KU Leuven
Departement Computerwetenschappen

P&O: Computerwetenschappen

Tussentijdsverslag 3

Team:
Zilver

Sam Gielis
Sophie Marien
Toon Nolten
Nele Rober
Gerlinde Van Roey
Maxim Van Mechelen

Academiejaar 2012 – 2013

Samenvatting

Het P&O-project bestaat erin een robot autonoom een doolhof te laten verkennen. Dit verslag beschrijft de invulling die team Zilver aan het project gaf. De robot wordt voorzien van een lichtsensor en een ultrasone sensor. Deze staan vast gemonteerd en kunnen niet onafhankelijk van de robot bewegen. De aansturing van de robot gebeurt via bluetoothverbinding. Een Grafische User Interface (GUI) maakt deze aansturing op een gebruiksvriendelijke manier mogelijk. De GUI geeft de baan van de robot weer. Ook een historiek van de sensorwaarden wordt weergegeven in de GUI.

De robot kan zich autonoom door een doolhof voortbewegen zonder op muren te botsen. Tijdens het rijden slaat de robot een map op van de doolhof. Op elke nieuwe tegel kijkt de robot rond en geeft de gedetecteerde muren door. Wanneer de hele doolhof verkend is, gebruikt de robot het $A*$-algoritme om de kortste weg naar de finish te bepalen. De finish wordt aangegeven met een bepaalde barcode. Andere barcodes laten de robot een opdracht uitvoeren.

Om de afwijking op de aansturing te minimaliseren, oriënteert de robot zich regelmatig op een witte lijn. Zo blijft de robot steeds zo dicht mogelijk bij het midden van de tegel.

Een computerprogramma simuleert de werking van de robot. Het is voor de simulator mogelijk door een virtuele doolhof te rijden en deze op gelijkaardige wijze te verkennen. De sensorwaarden worden gesimuleerd, met een afwijking om de werkelijke robot zo goed mogelijk te benaderen, maar verder worden dezelfde algoritmes gebruikt. Dit laat toe de algoritmes te testen zonder de robot hiervoor te gebruiken.

Inhoudsopgave

1 Inleiding

In het kader van het vak 'Probleemoplossen en Ontwerpen: computerwetenschappen' wordt gewerkt rond autonome intelligente robots. Verschillende teams bouwen en programmeren een robot met behulp van LEGO Mindstorms. Deze robot moet uiteindelijk volledig autonoom een doolhof kunnen verkennen.
Op de derde demonstratie kan de robot alle taken van de vorige demonstratie nog steeds uitvoeren. De robot kan zich volledig autonoom voortbewegen. Wanneer de robot zich in een doolhof voortbeweegt, kan hij deze in kaart brengen. Bij het inlezen van barcodes voert de robot een bepaalde opdracht uit. Op het moment dat de volledige doolhof is ingelezen, bepaalt de robot de kortste weg naar zijn beginpositie en rijdt hier in hoge snelheid naartoe.

2 Bouw van de robot

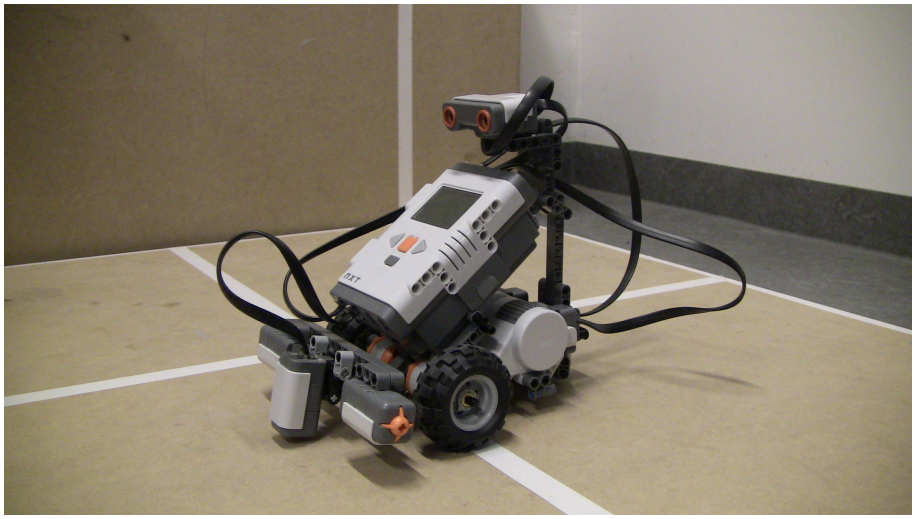
LEGO Mindstorms [1] biedt een bouwpakket voor een robot aan. Een NXT-microcomputer laat toe de robot te programmeren met Java.

2.1 Fysieke bouw

Bij het bouwen van de robot (zie figuur 1) werd het ontwerpboekje gevolgd. Deze compacte samenstelling leek geen directe nadelen te hebben. Twee grote wielen worden elk met hun eigen motor aangestuurd. Kleine wielen achteraan zorgen voor meer stabiliteit en worden niet aangedreven. De sensoren werden als volgt geïnstalleerd:

- *lichtsensor:* vooraan en dicht tegen de grond.
- *ultrasone sensor:* bovenaan, naar voren kijkend. De sensor staat vast gemonteerd.
- *druksensoren:* aan beide zijkanten, één aan de linkerkant en één aan de rechterkant.

Een alternatieve opstelling bestaat erin de ultrasone sensor op een derde motor te monteren zodat deze onafhankelijk van de robot kan ronddraaien. Opdat hier plaats genoeg voor is, moet de NXT plat gelegd worden met de wielen aan weerszijde. Dit vraagt echter een nieuwe calibratie.



Figuur 1: Robot

(a) Ultrasone sensor (b) Licht- en Druksensor

Figuur 2: Sensoren

Bovendien zou de meetwaarde van de ultrasone sensor afhangen van zijn positie ten opzichte van de robot, wat het interpreteren moeilijker maakt. Voor deze opstelling werd daarom niet gekozen.

