Robotica: Hexapod

I. van Alphen, S. van Doesburg, E. Salsbach, M. Visser

24 januari 2016

Inhoudsopgave

1 Inleiding

Deze opdracht betreft het ontwikkelen van de besturing en simulatie van een hexapod robot.[**?**] De robot die gebruikt gaat worden is de PhantomX AX Hexapod van TrossenRobotics. [**?**].

In de huidige situatie wordt een hexapod handmatig met een afstandbediening bestuurd en kent geen vorm van intelligentie. Ter voorbereiding op het werken met kunstmatige intelligentie, is er voor gekozen om een simulatiemodel van de robot te ontwikkelen. Om praktische informatie te verzamelen is er een koppeling nodig tussen het simulatie model en de hardware van de robot. Met behulp van deze koppeling kan er onderzoek gedaan worden naar bijvoorbeeld efficiënte looppatronen en zelf lerende functies.

Het verslag is opgebouwd uit het onderzoek naar een hexapod met daarbij de hoofdvraag en deelvragen. Gevolgd door de specificaties en de implementatie van het ontwerp.

2 Opdrachtdefinitie

De hexapod is momenteel alleen te besturen met behulp van een afstandbediening. De huidige besturingssoftware op de robot is niet in staat naast de afstandbediening om externe commando's te verwerken. Daarnaast kent het in de huidige toestand geen vorm van intelligentie. Zo heeft de robot momenteel geen besef van wat zich in zijn directe omgeving bevindt.

Als voorbereiding op het werken met kunstmatige intelligentie op de robot, is er voor gekozen om een simulatiemodel voor en van de robot te ontwikkelen. De redenen hiervoor zijn onder andere dat er in een model oneindig veel verschillende situaties voor de robot gecreeërd kunnen worden. Hiernaast is het simuleren van de hexapod sneller dan real-time testen en is er minder kans op schade van het materieel. Bovendien kan het vanuit financieël oogpunt in situaties nuttig zijn om niet met de echte hardware van de robot te werken. Door een real-time koppeling te maken tussen het simulatiemodel en de hardware van de robot is het mogelijk in staat om meer informatie te verzamelen om de intelligentie te verbeteren.

Het uiteindelijke doel is om in een simulatieomgeving de bewegingseigenschappen van de robot te optimaliseren om deze software vervolgens te testen op het echte model. Er moet dus onderzocht worden hoe een ontwikkelingsomgeving opgezet kan worden waarbij er een koppeling is tussen een simulatie en de robot zelf. Daarnaast moet de robot extern te besturen zijn met behulp van een computer. De robot en het simulatiemodel moeten de mogelijkheid hebben om de stand van de servomotoren real-time naar elkaar over te brengen. Om beschadiging te voorkomen moet het in staat kunnen zijn om fouten te detecteren. De onderlinge poten zouden niet met elkaar in contact moeten komen. Het onderzoeken van het gedrag van een hexapod is van belang om uiteindelijk de hexapod zelf lerende functies te geven.

3 Theoretisch kader

- Inhoudelijke verkenning, kennis benodigd voordat met het ontwerp gestart kan worden (o.a. normen en regelgeving)
- Relevante onderzoeksvragen worden hierin uitgewerkt
- Welke literatuur en/of theorieën zijn relevant en wat betekent dit voor het ontwerp
- Overzicht van bestaande oplossingen van het probleem en waarom voldoen deze in dit specifieke geval wel/niet.

4 Specificaties

Voor de specificaties van dit project, is het van belang een onderscheid te maken tussen functionaliteiten die noodzakelijk of gewenst zijn bij het ontwerp. De noodzakelijke functies moeten in ieder geval geïmplementeerd worden, terwijl het overige optioneel is, afhankelijk van de tijdrestrictie.

