Autonoom Voertuig

Beschrijving projectopdracht

Begrippenlijst

Neuraal netwerk – Een combinatie van hard- en software die zich gedraagt als verzameling onderling verbonden knooppunten. De verbindingen geven signalen door met instelbare versterkingsfactoren en de knooppunten voeren op de som van de binnenkomende signalen vaak een niet-lineaire bewerking uit. De versterkingsfactoren worden, meestal met behulp van supervised learning, zo ingesteld dat afhankelijk van de aan het netwerk aangeboden ingangssignalen, wenselijke, goed gedefinieerde uitgangssignalen ontstaan. Deze uitgangssignalen dienen vaak als basis voor herkenning of besturing.

Node – Een knooppunt zoals in boven beschreven netwerk.

Edge – De verbinding tussen twee knooppunten.

Relu function (REctified Linear Unit function) – De functie *lambda x: 0 if x < 0 else x*

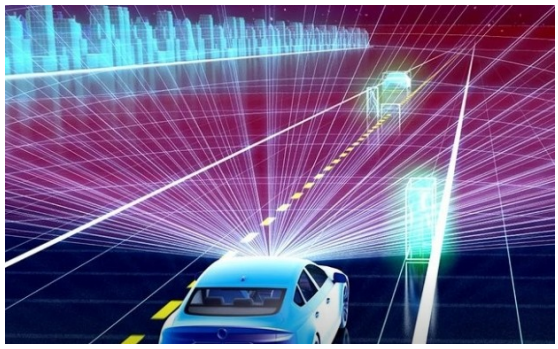
Deze functie wordt vaak gebruikt als niet-lineaire bewerking na een node omdat hij snel te berekenen is. Een niet-lineaire bewerking is nodig omdat een meerlaags fully connected neuraal netwerk anders equivalent is met slechts één laag, dus zinloos. Het effect van de niet-lineaire bewerking is een toename van het aantal mogelijk ingangspatronen dat aan de uitgang kan worden gerepresenteerd. Niet-lineaire verbanden zijn moeilijk te analyseren maar in de natuur schering en inslag.

Fully connected net – Alle nodes van een laag zijn verbonden met alle nodes in de volgende laag. Er zijn ook neurale netwerken die niet fully connected zijn, zoals convolutional nets die veel worden gebruikt om beelden te herkennen.

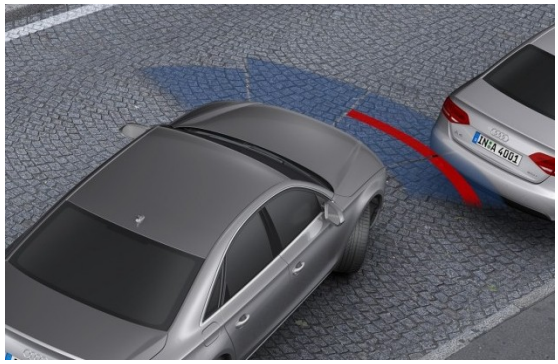
Convolutional net – Een meerlaags neuraal netwerk waarbij signalen aan de ingang van knooppunten van een bepaalde laag een gewogen som zijn van signalen aan de uitgang van lokale knooppunten in de voorgaande laag. Het effect is het bevoordelen en daarmee herkennen van bepaalde lokale patronen.

Netwerk topologie – De manier waarop de knooppunten binnen een neuraal netwerk met elkaar verbonden zijn, met andere woorden de “plattegrond” van het netwerk.

LIDAR (Light Detection And Ranging – Een apparaat dat door middel van reflectie van een lichtbundel de positie van objecten kan bepalen. Een “Scanning LIDAR” is een LIDAR waarvan de lichtbundel z’n omgeving in het platte vlak (2D) of ruimtelijk (3D) aftast.



SONAR (Sound Detection And Ranging) – Een apparaat dat door middel van reflectie van een geluidsbundel afstanden van objecten kan bepalen. Naast het feit dat geluidsgolven een medium nodig hebben en lichtgolven niet, is het ook zo dat geluidsgolven veel langer zijn dan lichtgolven. Daardoor waaiert een geluidsbundel meer uit, waardoor de hoek waaronder SONAR een voorwerp kan detecteren veel minder nauwkeurig is dan bij LIDAR.



Medium – Materiaal waar doorheen golven zich kunnen voortplanten (voortbewegen), bijvoorbeeld lucht of water.