2.2 Calibratie van de motoren

De robot wordt aangedreven door twee motoren, elk verbonden met één van de twee grote wielen. De aansturing gebeurt door te bepalen hoeveel graden de wielen moeten draaien. Beide motoren kunnen onafhankelijk ingesteld worden. De robot kan vooruit bewegen en rond zijn as. De resultaten van de calibratie worden weergegeven in tabel 1.

	linkerwiel	rechterwiel	aantal graden
1 cm vooruit	voor	voor	20,8°
1 cm achteruit	achter	achter	20,8°
180°draaien linksom	achter	linker	701°
180°draaien rechtsom	voor	achter	701°

Tabel 1: Resultaten calibratie motoren

2.2.1 Eén cm vooruit bewegen: bepalen van x

De parameter x bepaalt het aantal graden dat de wielen moeten draaien opdat de robot één cm vooruit beweegt.
Een schatting voor x via de *diameter* (in centimeter) gebeurt als volgt:

$$x_0 \approx \frac{360}{\pi \cdot diameter}$$

Een verdere bepaling van x gebeurt via tests.
De robot wordt naast een lintmeter geplaatst en krijgt opdracht beide wielen $100 \cdot x$ graden te laten draaien. Bij een perfect gekozen waarde voor x, legt de robot precies één meter af. Indien niet, wordt x aangepast en wordt de test herhaald tot een voldoende nauwkeurig resultaat bekomen wordt.

Een volgende test bestaat erin de robot beide wielen tien maal $10 \cdot x$ graden te laten draaien. Een boxplot van de totaal afgelegde afstand wordt weergegeven in figuur 3. Wanneer geen afwijking op één meter wordt waargenomen, heeft het starten en stoppen geen invloed op de totale afgelegde afstand. Dit blijkt echter wel het geval. In verdere algoritmes wordt hier rekening mee gehouden. De robot legt steeds één lange afstand af in plaats van vele korte.
Uit deze test blijkt bovendien dat de robot een afwijking naar links heeft wanneer hij rechtdoor rijdt. De boxplot van deze afwijking wordt eveneens weergegeven in figuur 3.

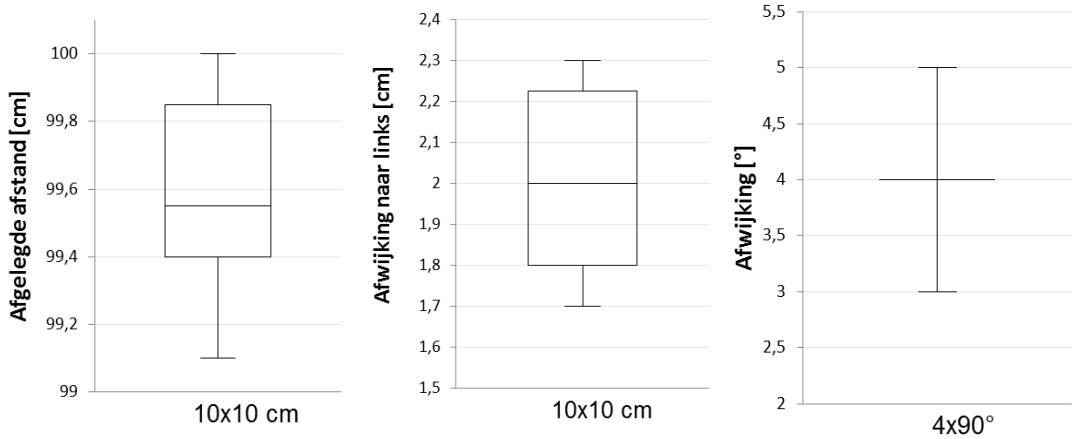
2.2.2 Volledig rond de as draaien: bepalen van y

De parameter y bepaalt het aantal graden die de wielen moeten draaien opdat de robot 360°rond zijn as zou draaien. Beide wielen draaien hierbij in tegengestelde richting.
Een schatting voor y via de diameter van het wiel, met as_{robot} (afstand tussen beide wielen) en *diameter* in centimeter:

$$y_0 \approx \frac{(2 \cdot \pi) \cdot (as_{robot}/2)}{diameter_{wiel}/2}$$

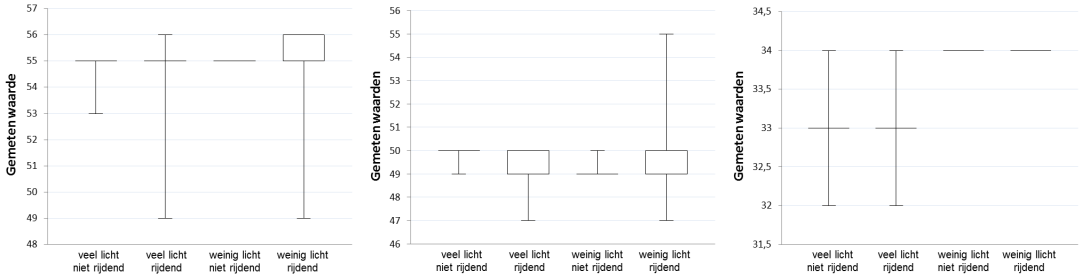
De robot wordt naast een lijn geplaatst en krijgt de opdracht zijn wielen y graden in tegengestelde richting te laten draaien. y wordt aangepast tot de robot na het draaien opnieuw precies naast de lijn uitkomt.

Door de robot in stappen rond zijn as te laten draaien ($4x90°$) kan de invloed op het starten en stoppen van de motoren bepaald worden. Figuur 3 geeft een boxplot van de gemeten afwijkingen op 180°. De algoritmes houden met deze afwijking rekening. Nadat de robot naar één kant draait, draait hij, zo mogelijk, evenveel graden naar de andere kant terug. Op deze manier wordt de afwijking ongedaan gemaakt.



(a) Afgelegde afstand rechtdoor (b) Afwijking naar links (c) Afwijking op 360°

Figuur 3: Boxplots calibratie motoren.



(a) Witte lijn (b) Paneel (c) Zwarte lijn

Figuur 4: Boxplots calibratie lichtsensor.