4.1 Noodzakeljke specificaties

Het primair doel van dit project is om een verbinding te creëren tussen een fysieke robot hexapod en een simulatiemodel. De verbinding tussen hexapod en de computer dient draadloos te zijn ten gunste van de bewegingsvrijheid van de robot. De datasnelheid van moet ook vastgesteld worden om de hexapod zo responsief mogelijk te maken. De hexapod moet aangestuurd kunnen worden door het simulatiemodel in het programma VREP. Veranderingen aan de stand van de poten moeten direct terug te zien zijn in het simulatiemodel. Het model moet in ieder geval bestaan uit een body en 6 ledematen. Ieder ledemaat moet onderverdeeld worden in drie hoofdcomponenten die gescheiden zijn door gewrichten en ook door middel van een gewricht zijn verbonden aan de body. Verder moeten de afmetingen en verhoudingen tot een halve centimeter nauwkeurigheid overeenkomen met de hexapod. Om te voorkomen dat de hexapod zichzelf kan beschadigen is het noodzakelijk dat de maximale bewegingsvrijheid van de gewrichten (per situatie) wordt uitgerekend of ingesteld. Zodat de poten onderling niet met elkaar botsen of dat de bekabeling beschadigd raakt.

4.2 Gewenste specificaties

Er zijn een veel mogelijkheden wat betreft additionele functies die geïmplementeerd kunnen worden. In deze subsectie zijn een aantal functionaliteiten opgesomd die mogelijk geïmplementeerd kunnen worden, maar die niet noodzakelijk zijn voor het uiteindelijke eindproduct.

- De hexapod is er zich van bewust als hij ondersteboven is geplaatst, en kan de stand van zijn poten daarop aanpassen. Wanneer de hexapod horizontaal gepositioneerd wordt, dan zorgt dit systeem er voor dat alle poten op of richting de ondergrond zijn geplaatst.
- Is in staat om zijn poten in- en uit te strekken, zodat het eenvoudig opgeborgen en opgezet kan worden.
- De spin kan muren en/of objecten detecteren en zijn looproute hier op aanpassen.
- Met een beperking aan een of meerdere poten is het in staat om het standaard looppatroon aan te passen.
- Kan zijn looppatroon aanpassen indien nodig, afhankelijk van het gewicht van de eventuele ballast.
- Eventuele functionaliteiten zoals aan/uit, stand van poten en bewegingen via de computer kunnen activeren bijvoorbeeld via een console.

4.3 Testplan

Om te kijken of de specificaties gehaald zijn wordt er als eerst gekeken naar de noodzakelijke specificaties. Deze specificaties zijn geprobeerd zo op te stellen dat deze in ieder geval haalbaar zijn. De haalbaarheid van de gewenste specificaties zijn voor lastig in te schatten omdat dit veelal buiten onze huidige kennis is en daarom kan het blijken dat een of meerdere specificaties te makkelijk of te lastig zijn opgesteld naar mate het onderzoek vordert. In dat geval zal er samen met de begeleider gekeken worden of er een alternatieve specificatie mogelijk is. Of de resultaten wel of niet voldoen aan de eisen kan in veel gevallen objectief beoordeeld worden door te kijken naar de exacte eisen en de bijbehorende resultaten. Echter is dit bij sommige specificaties niet mogelijk en zal dit met logische redenering en duidelijke uitleg worden ondersteund. Dit betekent dat de resultaten goed gedocumenteerd worden zodat het bij de conclusie duidelijk is wanneer de specificaties gehaald zijn. Bij twijfel of een specificatie gehaald is zal dit met de begeleider besproken worden.

5 Ontwerp

5.1 Het ontwerpen in Inventor

Het model van de hexapod bestaat uit verschillende componenten. De body bestaat uit twee platen, en elke poot is onder te verdelen in drie componenten. Bovendien hoort bij elk gewricht een motor, in totaal 18 motoren. Als eerste zijn de afmetingen van de spin zo exact mogelijk opgemeten. Er is maar een poot opgemeten, en is in inventor vijf maal gedupliceerd. Het programma inventor geeft de mogelijkheid om met behulp van constraints het voorvlak van elk object te schetsen, en vervolgens uit te breiden tot een driedimensionaalobject. Ook met constraints, is het weer mogelijk om alle objecten bij elkaar te voegen.