Hoek-resolutie – Het kleinste waarneembare verschil in hoek van een object ten opzichte van de waarnemer.

Imperatief programmeren – Een wijze van programmeren waarbij de computer stap voor stap hardcoded krijgt voorgeschreven wat hij afhankelijk van de situatie moet doen. Zo’n situatie-afhankelijke reeks stappen heet een algoritme, niet te verwarren met het populaire hedendaagse spraakgebruik, waarbij met algoritme software voor het automatisch analyseren van “big data” wordt bedoeld.

Hardcoded – Uitgeschreven in de vorm van programma-statements die de het gedrag van het programma onder gegeven omstandigheden volledig vastleggen.

Programma-statement – Atomaire (letterlijk: ondeelbare) opdracht in een computer programma. Een statement kan wel executie van een groot aantal andere statements tot gevolg hebben. Zo'n statement heet een functie-call.

Big data – Omvangrijke verzamelingen gegevens, vaak in een verscheidenheid aan data-formaten.

Data-formaat – De manier waarop gegevens worden gerepresenteerd.

Simulatie (in de context van deze opdracht) – Een computerprogramma waarvan de programma-objecten het gedrag van fysieke objecten nabootsen.

Socket-verbinding – Een communicatiekanaal tussen twee computerprogramma's die al dan niet op verschillende computers kunnen draaien. Via zo'n kanaal kunnen boodschappen worden verstuurd, bijvoorbeeld in de vorm van strings, vaak in JSON formaat.

Client – Een computerprogramma dat uitwisseling van diensten initieert en daarna onderhoudt met een ander computerprogramma, bijvoorbeeld via een socket-verbinding.

Server – Een computerprogramma dat op initiatief van een client een uitwisseling van diensten onderhoudt met die client. Merk op dat de termen client en server slaan op de richting van het initiatief tot contact, niet op wie aan wie een dienst verleent. De benaming "client" en "server" zijn wat dat betreft misleidend.

Voorbeeld A - Gedienstige server:

> Client: Knippen en scheren alstublieft. Server: Gaat u zitten. Wilt u thee?

Voorbeeld B - Gedienstige client:

> Client: Sire, hierbij meld ik mij present. Server: Klets niet en poets mijn schoenen.

Het initiatief ligt in beide gevallen bij de client. In dit project is de besturing de client en het gesimuleerde voertuig de server. De server wordt als eerste opgestart en wacht tot de client zich meldt. Het ligt voor de hand te zeggen dat de besturing ten dienste staat van het voertuig, net als de motor en de koplampen. Zo beschouwd lijkt dit project op voorbeeld B, de client neemt het initiatief tot het verlenen van een dienst aan de server.

String (zoals in "string of pearls", parelketting) – Aaneengeregen rijtje lettertekens.

JSON (JavaScript Object Notation) – Een gestandaardiseerde, leesbare, CO_2-productieve manier om objecten en lijsten van objecten in strings om te zetten. Ontleend aan de programmeertaal JavaScript, echter in principe bruikbaar binnen elke programmeertaal.

Ill conditioning – Het optreden van grove reken-onnauwkeurigheden ten gevolge van het naast elkaar voorkomen van getallen die heel veel en heel weinig van nul verschillen in het zelfde numerieke wiskundige probleem.

Numeriek wiskundig probleem – Een wiskundig probleem dat niet wordt opgelost met formele wiskundige analyse maar met een hoop rekenwerk op de computer.

Terminal (command prompt) – Een systeemprogramma dat commando's accepteert vanaf het toetsenbord, meestal op je scherm zichtbaar als een venster met een zwarte, grijswitte of blauwe achtergrond. Werken via een terminal geeft je een hele directe, nauwkeurige manier om de computer naar je hand te zetten. Terminalcommando's kunnen aaneengeregen worden tot scripts, die weer kunnen worden aangestuurd vanuit o.a. Python.

Systeemprogramma – Een programma dat deel uit maakt van de "eigen intelligentie" van je computer, oftewel van het besturingssysteem, zoals Linux, MacOS of Windows. Dit in tegenstelling tot programma's voor een specifieke toepassing, zoals tekstverwerkers, browsers en spreadsheets.

Operationeel – Letterlijk: Uitvoerbaar. Een beschrijving die zodanig concreet is dat hij kan worden gebruikt als stap voor stap handleiding bij een activiteit.