2.3 Calibratie van de lichtsensor

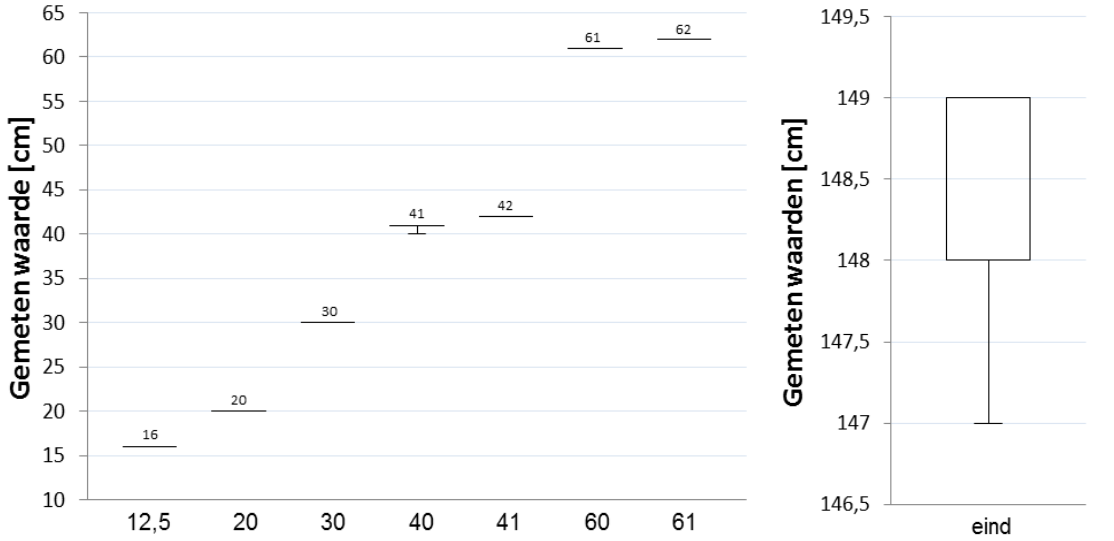
De lichtsensor meet de lichtintensiteit van de omgeving. De sensor kan zelf ook een rood licht uitsturen. Hoe meer dit licht geabsorbeerd wordt door de omgeving, hoe donkerder de omgeving. Op deze manier kunnen de meetwaarden geïnterpreteerd worden als witte of zwarte lijnen.
De lichtsensor wordt in verschillende omstandigheden getest: bij direct kunstlicht, in de schaduw, terwijl de robot rijdt en terwijl hij stilstaat. Dit voor alle soorten ondergrond die in de doolhof voorkomen: een paneel, een witte lijn en een zwarte lijn. Boxplots worden weergegeven in figuur 4. Deze toont dat het gemiddelde enkel bij een zwarte lijn afhangt van de omstandigheden. De afwijking wordt echter sterk bepaald door het feit of de robot rijdt of stilstaat. Bij het simuleren en het interpreteren van de meetwaarden wordt hier rekening mee gehouden.

2.4 Calibratie van de ultrasone sensor

Een ultrasone sensor zendt ultrasone geluidsgolven uit. De golven weerkaatsen op een object dat zich binnen het bereik bevindt. Bij ontvangst van zijn eigen geluidsgolven, weet de robot dat er een object in de buurt is en op welke afstand het object zich bevindt.

In het kader van een doolhof, kan een robot muren tegenkomen. Ook de paaltjes, die de muren omhoog houden, worden als object gedetecteerd. Om te vermijden dat de paaltjes als muur worden geïnterpreteerd worden enkel meetwaarden kleiner dan 29 cm als muur beschouwd.

De afwijking van de ultrasone sensor wordt gemeten door de robot op een bepaalde afstand van een muur te zetten. Verschillende afstanden worden zo gemeten. Boxplots in figuur 5 geven de gemeten waarden (boven de lijnen) ten opzichte van de werkelijke waarden weer. Hoewel de



(a) meetwaarden

(b) laatste geldige meetwaarde

Figuur 5: Boxplots calibratie ultrasone sensor.

variantie op de gemeten waarde niet groot is, komt de gemiddelde gemeten waarde niet altijd overeen met de werkelijke afstand. Vooral voor afstanden kleiner dan 20 cm en groter dan 145 cm, wijkt de gemeten waarde af van de werkelijke waarde. Waarden buiten het interval [20,145] worden daarom niet als betrouwbaar veronderstelt.

3 Algoritmes

Verschillende algoritmes zorgen ervoor dat de robot alle opdrachten kan uitvoeren.

3.1 Het rechtzetten op een witte lijn

Indien regelmatig gecorrigeerd wordt op de calibratieafwijkingen kan de impact ervan geminimaliseerd worden. Deze correctie gebeurt door de witte lijnen als referentie te nemen en hier loodrecht op te oriënteren. Onderstaand algoritme kan gebruikt worden wanneer de robot vlakbij een witte lijn staat (wanneer hij volledig rond zijn as draait zou hij de witte lijn moeten passeren):

Listing 1: Witte Lijn algoritme (pseudocode)

```
1 AngleTurned = 0
2
3 Draai naar rechts tot witte lijn.
4 Draai naar links tot witte lijn EN verhoog AngleTurned.
5 Draai AngleTurned/2 naar rechts.
```

Bij demo 3 is er iets veranderd aan het algoritme. Tijdens het testen zagen we dat als er een zwarte lijn op een witte streep stond (scheiding van twee tegels die tegen elkaar worden gezet), dat de robot zich niet loodrecht kon zetten op de witte lijn. Om dit tegen te gaan is ervoor gezorgd dat als de robot een witte lijn detecteert, hij er eerst helemaal overgaat en dan pas draait om zich loodrecht op de witte lijnen te plaatsen. Het resultaat was dat de robot zich veel beter recht zette op de witte lijn bij het testen.

Alternatief is het mogelijk te corrigeren op de muren, maar dan moet de robot zicht bevinden op een tegel die muren aan beide kanten bevat.

3.2 Onderzoeken van het doolhof

Alvorens de robot het kortste pad tussen twee punten bepaalt, dient het hele doolhof gekend te zijn. Dit verkennen doet de robot autonoom. Elke tegel die de robot reeds passeerde, wordt gemarkeerd. Indien alle tegels gemarkeerd zijn, is de volledige doolhof doorzocht. Het markeren gebeurt via een boolean die op true of false gezet wordt.

Bij elke tegel onthoudt de robot alle uitwegen. Vervolgens slaat hij de eerste nog-niet-bekeken uitweg in. Wanneer de robot op een doodlopend stuk komt, keert hij terug tot een kruispunt waar nog niet alle uitwegen van bekeken werden. Nadat de robot het hele doolhof bekeken heeft, kijkt hij na of de doolhof de 'finish'-barcode bevat. Indien niet, wordt het hele doolhof opnieuw onderzocht.