Het 3D model van de Robot is gemaakt in het programma Inventor. Hiervoor is gekozen omdat dit al bekend is en gratis te gebruiken voor studenten. Tevens kunnen Inventor bestanden gemakkelijk over worden gezet naar bestanden die leesbaar zijn voor de simulatie software.

Om de robot te reconstrueren in het tekenprogramma moet de robot gezien worden als losse onderdelen. Elk onderdeel moet apart opgemeten worden met alle hoeken en uitsparingen erbij.

Het model is dan ontleed tot losse solids (onderdeel bestaande uit een stuk) die samen te voegen zijn tot de uiteindelijke assembly (model opgebouwd uit meerdere onderdelen). De servomotoren zijn gedowload van website voor motoren, maar de andere onderdelen zijn allemaal getekend. De poten bestaan uit drie sigmenten die worden verbonden met servomotor en tevens op dezelfde manier aan de main body vastzitten.

5.2 De simulatieomgeving

Om het inventor bestand te importeren naar V-rep, is het eerst nodig om het model om te zetten naar het STL bestandstype.

Het object wordt initieel in V-rep gezien als geheel object. Met 'divide selected shapes', is het mogelijk om objecten onder te verdelen in zo klein mogelijke stukken. Met behulp van gewrichten worden de 18 pootdelen en de main body met elkaar verbonden. Met de functie 'object/item position/orientation' zijn de objecten en gewrichten op de juiste positie gezet.

Omdat V-rep veel berekeningen moet doen bij het simuleren van complexe objecten, is het niet gunstig om te gaan simuleren met de objecten die geïmporteerd zijn. Het is daarom ook nodig om de vorm van de objecten te vereenvoudigen. Dit kan met 'Morph selection into convex shapes', en dat zorgt er uiteindelijk voor dat simulaties vele malen sneller gaan. Om het uiterlijk van de spin te behouden, worden de geïmporteerde objecten gebruikt als mask. Om dit te bereiken, moet in het 'dynamic properties dialog' venster alles worden uitgevinkt. Dat zorgt ervoor dat het object niet met zijn omgeving reageert, en dat het niet dynamisch gesimuleerd wordt. De convex shapes daarentegen, moeten wel dynamisch zijn en kunnen reageren met de omgeving. Verder worden deze objecten doorzichtig gemaakt.

Voor de positie bepaling en het instellen van de doel coördinaten bij invers kinematica wordt er gebruikt gemaakt van de zogeheten 'dummy-dummy link'.

Hierbij worden er dummies bevestigd aan de tip van elke poot, de zogeheten Tip Dummies. Daarnaast worden er evenveel Target Dummies aangemaakt. Via het 'scene object properties dialog', kunnen de Tip Dummies gelinkt worden aan hun corresponderende Target Dummies. Wat belangrijk is voor het simuleren, is dat voor elk object de optie 'invers kinematics mode' aan staat.

Het is belangrijk om de scene hierarchy goed te ordenen, omdat dit bepaald hoe objecten bewegen ten opzichte van elkaar. Dat betekent dat het hoofd-object(de body) bovenaan staat, en dat alle onderliggende objecten geplaatsen moeten worden onder het object waar het aan gekoppeld is. Dit geldt overigens voor de convex shapes. Alle convex shapes zijn overigens met elkaar verbonden door middel van de joints. De mask moet steeds geplaatst worden onder zijn eigen convex shape. Ook de dummies krijgen hun vaste plek. De Tip Dummy komt onder de tip van de poot terecht en de target dummy direct onder de body.

5.3 Het script

5.3.1 Loopscript

Wanneer de spin in de fysieke wereld moet lopen, en de koppeling tussen simulatie en de robot is gemaakt, moet het simulatie model ook een loop beweging uitvoeren. Hiervoor is een Lua script geschreven, wat er voor zorgt dat het simulatie model zich lopend voort beweegt door de ruimte.

