

Peer review Kinematica

Student: Lars van Duijnhoven

Studentnummer: 2103948

Docent: Joost Kraaijeveld

Datum: 21-11-2024

Tijd: 9:15 tot 11:15

Course: World of Robots – World

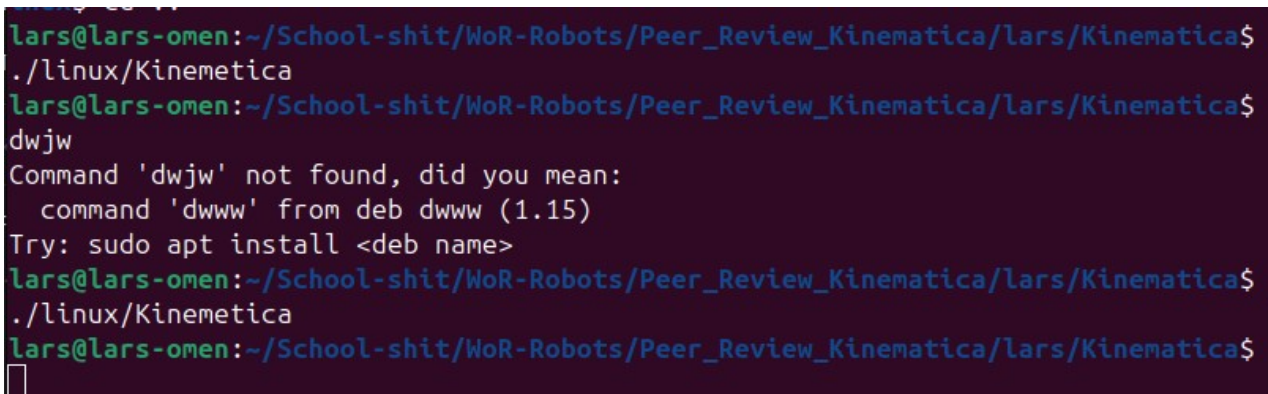
Ik heb de inlevering gereviewed van Tan Hoang.

Waarneming:

Functionaliteit en Requirements

Ik zie nergens een lijst staan van requirements, er is ook niet eens een readme toegevoegd, dus daar kan ik niks uit halen. Voor de volgende keer zou een readme wel erg fijn zijn zodat ik weet hoe ik kan compilen...

Kwaliteit en onderhoudbaarheid

A terminal window with a dark purple background. The prompt is 'lars@lars-omen:~/School-shit/WoR-Robots/Peer_Review_Kinematica/lars/Kinematica\$'. The user enters './linux/Kinemetica', followed by 'dwjw'. The terminal shows an error: 'Command 'dwjw' not found, did you mean: command 'dwww' from deb dwww (1.15)'. It then suggests 'Try: sudo apt install <deb name>'. The user enters './linux/Kinemetica' again, and the prompt returns.

```
lars@lars-omen:~/School-shit/WoR-Robots/Peer_Review_Kinematica/lars/Kinematica$
./linux/Kinemetica
lars@lars-omen:~/School-shit/WoR-Robots/Peer_Review_Kinematica/lars/Kinematica$
dwjw
Command 'dwjw' not found, did you mean:
  command 'dwww' from deb dwww (1.15)
Try: sudo apt install <deb name>
lars@lars-omen:~/School-shit/WoR-Robots/Peer_Review_Kinematica/lars/Kinematica$
./linux/Kinemetica
lars@lars-omen:~/School-shit/WoR-Robots/Peer_Review_Kinematica/lars/Kinematica$
```

Als je niks meegeeft met het programma gebeurt er ook niks? Het is nu heel onduidelijk wat het programma doet voor de gebruiker.