Meta-functioneel – Letterlijk: De functionaliteit overstijgend. Het gaat hier om bedrijfsmatige, ethisch-maatschappelijke, juridische, regulerings- en technische kaders die de directe functionele eisen overstijgen.

Kwantitatief – Beschreven in getallen, meer dan in termen die een subjectief oordeel inhouden.

Kwalitatief (in de context van deze opdracht) – Beschreven in subjectieve termen zoals veel, weinig, goed, slecht, veilig, onveilig, snel, traag, efficient, inefficient.

Versiebeheer – Een werkwijze waarbij alle versies van een document of broncode-bestand eenduidig benoemd en gedateerd zijn, en permanent beschikbaar blijven.

MoSCoW (Must have, Should have, Could have, Won't have) – Een manier om onderdelen van het projectresultaat te klassificeren op een schaal van essentieel tot "toeters en bellen".

Timeboxing – Indien qua volgorde volgens de MoSCoW klassificatie wordt gewerkt is dit een manier om de kosten van een project te begrenzen. Het idee is dat een project wordt afgekapt als het geld op is. Doordat met de belangrijkste zaken (Must have's) wordt begonnen is het project resultaat vaak toch practisch bruikbaar, ook al is niet aan alle wensen voldaan. Timeboxing is een antwoord op ongebreidelde budget-overschrijdingen bij IT projecten.

Scrum – Een eclectische (overal het beste van uitpikken) werkwijze die in veel bedrijven losjes wordt toegepast om opdrachtgevers in staat te stellen een vinger aan de pols te houden en ontsporen van IT projecten vroegtijdig te signaleren en te voorkomen.

MoSCoW en Timeboxing worden vaak toegepast in de context van Scrum. Verder kenmerk van Scrum is regelmatig kort en bondig overleg tussen ontwikkelaars en stakeholders (belanghebbenden).

Scrum is onderdeel van Agile. De term Scrum komt uit de rugby-sport en verwijst naar nauw contact, in het geval van rugby lijfelijk.

Agile (behendig) – Een paradigma (denkwijze) uit de bedrijfskunde waarbij flexibiliteit van de bedrijfsprocessen centraal staat. Bedrijfsprocessen zijn flexibel als ze te allen tijde bij te sturen zijn. Bij software-ontwikkeling voorziet Scrum hierin door frequent overleg met de betrokkenen.

DevOps – Door velen gezien als de opvolger van Scrum. Een werkwijze waarbij naast ontwikkeling van software ook het gebruik ervan in de feedback loop is opgenomen. De houdbaarheid van IT methoden zoals Scrum en DevOps ligt tussen de 5 en 10 jaar. Dit is minder erg dan het lijkt, omdat er, met enig vallen en opstaan, sprake is van evolutionaire progressie.

TensorFlow/Keras – C++ library van Google voor het maken van neurale netwerken, met daaroverheen een Python laagje (Keras) om hem makkelijker te kunnen gebruiken. TensorFlow kan gebruik maken van de rekencapaciteit van sommige grafische kaarten. Voor dit project is dat niet nodig.

Tensor – Een blok getallen.

SciKitLearn – Python library voor Machine Learning, zoals gebruik van neurale netwerken en het doen van regressie-analyse. Kenmerk is dat snel en “eenvoudig” applicaties kunnen worden opgezet en ook zonder inzet van een grafische kaart efficient wordt gerekend.

Regressie-analyse – Het ontdekken van verbanden in numerieke gegevens die de werkelijkheid beschrijven.

SimPyLC – Python library voor realtime simulatie met daarin o.a. het auto voorbeeld.

Dependencies – De dependencies van een library L zijn de libraries die L direct of indirect gebruikt.

Resources – Hulpbronnen, vaak met een financieel of ecologisch prijskaartje.

Safety – Beveiliging tegen ongelukken.

Security – Beveiliging tegen opzettelijke bedreigingen.

Layered Safety c.q. Security – Beveiliging bestaande uit een aantal “verdedigingslinies” die onafhankelijk zijn, waardoor de totale kans op een calamiteit het product van de kansen op het doorbreken van één van de linies is, en dus sterk afneemt met het aantal linies.

Doel

Het doel van deze opdracht is, practische ervaring op te doen met het gebruik (in Python) van een neuraal netwerk voor het besturen van een realtime systeem, in dit geval een voertuig.