Het Luascript is geïnspireerd op het script van de AntRobot, een standaard robot uit de V-REP bibliotheek, welke qua fysieke kenmerken redelijk overeen komt met de hexapod. De Ant heeft ook een main body met daaraan zes poten welke op dezelfde wijze kunnen scharnieren als de hexapod, wat de beweging van de poten vrij identiek aan elkaar maakt. De Ant maakt eveneens gebruik van Inversed Kinematics, wat voor het toepassen van bepaalde aspecten van dit script op de hexapod ideaal is.

Het design van de code is hetzelfde gebleven, alleen zijn er bepaalde aanpassingen gemaakt waardoor het toepasbaar is voor de Hexapod. Het script is te vinden in de bijlage.

5.3.2 Positiescript

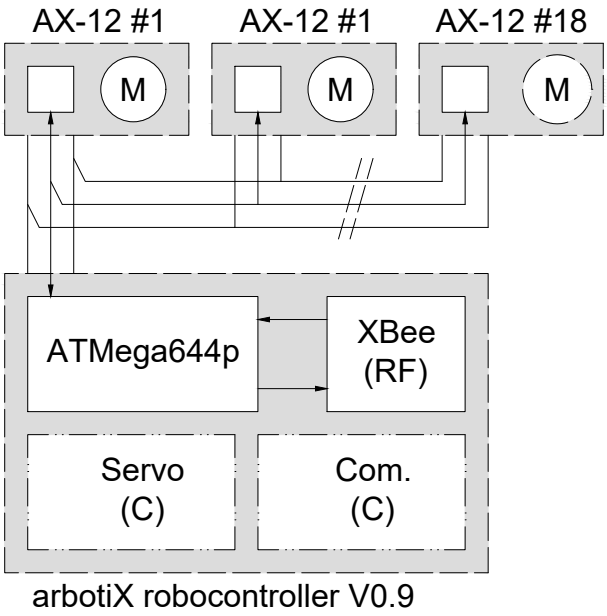
De koppeling van simulatie naar de robot kan alleen worden gemaakt wanneer de simulatie aan staat. Als deze bezig is, moeten de hoeken van de servo's worden uitgelezen om de hexapod dezelfde beweging te geven. Het uitlezen van de hoeken gebeurt met een python script wat per gewricht de hoek uitleest.

5.4 Embedded software spin

De spin bestaat uit 18 servo motoren die aangestuurd worden een arbotiX robocontroller board (zie figuur **??**). Deze controller bestaat uit een ATMega644P, een XBee, Servo connectoren en wat algemene I/O.

5.4.1 Aansturen Servo motoren

De servo motoren die gebruikt zijn de Dynamixel AX-18F. Deze motoren werken op 12V en kunnen maximaal 2,2A aan stroom trekken. Voor het aansturen gebruiken de motoren een eigen serieel protocol.



Figuur 1: Versimpelde weegave in een blokschema van de samenhang van de verschillende systemen en programma's op de hexapod.

Het protocol werkt met master die steeds een pakket over de communicatie bus stuurt en mogelijk een pakket terug kan krijgen. Elk pakket is gericht aan één, meerdere of alle motoren (slaves). Een instructie pakket (master naar servo) bestaat uit de volgende stukken data:
0xFF 0xFF ID LENGTH INSTRUCTION PARAMETER0 PARAMETERS-N CHECKSUM

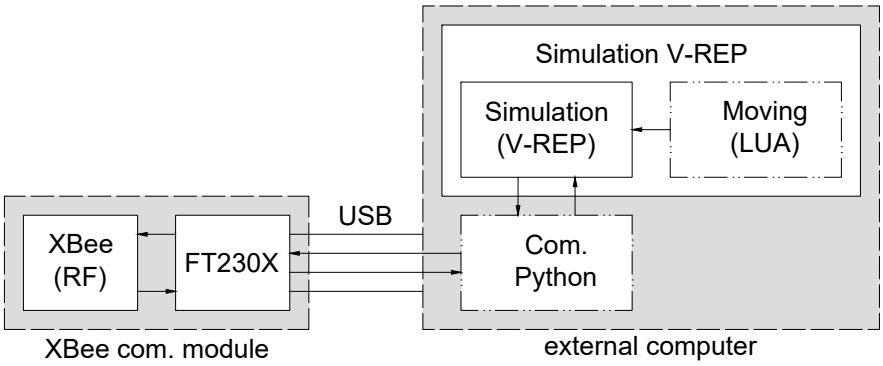
0xFF 0xFF geeft het begin van een pakket aan. Wanneer dit ontvangen wordt weet iedereen dat er een pakket aankomt.