Main.cpp:

```

src > G Main.cpp > ...
1  #include "RobotArm.h"
2
3  int main(int argc, char** argv)
4  {
5
6      RobotArm robotArm(15.0,19.0,6.0, 0.0, 90.0, 0.0);
7      if(argc == 5)
8      {
9          robotArm.kinematica(std::stof(argv[1]), std::stof(argv[2]), std::stof(argv[3]), std::stof(argv[4]));
10     }
11     return 0;
12 }
13

```

Dit (hierboven) kan beter door het creeëren van de constructor binnen de if-statement te hebben. Zo heb je al minder onnodig geheugengebruik. Daarnaast kan je ook nog een else toevoegen met een errorhandling als er geen argumenten meegegeven worden, omdat het nu erg onduidelijk is wat er gebeurt en waarom hij niks doet :/.

Verder is de onderhoudbaarheid best wel matig, omdat er ineens een aparte folder in de src folder zit die naar mijn mening niks te maken heeft met de hele opdracht?

Meer over de onderhoudbaarheid in de conclusie, omdat ik het hier op kan sommen.

Applicatie

```

1  cmake_minimum_required(VERSION 3.5)
2
3  project( Kinemetica )
4  add_executable( Kinemetica src/Main.cpp src/Matrix.inc src/RobotArm.cpp src/RobotController/serialHandler.cpp src/RobotController/RobotLL.cpp src/RobotController/Robot
5
6
7
8

```

CmakeLists.txt: Er is hier geen gebruik gemaakt van alle warnings met Wall, als je dit wel doet krijg je ineens een waslijst met warnings. Zie het bestand make_output voor de vergelijking. Voeg dus aan de Cmake ergens dit toe: add_compile_options(-Wall -Wextra -Wconversion). Zie internet voor voorbeelden, er hoort nog iets bij namelijk.

Cppcheck commando: cppcheck --enable=all --inconclusive --force --inline-suppr --std=c++17 --suppress=missingIncludeSystem ./src/ 2> cppcheck_output.txt

Dit txt bestandje heb ik daarna verplaatst naar de correcte tooling output folder.

```

^
src/Matrix.hpp:74:0: style: The function 'getColumns' is never used. [unusedFunction]
.....
inline static std::size_t getColumns()
.....
^
src/Matrix.inc:363:0: style: The function 'solve' is never used. [unusedFunction]
.....
Matrix< T, M, 1 > Matrix< T, M, N >::solve() const
.....
^
src/RobotArm.cpp:103:0: style: The function 'getJointAngle' is never used. [unusedFunction]
.....
double RobotArm::getJointAngle(unsigned short joint) const
.....
^
src/RobotArm.cpp:108:0: style: The function 'setJointAngle' is never used. [unusedFunction]
.....
double RobotArm::setJointAngle(unsigned short joint, double angle)
.....
^
src/RobotArm.cpp:114:0: style: The function 'getXPosition' is never used. [unusedFunction]
.....
double RobotArm::getXPosition() const
.....
^
src/RobotArm.cpp:119:0: style: The function 'getYPosition' is never used. [unusedFunction]
.....
double RobotArm::getYPosition() const
.....
^
src/RobotArm.cpp:125:0: style: The function 'getCurrentPosition' is never used. [unusedFunction]
.....
std::pair<double, double> RobotArm::getCurrentPosition()
.....
^
src/RobotController/RobotHL.cpp:46:0: style: The function 'waitUntilDone' is never used. [unusedFunction]
.....
bool RobotArm_HighLevel::waitUntilDone()
.....
^
src/RobotController/RobotHL.cpp:119:0: style: The function 'moveToPark' is never used. [unusedFunction]
.....
void RobotArm_HighLevel::moveToPark()
.....
^
src/RobotController/RobotHL.cpp:138:0: style: The function 'moveToStraightUp' is never used. [unusedFunction]
.....
void RobotArm_HighLevel::moveToStraightUp()
.....
^
src/RobotController/serialHandler.cpp:39:0: style: The function 'readFromSerialUntil' is never used. [unusedFunction]
.....
std::string serialHandler::readFromSerialUntil(char delimiter)
.....