3.3 Het vinden van het kortste pad

Het kortste pad algoritme maakt gebruik van een graf die opgesteld wordt tijdens het verkennen van de doolhof. Het algoritme is een A* algoritme waarbij de Manhattan heuristiek en een kost wordt gebruikt. De heuristiek is de verwachte afstand tot de "finishTileën is een onderschatting, er wordt dus geen rekening gehouden met muren. De "finishTile"krijgt de heuristiekwaarde 0, de buren hiervan (horizontaal en verticaal, niet diagonaal) de waarde 1 , enzoverder tot alle tegels een heuristiek waarde gekregen hebben. De kost van een tegel is het werkelijke aantal tegels nodig om van de "startTile"tot de "finishTile"te komen.

In de begin situatie zijn alle tegels ongemarkeerd en de kost van de "startTile"wordt op nul gezet. Daarna wordt de kost opbouwend aan elke tegel meegegeven. Hierbij worden de muren in rekening gebracht.

Speciaal voor dit algoritme zijn dus 3 extra velden aan de klasse Tile toegevoegd, nl. een booleanveld isMarked die bijhoudt of dit vakje al onderzocht is of niet, een veld dat de heuristiekwaarde bijhoudt en een veld dat de kost van de tile bijhoudt.

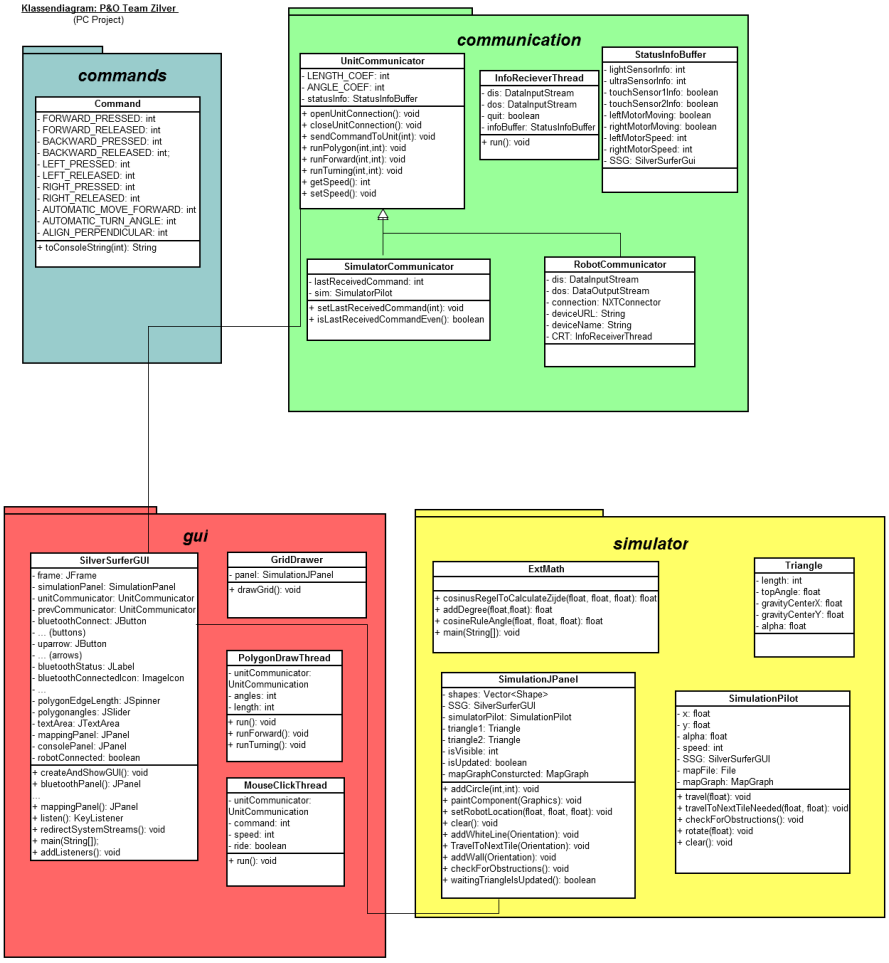
4 Software

De software bestaat uit twee delen: een project dat op de NXT van de robot loopt en een project dat op de computer loopt. Alles wordt aangestuurd via de Graphical User Interface (GUI). Deze toepassing laat toe de robot te besturen (via bluetooth) en de reacties van de robot te simuleren met de simulator. Een Communication-pakket stuurt de commando's van de GUI door naar de juiste unit. De simulator kan gebruikt worden om de software op te testen zonder telkens op de robot te moeten wachten.

4.1 Software ontwerp

Zoals reeds vermeld bestaat de software uit twee projecten: één draait op een computer en één draait op de NXT-Brick. Figuren 7 en 6 tonen een klassendiagram.
Beide projecten hebben een identiek package *commands* met één klasse *Command*. Hierin staan de final static integers die met de mogelijke bluetoothsignalen overeenkomen. Een verdere beschrijving van de software op de NXT-brick wordt in de sectie robot gegeven.

Het computerproject heeft nog zes andere packages: *communication*, *gui*, *simulator*, *mazeAlgorithm*, *audio* en *mapping*. De klasse *SilverSurferGUI* uit de package *gui* implementeert de GUI. De GUI communiceert met de simulator of de robot via de klassen in het package *communication* door een object van de superklasse *UnitCommunicator* bij te houden. Aan deze *UnitCommunicator* is een object van de subklassen *RobotCommunicator* of *SimulatorCommunicator* toegekend. Zo worden de commands dynamisch naar de juiste unit gestuurd: de *RobotCommunicator* communiceert met het NXT-Project, de *SimulatorCommunicator* met de simulator klassen.
Andere klassen van de package *gui* zijn: *MouseClickThread* , *PolygonDrawThread*, *RunForwardThread* en *TurnAngleThread*. De vier Threads zorgen ervoor dat het tekenen van de baan van de robot de rest van het programma niet stillegt.

