Rubik's Cube Solver

Silke Knossen (11025948) Daniël van de Pavert (11045418)
Kim Gouweleeuw (11039841) Daniël Vink (10675140)

Juni 2016

Inleiding

Dit project gaat om een robot iets te laten doen wat grenzen verlegt en innovatief is. Wij gaan met de Mindstorm EV3 pakketten zelf een robot bouwen die een Rubik's Cube oplost. Om dit tot stand te laten komen moet de robot een bakje hebben waar de Rubik's Cube inligt. Dit bakje moet horizontaal gedraaid kunnen worden. Bovendien moet de robot een grijparm hebben om de Rubik's Cube verticaal om te draaien en hem vast te houden zodat alleen de onderste rij gedraaid wordt. Verder heeft de robot een kleurherkenningssensor nodig waardoor de robot kan zien hoe de Rubik's Cube er op dat moment uitziet. De robot moet kunnen zien of er daadwerkelijk een Rubik's Cube in zijn bakje ligt. Daarvoor wordt een infraroodsensor gebruikt. Daarmee kan de afstand tussen de sensor en een object worden bepaald. Wanneer de afstand tot een voorwerp kleiner is dan een bepaalde waarde, ligt er dus een kubus in het bakje en mag de robot beginnen met het oplossen. Om het probleem op te lossen is het noodzakelijk een algoritme voor het Rubik's Cube probleem te schrijven nadat de robot gebouwd is.

Het algoritme gaat uit van een beginstaat en een eindstaat. Er moet een weg worden gevonden van de beginstaat naar de eindstaat. Hiervoor is een representatie van de Rubik's Cube nodig. De representatie van de beginstaat wordt steeds geüpdate naar een tussenstaat tot de eindstaat is bereikt. Deze weg zal het snelst en best worden gevonden met behulp van het A* algoritme.

Voortgangsreportage

Dag 1

Robot bouwen

Om de robot te bouwen hebben wij ons laten inspireren op een voorbeeld. Dit voorbeeld vereiste echter onderdelen die wij niet tot onze beschikking hadden. We hebben dit opgelost door de werking van bepaalde delen van de robot te bestuderen en dit met de onderdelen die we wel tot onze beschikking hadden na te bouwen. De robot is uiteindelijk in theorie functioneel voor ons probleem. Het moet nog blijken of dit in de praktijk ook het geval is. Dit kan pas worden getest wanneer er een algoritme is geschreven. Midden op de robot is een bakje gemaakt met de mogelijkheid om te draaien door de aansluiting op een grote motor. Dit is het bakje waar de Rubik's Cube in kan liggen. Ook is er, aangesloten op de tweede grote motor, een grijparm gemaakt die de Rubik's Cube vast kan houden terwijl het bakje draait. Dit zorgt ervoor dat alleen de onderste laag van de kubus gedraaid wordt. Bovendien maakt deze arm het mogelijk de Rubik's Cube een kwartslag te kantelen. Deze drie elementaire bewegingen kunnen samen alle bewegingen maken die nodig zijn voor het oplossen van de Rubik's Cube. Verder hebben we een kleurensensor aangesloten op de kleine motor, zodat deze kan bewegen en de kleuren van de Rubik's Cube in kan scannen. Als laatste hebben we ook nog een infraroodsensor gemonteerd op de robot, die de afstand tot de Rubik's Cube kan bepalen. We sluiten de motor van het bakje aan op output A; de motor van de grijparm aan op output B en de motor van de kleuren sensor aan op output C. Daarnaast wordt de kleurensensor aangesloten aan input 1 en de infraroodsensor zelf wordt aangesloten aan input 2.

Alogritme

Doordat de keuze is gevallen op een Rubik's Cube oplossende robot, moest er een een representatie gemaakt worden van een kubus. Doordat nog niet duidelijk was welke taal toepasbaar was op de Mindstorm Brick is het concept op papier gezet in Java omdat we daar het meeste ervaring mee hebben en omdat dit redelijk eenvoudig te schrijven valt. Voor de kubusrepresentatie is gekozen voor een lijst met daarin zes lijsten, elk voor een kant van de kubus. Deze zes lijsten bevatten allemaal 9 elementen, de vlakjes van een kant van de kubus. Voor de elementen gebruiken we integers, 1 voor blauw, 2 voor oranje, 3 voor wit, 4 voor rood, 5 voor groen, en 6 voor geel. Het einddoel van de kubus is om in elk van de zes lijsten in zijn geheel gevuld is met hetzelfde element. Dus één lijst met zes 1'en, één lijst met slechts 2'en, enz.

