig dator och som skriver och läser från filer istället för att kommunicera med servon.

Simulatorn själv kör man genom att köra

```
cargo run
```

i simulators/walk

10 Slutsatser

Det som skulle kunna förbättras är bland annat implementation av hindergång för roboten, då roboten redan är förberedd för detektion av hinder i sensorenheten. Felen i servokommunikationen skulle kunna åtgärdas, för bättre kontroll av servona. I gångstilen skulle tillåtet område för fotplacering skulle kunna utökas om nedskalningskoden görs tolerant för konkava områden, samt om undvikning av benkollision kan göras dynamiskt istället för med hårdkodade gränser.

Bilagor

A Programlistning

Här visas en lista på de olika program roboten använder sig av.

Program	Programmerare	Datum	Version
Inverterad Kinematik	Emil Segerbäck, Hannes Tuhkala	2016-12-02	1.0
Gångstil	Olav Övrebö	2016-12-13	1.0
AVR-Kommunikation med SPI	Malcolm Vigren, Noak Ringman	2016-12-13	1.0
SPI på centralenheten	Malcolm Vigren, Noak Ringman	2016-12-13	1.0
Webbserver/GUI	Emil Segerbäck, Hannes Tuhkala	2016-12-13	1.0
Beslutsfattning	Robin Sliwa, Noak Ringman	2016-12-13	1.0
PID-reglering	Robin Sliwa, Olav Övrebö	2016-12-13	1.0
Motorik	Olav Övrebö, Frans Skarman	2016-12-13	1.0
Servokontroll	Frans Skarman	2016-12-13	1.0
Sensorenheten	Malcolm Vigren, Noak Ringman	2016-12-01	1.0

Tabell 5: Lista över program