Eindverslag project

Groep 6

9 januari 2012

Hoofdstuk 1

Inleiding

Dit rapport beschrijft de ervaringen van de projectgroep met het computer systeem project. Het doel van dit project is het programmeren van een robot.

De robot wordt in een doolhof geplaatst en moet zelf de uitgang vinden. De robot is vormgegeven als een tank. Om uit het doolhof te komen kan de tank gebruik maken van een kompas. De tank heeft ook afstandsensoren zodat de muren geregistreerd kunnen worden.

De taak van de projectgroep is om de tank te programmeren, zodat de tank gebruik kan maken van het kompas, de sensoren en de rupsbanden. Het projectdoel is dat de tank, vanaf het moment dat je de tank aan zet, zonder menselijk ingrijpen zelfstandig uit het doolhof kan rijden.

De projectgroep gaat de Pledge methode, of wel algoritme, gebruiken. Het Pledge algoritme laat de robot rechtdoor te rijden tot hij een obstakel tegen komt. Als de robot een obstakel tegen komt zal dit obstakel gevolgd worden tot dat de robot weer zijn oorspronkelijke richting kan vervolgen. Om de oorspronkelijke richting te bepalen gebruikt het Pledge algoritme een kompas. Het voordeel van het Pledge algoritme is dat er altijd een uitgang gevonden wordt en dat er niet geregistreerd hoeft te worden waar de robot zich bevind in het doolhof.

Om Pledge uit te kunnen voeren moet de robot kunnen sturen, wat wordt toegelicht in hoofdstuk 2. Als de robot een obstakel tegenkomt moet hij dit obstakel volgen, hoe dit precies in zijn werk gaat wordt beschreven in hoofdstuk 3. Terwijl dit obstakel gevolgd wordt moet de robot weten wanneer hij zijn originele richting weer heeft, dit proces wordt omschreven in hoofdstuk 4. In hoofdstuk 5 staan de conclusie en aanbevelingen. Ten slotte heeft dit verslag nog 2 bijlagen met daarin de werkverdeling en een gedetailleerde beschrijving van het Pledge algoritme.

Hoofdstuk 2

Sturen

Omdat het uiteindelijke doel van het project is om een tank zich door een doolhof te laten bewegen is het sturen essentieel. Om ook nog te zorgen dat dit redelijk snel kan is het belangerijk dat de rupsbanden meer dan alleen aan en uit kunnen zijn.

2.1 Definitie Probleem

De tank heeft twee rupsbanden die elk worden aangedreven door een aparte motor. Om te zorgen dat de tank bochten kan maken zou er gekozen kunnen worden voor een system waar één wiel naar voren gaat en het andere wiel naar achter zodat de tank op zijn plaats kan draaien. Het probleem is dat dit een tijdrovende ingrijp is waar bij de tank geen afstand aflegt. Als de tank een muur aan het volgen is zal deze niet exact parralel aan de muur rijden waardoor er vaak bij gedraaid zou moeten worden en als er bij elke bijdraaing zal worden stil gestaan zal de race waarschijnlijk niet gewonnen worden.

2.2 Aanpak

Om de tank sneller te laten draaien is er voor een Pulse Width Modulation, afgekort PWM, systeem gekozen. PWM zorgt ervoor dat de motor in een tijdsperiode enkele keren aan en uit zal gaan, zodat de motor niet constant op vol vermogen en dus trager draait. Met PWM kan dus de snelheid van de wielen worden bepaald en kan tijdens het rijden bijgestuurd worden door één wiel trager te laten draaien dan het andere wiel.

Om het PWM systeem te implementeren is timing essentieel. Dit dwingt af om PWM te implementeren in een kernel module omdat dit volgens ons de enige manier is om een consistente timer te maken. We hebben geprobeerd PWM zo min mogenlijk processorintensief te maken door gebruik te maken van een

Duty Cycle model waarin 3 timers lopen. één basistimer en één timer voor elke motor.

2.3 Implementatie

Het doorsturen van informatie naar de kernel module verloopt als volgt. De kernel module accepteerd een functie call, de write functie. Ons programma kan de file /dev/lart openen en hier naar toe schrijven.

De driver accepteerd een structure met 4 getallen:

```
uit pwm_module.h:
// public struct
static struct pwm_input{
  unsigned long pin_input;
  unsigned long base_time;
  unsigned long left_time;
  unsigned long right_time;
};
```

Het eerste getal is het register, de representatie van hoe je de 8 verschillende bits wilt hebben, het tweede getal is de basistime, die standaard op 100ms loopt en de twee andere timers aanroept. De laatste twee zijn de linker en rechter motor die aangeven hoeveel tijd van die 100ms de motors aan moeten staan.
Een ander probleem is het uitschakelen van de juiste motor. De oplossing is door het definieren van pinnen die alleen door links gebruikt worden en zo ook voor rechts. Het begon met:

```
rechts = 1 << 5 | 1 << 6 | 1 << 7;
```

en als de rechter motor moest stoppen gebeurde er:

```
GPSR = ~(pin_input & rechts);
```

Hier hebben we nog last van gehad omdat voor een tijd rechts gedefinieerd was als de 6,7,8ste pinnen, waardoor als de tank achteruit ging rijden hij ook niet meer zou stoppen. Deze oplossing werd verbeterd naar:

```
rechts = 1<<5|1<<6;

GPCR = rechts;
```

De enable bit 7 is weg gelaten en in plaats van de registers te zetten gebruiken we het Cleanup register om de GPSR te legen. De andere oplossing was natuurlijk de enable bit cleanen, maar door tijd gebrek is dit (nog) niet getest en zal het waarschijnlijk hierbij gelaten worden.