Dag 2

Errors Java

Gezien er op de EV3 brick een andere JDK functioneel was dan op de laptop waarmee er op dat moment gewerkt werd, was er een onverwachte error. Dit is verholpen door de JDK waarmee op de laptop gewerkt werd terug te draaien van 1.8 naar 1.7. Toen deze laatste error verholpen was, bleek de connectie tussen de laptop en brick goed te werken en de verschillende testen met het LCD scherm en de speaker van de brick waren succesvol. Het volgende probleem kwam naar voren zodra er geprobeerd werd met de motoren of sensoren te communiceren.De error vertelde ons dat de gevraagde port niet geopend kon worden. Deze error hebben wij niet kunnen verhelpen. Er is hiervoor hulp gezocht, zonder resultaat. Aan het eind van de dag zijn we tot de conclusie gekomen dat Java niet werkte voor wat wij wilde doen. Daarom besloten we de volgende dag te beginnen aan het programmeren in MatLab.

Algoritme

Gekozen is om de bewegingen die de machine kan maken te hardcoden waarbij letterlijke vertalingen van Fridrichs F2L (Rubik's Cube oplosmethode) algoritmes in Java zijn gezet. Dit kostte behoorlijk wat tijd en inspanning (7 pagina's voor slechts het eerste vlak).

Dag 3

MatLab

We zijn begonnen met werken in MatLab omdat Java niet bleek te werken. We hebben verbinding gemaakt met de Mindstorm EV3 door hem aan te sluiten via de USB-port. We misten namelijk een onderdeel om hem via wifi te verbinden. Nadat we de connectie hadden getest zijn we begonnen een functie te schijven waardoor de motor waarop het bakje is aangesloten gaat draaien. Deze daaiing moet echter steeds 90 graden verschillen van een uitgangsstand. Het bakje moet dus 90, 180 of 270 graden kunnen draaien. Doordat er echter tandwielen aan de motor verbonden zijn moet de draaiing van de motor groter zijn dan 90 graden. Om het bakje 90 graden te draaien moet de motor dus een grotere rotatie maken. We hebben naar aanleiding van de tandwielen uitgerekend dat om het bakje 90 graden te draaien een 3 keer zo grote rotatie van de motor nodig is. De motor draait uiteindelijk dus 3*90 graden om het bakje één slag linksom te draaien. Toen we dit testte bleek het bakje verder te draaien dan gepland. Dit komt natuurlijk doordat de motor snelheid krijgt en wanneer de hoek bereikt is, draait de motor nog een beetje door. Om dit op te lossen hebben we een functie geschreven (turnLeft) die werkt alsvolgt: De rotatie wordt meegegeven aan de functie. Zo lang de hoek nog niet bereikt is, draait het bakje door. Als de hoek bereikt is en de motor gestopt is, wordt uitgerekend hoeveel de motor te ver heeft doorgedraaid. De rotatie die gemaakt is door de motor wordt opgevraagd en het overschot wordt omgezet in een terugdraaiing. Als de snelheid van de rotatie groot is, is de overschot groot. Als er een kleine hoek gemaakt moet worden, zal er dus veel overschot zijn. Om dit te verhelpen hebben we de snelheid van de draaiing variabel gemaakt. Wanneer er een grote rotatie moet plaatsvinden zal de snelheid groter zijn dan wanneer er een kleine rotatie moet plaatvinden. (Deze methode is later toegepast op alle motoren, omdat alle motoren hetzelfde probleem hebben). Zodoende wordt bijna exact over de hoek die meegegeven wordt gedraaid. We draaien absoluut ten opzichte van het nul punt, want zo blijft het nul punt vaststaan. We

willen namelijk niet dat een kleine afwijking steeds wordt meegerekend in het berekenen van de volgende hoek.