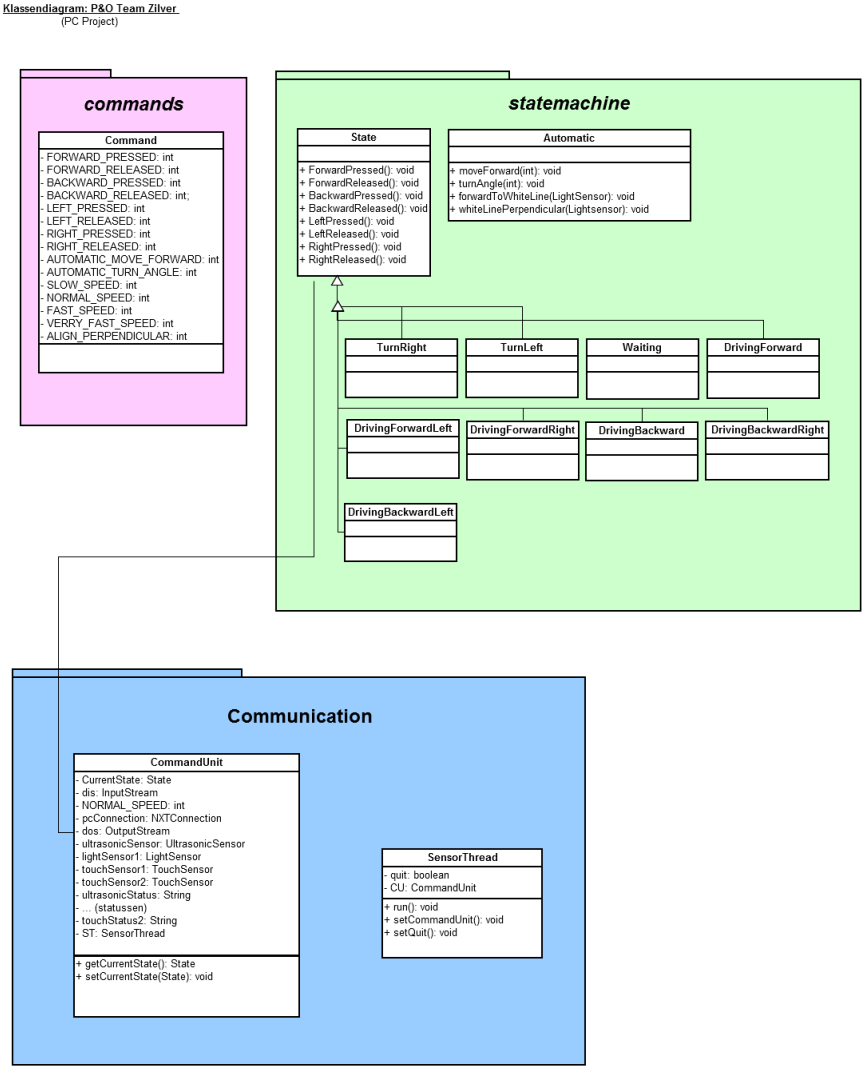


Figuur 6: Klassendiagram van de software die op de PC loopt

Het package *simulator* implementeert de functionaliteit van de simulator.
Het *mapping-package* heeft klassen zoals *Tile, Edge, Obstruction, Barcode, ...* die elementen uit de wereld van een robot voorstellen. Een tile stemt overeen met één tegel van het doolhof en heeft vier edges: één voor elke zijde. Die edges houden de twee aanliggende tegels bij en eventueel een Obstruction, bijvoorbeeld een muur. De klasse *MapGraph* brengt al deze elementen samen. Het houdt een begin-tegel bij en en huidige-tegel. Het biedt functionaliteiten aan om van de huidige tegel naar de tegel Noord, Oost, Zuid of West ervan te reizen en de map dynamisch uit te breiden. Zo wordt impliciet een hele graaf bijgehouden die dynamisch kan aangevuld worden. De klasse *MapReader* kan uit een bepaalde textfile een MapGraph opstellen die overeenkomt met het doolhof dat in het bestand gedefinieerd wordt.

4.2 Het doorgeven van commando's

De GUI zet een actie van de gebruiker om in een commando. Dit commando wordt gerepresenteerd door een integer dat naar de *UnitCommunicator* wordt gestuurd. Twee van de commando's hebben echter extra informatie nodig: *automatic move x cm forward* en *rotate x degrees*. Deze informatie wordt toegevoegd aan de integer door de integer uit te breiden met extra cijfers (dit wordt in de volgende sectie in detail beschreven).
De *UnitCommunicator* stuurt de bekomen integer door naar ofwel de *RobotCommunicator* ofwel de *SimulatorCommunicator*. Deze zetten de integer weer om in de juiste actie van respectievelijk



Figuur 7: Klassendiagram van de software die op de robot loopt

de robot en de simulator. De wijze waarop dit gebeurt, verschilt licht voor beide gevallen. Zo stuurt de *RobotCommunicator* zijn commando's via Bluetooth door naar de NXT-brick, terwijl de *SimulatorCommunicator* zo'n verbinding niet nodig heeft.

4.2.1 De bewerking op de integers

Integers stellen de commando's voor. In twee gevallen is echter meer informatie nodig: om de robot een bepaalde afstand te laten afleggen en om de robot een bepaald aantal graden te laten draaien. Deze afstand en dit aantal graden moet mee doorgegeven worden met de integer. De eenheden waarin de afstand en de hoek worden doorgestuurd zijn respectievelijk cm en graden.

De doorgegeven integer wordt als volgt opgebouwd:

- de waarde van de afstand (hoek) wordt vermenigvuldigd met 1000.
- de integer die het commando representeert, wordt hierbij opgeteld.

Om de bekomen resultaten terug op te splitsen in de twee oorspronkelijke gegevens worden volgende stappen gevolgd:

- een modulo-operatie van tien geeft het laatste cijfer terug. Dit stelt het soort commando voor.
- dit getal wordt van de integer terug afgetrokken.
- de oorspronkelijke afstand (hoek) wordt bekomen door de integer door 1000 te delen.

Deze werkwijze brengt een beperking met zich mee: de waarde van de afstand (hoek) kan slechts tot 2 cijfer(s) na de komma doorgegeven worden. De robot kan niet nauwkeuriger dan 0,1 cm aangestuurd worden. Hierdoor is het niet nodig de afstand nauwkeuriger door te geven. De nauwkeurigheid van de hoek is gevoeliger. De veelhoek stapelt immers veel afrondingsfouten op naarmate de lengte van de zijde en/of het aantal hoeken stijgt. Doordat de begin- en eindpunten niet samen vallen is te zien dat de som van de berekende hoeken samen geen 360 graden vormt.