```

Dit is een screenshot uit het commando/ txt bestand van het commando dat ik hierboven uitgelegd heb. Zoals je ziet zijn er erg veel unused functions. Als deze weggehaald waren dan was de code compacter, makkelijker te lezen en sneller te compilen. Het is nu ook niet alsof ze echt nut hebben, want bijvoorbeeld de readFromSerialUntil is een readSerial functie. Waarom zouden we nu van Serial willen lezen? We sturen alleen commando's van hoe de servo's moeten staan.

Verder is er helemaal geen

Algemene code

```
calculateInverseKinematics(std::make_pair(length1+ afwijking, 2), 0.5);

std::cout << "hoek 1 : " << joints[0].currentAngle << std::endl;
std::cout << "hoek 2 : " << joints[1].currentAngle << std::endl;
std::cout << "hoek 3 : " << joints[2].currentAngle << std::endl;

robotController.singleServoCommandDegrees(5000, 1 , joints[0].currentAngle);
robotController.singleServoCommandDegrees(5000, 2 , joints[1].currentAngle);
robotController.singleServoCommandDegrees(2000, 3 , joints[2].currentAngle);

std::this_thread::sleep_for (std::chrono::seconds(4));

robotController.singleServoCommandDegrees(1500, 5, 70);

std::this_thread::sleep_for (std::chrono::seconds(3));

robotController.moveToReady();
robotController.singleServoCommandDegrees(1500, 5, 70);
```

Hier worden altijd couts geprint, wat opzich wel kan, maar ik had dit liever in een aparte functie gehad, omdat dit vaker herhaald wordt. Denk aan een functie “printCurrentAngles”, dan is deze Kinematica functie ook wat kleiner.

Verder zijn de sleeps hier hardcoded? Dit vind ik best wel apart en had eenvoudig aangepast kunnen worden. De singleServoCommandDegrees had ook vervangen kunnen worden door een multipleServoDegrees, waarna er altijd de meegegeven tijd gewacht werd, zodat je in ieder geval zeker weet dat het daadwerkelijk afgelopen is, ipv hopen dat dit lang genoeg is. Verder is sleep_for() heel erg blocking en zorgt het ervoor dat het gehele programma 4 seconde niks meer doet, wat in veel gevallen niet optimaal of zelfs gevaarlijk is.

```
35
36 void RobotArm_LowLevel::singleServoCommand(uint8_t pin, uint16_t pulseWidth, uint16_t speed, uint16_t time)
37 {
38     std::string commandString = "";
39     // std::cout << "hello" << std::endl;
40     commandString += generateCommandString(pin, pulseWidth, speed, time, true);
41     sendCommand(commandString);
42 }
43
```

RobotLL.cpp

Hier staat een cout nog uitgecomment, dit is niet netjes en mag eigenlijk niet in de code blijven staan. Haal dit weg voor extra leesbaarheid.

```

uint16_t RobotArm_HighLevel::convertDegreesToPulseWidth(int16_t degrees, int8_t channel) {
    int16_t inputMin = -90;
    int16_t inputMax = 90;
    int16_t offset = 0;
    bool isInverted = false;
    for (size_t i = 0; i < robotJoints.size(); i++) {
        if (robotJoints[i].pinNumber == channel) {
            if (robotJoints[i].maxPosition > 90) {
                inputMin = 0;
                inputMax = 180;
            }
            isInverted = robotJoints[i].inverted;
            offset = robotJoints[i].offset;
            break;
        }
    }

    if (isInverted) {
        degrees = -degrees;
    }

    const uint16_t outputMin = 500;
    const uint16_t outputMax = 2500;
    uint16_t pulseWidth = outputMin
        + ((degrees - inputMin) * (outputMax - outputMin))
        / (inputMax - inputMin);
    // std::cout << "pulsewidth " << pulseWidth << std::endl;
    return pulseWidth + offset;
}

```

RobotHL.cpp

Hetzelfde als hierboven, ook weer een debug statement die uitgecomment is. Deze moet ook weg.