De volgende functies die we hebben geschreven zijn voor de grijparm. De eerste functie van de grijparm (rolCube) zorgt ervoor dat de Rubik's Cube in zijn geheel kan kantelen. Deze functie is een kopie van de functie die het bakje laat draaien. De hoek die wordt meegegeven is echter van kleinere waarde, namelijk 200 graden. Dit komt doordat deze motor niet geheel om zijn as kan draaien. Verder moet deze grijparm na elke beweging weer terug naar zijn originele stand. Hierom wordt de functie voor het draaien twee maal aangeroepen. Een keer een draaiing tegen de klok in, en daarna dezelfde rotatie met de klok mee.

De tweede functie voor de grijparm (grabCube) zorgt ervoor dat de Rubik's Cube op zijn plek gehouden wordt. Vervolgens moet het bakje gaan draaien waardoor de onderste laag van de kubus wordt gedraaid. De functie voor de grijparm moet dus hoek maken waardoor de arm de kubus in een greep houdt. Hiervoor hebben we geen berekening kunnen bedenken, maar na een paar keer proberen zijn we op een hoek van 90 graden gekomen. De motor van de arm wordt op dat punt stil gezet.

De kleurensensor die aangesloten is op de input-port 2 bleek niet goed te werken voor de Rubik's Cube die wij gebruikten. We hebben geprobeerd de lichtintensiteit te beïnvloeden waardoor de kleuren beter herkenbaar zouden zijn. Dit gaf echter geen resultaat. We hebben aan een medestudent gevraagd of zij andere Rubik's Cubes had, die we zouden kunnen testen. Zij zou ze de volgende dag voor ons meebrengen.

Java runnen in MatLab

Er is op meerdere manieren geprobeerd Java klasses in MatLab te runnen, zowel door de .class files aan het classpath.txt betand te voegen als door de .jar files in het pad van matlab te zetten. Beide methoden bleken niet te werken gezien er geen object aangemaakt kon worden van de geimporteerde klassen. Dus we hebben besloten om alles in MatLab te schrijven.

Algoritme

In plaats van F2L (de menselijke tactiek) te gebruiken, hebben we besloten over te stappen op een depth-first search systeem. Om dat mogelijk te maken hebben we alle bewegingen die de machine kan maken om moeten zetten naar mogelijke state-veranderingen. De kubus kan bijvoorbeeld 1, 2 of 3 maal om zijn x-as draaien, 1, 2 of 3 maal om zijn y-as draaien, en er kan slechts 1 laag 1, 2 of 3 maal om zijn y-as gedraaid worden. Dit hebben we uitgewerkt op papier. Daarnaast zijn al deze state-veranderingen vertaald naar Java functies die de representatie correct aanpassen door de elementen of lijsten van index te veranderen. Nu moeten we nog een werkend depth-first zoekalgoritme creëeren dat deze states kan lezen en probeert naar het einddoel te bewegen.

Dag 4

MatLab

De Rubik's Cubes die wij deze dag tot onze beschikking hadden hebben we getest. Er kwam uiteindelijk één kubus uit die alle kleuren juist aangaf, behalve de kleur oranje. Dit komt omdat oranje niet wordt herkend door de kleurensensor. Omdat het teveel tijd zou kosten dit in de kleurensensor aan te passen hebben we ervoor gekozen zwarte stickers op de oranje vlakjes te plakken. Zwart wordt namelijk wel herkend door de kleurensensor. Zo hebben we uiteindelijk zes verschillende kleuren die we kunnen gebruiken in het algoritme.

De functies voor de grijparm werkte nog niet optimaal. We hebben de snelheid van de terugrotatie na het grijpen aangepast. Wanneer de snelheid namelijk hoger is, is de kans op vastlopen kleiner. Hierdoor maakt de grijparm een soepelere beweging. Wat bleek was dat de motor van de grijparm niet stopte nadat de arm was tergugedraaid. Dit hebben we ook kunnen aanpassen door de functie een vaste hoek mee te geven en wanneer de rotatie groter is dan die hoek, stopt de motor.

Het inscannen van de kleuren kan alleen als de kleurensensor loodrecht op het goede vlakje van de Rubik's Cube staat. Hiervoor moest de motor die deze kleurensensor bestuurt de goede hoek maken. We hebben een functie geschreven die de verschillende standen van de motor kan uitvoeren (readColor). Ook deze rotaties zijn absoluut.