4.3 GUI

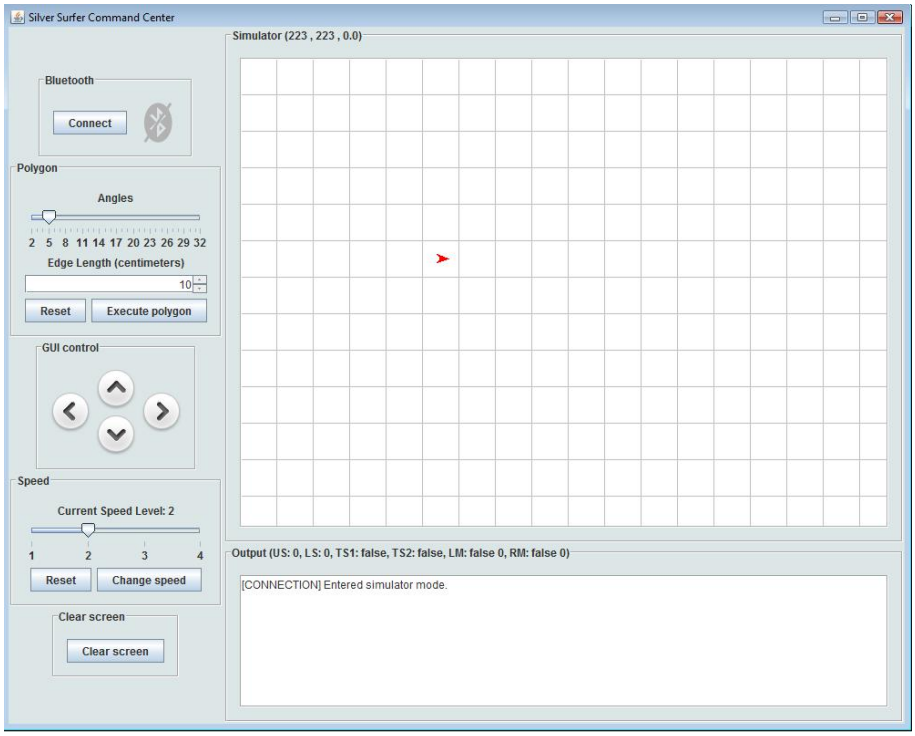
De GUI bestaat uit twee vensters, enkele knoppen, een menubar en enkele instelmogelijkheden: zie figuur 8. Deze knoppen besturen ofwel de robot en de simulator ofwel enkel de simulator, naargelang de bluetooth verbinding aan staat.

Voor visualisatie van de sensoren zijn er grafieken die de waarden weergeven. De robot zelf op de simulator heeft een blauwe straal die aangeeft of er een muur voor hem staat of niet. Daarnaast zijn er ook nog knoppen die bepaalde functies uitvoeren, zoals "turn left/right", "look aroundën "Align on the white line/ walls".

4.4 Bluetooth

De communicatie tussen robot en computer gebeurt volledig via bluetooth. De GUI voorziet een knop om deze verbinding te maken en geeft de status van de verbinding weer. De GUI stuurt commando's door naar de robot via de klassen *UnitCommunicator* en *RobotCommunicator*.

De leJOS-API2̂ voorziet de *NXTConnector* klasse. De methode *connectTo((String, String, int, int)* zet de bluetoothverbinding tussen computer en brick op. De belangrijkste argumenten hiervoor zijn de naam van de NXT-Brick en zijn DeviceUrl - in het geval van de gebruikte NXT: 'Silver' en '00:16:53:0A:04:5A'.



Figuur 8: Grafische User Interface

4.5 Robot

Het project op de NXT-brick bestaat uit één hoofdklasse, één commandoklasse en enkele thread-klassen. Bij het opstarten, initialiseert de robot zijn sensoren. Wanneer de computer verbinding maakt, initialiseert de robot ook zijn 'datastreams' en start hij een *sensorthread* op.

De computer stuurt commando's in de vorm van integers. De robot vertaalt deze met behulp van de klasse *Command* en voert de juiste actie uit. Sommige methodes geven een resultaat mee aan de computer. Dit gebeurt door ze op een datastream te zetten, voorafgaand door een tag (bijvoorbeeld [US] voor ultrasonic sensor data).

Enkele belangrijke methodes:

- *updatecoordinates:* de robot houd zijn positie bij. bij elke beweging zullen de coordinaten worden geupdated.
- *updateStatus:* stuur alle statusgegevens (lichtsensor, coordinaten, ...) naar de computer.
- *main:* wacht tot de computer een commando geeft en voer dit uit.

De belangrijkste commandos zijn: - AlignPerpendicular (zet zich goed op de witte lijn, dit commando roept de methode alignonwhiteline in de klasse automatic op (zie onder)) - AlignWall (zet zich goed tussen 2 muren, dit commando roept de methode alignWall in klasse automatic op (zie onder)) - CloseConnection (sluit de connectie door de sensorthread the sluiten, de datastreams af te sluiten, de connectie te verbreken en het programma te laten eindigen) - lookAround (kijkt rond voor openingen/muren, roept methode lookaround in klasse automatic op) - playSong (maakt een songthread aan die een liedje afspeelt- - AutomaticMoveForward en AutomaticTurnAngle De laatste 2 commandos krijgen als input een invoer in cm of graden (naargelang het command), dit vermenigvuldigd met 100 en daarbij opgeteld de int van het juiste commando. Dit wordt terug omgezet naar de juiste invoer in cm of graden, en vervolgens naar een invoer in graden die de wielen moeten draaien om de juiste hoek of afstand af te leggen. Deze input wordt meegegeven aan een methode in autoamtic (zie onder).