ID geeft het ontvangers ID aan. Elke servo heeft een uniek ID die gebruikt kan worden. Verder is er een speciaal BROADCAST id waar elke servo naar luistert.

LENGTH bevat de lengte van het pakketje. Hierdoor weet te ontvanger hoeveel parameters hij kan verwachten

INSTRUCTION is het type pakket. Dit kan een READ, WRITE of SYNC-WRITE zijn. Bij READ word er een waarde uitgelezen, Bij WRITE een waarde weggeschreven en bij SYNCWRITE kan er bij meerdere Servo motoren tegelijk worden geschreven.

PARAMETER0 is het lees/schrijf adres PARAMETER-N is alle data die weggeschreven moet worden. Bij en SYNCWRITE instructie bevat dit de servo ID's



Figuur 2: Versimpelde weegave in een blokschema van de samenhang van de verschillende systemen en programma's op de externe computer.

en DATA per servo zodat meerdere servo tegelijk beschreven kunnen worden.

CHECKSUM bevat een checksum van het gehele pakket zodat gecontroleerd kan worden of alles goed doorgekomen is.

Een Status pakket (Servo naar master) bestaat uit de volgende stukken data:
0xFF 0xFF ID LENGTH ERROR PARAMETER-N CHECKSUM
0xFF 0xFF geeft het begin van een pakket aan. Wanneer dit ontvangen wordt weet iedereen dat er een pakket aankomt.

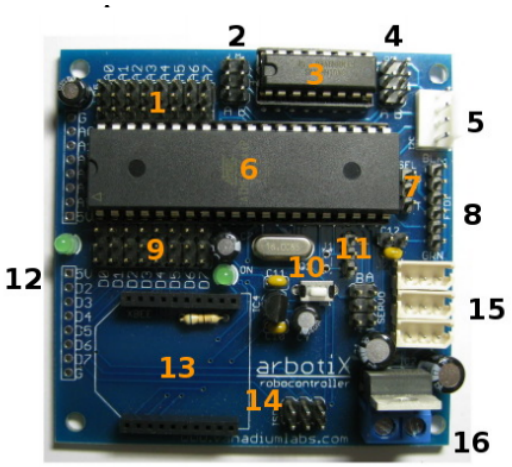
ID geeft het ontvangers ID aan. Elke servo heeft een uniek ID die gebruikt kan worden. Verder is er een speciaal BROADCAST id waar elke servo naar luistert.

LENGTH bevat de lengte van het pakketje. Hierdoor weet te ontvanger hoeveel parameters hij kan verwachten

ERROR geeft een binaire code terug met de fouten die op dat moment actief zijn in de servo. Zo een fout kan bijvoorbeeld zijn: Overbelasting, Checksum fout, Ingangspanning fout.

PARAMETER-N geeft de data terug die door de instructie pakket is opgevraagd

CHECKSUM bevat een checksum van het gehele pakket zodat gecontroleerd kan worden of alles goed doorgekomen is.