Daarbij gaat het vooral om het fundamentele verschil tussen het imperatief programmeren van een door menselijk vernuft bedachte, situatie-afhankelijke, expliciete strategie enerzijds, tegenover de a-specifieke benadering waarbij een neuraal netwerk impliciet patronen leert herkennen, zonder dat er sprake is van zo’n strategie.

Specifiek worden twee situaties onderzocht, besturing met behulp van een 2D Scanning LIDAR, waarbij het voertuig een nauwkeurig 2-dimensionaal beeld van z’n omgeving krijgt en besturing door een drietal losse SONAR modules, waarbij het voertuig slechts een grove indicatie krijgt van de positie van obstakels.

Inzicht in de verschillen in hoekresolutie tussen LIDAR en SONAR technologie leiden bij imperatief programmeren tot verschillende expliciete besturings-algoritmen.

Bij uitvoering van dit project wordt je geconfronteerd met een aantal concrete punten waarin gebruik van een neural net afwijkt van gebruik van een expliciet, imperatief geprogrammeerd algoritme.

Besteed als je aan dit project werkt aandacht aan de meta-functionele verschillen tussen een neural net en een expliciet algoritme. Dit betreft onder andere de mate waarin:

- detailkennis over de technologie van het te besturen systeem nodig is om de besturing te maken
- het gedrag van de besturing in specifieke situaties te verklaren is
- het gedrag van de besturing in nieuwe situaties te voorspellen is
- een mens ethisch gezien verantwoordelijk is, dan wel juridisch verantwoordelijk kan worden gesteld voor ongelukken

Besteed indien bovengenoemde punten invloed hebben op je ontwerpkeuzen, hieraan kort aandacht in je projectplan. Stel bovengenoemde punten daarnaast uitgebreider aan de orde je projectevaluatie, ook indien ze geen directe invloed op je keuzen hebben gehad.

Wat heeft meer invloed op het resultaat, de verschillen tussen enerzijds SciKitLearn en anderzijds Tensorflow/Keras of de verschillen in gekozen netwerk-topologie, ongeacht welke library je gebruikt.

Een bijkomend doel van dit project is ervaring op te doen met het flexibel en ordelijk opzetten van software met behulp van Python. Probeer je programma's zo op te zetten dat voor overstap van SciKitLearn op Keras/Tensorflow slechts geringe wijzigingen nodig zijn en de overall structuur van je programma gehandhaafd blijft. En dergelijke programma-opzet heet "stabiel" omdat het de tand des tijds kan doorstaan. Het is zogezegd duurzaam en bespaart daarmee mens-uren en andere resources.

Inhoud

Gegeven is een eenvoudige simulatie van een auto, die wel aan de natuurwetten voldoet: Als je te hard rijdt, vlieg je uit de bocht. De gesimuleerde auto kan worden bestuurd via een socket-verbinding. Begrip van de werking van de simulator zelf is voor uitvoering van dit project niet nodig.

Gegeven is tevens een hardcoded besturing in het bestand *hardcoded_client.py* in de folder *control_clients*. Gebruik deze code tevens als voorbeeld voor communicatie met de gesimuleerde auto, die de rol van server vervuld. Merk op dat het dictionary *sensors* de meetgegevens van de LIDAR of SONAR bevat en dictionary *actuators* de gewenste stuurhoek en snelheid. De gegevens van de drie vaste SONAR units worden, samen met de resulterende stuurhoek, volledig gelogd. De gegevens van de LIDAR worden gereduceerd in dimensionaliteit, door de gegevens van de LIDAR niet bijvoorbeeld per graad door te geven maar de zichthoek van de LIDAR in een aantal (bijvoorbeeld zestien) sectoren te verdelen. Ook met deze opsplitsing heeft de LIDAR gemiddeld een 16/3 keer zo nauwkeurige hoek-resolutie als de SONAR.
Vergelijk de LIDAR-besturingsstrategie in method *lidarSweep* met de SONAR-besturingsstrategie in

functie *sonarSweep*. Probeer in samenspraak met andere studenten en, indien nodig, de docent, te komen tot een formulering in “gewoon nederlands” van beide strategieen en de verschillen ertussen.

Laat het voertuig daarna achtereenvolgens met beide strategieen rijden op de bijpassende baan (*lidar.track* of *sonar.track*) en de bovengenoemde gegevens loggen.