De robot heeft een state die aangeeft wat het momenteel aan het doen is. De mogelijkheden zijn: - Driving forward (naar voor rijden zolang de robot in deze state is- - Driving forward left (linksvoor) - Driving forward right (rechtsvoor) - Driving backward - Driving backward left - Driving backward right - Turn left - Turn right - Waiting (wachten tot er een nieuwe state wordt gegeven) - Automatic (in deze state zal de robot een bepaalde methode met bepaalde parameters uitvoeren die hij meekreeg van de main klasse, na de uitvoering zal de state terug naar waiting gaan)

De klasse automatic is de meest belangrijke van de states. De gebruikte methodes hierin zijn: - beweeg naar voor met barcodedetectie - beweeg naar voor zonder barcodedetectie - draai - zet loodrecht op witte lijn - zet halverwege tussen 2 muren - kijk rond en zoek naar muren/openingen - lees de barcode de methode "beweeg x cm naar voor met barcodedetectieïs de standaardbeweegmethode, de andere wordt alleen gebruikt bij het lezen van de barcode zelf (zie verder). De beweeg en draai methodes krijgen een input van x in graden die het wiel moet draaien. Deze graden worden berekent door de main klasse (zie hoger). De methode "zet halverwege tussen 2 muren"zal een boolean teruggeven die zegt of het gelukt is. De methode "lees de barcode"zal de barcode lezen door 2 cm naar voor te gaan, de lichtsensorwaarde te lezen, deze toe te voegen aan een string, en dan de volgende bar lezen. Het resultaat wordt teruggegeven aan de computer.

Tenslotte zijn er nog enkele thread: - SensorThread - BarcodeThread - WhiteLineThread - Songthread (zie boven) De sensorthread zal elke 50 miliseconden alle data van zijn sensoren opslagen en deze naar de computer sturen. Dit gebeurt vaak genoeg voor up to date info maar niet teveel zodat we de stream niet overflowen. De barcodethread wordt gemaakt bij een moveforward commando, deze zal de beweging stoppen wanneer een zwarte lijn wordt gedetecteerd. De moveforward zal dan de methode readbarcode oproepen, die de barcode leest. De Whitelinethread zal naar voor bewegen tot hij wordt gestopt door de methode alignonwhiteline wanneer deze een witte lijn detecteerd.

Een *Finite State Machine* geeft de robot een concreet uitvoeringsschema. Een object van de klasse *CommandUnit* houdt een currentState bij - een object van de superklasse *State*. Negen subklassen verzorgen het nodige onderscheid: *Waiting*, *DrivingBackward*, *TurnLeft*, *DrivingRightForward*, en één speciale toestand: de klasse *Automatic*. De *CommandUnit* ontvangt commando's via bluetooth. De *CommandUnit* wijzigt de currentState naar een nieuw object van een *State*-subklasse. De constructoren van elke subklasse passen het gedrag van de motoren aan bij de overgang naar de bepaalde state. Bijvoorbeeld: De robot staat stil en zijn currentState is *Waiting*. Als de gebruiker de robot voorwaarts wil laten gaan, drukt hij de up-toets in. De *CommandUnit* wijzigt de currentState van de robot naar currentState.ForwardPressed(). Dit levert een currentState van het type *DrivingForward* op. De robot beweegt voorwaarts. Tot slot is er de klasse *Automatic*. Deze heeft vier methoden: *turnAngle(int)*, *driveForward(int)*, *forwardToWhiteLine(LightSensor)* en *whiteLinePerpendicular(LightSensor)*. De *CommandUnit* roept deze methoden op met argumenten die hij geparsed heeft uit de informatie van de ontvangen signalen, zoals beschreven in sectie 4.2.

4.6 Simulator

De simulator bootst de werking van de robot virtueel na. Hij kan dezelfde commando's uitvoeren als de werkelijke robot. De GUI stuurt de simulator aan via de klassen *UnitCommunicator* en *SimulatorCommunicator*. Deze ontvangen de uit te voeren commando's, analyseren ze en sturen ze door naar de simulator.
De simulator zelf bestaat uit zeven klassen:

- *Wall:* tekent de muren van het doolhof.
- *State:* wordt gebruikt om te zeggen of iets horizontaal of verticaal moet staan.
- *SimulationSensorData*
- *SimulationPilot:* houdt de positie en de richting van de 'robot' bij.
- *SimulationPanel:* tekent de baan van de 'robot' in het tekenpaneel.

- *GridDrawer:* zet het grid op.
- *Triangle:* berekent de hoekpunten van de driehoek die de 'robot' voorstelt.
- *ExtMath:* doet enkele berekeningen.
- *Bag:*[3] een geheugen opslagplaats waar dubbels in worden opgeslagen.

Het opzetten van het tekenpaneel gebeurt in de klasse *SilverSurferGUI*. De klasse *GridDrawer* voegt hier een grid aan toe. De roosters hebben dezelfde afmetingen als de secties van de panelen. Zo kan een muur enkel op een lijn van het grid staan. Wanneer de 'robot' een pad aflegt, tekent de simulator dit in het tekenpaneel als een rode lijn (herschaald: één cm = één pixel). De lijn bestaat uit verschillende cirkels die elkaar gedeeltelijk overlappen. Het *SimulationPanel* houdt alle bezochte coördinaten bij. De klasse bevat een methode die deze cirkels één na één tekent. Dit zorgt ervoor dat de lijn continu bijgewerkt wordt. De huidige positie en de huidige orientatie van de 'robot' wordt bovenaan het simulatorpaneel in de GUI weergegeven.

5 Besluit

De uiteindelijke fysieke bouw van de robot bestaat uit de NXT, de wielen met aandrijvingen, een ultrasone sensor, een lichtsensor en twee druksensoren. De calibratie van de robot zorgt ervoor dat hij precies aangestuurd kan worden en dat de sensoren waarden teruggeven die overeenkomen met de werkelijkheid. De GUI gebruikt een eenvoudige interface en bevat knoppen waarmee de robot handmatig bestuurd kan worden. Door het ingeven van parameters in de GUI kan de robot een willekeurige veelhoek rijden. De uitleeswaarden van de sensoren vertellen de gebruiker wat de robot 'ziet'. Op deze manier krijgt de gebruiker een idee over de doolhof waar de robot in rijdt. Bovendien kan de robot zich loodrecht oriënteren op een witte lijn. De simulator is ook verbonden aan de GUI. Deze voert dezelfde opdracht uit als de robot. Het is mogelijk een virtueel doolhof te laten en de simulator hier in te laten 'rijden'. De simulator is handig om testen op uit te voeren, zonder steeds op de robot te moeten wachten.