2.4 Aanbevelingen

Het lastigste is consistent blijven. Tijdens het maken van de drivers kwamen er problemen opduiken die niet altijd even goed zijn op gelost. Zo was het orgiginele plan om de driver enkel en alleen een timer te laten zijn die de twee motors uit zette en dat de originele user_gpio.c de rest kon afhandelen. Dit bleek niet mogelijk waardoor de driver ook ineens de register input moest accepteren en dus moest de functionaliteit van user_gpio.c worden nagemaakt om te werken met de driver. Het nadeel is dat de LED/button pinnen op het moment van schrijven niet als input kunnen dienen. Als er meer ervaring in de groep was met kernel modules was het misschien beter geweest om een directe memory map te maken naar de GPSR en twee andere registers die de snelheid van de motors moeten weergeven. Hiermee zouden zowel de knopjes als de ledjes kunnen worden benut.

Hoofdstuk 3

Wall Hugging

3.1 Definitie probleem

Wall Hugging, ofwel het volgen van (de loop van) de muur, is nodig om het Pledge algoritme succesvol te kunnen implementeren. Om een muur te kunnen volgen moet de tank eerst weten wat zijn relatieve positie is ten opzichte van de muur. Ook moet de tank met verschillende situaties kunnen omgaan, zoals bijvoorbeeld een doodlopende gang

3.2 Aanpak

Om de afstand tot de muur te kunnen bepalen kan de tank de vier afstandssensoren uitlezen. Deze meetwaardes worden verwerkt tot waarden waarmee het programma verder kan redeneren wat er moet gebeuren.

3.3 Implementatie

Er is gekozen voor een aanpak waarbij gekeken wordt waar de tank heen moet afhankelijk van de gefilterde sensorwaarden. In grote lijnen probeert de tank rechtdoor te rijden terwijl de muur binnen een bepaalde afstandsmarge gehouden dient te worden. Indien dit niet (meer) het geval is wordt er bijgestuurd. *zie plaatje ontwerpverslag*. In gevallen waarbij de muur die gevolgd wordt zo goed als rond is levert dit wel de meeste vertraging op; de tank probeert zal elke keer weer een stukje rechtdoor rijden, gevolgd door een fase waarin de tank de afstand tot de muur te groot acht en weer bijstuurd. Rechte muren met scherpe bochten worden wel snel genomen, aangezien de te tank opeens geen muur meer ziet wordt er alles aan gedaan om weer 'een hand aan de muur' te kunnen krijgen.

3.4 Aanbevelingen

soon...

Hoofdstuk 4

Bochten Tellen

Een onderdeel van het Pledge algoritme is het tellen van de gemaakte bochten. Dit is nodig om te voorkomen dat de tank in een eindeloze herhaling valt.

4.1 Definitie probleem

De tank maakt geen bochten van 90 graden. Dit komt door belemmering van de hardware. Om er alsnog achter te komen hoeveel bochten (= 90 graden gedraaid) moet er met het kompas worden gewerkt.

4.2 Aanpak

De oplossing hiervoor is de richting van de tank te abstraheren naar 4 richtingen, elke richting is precies 90 graden van een andere richting verwijderd. Als de tank een richting veranderd verandert de hoeveelheid bochten ook meteen.

4.3 Implementatie

Het kompas is gemplementeerd in het Compass.c bestand. De huidige richting wordt bijgehouden in een globale variabele, direction genaamd. In de Compass_update() functie wordt de nieuwe meetwaarde van de hardware gelezen en opgeslagen. Vervolgens bevat het Compass.c bestand nog een aantal functies om te kijken of de huidige richting van de tank binnen bepaalde waardes ligt en wat de richting van de tank is. Deze richtingen zijn relatief of de beginrichting en niet op het geografische noorden. De grootste problemen waren het uitlezen van de hardware en het bepalen of de richting van de tank zich wel of niet binnen bepaalde waardes bevind.

4.4 Aanbevelingen

Het bijhouden door middel van bochten gaat prima. Het is een stuk betrouwbaarder dan richting bepalen via de motoren omdat je geen last hebt van niet te voorspellen factoren zoals wrijving. Je zult een paar kleine problemen tegenkomen die men kan verhelpen door een beetje logisch na te denken.

Hoofdstuk 5

Conclusie

Bijlagen

TODO: Sol inkorten, Luyt kiezen wat erin moet, Jelle+Mick laatste stukje nog