Verder zijn er veel overbodige includes, zie ze hieronder:

RobotArm.cpp: <thread> en <chrono>

RobotArm.h: <iostream>, <vector> en <cmath>

Matrix.hpp: <array>, <cstdint> en <initializer_list>

De andere bestanden heb ik niet meer gecontroleerd, maar ik vermoed dat ik daar ook overbodige includes zou vinden. De includes hierboven betekent trouwens niet dat ze uit dat specifieke bestand moeten, maar meer dat er dus op 2 plekken iets wordt geïnclude terwijl dat niet nodig is. Hier moet dus naar gekeken worden om de optimale locatie van de include te vinden en niet om op random plekken de includes te verwijderen. Als deze includes weg zijn, gebruikt het programma minder geheugen, omdat er dan ook minder code is. Dit zorgt ook weer voor een snellere compiletijd


```

/home/lars/School-shit/WoR-Robots/Peer_Review_Kinematica/lars/Kinematica/src/RobotArm.cpp:142:60: warning: conversion from 'double' to 'int16_t' (aka 'short int') may change value [-Wfloat-conversion]
142 |         robotController.singleServoCommandDegrees(3500, 0, angle1);
    |         ^~~~~~
/home/lars/School-shit/WoR-Robots/Peer_Review_Kinematica/lars/Kinematica/src/RobotArm.cpp:150:71: warning: conversion from 'double' to 'int16_t' (aka 'short int') may change value [-Wfloat-conversion]
150 |     botController.singleServoCommandDegrees(5000, 1, joints[0].currentAngle);
    |     ^~~~~~
/home/lars/School-shit/WoR-Robots/Peer_Review_Kinematica/lars/Kinematica/src/RobotArm.cpp:151:71: warning: conversion from 'double' to 'int16_t' (aka 'short int') may change value [-Wfloat-conversion]
151 |     botController.singleServoCommandDegrees(5000, 2, joints[1].currentAngle);
    |     ^~~~~~
/home/lars/School-shit/WoR-Robots/Peer_Review_Kinematica/lars/Kinematica/src/RobotArm.cpp:152:71: warning: conversion from 'double' to 'int16_t' (aka 'short int') may change value [-Wfloat-conversion]
152 |     botController.singleServoCommandDegrees(2000, 3, joints[2].currentAngle);
    |     ^~~~~~
/home/lars/School-shit/WoR-Robots/Peer_Review_Kinematica/lars/Kinematica/src/RobotArm.cpp:169:60: warning: conversion from 'double' to 'int16_t' (aka 'short int') may change value [-Wfloat-conversion]
169 |         robotController.singleServoCommandDegrees(3500, 0, angle2);
    |         ^~~~~~
/home/lars/School-shit/WoR-Robots/Peer_Review_Kinematica/lars/Kinematica/src/RobotArm.cpp:178:71: warning: conversion from 'double' to 'int16_t' (aka 'short int') may change value [-Wfloat-conversion]
178 |     botController.singleServoCommandDegrees(5000, 1, joints[0].currentAngle);
    |     ^~~~~~
/home/lars/School-shit/WoR-Robots/Peer_Review_Kinematica/lars/Kinematica/src/RobotArm.cpp:179:71: warning: conversion from 'double' to 'int16_t' (aka 'short int') may change value [-Wfloat-conversion]
179 |     botController.singleServoCommandDegrees(5000, 2, joints[1].currentAngle);
    |     ^~~~~~
/home/lars/School-shit/WoR-Robots/Peer_Review_Kinematica/lars/Kinematica/src/RobotArm.cpp:180:71: warning: conversion from 'double' to 'int16_t' (aka 'short int') may change value [-Wfloat-conversion]
180 |     botController.singleServoCommandDegrees(1500, 3, joints[2].currentAngle);
    |     ^~~~~~