A Demo 1

De robot wordt voor demo 1 nog niet voorzien van sensoren. De focus ligt vooral op de nauwkeurigheid van de besturing en op het implementeren van alle softwarecomponenten. Deze software bestaat uit twee projecten: een op de computer en een op de robot. De computersoftware bestaat uit een Grafical User Interface (GUI), enkele Communication klassen die informatie doorsturen en enkele klassen die de werking van de robot simuleren (simulator).

A.1 Resultaten

Bij het rijden van de veelhoek had de robot een kleine afwijking. De afwijking werd groter bij grotere veelhoeken en meerdere hoeken om wille van de geaggregeerde fout.

A.2 Conclusies

Calibratie moet meer getest worden. De GUI kan gebruiksvriendelijker, ook moet het scherm van de simulator herschaald worden, zodat de lijn van simulator nog wordt weergegeven als de simulator van de robot uit het scherm gaat.

A.3 Oplijsting aanpassingen verslag

Volgende secties werden aangepast ten opzichte van de eerste demonstratie:

- *2.1 Fysieke bouw:* de sensoren werden toegevoegd.
- *2.2 Calibratie van de motoren:* opnieuw gedaan.
- *2.3 Calibratie van de lichtsensor:* nieuwe sectie.
- *2.4 Calibratie van de ultrasone sensor:* nieuwe sectie.
- **??** *Het rijden van een veelhoek:* nieuwe sectie.
- *3.1 Het rechtzetten op een witte lijn:* nieuwe sectie.
- *4.1 Software ontwerp:* enkele nieuwe klassen en mapping-package.
- *4.3 GUI:* enkele nieuwe functionaliteiten.
- *4.6 Simulator:* grid en triangle, virtuele doolhof.

B Demo 2

De sensoren worden voor demo 2 wel geïnstalleerd. Na calibratie kunnen ze informatie doorzenden naar de robot. Threads zorgen ervoor dat de robot tegelijkertijd sensorwaarden kan lezen doorsturen.

De meetwaarden worden in de GUI weergegeven zodat een gebruiker de robot kan besturen zonder deze te zien. Bovendien is de simulator gekoppeld aan de robot. Wat de robot doet, doet de simulator ook en wordt getekend in de GUI. De simulator kan ook onafhankelijk van de robot opereren. Het is mogelijk een virtuele doolhof te laden en te simuleren dat de 'robot' zich hierdoor beweegt. Zowel robot als simulator kunnen zich rechtzetten op een (virtuele) witte lijn. Het is bovendien mogelijk de robot zich in het midden van de tegel te laten zetten.

B.1 Resultaten

Het verslag voor de tweede demonstratie bevatte enkele paragrafen die tegen het einde nog snel geschreven waren. Deze paragrafen waren van mindere kwaliteit. Ook sommige afbeeldingen werden verkeerd ingevoegd.

Op de demonstratie reed de robot de eerste tegels zoals het hoorde. Ook het witte lijnalgoritme werd goed uitgevoerd. De sensorwaarden werden weergegeven in de GUI, zodat de bestuurder, die de robot en het doolhof niet zag, zich een beeld kon vormen van het doolhof. Op een gegeven moment gaf de bestuurder de robot opdracht zich midden op een tegel te zetten. Dit algoritme maakt gebruik van de muren rond de tegel. De robot bevond zich op dat ogenblik echter op een tegel die door geen enkele muur omsloten werd. De bestuurder had dit niet eerst nagekeken. Dit had als gevolg dat de robot helemaal scheef stond, zonder dat de bestuurder dit wist.

De sensorwaarden werden niet gesimuleerd in de simulator. De simulator maakte in zijn algoritmes rechtstreeks gebruik van de virtuele doolhof. Robot en simulator maakten met andere woorden gebruik van andere algoritmes, wat niet het doel is van een simulator.

De simulator kon wel door een virtuele doolhof manoeuvreren. Meestal 'botste' hij tegen de muren, maar soms reed hij toch door een muur.

B.2 Conclusies

Aan het verslag zou vroeger begonnen moeten worden zodat domme fouten vermeden kunnen worden.

De allign-algoritmes (op een witte lijn en in het midden van een tegel) werken enkel in specifieke situaties. De algoritmes dienen enkel in deze omstandigheden uitgevoerd te worden.

De simulator dient de sensorwaarden te simuleren en zou dezelfde algoritmes moeten gebruiken als de robot. Op deze manier kunnen de algoritmes getest worden zonder steeds op de robot te wachten.

B.3 Oplijsting aanpassingen verslag

Volgende secties werden aangepast ten opzichte van de tweede demonstratie:

- *Abstract:* aangepast.
- *2.3 Calibratie van de lichtsensor:* meetresultaten toegevoegd.
- *2.4 Calibratie van de ultrasone sensor:* meetresultaten toegevoegd en aangepast.
- *Het rijden van een veelhoek:* er uitgehaald.
- *3.1 Het rechtzetten op een witte lijn:* aangepast.
- *3.2 Algoritme onderzoeken doolhof:* nieuwe sectie.
- *3.3 Algoritme Kortste pad:* nieuwe sectie.
- *4.1 Software ontwerp:* enkele nieuwe klassen en packages.
- *4.3 GUI:* aangepast.
- *4.6 Simulator:* toevoegingen.

Referenties

[1] *Lego Mindstorms*: Een uitbreiding op de LEGO bouwstenen waarmee kleine, aanpasbare en programmeerbare robots gebouwd kunnen worden. Een centrale besturingsmodule ('the brick') kan geprogrammeerd worden met verschillende programmeertalen. In eerdere versies werd een RCX gebruikt voor de brick, nu wordt met NXT gewerkt. De brick kan enkele motoren aandrijven. Bovendien kunnen er verschillende sensoren, o.a. een ultrasone sensor en een lichtsensor, aangesloten worden. [www.lego.com] [http://en.wikipedia.org/wiki/Lego–Mindstorms]

[2] *leJOS*: Een kleine Java Virtuele Machine die toelaat de NXT-brick te programmeren. leJOS voorziet verschillende klassen die o.a. de motoren aansturen en een bluetoothverbinding opzetten. [http://lejos.sourceforge.net/]