We hebben een functie geschreven (scanColors) die een vlak van de Rubik's Cube inscant. De kleurenscanner beweegt naar het midden van de kubus toe. Daar scant het de kleur van het middelste vlakje. Daarna beweegt de sensor een klein beetje terug, waardoor het boven het dichtstijzijnde vlakje staat. Deze kleur wordt dan ook gescand. Vervolgens draait het bakje steeds 45 graden naar links en wordt elke volgende kleur gescand. Echter bleek dat de kleurensensor niet altijd de juiste kleur scant. De oorzak hiervoor was dat het bakje niet precies om het middelpunt draait.We hebben nog eens naar de constructie gekeken. De pin van de motor bleek niet in het midden van het bakje te blijven. Het was niet stevig genoeg in elkaar gezet, waardoor het midden verschoof. We hebben dit opgelost door de constructie te verstevigen waardoor het draaipunt perfect in het midden bleef. Hierdoor kan de kleurensensor de vlakjes van de Rubik's Cube beter inscannen.

We hebben vervolgens de functie scanColors uitgebreid waardoor de hele Rubik's Cube wordt ingescand. De scanner beweegt naar het middelpunt en scant eerst een heel vlak. Vervolgens wordt door de grijparm de kubus een kwartslag gedraaid. Daarna scant de kleurenscanner weer het hele vlak. Dit wordt twee keer herhaald. Vervolgens wordt de kubus 90 graden gedraaid en gekanteld. Dan wordt dat vlak weer gescand. Voor het laatste vlak wordt de kubus twee keer gekanteld en wederom het vlak ingescand. Alles gaat ten stotte terug naar de beginpositie.

Na ons moment van euforie bleek echter dat de motor van de kleurenscanner niet meer goed naar boven het middelste vlakje van de kubus wilde bewegen. Dit is voor ons nog altijd een raadsel, aangezien we niks in de code veranderd hebben en het hiervoor wel gewoon werkte.

Ook hebben we twee functies geschreven om een lijst met kleuren om te zetten naar een lijst van de bebehorende int's. Deze functies zijn switchColor en forLoop, de laatste is helaas nooit gebruikt omdat het handiger bleek te zijn om switchColor direct aan te roepen vanuit scanColor.

Algoritme

We hebben veel tijd besteed aan het werkend krijgen van Java in MatLab. Dat is niet gelukt waardoor we gekozen hebben om de functies voor de representatie van de kubus in MatLab te schrijven. In plaats van zes lijsten in een lijst is er geopteerd voor een matrix van 6 x 9, 6 zijdes met elk 9 vakjes. Daarna zijn er vier verschillende soorten functies geschreven die de representatie van de kubus correct aanpassen gebaseerd op de bewegingen die de robot kan maken.

1. flipCube(Twice/Thrice)
   Deze functie komt overeen met de kubus één of meerdere keren een kwartslag om zijn x-as te draaien. Kan maar één kant op dus vandaar een driedubbele move in plaats van één tegen de klok in. De volgorde van de kolommen wordt hier aangepast. Kolommen wisselen van plek door te roteren met de klok mee.

2. turnCube(Twice/Counter)
   Deze functie komt overeen met de kubus één of twee keer om zijn y-as te draaien. Counter draait de kubus tegen de klok in. De volgorde van de kolommen wordt hier aangepast. Kolommen wisselen van plek door te roteren met de klok mee.

3. turnSide(Twice/Counter)
   Deze functie is hetzelfde als turnCube, alleen dan voor slechts de onderste laag in plaats van de hele kubus. Subkolommen van 3 elementen worden hier gewisseld met elkaar.

4. orderSide(Counter)
   Deze functie draait een zijde met de klok mee of tegen de klok in. Nodig voor de zijkanten bij flipCube of de boven en onderkant bij turnCube en turnSide. Het verwisselt elementen in een kolom met elkaar.

Om de kleuren te representeren is gekozen voor integers en om te testen is een testcube opgebouwd in opgeloste staat waar de functies op losgelaten zijn. Ze lijken perfect te werken. Nu is het nog nodig om de kleurenscanner de integers in de lijst in te vullen en om een zoekalgoritme in Matlab te schrijven.

Dag 5

MatLab

De motor van de kleurensensor werkte nog steeds niet optimaal. We hebben hier weer naar gekeken, zonder resultaat. We hebben een paar aanpassingen geprobeerd, zoals een grotere snelheid wanneer er over een kleine hoek wordt gedraaid. We dachten namelijk dat de motor een te kleine hoek moest maken en daar moeite mee had. Dit bleek niet zo te zijn. Verder zou het kunnen liggen aan het feit dat de motor zichzelf steeds corrigeert en daar nooit mee stopt. Dit is echter uitgesloten, want dan zou het geval zijn dat de motor kleine bewegingen maakt. De motor maakte echter geen beweging meer.