```

Hierboven is een stukje uit make_output.txt gepakt, omdat hier hele gevaarlijke warnings in zitten. Hieronder zijn wat regels waar het fout gaat.

```

178 |     robotController.singleServoCommandDegrees(5000, 1, joints[0].currentAngle);
179 |     robotController.singleServoCommandDegrees(5000, 2, joints[1].currentAngle);
180 |     robotController.singleServoCommandDegrees(1500, 3, joints[2].currentAngle);
181 |

```

Het lijkt op zich niet zo heel erg, maar singleServoCommandDegrees verwacht een uint16_t, terwijl currentAngle, zoals gedefinieerd is in de struct, een double is. Een double naar long conversion is al erg en kan al snel voor problemen zorgen. Een double naar unsigned long conversion kan al helemaal voor problemen zorgen, omdat je alle originele negatieve getallen juist extreem verhoogt, omdat het lager is dan de minimum en dan begint hij weer van boven. Wat hier meermaals gedaan wordt is nog een gradatie erger, namelijk een double omzetten naar een **unsigned 16 bit** getal. Dus kommagetallen haal je dan al weg, alle negatieve getallen haal je dan weg én alle getallen boven het 16 bit limiet haal je weg. Dit zorgt ervoor dat de kans op onverwachte resultaten heel erg groot wordt, waardoor dit een behoorlijk gevaarlijke fout kan zijn. Dit kan verbeterd worden door de desbetreffende uint16_t's naar floats te veranderen of juist andersom.

Namespaces (en een klein beetje naamgeving)

Ik heb in RobotArm.h geen Doxygen gezien, maar in andere header files wel. Dit is al een beetje inconsistent en ik had liever ook hier Doxygen gezien. De naamgeving is overigens ook onlogisch. Als voorbeeld in RobotArm.cpp wordt regelmatig

Nederlands en Engels door elkaar gehaald. Hier moet echt een keuze in gemaakt worden, want anders weet je over 1 week zelf niet meer waarom “length” niet werkt. Zie hieronder het desbetreffende voorbeeld van RobotArm.cpp:

Hier wordt in de meegegeven argumenten heel raar gebruik gemaakt van Nederlands en Engels. Het eerste argument is namelijk een combinatie van “goal”, “coordinaten” en “pair”, dus 2 engelse woorden en 1 nederlands woord? Hierna een volledig

```
void RobotArm::calculateInverseKinematics(std::pair<double, double> goalCoordinatenPair, double stapGrootteFactor, double precision)
{
    Matrix<double, 2, 1> currentCoordinaten;
    std::pair<double, double> currentCoordinatenPair = calculateForwardKinematics();
    currentCoordinaten.at(0, 0) = currentCoordinatenPair.first;
    currentCoordinaten.at(1, 0) = currentCoordinatenPair.second;

    Matrix<double, 2, 1> goalCoordinaten;
    goalCoordinaten.at(0, 0) = goalCoordinatenPair.first;
    goalCoordinaten.at(1, 0) = goalCoordinatenPair.second;

    Matrix<double, 2, 3> jacobimatrix;
    Matrix<double, 3, 2> inverseJacobimatrix;
    Matrix<double, 2, 1> deltaPosition;
    Matrix<double, 2, 1> stapGrootte;
    Matrix<double, 3, 1> deltaHoeken;

    unsigned long loopCount = 0;
    unsigned long maxIterations = 1000000;
```

nederlands woord en daarna weer een volledig engels woord. Dit geeft gewoon geen structuur en is heel onduidelijk. Een paar regels naar onder gebeurd het weer, namelijk met “deltaPosition”, “stapGrootte” en “loopCount”.