Maak vervolgens in Python een fully connected neuraal netwerk met tussen de 2 en de 5 lagen en per laag tussen de 8 en de 256 nodes met behulp van SciKitLearn. Gebruik een *relu* activation function. Er is één analoge uitgangsnode. Gebruik de eerder gegenereerde logfiles als trainingsdata. Houd rekening met het volgende:

- Omdat alleen om de trainingsbaan hoeft te worden gereden, is de trainingset gelijk aan de testset. Er hoeft dus, net als bij de Formule 1, geen rekening gehouden te worden met overfitting, het gaat alleen om de prestaties op *dit* “circuit”.
- Begrens als eerste stap alle afstanden op 20 meter, om ill conditioning te voorkomen.
- Pas daarna voorafgaand aan de training per kolom uniforme schaling toe naar het bereik [-1, 1]
- Vergeet niet weer terug te schalen bij het gebruik van het getrainde netwerk om de auto te besturen!

Probeer verschillende laagbreedtes en aantallen lagen uit. Je mag ook nog ander zaken proberen, zoals een andere activation function. Kijk wat het beste werkt.

Maak, als je een optimum hebt bereikt voor LIDAR en SONAR, precies hetzelfde netwerk met TensorFlow/Keras. Wat zijn de verschillen?

Werkwijze

Installeer de simulator, bijvoorbeeld met:

python -m pip install SimPyLC

Maak een folder voor de programma’s die je voor dit project gaat schrijven.

Open een terminal binnen deze folder en haal de voorbeeld-simulaties en de documentatie op:

python -m simpylc -h en dan de aanwijzingen opvolgen.

Plaats indien je met Windows werkt de DLL’s van de simulator in een folder in je DLL zoekpad, bijvoorbeeld *C:\Windows\System32*.

Installeer SciKitLearn en TensorFlow/Keras (Google is your friend).

Het is niet nodig TensorFlow op je graphics card te laten werken. De warnings die dit produceert kun je negeren.

Probeer de simulator uit aan de hand van de documentatie die je verkrijgt met terminal commando

python -m simpylc -d

Ga naar de folder *car_neural_nets* en start de simulatie met terminal commando

python world.py

Ga daarna naar de folder *control_clients* en start vanuit een terminal de hardcoded control met terminal commando

python hardcoded_client.py

Probeer SciKitLearn en Tensorflow/Keras uit aan de hand van één of meer kleine voorbeelden uit de documentatie.

Stel een projectplan op. Dit document bevat ten minste:

Context – Een beschrijving van de fysieke, bedrijfsmatige en/of maatschappelijke omgeving waarin het projectresultaat zal worden gebruikt. De context heeft invloed op zowel eisen aan het projectresultaat als ontwerpkeuzen. Bij dit project maakt het bijvoorbeeld veel verschil of het bestuurde voertuig in de binnenstad van Rotterdam of op een afgesloten race-circuit moet rijden.

Bewustzijn van de context zorgt ervoor dat geen requirements worden vergeten en dat het ontwerp een mate van degelijkheid vertoont die past bij de situatie. Een besturing van een modeltrein en een echte trein kunnen qua functionele eisen veel op elkaar lijken, maar qua betrouwbaarheid zijn er verschillen die tot uitdrukking komen in het ontwerp, bijvoorbeeld wat betreft “layered safety”.

Requirements – Eisen aan het eindresultaat, puntsgewijs, nauwkeurig (liefst kwantitatief) geformuleerd en met MoSCow aanduidingen.

Testspecs – Operationele beschrijving van de (eveneens bij voorkeur kwantitatieve) tests die nodig zijn om vast te stellen of aan de requirements wordt voldaan.

Ontwerp – Een beschrijving van de te volgen oplossingsstrategieen met voldoende details om programmering door een weldenkde programmeur (in tegenstelling tot “coder”) mogelijk te maken, maar niet meer dan dat. Diagrammen, lijstjes en berekeningen kunnen deel uitmaken van een ontwerp. Het ontwerp zelf is echter de *boodschap, niet de vorm* waarin deze is gegoten.

Een goede manier van ontwerpen is Literate Programming (zie Google), waarbij documentatie zoveel mogelijk voorafgaand aan programmering in de broncode-bestanden wordt geplaatst. Dit heeft als voordeel dat documentatie altijd bij de hand is en dus wordt bijgewerkt.