De grijparm bleek niet elke keer de kubus goed om te gooien. De hoek die gemaakt werd, was te groot. Daardoor schoof de arm van de kubus af en schoof hem op dezelfde kant weer terug tijdens de terugdraaiing. Dit hebben we opgelost door de hoek die de grijparm maakt kleiner te maken, namelijk 190 graden in plaats van 200.

Vanwege het tijdsgebrek is het niet gelukt het algoritme te implementeren. In plaats hiervan hebben we een code geschreven die uitgaat van een opgeloste Rubik's Cube en daar een patroon van een bloem in maakt. De bewegingen die gemaakt moeten worden zijn gehardcoded met de eerder gemaakte functies.

De kleuren die gescand werden door de kleurensensor moesten nog in een cellarray gezet worden. We hebben de functie scanColors uitgebreid. De functie roept switchColor aan, waardoor de naam van de kleur wordt omgezet in een cijfer tussen 1 en 6. Deze cijfers worden vervolgens in de array gezet. Zo krijgen we een matrix van 6 bij 9. Na het inscannen staan deze cijfers echter nog niet op de juiste plek met betrekking tot de representatie van de kubus die gebruikt wordt voor het algoritme. Daarvoor is de functie newOrder gemaakt. Deze zet de cijfers die worden ingescand op de juiste plek voor de representatie.

Algoritme

Nu de representatie van de kubus in Matlab werkte hebben we geprobeerd om een A*-algoritme te schrijven dat als beginpunt de kubus gebruikt die wordt ingelezen door de kleurensensor en als doel een opgeloste kubus. Doordat er niet veel tijd over was tot de presentatie hebben we ervoor gekozen om slechts één kant, de witte, op te lossen. Voor A* is het nodig om waardes aan de states te geven, die hebben we gemaakt aan de hand van de amountOfWhite(kubus) functie. Deze functie zoekt in de ingelezen kubus naar de kolom waar het vijfde element wit is. Het vijfde element is namelijk het middelste vakje van een kubuszijde en kan niet van positie ten opzichte van de andere blokjes veranderen. Daarom moet er om die vakje heen worden gebouwd. Daarna wordt het aantal witte vlakjes in deze kolom geteld wat vergeleken wordt met de doelwaarde 9 (een volledig gekleurde kant). Hoe hoger de waarde die uit de amountOfWhite functie komt, hoe beter dus. Deze functie wordt dan aangeroepen na elke mogelijke stateverandering (de functies van de vorige dag) en de waarde zou moeten toegekend worden aan elke stateverandering waardoor de best mogelijke zet gekozen kan worden. Nu was het nog de zaak om de kosten van gemaakte zetten mee te rekenen in de waarde van een zet en om de stateveranderingen te koppelen aan bewegingen van de robot. Hier hebben we helaas geen tijd meer voor gehad voor onze presentatie.

Discussie

Het project was een grote uitdaging voor de tijd die we ervoor hadden. We zijn tot de conclusie gekomen dat er veel standaard afwijkingen zitten in de motoren van de robot. Wat verder jammer is, is dat het bakje net groter is dan de Rubik's Cube zelf. Dit is nodig om de kubus om te rollen, maar dit zorgt er

ook voor dat wanneer de onderste laag gedraaid wordt, de kubus soms niet netjes uitgelijnd wordt. We zijn erachter gekomen dat het mogelijk is een Rubik's Cube solver te maken in MatLab, maar dit vergt echter meer tijd. Een vervolg stap zou dan ook zijn het A* algoritme toe te passen die de combinatie maakt met de representaties die wij geschreven hebben en de functies van de robot. We hebben van dit project geleerd hoe we programmeren in MatLab en dat de koppeling naar de robot heel nauwkeurig bepaald moet worden en hoe een Rubik's Cube moet opgelost worden (slechts één van ons had de vaardigheden dit te doen).

Appendix

Dit project werkte met Mindstorms EV3 en werd geprogrammeerd in MatLab.