Er worden verder alleen namespaces gebruikt voor klassen, wat naar mijn mening is zoals het hoort. Dit vind ik dus goed, omdat je dan altijd kan herleiden waar iets vandaan kwam ipv een standaard namespace gebruiken.

Ook is er geen UML-diagram toegevoegd, dus moest ik naar de code kijken om een oordeel te doen over de coupling en cohesion. Vanwege een gebrek aan tijd heb ik besloten om hier niet diep op in te gaan. Van wat ik snel opgepikt hebt kon ik niks zorgwekkends vinden. Hieronder is nog wel een klein stukje geschreven over de verantwoordelijkheid van de klassen, maar meer dan dat is er niet.

Klassen

Ik vind dat er een vaag gebruik is van klassen. Je hebt namelijk de RobotHL, RobotLL en RobotArm klassen, maar ze hebben alle 3 een andere functionaliteit. De naamgeving had hier sowieso al beter gekund, maar er is ook voor RobotArm geen Doxygen geschreven. Mijn vraag is, waarom is RobotArm een Higher High Level driver van de robotarm? Ik begrijp het verschil tussen deze 2 niet en RobotArm had eigenlijk in de High Level driver moeten zitten, omdat dat is waar een High level driver voor is.

Verder bevatten alle klassen wel een header en source, wat het overzichtelijker maakt, dus dit is goed gedaan.

De responsibility is naar mijn mening ook niet helemaal goed afgehandeld, omdat de RobotArm klasse nu verantwoordelijk is voor het bijhouden van joints (terwijl RobotHL dit ook heeft??), het aansturen van de RobotHL klasse én de timing regelt d.m.v. sleeps.

Functies

Hier valt nog behoorlijk aan te werken, zo zijn in RobotArm.cpp er functies van maar 1 regel, zoals “getCurrentPosition”, maar ook van 60 regels, zoals “kinematica”. 60 regels vind ik te veel voor een functie en dit had makkelijk verkort kunnen worden door wat prints te verwijderen. Verder had deze functie ook in verschillende gedeeltes/ functies opgedeeld kunnen worden om zo de verantwoordelijkheden te verspreiden. Nu past deze functie namelijk ineens direct de currentAngle van de joints aan? Zie foto hieronder, dit had zeker wel in een aparte functie gekund.

```
std::this_thread::sleep_for (std::chrono::seconds(4));

robotController.singleServoCommandDegrees(1500, 5, 70);

std::this_thread::sleep_for (std::chrono::seconds(3));

robotController.moveToReady();
robotController.singleServoCommandDegrees(1500, 5, 70);

joints[0].currentAngle = 0.0;
joints[1].currentAngle = 90.0;
joints[2].currentAngle = 0.0;

std::this_thread::sleep_for (std::chrono::seconds(3));

robotController.singleServoCommandDegrees(3500, 0, angle2);
/
std::this_thread::sleep_for (std::chrono::seconds(3));
calculateInverseKinematics(std::make_pair(length2 + afwijking,2 ), 0.5);
```

Daarnaast zijn er, zoals bij het kopje “Applicatie” ook besproken is, meerdere functies die wel public zijn maar niet gebruikt worden. Dit kan beter door ze private

te maken of zelfs helemaal te verwijderen. Verwijderen zorgt voor overzichtelijkere code, minder gebruik van geheug en een snellere make tijd.

Variabelen

De variabelen hebben, zoals in het kopje namespaces al genoemd is, vaak hele aparte namen. Wel is er een goede poging gedaan om RAII toe te passen, omdat de variabelen grotendeels verwerkt zijn in de objecten of functies.

Conclusie:

De code bevat een aantal goede dingen, zoals het gebruik van duidelijke header- en cpp bestanden voor de verschillende klassen, wat zorgt voor een overzichtelijke structuur (alleen begrijp ik de RobotController map niet helemaal).