- 1 - Analog port headers
- 2 - Left motor/encoder headers
- 3 - Dual motor driver, max current 1A
- 4 - Right motor/encoder headers
- 5 - I2C header
- 6 - ATMEGA644P
- 7 - Power selection header
- 8 - FTDI serial0/programming
- 9 - Digital port headers
- 10 - Reset Switch
- 11 - Serial1 header (also J1)
- 12 - Prototyping headers and user led
- 13 - XBEE socket
- 14 - In-system programming (ISP)
- 15 - 3 Bioloid headers
- 16 - Power terminals

Figuur 3: Robocontroller overzicht

Implementatie Het controller board werkt met een ATMega644P waarbij de Servo connectoren direct aangesloten zijn op een UART ingang van de atmega. De servo-bus werkt met een enkele seriele lijn en dat betekent dat zowel de verzend als de ontvang kant via dezelfde lijn werkt. Daarom moet elke keer wanneer data verzonden wordt de atmega uart ontvang hardware uitgezet worden en elke keer wanneer er data ontvangen wordt de atmega uart verzend hardware uit worden gezet.
De AX18FWrite functie werkt vrij vanzelfsprekend. De UART Tx module wordt aangezet, de transmit buffer wordt leeg gehaald en vervolgens wordt de buffer weer gevuld met informatie van het pakketje. Op het eind wordt nog een checksum toegevoegd en de Tx module weer uitgezet.

```
46 /**
47  * Writes to the Servo
48  * @param id       Servo identifier
49  * @param address Servo memory write address
50  * @param data     Data to write to memory
51  * @param length   Length of data
52  */
53 void AX18FWrite(unsigned char id, unsigned char address, unsigned
        char *data, unsigned char length) {
54
55   // Enable Uart Tx so we can send
56   uart1_RxDisable();
57   uart1_clearTxBuffer();
```

```
58    uart1_TxEnable();
59
60    uart1_putc(AX_START);
61    uart1_putc(AX_START);
62    uart1_putc(id);
63    uart1_putc(length + 3);
64    uart1_putc(AX_WRITE_DATA);
65    uart1_putc(address);
66
67    for(unsigned char i = 0; i < length; i++) {
68      uart1_putc(data[i]);
69    }
70
71    uart1_putc(generateTxChecksum(id, address, data, length));
72    //_delay_us(500);
73    uart1_TxWaitDisable();
74  }
```

Listing 1: AX18ServoDriver.c - AX18Write functie

De Read functie is iets uitgebreider omdat het de data echt moet kunnen analyseren. Eerst wordt een pakket verzonden naar de servo met een aanvraag voor data. Dit gebeurd op dezelfde manier zoals in de AX18FWrite functie. Daarna wacht de functie tot er data terug komt van de Servo. Elke byte die dan terug komt wordt in een state machine gestopt die alles byte voor byte analyseert. Zo zijn state 1 en state 2 de start bytes. Wanneer een andere byte dan de start byte is gevonden wordt de state weer naar 0. De rest van de states zijn ook duidelijk.

```
119 /**
120  * Reads from the servo
121  * @param id      Servo identifier
122  * @param address Servo memory read address
123  * @param buffer  Buffer to save data
124  * @param length  Length to read
125  * @return        Error occurred, return 0 on success, 1 on error
126  */
127 unsigned char AX18FRead(unsigned char id, unsigned char address,
        unsigned char *buffer, unsigned char length) {
128
129   unsigned char checksum = generateRxRequestChecksum(id, length,
        address);
130
131   // Enable Uart Tx so we can send
132   uart1_clearTxBuffer();
133   uart1_TxEnable();
134   uart1_RxDisable();
135
136   uart1_putc(AX_START);
137   uart1_putc(AX_START);
138   uart1_putc(id);
139   uart1_putc(4);
140   uart1_putc(AX_READ_DATA);
141   uart1_putc(address);
142   uart1_putc(length);
143   uart1_putc(checksum);
144
145   uart1_TxWaitDisable();
146   uart1_RxEnable(); // TEMP
147
148   // Wait for response
149   while(uart1_canRead() <= 0);
```

```
150
151   //_delay_us(TX_READ_DELAY_TIME);
152   unsigned char RxState = 0, RxDataCount = 0, RxServoId = 0,
        RxLength = 0, RxError = 0, RxChecksum = 0, Error = 0;
153
154   // Wait a couple of micro seconds to receive some data
155   // Loop trough all received bytes
156   while(uart1_canRead() > 0) {
157     //printf("Buffer size: %d\r\n", uart1_canRead())
158     char c = uart1_getc();
159   printf("(%d): 0x%x\r\n", RxState, c);
160     switch(RxState) {
161
162       // 1) First Start byte
163       case 0:
164         if(c == AX_START) {
165           RxState = 1;
166         }
167       break;
168
169       // 2) Second start byte
170       case 1:
171         if(c == AX_START) {
172           RxState = 2;
173         } else {
174           RxState = 0;
175         }
176       break;
177
178       // 3) Id byte
179       case 2:
180         if(c != AX_START) {
181           RxServoId = c;
182           RxState = 3;
183         } else {
184           RxState = 0;
185           Error = 1;
186         }
187       break;
188
189       // 4) Length byte
190       case 3:
191         RxLength = c;
192         RxState = 4;
193       break;
194
195       // 5) Error byte
196       case 4:
197         RxError = c;
198         RxState = 5;
199       break;
200
201       // Data bytes and checksum byte
202       case 5:
203         if(RxDataCount >= RxLength - 2) {
204           RxChecksum = c;
205           RxState = 6;
206           break;
207         }
208         printf("Data (%d): 0x%x\r\n", RxDataCount, c);
209         buffer[RxDataCount++] = c;
210       break;
```

```
211
212         // There is no state 6 unless we got more data then expected
    ...
213         case 6:
214           Error = 1;
215         break;
216
217       }
218     }
219
220     // Check if packet is correct by comparing the checksum
221     if(generateRxChecksum(RxServoId, RxError, address, buffer, length
      ) != RxChecksum) {
222       Error = 1;
223     }
224
225     printf("error: 0x%x\r\n", RxError);
226
227     return Error;
228
229 }
```

