WoR-Robots Algoritme

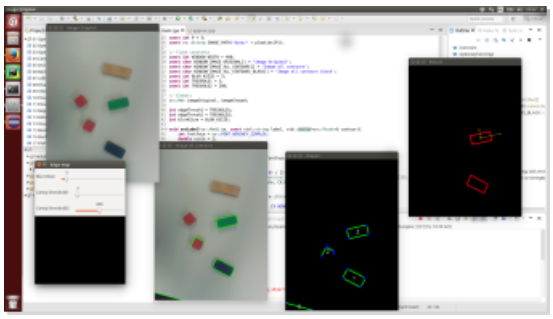
Joost Kraaijeveld, Jorg Visch en Chris van Uffelen

Wednesday 7th September, 2022

Een autonome robot moet zijn wereld kunnen zien en interpreteren. In deze opdracht wordt de basis gelegd voor deze vereisten.

1 Het programma

Schrijf een programma dat de locatie en positie van een willekeurig opgegeven gekleurde "Post-it" met een bepaalde vorm bepaalt in een chaotische omgeving op basis van camerabeelden met behulp van automatische beeldherkenning. Als de opgegeven Post-it niet aanwezig is moet het programma dit correct detecteren en melden. Uiteraard moet het programma de gebruiker ondersteunen bij eventuele foute invoer en dus heldere feedback geven bij een onjuiste invoer.



1.1 Vereisten

Het programma dient zowel interactief als in batch-vorm te kunnen werken.

In de interactive modus wordt de specificatie met behulp van het toetsenbord ingevoerd. Het programma blijft net zo lang actief totdat "exit" als specificatie wordt ingevoerd. Een specificatie is van de vorm "[vorm][whitespace][kleur]".

Hierbij is [vorm] één van “cirkel”,”halve cirkel”,”vierkant”,”rechthoek” of ”driehoek”. De volgende kleuren moet worden herkend: “groen”, “roze”, “geel” en “oranje” uit het Neon Post-it kleurenassortiment[1]. Van alle kleuren moeten tenminste de volgende vormen in meervoud aan wezig zijn bij het assessment: “halve circel” en “driehoek”.

In de batch-vorm wordt de specificatie uit een bestand gelezen dat als argument aan het programma wordt meegegeven. In dit bestand staat telkens op één regel één specificatie. Op iedere regel kan commentaar staan. Tekst na “#” tot aan het einde van de regel is commentaar en moet verder genegeerd worden.

In de interactieve modus moet het programma de contouren van het gevonden object markeren en de x– en y–coordinaten van het middelpunt van het object aangeven *op de afbeelding*. De oppervlakte van het object en de tijd die nodig was om het object te herkennen in klok-tikken moeten op een zelf te bepalen manier zichtbaar zijn in het programma. Als het object niet gevonden wordt moet dit eveneens op een zelf te bepalen manier zichtbaar worden, waarbij in ieder geval het aantal klok-tikken dat voor het niet–herkennen nodig was zichtbaar is.

In de batch-modus moeten de bovenstaande gegevens minus het markeren van het object (wat wel visueel zichtbaar moet zijn) naar STDOUT geprint worden.

De applicatie moet werkend gedemonstreerd worden tijdens een assessment. Het is tijdens het assessment toegestaan om de applicatie te calibreren op vorm, kleur en afstand. Het programma mag tijdens het assessment geen overbodige output (debug, status etc.) tonen.

Tijdens het assessment zulllen vragen gesteld worden over de gebruikte algoritmes. Hierbij is speciale aandacht voor de berekenbaarheid en complexiteit van de algoritmes [2]. Ook moet je de werking van het programma kunnen uitleggen waarbij de tijdsconcepten zoals in verwoord in de slides van de tweede les (zie onderwijsonline.han.nl) gebruikt moeten worden.

1.2 Compilatie en run–time vereisten

De primaire eis is uiteraard dat het programma correct werkt. De code moet minimaal op het niveau van de basis–semesters in termen van ontwerp en realisatie zijn uitgewerkt. De code moet in een Git-repository bijgehouden en ingeleverd worden. Maar aanvullend daarop nog wat andere criteria.

De volgende beordelingcriteria zijn z.g. “knock-out”-criteria. Als je niet aan deze onderstaande criteria voldoet dan is het cijfer sowieso onvoldoende:

[1] Als je niet de beschikking hebt over Post-it’s dan kan je zelf met behulp van de kleurstiften van je kleine broertje of zusje zelf verschillende vormen van verschillende kleuren maken. Of op een andere manier. Maakt niets uit. Zolang er maar de gewenste *hoeveelheid* vormen en kleuren aanwezig zijn...

[2] Zie voor meer informatie ook Wikipedia: berekenbaarheid en complexiteit

- Het programma moet als een standaard C++–programma in Eclipse compileren op een "stock" Linux Debian Testing (Bullseye) of Ubuntu 20.04 LTS (Focal Fossa) distributie met tenminste GCC 9.3.
- Als je er voor kiest om een ander platform dan Eclipse te gebruiken om te compileren, e.g. configure, cmake of andere vorm van compilatie–script, dan moet je bron–code een duidelijke compilatie–instructie of script bevatten, in een README–bestand of anderszins gedocumenteerd. Denk eraan dat ik dom, oud en opvallend vaak chagrijnig ben. Ik ben weinig tot niet geneigd na te denken over, of me bezig te houden met, het debuggen van andermans compilatie–instructies. . . .
- Alle volgende opmerkingen gaan over de zelfgeschreven code (niet over externe bibliotheek-code):
  - Het programma moet met "–Wall –Wextra –Wconversion" zonder errors en zonder warnings compileren met GCC.
  - Bij het compileren moet *alle* output van de compiler te zien zijn.
  - Mocht het zo zijn dat bepaalde warnings niet te voorkomen zijn dan moet in een begeleidende tekst duidelijk omschreven worden over welke warnings het gaat en waarom deze niet te voorkomen zijn. Ook moet aangegeven worden waarom deze warnings onder deze omstandigheden niet bezwaarlijk zijn.
  - Als het programma wordt onderworpen aan de statische codecheckers "Codan" (Zit in Eclipse 2020-12 (4.18.0)) of "CPPCheck" (2.3 of hoger[3]) dan mogen er geen problemen (warnings of errors) gedetecteerd worden op het hoogste waarschuwingsniveau van de tool [4].
- Ook het runnen van het programma met Valgrind met alle warning– en error–checking–mogelijkheden ingeschakeld moet zonder problemen gedaan kunnen worden.
- De headers moeten voorzien van relevant Doxygen–commentaar.
- De repo mag geen overbodige bestanden bevatten. Overbodige bestanden zijn bestanden die door tools gegenereerd worden (o.a. maar niet beperkt tot, object-bestanden en executables, of bestanden die typisch lokaal zijn, o.a. maar niet beperkt tot, Eclipse-project bestanden of CMake-artefacten).

2 Inleveren

Het inleveren gaat als volgt. Allereerts: je mailt een link naar jouw repository, samen met een zip met daarin je repository, naar de docenten. En op

[3] Let op: dit is NIET de standaard versie in Ubuntu 20.04!

[4] Beide codecheckers kennen methoden om warnings en errors uit te zetten voor specifieke delen van de code. Hierbij moet je dan wel aangeven waarom de code wel correct is en waarom je de warning of error hebt "uitgezet" voor dat stuk code

de tweede plaats: de Git-repository met de code van het programma moet uiterlijk aan het einde van week 4 op vrijdag 16:00 van de course op ISAS ge-upload worden. Tot slot: het programma moet werkend gedemonstreerd worden aan de docenten (assessment!).