Richtlijn

Project Robotica TI2

A.M. Gieling & M. Oldenburg

28 januari 2022

Randvoorwaarden

- Een robot bestaat uit **zowel fysieke hardware als immateriële software** (alleen een simulator bouwen is daarom niet toegestaan!).
- Er wordt gebruik gemaakt van een **gedistribueerde controller-architectuur**, op zijn minst onderverdeeld in een primaire en secundaire controller. De primaire controller MOET in staat zijn fully-managed objectgeoriënteerde software te draaien (dus geen 8 bit microcontroller aan de primaire kant bijvoorbeeld). Ook een Arduino (nep-OO) is om die reden niet toegestaan.
- Er wordt gebruik gemaakt van zowel **sensoren als actuatoren**.
- De software voor de primaire controller (de systeembrede aansturing) **wordt objectgeoriënteerd gespecificeerd m.b.v. UML**.
- De software voor de primaire controller wordt gerealiseerd in een algemeen inzetbare (general purpose) **objectgeoriënteerde taal** naar keuze. DSL's (domain-specific languages) zijn niet toegestaan, ook niet als ze OO zijn.
- De functionaliteit van de robot is gebaseerd op de integratie van een door elke student **zelf opgestelde user story/narrative**. *Een user story is een functionele voorstelling van een praktijksituatie waaruit duidelijk wordt wat de robot allemaal kan.* Alle user stories in een projectgroep maken tezamen een integraal en samenhangend geheel. User stories dienen binnen twee weken na aanvang van het project opgesteld te zijn (in de tweede week mag een ruwe schets worden overlegd, in de derde week dient de user story volledig te zijn). De use case mag gebaseerd zijn op een opdrachtomschrijving uit het bedrijfsleven (graag zelfs - hoe reëler, hoe beter).
- In de vierde week levert de groep een **mock-up** op die de mechanica of fysieke werking van de robot visualiseert. Dit mag een kartonnen model zijn, een filmpje van een mime-voorstelling, een simulatie in een rekenpakket, maar een 3D model in een ontwerppakket (CAD) is ook goed – zo lang het maar **zelfgemaakt** is.

- Als de groep zelf elektronica ontwikkelt, dient zij een bespreekbaar **blokschema** (uiterlijk in week 4) en daarna een volledig **elektrisch schema** (uiterlijk in week 6) te overleggen van het prototype. Alleen dan zal er genoeg tijd zijn alle onderdelen aan te schaffen en te assembleren.
- Een groep die met een Darwin2 aan de gang gaat hoeft geen mock-up te maken, en geen elektrisch (blok)schema. De hardware ligt in dat geval vast. De software is echter een veel grotere uitdaging.
- In de user story van de robot komt duidelijk naar voren dat de robot de kerntaken van de opdracht **autonoom** kan uitvoeren op basis van sensor-actuator technologie, zonder tussenkomst van een operator. Dat betekent **NIET** dat **ALLE** taken van het systeem autonoom behoeven te worden uitgevoerd, alleen de kerntaken die als individuele opdrachten zijn gekozen.
- Elke groep bestaat uit **3 studenten**, behalve misschien 1 groep die voornamelijk uit herkansers bestaat.
- Als er twee studenten over zijn, bijvoorbeeld omdat het aantal studenten in de klas niet deelbaar is door drie, of omdat er in de loop van het project een student afhaakt van een groep van 3, dan wordt van de overblijvers de ambitie verwacht dat zij de opdracht als tweetal uitvoeren. Mogelijk dienen er aanpassingen aan de eisen of user stories te worden gemaakt, dit alles in goed overleg met de begeleiders. Als er één student over is, dan wordt van deze student verwacht dat hij ofwel een passende variant van het project op basis van de eigen user story afrondt, of hij wordt naar het inzicht van de begeleidend docenten aan een bestaande groep toegevoegd, waarbij hij een nieuwe user story zal moeten maken die aansluit op het project van de nieuwe groep. Alles in goed overleg met de begeleiders.
- Vóór het tweede begeleidingsmoment schrijft **ELKE** deelnemer een **user story** die deel uit gaat maken van het project in grote lijnen. Ook kiest **ELKE** deelnemer een **specialisme**. Tijdens de tweede projectbijeenkomst worden de projectopdrachten, de user stories en de specialismen door de begeleiders gekeurd.
- Het gekozen specialisme **bepaalt de focus** van de hardware/software-ontwikkeling en is bepalend voor de rode draad van het **op te leveren artikel**.
- Begeleiders bepalen of een projectkeuze voldoet aan de genoemde voorwaarden. Een voorbeeld: sommige studenten willen hun project in PHP schrijven. Ze argumenteren dat dit een objectgeoriënteerde taal is, omdat je er klassen mee kunt aanmaken. Maar, zoals de meeste ervaren programmeurs misschien wel weten: "just because you are using classes, that does not make it OOP". En er zijn nou eenmaal talen die het paradigma sterker ondersteunen dan andere. Uiteraard is PHP Object-Capable, maar deze voldoet niet als OO-taal voor dit project, omdat de taal met haar functiebibliotheek ook net zo makkelijk Procedure-Capable is. Kortom, de taal helpt helemaal niet om je de discipline te geven die je nodig hebt