Verder is de main.cpp erg leeg, wat netjes is en laat zien dat hier overna is gedacht. Als laatste zijn de namespaces (naar mijn mening) goed gebruikt en is er geen standaard namespace ingesteld voor een pagina. Dit zorgt dat er altijd te herleiden is waar een functie of variabele vandaan gekomen is, wat voor extra duidelijkheid zorgt.

Inconsistenties in codestijl: De afwisseling tussen Nederlandse en Engelse termen, zoals in de variabelennamen (goalCoordinatenPair, deltaPosition, stapGrootte), maakt de code moeilijk te begrijpen en kan voor verwarring zorgen tijdens het lezen ervan.

Gebrek aan documentatie: Het gebrek aan een readme en doxygen in bepaalde delen van de code, zoals de RobotArm-klasse, zorgt voor onduidelijkheid. Ik had mijn eigen readme erbij gepakt voor deze opdracht, omdat die meer duidelijkheid gaf over de code dan alles wat hier te vinden was. En die is geschreven voor een andere oplevering!!

Omzettingsgevaar: Er zijn gevaarlijke conversies in de code, zoals het omzetten van double naar uint16_t, wat kan leiden tot hele aparte errors, veel hoofdpijn en veel extra tijd.

Onnodige en overbodige code: Er zijn functies die niet worden gebruikt, debugstatements die zijn uitgecomment en overbodige includes. Dit verhoogt het geheugen en de compilatietijd zonder iets daadwerkelijk toe te voegen.

Te grote functies: Functies zoals kinematica bevatten te veel regels (60+), wat zorgt voor lastig te lezen code. Je hebt gewoon meer tijd nodig om te ontcijferen wat er nou eigenlijk staat.

Hardcoded sleeps en blocking calls: Het gebruik van sleep_for-functies zou ik niet aanraden in de meeste gevallen, omdat dit blocking is. Verder zijn de tijden ook nog eens hardcoded, waardoor je niet eens weet of het de juiste tijd is, dit kan dynamischer.

Onjuiste verantwoordelijkheid van klassen: Er is onduidelijkheid over de verantwoordelijkheden van RobotArm, RobotHL, en RobotLL. Deze klassen overlappen in functionaliteit, zoals het bijhouden van joints door beide (RobotArm en RobotHL) een joint vector te hebben.

Kortom, de inconsistenties in stijl, ontbrekende documentatie, aparte codeconstructies en meer zorgt ervoor dat deze code alleen met veel hoofdpijn en herschrijven uit te breiden is. Niet optimaal.

Aanbeveling:

Documentatie:

Voeg een duidelijke README.md toe waarin staat hoe je kan compilen, runnen en dat soort dingen.

Gebruik **consistent** doxygen in alle headerbestanden.

Codestijl:

Kies voor één taal (Engels of Nederlands) en houdt deze overal in de code aan.

Typeconversies:

Convert alsjeblieft **nooit** meer een double naar een integer, al helemaal niet unsigned en al HELEMAAL geen kleine integers.

Verwijderen van overbodige code:

Haal ongebruikte functies weg of maak ze private als ze binnen de klasse worden gebruikt.

Controleer om de zoveel tijd of je includes wel echt nodig zijn.

Splitsen van grote functies:

Splits grote functies zoals kinematica op in kleinere functies met specifieke verantwoordelijkheden, zoals `adjustCurrentAngles` en `executeServoCommands`.

Verbetering van timing:

Gebruik software-timers of andere manieren om blocking timers te voorkomen, tenzij het een goed gedocumenteerde keuze is geweest.

Klassestructuur:

Denk de volgende keer misschien een beetje na over wat je precies wilt bereiken met elke klasse, om zo verantwoordelijkheden te splitsen.

Cmake warnings/ compiler warnings:

Pas de `CmakeLists.txt` aan zodat je **alle** compiler warnings te zien krijgt, anders geeft het je valse hoop.

Los alle waarschuwingen en foutmeldingen op die door `cppcheck` en de compiler worden gemeld.

-----Einde Peer review-----