Listing 2: AX18ServoDriver.c - AX18Read functie

5.4.2 Communicatie protocol

Om de robot draadloos aan te sturen wordt er gebruik gemaakt van xbee modules. Deze modules werken via een serië ele verbinding. XBee bevat de mogelijkheid om de serië ele data in data pakketten te versturen om zo individuele pakketten te kunnen onderscheiden. Echter maken wij daar geen gebruik van om het geheel wat simpeler te houden. Dit betekent dat er één continuë stroom van data is die geanalyseerd moet worden om losse data berichten te onderscheiden.
Dit wordt gedaan op dezelfde manier zoals de servo motoren worden aangestuurd. Het pakket begint wanneer er vier start bytes zijn gevonden. Daarna komt de lengte van de data, het commando, de data zelf en als laatst een checksum om het gehele pakket te controleren op fouten. Zie tabel **??**. Deze pakketten kunnen op dit moment nog één kant op worden gestuurd (van de Computer naar de spin) maar dit is later gemakkelijk uit te breiden om ook de andere kant op te gaan op dezelfde manier zoals bij de servo motoren het geval is.

byte nr	**Example value**	**meaning**
0-3	0x5A - 0x3C - 0x42 - 0x99	Start bytes
4	0x02	Data length
5	0x00	Command
6-(6+length)	0x60	Additional data
7+length	0xFF	Checksum (length, command and data)

Tabel 1: XBee pakket

Implementatie Bij de implementatie van het communicatie protocol is geprobeerd het systeem zo robuust mogelijk te maken. De communicatie over de XBee werkt als een seriele lijn waar een lange stream aan data uitkomt. Omdat de XBee ook met buffers werkt en bij draadloos verkeer alle bytes niet altijd aankomen en opnieuw verzonden moeten worden kan het voorkomen dat de er