als je OO echt aan het leren bent. De kans dat je de taal op een niet-OO-conforme wijze in gaat zetten is in onze ervaring ca. 100%, waardoor de taal afvalt als optie.

Uitleg over het gekozen specialisme

Het gekozen specialisme is bepalend voor de nadruk die elke deelnemer legt op een bepaald aspect van het ontwikkelproces. In zo'n groot project moeten ontwikkelaars keuzes maken: sommige componenten moeten worden 'ingekocht'. Met inkopen wordt hier eerder bedoeld 'outsourcen' / 'aanbesteden'. Een voorbeeld: een groep maakt een drone die gelijktijdig moet kunnen navigeren, objecten moet kunnen ontwijken en beelden moet kunnen interpreteren van gewassen in een kas. Hier zitten meerdere hele zware specialismen in:

- Low-level navigeren (de firmware van de IMU en de flight controller);
- High-level applicatiespecifiek navigeren (een veilige 'flight path' zoeken door een kas);
- Low-level beeldherkennen (cameradriver maken om de camera te kunnen gebruiken onder Linux);
- High-level applicatiespecifiek beeldherkennen (het herkennen van individuele planten en hun ontwikkeling);
- Low-level connectivity (het uitlijnen van hardwarebussen en softwaredrivers, d.w.z. UART-, USB-, Ethernet-, SPI-, I^2C-, en GPIO-drivers maken voor de gekozen hardware, zodat de higher-level software deze modules met eenvoudige calls kan aanspreken);
- High-level applicatiespecifieke connectivity (het versleuteld en betrouwbaar overbrengen van plantdata naar een server waarmee automatisch onderhandeld kan worden op een beveiligd netwerk, waarin het niet mogelijk is dat er data verloren gaat);
- Low-level concurrency (de hardware zodanig ontwerpen dat diverse primaire en secundaire CPU's of cores hun eigen taken krijgen en de stuurinformatie hierover bij elkaar brengen op de primaire controller);
- High-level applicatiespecifieke concurrency (de symbolische [software-]architectuur van de drone zodanig ontwerpen, dat meerdere computationeel zware taken gelijktijdig, met de juiste prioriteiten, altijd op tijd en zonder haperen kunnen worden uitgevoerd).

De low-level taken vallen eigenlijk allemaal af, omdat het doel van dit project het maken van een high-level toepassing is (geschikt voor OO-implementatie). De projectgroep kiest er dus voor om alle low-level hardware en firmware 'in te kopen' / 'te outsourcen'. Er zullen inkopen worden gedaan van kant-en-klare modules (hoewel er uiteraard 'glue-electronics' moet worden gerealiseerd om alles aan elkaar te koppelen) en er wordt voor de low-level software naar open-source pakketten gezocht, zodat de hardware aanstuurbaar is vanuit OO-programma's zonder ook maar 1 regel code te schrijven.