Het is vaak handig je programma al bij het ontwerpen in compartimenten te verdelen die redelijk los van elkaar staan (niet te veel onderlinge informatie-uitwisseling hebben) en dus ook los te testen zijn. Zulke compartimenten heten modules. Als je modules gebruikt, specificeer ze dan, inclusief de informatie-uitwisseling, in het ontwerp. Dit eerste project is beperkt van omvang en het gebruik van modules is niet persee noodzakelijk. Pas ze alleen toe als dit in jouw aanpak voordelen heeft.

Het automatisch genereren van diagrammen uit rudimentaire stukken broncode zoals class declarations en function definitions is te prefereren boven het met behulp van een grafisch een tool samenstellen van prachtige diagrammen die vervolgens bij de eerste codewijziging verouderd zijn en niet, of ten koste

van veel uren, worden bijgewerkt.

Planning – Een initiele, onderweg bij te stellen beschrijving van volgorde en doorlooptijd en benodigde mens-uren voor de activiteiten die onderdeel zijn van het project. Als je modulair (met modules) werkt, laat deze dan terugkeren in je planning (en in wat grotere “real life” projecten ook in de taakverdeling).

Het projectplan, inclusief de planning, staat onder versiebeheer.

Maak een begin met de projectevaluatie – Hierin komen meta-functionele overwegingen aan de orde. De projectevalutatie is een groeidocument, met andere woorden, hij wordt gaandeweg het project verder uitgewerkt. Vanwege het informele karakter van dit document hoeft hierop geen versiebeheer te worden toegepast.

Schrijf de broncode – Doe dit in kleine stappen en zorg dat je steeds iets hebt dat in ieder geval de goedkeuring van de Python interpreter kan wegdragen, al doet het nog niks.

Werk niet van begin naar eind maar van belangrijk naar onbelangrijk en houd de planning in de gaten om te zorgen dat je niet te veel in de details verdwaalt (MoSCoW gecombineerd met timeboxing). De SCRUM werkwijze wordt niet volledig gevolgd, houd je opdrachtgever (docent) echter wel in de lus betreffende wijzigingen en voortgang.

Valideer de broncode – Voer de tests uit zoals beschreven in de testspecificaties en verwerk de resultaten in een puntsgewijs, kwantitatief georienteerd testrapport.

Voltooi de projectevaluatie – Besteed hierin aandacht aan de meta-functionele kanten van het project zoals eerder beschreven. Geef ook een korte samenvatting van de bergen, dalen en hopelijk leerervaringen die je persoonlijk bent tegengekomen tijdens dit project.

Middelen

- Laptop, adaptor, muis, internet-verbinding.
- Pen en papier voor een snelle schets.
- Python 3.9 of hoger met de volgende libraries en hun dependencies.
  - SimPyLC
  - SciKitLearn
  - Tensorflow/Keras
- Tekstverwerkings-programma zoals MSWord of LibreOffice Writer.
- Programma om eenvoudige plaatjes te tekenen ter illustratie van je werk zoals MSPaint of Pinta.
- Programma-editor, bij voorkeur Code OSS, eventueel Notepad++, bij voorkeur NIET PyCharm, Visual Studio, Eclipse of een andere IDE (Interactive Development Environment).

Producten

- Projectplan
- Broncode
- Testrapport
- Projectevaluatie
- Overige documenten die je zelf als relevant beschouwt

Toetsing

Beoordeling vind plaats aan de hand van:

- Uitwerking van opdrachten die onderdeel zijn van de lessen.
- Uitwerking van de project-opdracht, inclusief inhoud van het projectplan.
- Verloop van de eindassesment, waarin de beschreven producten aan de orde komen, met nadruk op broncode en, als goede tweede, het ontwerpgedeelte van je project-plan.

Zorg dat je op de assessment goed “in je code zit”. Het is niet erg als je je hier en daar hebt laten inspireren door werk van anderen, echter dien je je code wel regel voor regel te kunnen uitleggen. Een argument als “stond ergens op Internet, geen idee waarom het werkt” is niet valide, het gaat immers om begrip.

Daarentegen is het niet persee onoverkomelijk als iets nog niet helemaal goed werkt. Ook hier is begrip weer het belangrijkste.

Heldere broncode, met doeltreffend gekozen structuur en benamingen, is van belang voor de beoordeling. Je wordt immers opgeleid tot ontwikkelaar en het valt te verwachten dat anderen verder zullen werken aan code die jij hebt geschreven.

--//--