soms 1 byte binnenkomt en soms 25 bytes. Elke byte moet verwerkt worden en bytes die een tijd later aankomen kunnen nog horen bij een pakket die eerder is begonnen.
Om dit allemaal in goede banen te leiden is er een 'XBee_ communication_ processSerialData' functie gemaakt waar de staat van een pakket wordt opgeslagen en het pakket via meerdere stukken opgebouwd kan worden. Deze functie werkt ook met een state machine, het grote verschil is dat het pakket inclusief de ontvang staat globaal opgeslagen wordt. Verder kunnen pakketten worden opgeslagen en direct door worden gegaan met het analyseren van de rest van de data. De pakketten worden opgeslagen in een ringbuffer hierdoor kan de buffer nooit overstromen met te veel data maar zullen de meest oude pakketten verwijderd worden van de buffer zodat de nieuwste pakketten in ieder geval nog uitgelezen kunnen worden.

```
119 unsigned char XBee_communication_processSerialData(char *data,
        unsigned char length) {
120   unsigned char countNewPackets = 0;
121
122   for(unsigned char i = 0; i < length; i++) {
123     unsigned char c = data[i];
124     //printf("State: %d (0x%x)\r\n", RxState, c);
125
126     if(RxState == 0) {
127         if (c == PACKET_START_BYTE_1) {
128           RxState = 1;
129           } else {
130           RxState = 0;
131         }
132       } else if(RxState == 1) {
133         if(c == PACKET_START_BYTE_2) {
134           RxState = 2;
135           } else {
136           RxState = 0;
137         }
138       } else if(RxState == 2) {
139         if(c == PACKET_START_BYTE_3) {
140           RxState = 3;
141           } else {
142           RxState = 0;
143         }
144       } else if(RxState == 3) {
145         if(c == PACKET_START_BYTE_4) {
146           RxState = 4;
147           } else {
148           RxState = 0;
149         }
150       } else if(RxState == 4) {
151         RxPacket.length = c;
152         RxChecksum |= c;
153         if(RxPacket.length > MAX_DATA_SIZE) {
154           RxState = 0;
155         } else {
156           RxState = 5;
157         }
158       } else if(RxState == 5) {
159         RxPacket.command = c;
160         RxChecksum |= c;
161
162         RxState = 6;
```

```
163       } else if(RxState == 6) {
164         if(RxDataCount < RxPacket.length) {
165           RxPacket.data[RxDataCount++] = c;
166           RxChecksum |= c;
167         }
168         if(RxDataCount >= RxPacket.length){
169           RxState = 7;
170         }
171       } else if(RxState == 7) {
172         RxPacket.checksum = c;
173 
174         if(RxPacket.checksum == RxChecksum) {
175           XBee_communication_saveRxPacket(&RxPacket);
176           countNewPackets++;
177         } else {
178           printf("Checksum failed: 0x%x vs 0x%x\r\n", RxPacket.
    checksum, RxChecksum);
179         }
180 
181         RxState = 0, RxChecksum = 0, RxDataCount = 0;
182 
183     }
184 
185   }
186 
187   return countNewPackets;
188 }
```

Listing 3: XbeeCom.c - processSerialData functie

6 Resultaten

7 Discussie en resultaten

8 Conclusies

9 Aanbevelingen

10 Onderzoeks opzet

Het project bestaat uit twee delen. Het eerste deel heeft als doel de hexapod in de simulatie te verwerken. Het tweede deel is ervoor zorgen dat er een verbinding ontstaat tussen de simulatie en de hexapod. Vanwege de wensen van de opdrachtgever zal er gewerkt worden met het simulatie-programma V-REP en zullen andere softwarepakketten buiten beschouwing worden gelaten.

Daarnaast zal er in de eerste deel gekeken worden naar toepassingen die andere hebben bedacht voor de hexapod. Hieruit is inspiratie te halen voor handige en leuke toepassingen.

Om de verbinding te maken tussen robot en computer dienen er verschillende bronnen te worden geraadpleegd. Een groot deel van die bronnen zal worden voorzien door gebruik te maken van het internet. Verder zal kennis die in de les wordt opgedaan, worden toegepast en de boeken kunnen worden bekeken.

De hexapod is al fysiek aanwezig, wat het testen van geschreven code gemakkelijk maakt. Op deze manier wordt gelijk duidelijk of het geschreven programma functioneert. Tijdens het schrijven van de verschillende functies zal de hexapod fysiek aanwezig zijn om direct het resultaat te ondervinden.

11 Bibliografie