Bouwinstructies waar wij onze robot op gebaseerd hebben:
`http://mindcuber.com/mindcub3r/MindCub3r-Ed-v1p1.pdf`

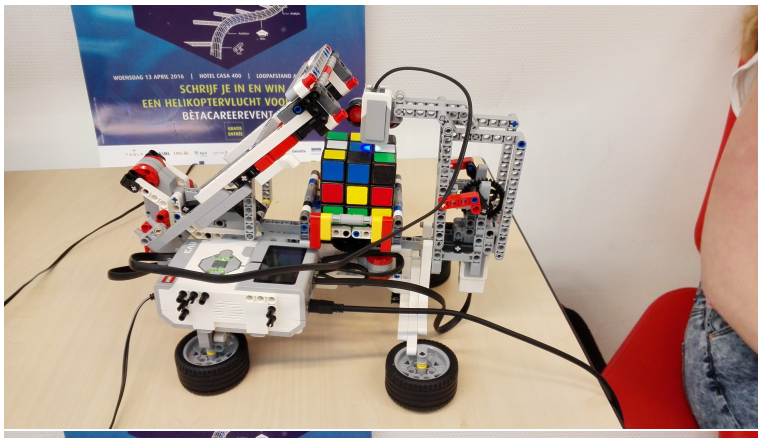
Het algoritme van Fridrich:
`https://solvethecube.com/#step1`

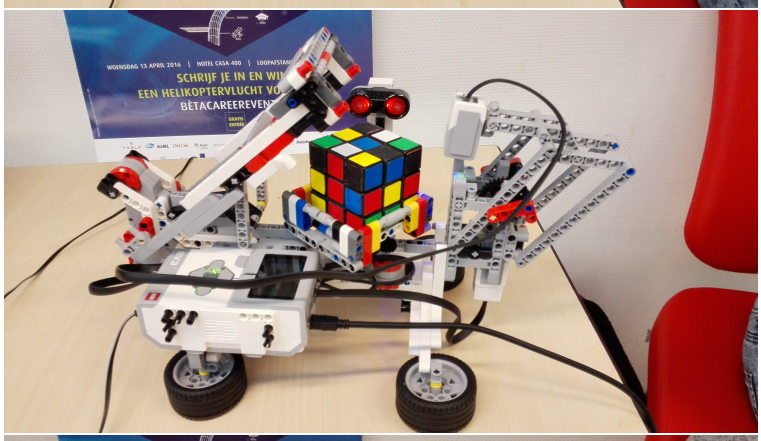
Link naar gezipte file met alle functies voor Ruby:
`https://www.dropbox.com/s/ak06sc9vejl6gl1/RUBI.zip?dl=0`

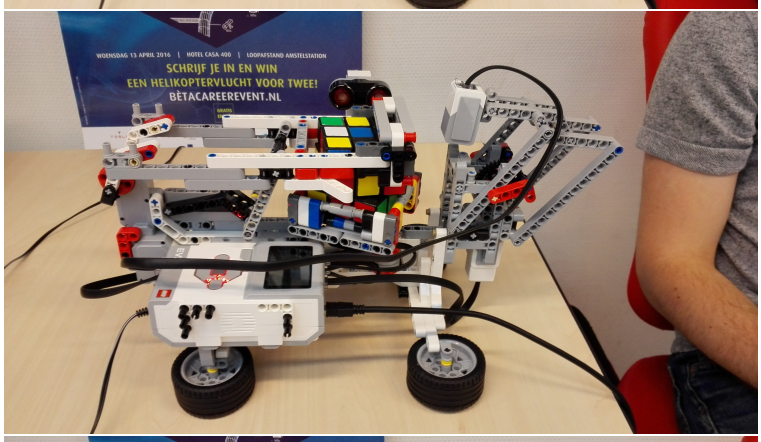
Het filmpje voor het inscannen van de vlakken:
`https://www.youtube.com/watch?v=ljBuPNe5YVE&feature=youtu.be`

Het filmpje voor het maken van een bloempatroon:
`https://www.youtube.com/watch?v=w_VwtZda1KA&feature=youtu.be`

Afbeeldingen van verschillende standen van de robot, ter verduidelijking:






WOENSDAG 13 APRIL 2016 | HOTEL CASA 400 | LOOPAFSTAND AMSTELSTATION
SCHRIJF JE IN EN WIN
EEN HELIKOPTERVLUCHT VOOR TWEE!
BÈTACAREEREVENT.NL