Omdat het gekozen specialisme *high-level* moet zijn, maken de studenten de volgende verdeling: één student gaat de high-level navigatie (pathfinding) ontwikkelen, een andere high-level beeldherkenning (computer vision) en weer een andere high-level concurrency (parallelle OO-architectuur). Er is één high-level taak die zodoende blijft 'liggen': de high-level connectivity. Ook dit onderdeel wordt daarom ge-outsourced, d.w.z. er moet een open source pakket / softwarestack voor worden gevonden. De groep besluit om een aantal bestaande protocollen te kiezen: HTTP voor een marked-up weergave van gebruiksdata in het front-end, SDP en RTP voor het versturen van real-time videodata, Telnet voor een gesprek tussen 2 controllers en SFTP voor het remote opslaan van backup-bestanden. Ze schrijven wat 'glue code' om deze modules in hun eigen architectuur op te nemen.

In de praktijk komt het er door deze keuzes op neer dat de student die in dit voorbeeld concurrency kiest, de architectuur van de primaire software moet ontwerpen. Immers, hij is degene die verantwoordelijk is voor een vloeiende samenwerking tussen de navigatie en de beeldherkenning, die beiden gelijktijdig op de primaire controller moeten gebeuren.

In het op te leveren onderzoeksartikel neemt het gekozen specialisme een centrale plaats in: student 1 schrijft een onderzoek naar 'drone pathfinding with obstacles', student 2 schrijft over 'computer vision for agricultural crops' en nummer 3 onderzoekt 'smooth drone operation under heavy parallel task load'. Merk op dat alle 3 een *functionele* omschrijving krijgen. Dit is **extreem** belangrijk, omdat *real engineering* nooit in een vacuüm plaatsvindt. Er wordt altijd ge-ingenieurd op basis van een zekere toepassing, je *doet het ergens voor*. Als ingenieur moet je dit *ten allen tijden helder hebben* voor jezelf.

Het eindresultaat van deze projectgroep heeft sensoren, actuatoren, een OO-ontwerp, OO-software, het werkt autonoom (vliegen, beeldverwerking en *smooth concurrent operation*), bestaat uit een gedistribueerde controller-architectuur, heeft zowel hardware als softwarecomponenten en is gebaseerd op een geloofwaardige en praktijkgerichte user-story. What could go wrong?

Voorbeelden van specialismen kunnen zijn:
High-level (ook wel: '*symbolisch voor te stellen*') ...

- ... pathfinding,
- ... algoritmiek / datastructuren,
- ... computer vision,
- ... concurrency,
- ... regelsystemen / besturingstechniek,
- ... real-timetechniek,
- ... meet- & regeltechniek (sensorkalibratie / -ijking),
- ... speltheorie (besliskunde, AI),
- ... zoektechnieken (AI),
- ... besturingssystemen,
- ... distributed systems,

- . . . machine learning / neural networks,
- . . . speech recognition,
- . . . telematica / infrastructuur,
- . . . etc. etc.

Let op: hoe verder het specialisme af staat van de opleiding, hoe meer de student zelf moet bijleren! Denk er ook aan dat je je specialisme in je user story **NIET** met bovenstaande bewoordingen mag beschrijven. In je user story *benoem je welk functioneel probleem je gaat oplossen.*

Een voorbeeld: je kiest het specialisme *machine learning* omdat je bijvoorbeeld spraakherkenning wilt realiseren die zich aanpast aan de spreker door in de loop van de tijd diens stem steeds beter te leren verstaan. In je user story benoem je in zo'n geval je specialisme als: 'zelflerende gebruikersspecifieke spraakherkenning voor het geven van commando's aan een dimbare lamp' (om zomaar een voorbeeld te noemen). Je maakt het dus **heel erg concreet en applicatiespecifiek**. We kunnen dit uit je user story opmaken, waarin je een context geeft waaruit blijkt dat (in dit voorbeeld) . . . *de gebruiker een lamp nodig heeft die zijn dimstand op basis van spraakcommando's kan regelen, en die na een inleerperiode 100% betrouwbaar reageert op alleen de stem van deze gebruiker.* Wees daarom creatief, concreet en specifiek in je user story!