

2020

Laborbuch

„HS-KL goes RoboCup 2021“

Team HS-KL, Leitung: Prof. Adrian Müller, FB I/MST

Inhalt

[Dokumenthistorie 3](#_Toc45894884)

[Team HS-KL 3](#_Toc45894885)

[Ansprechpartner 3](#_Toc45894886)

[Dokumentation 3](#_Toc45894887)

[Hinweise für Autoren, To-Do 4](#_Toc45894888)

[Unsere Strategie 4](#_Toc45894889)

[Hintergrundinformation 4](#_Toc45894890)

[Spielfeld, Testen in der Praxis 5](#_Toc45894891)

[Strategie: Optionen für das Vorgehen 5](#_Toc45894892)

[SimRobot (ein Simulator für den RoboCup) 6](#_Toc45894893)

[Voraussetzungen für das Arbeiten mit dem Simulator: PC mit Ubuntu 18.04 6](#_Toc45894894)

[Benötigte Abhängigkeiten auf Deinem PC installieren 6](#_Toc45894895)

[Installieren der IDE 7](#_Toc45894896)

[Klonen der Codebase 7](#_Toc45894897)

[Für unsere IDE *CodeLite* einen Workspace generieren 7](#_Toc45894898)

[Simulator: Neue Befehle *randomize* und *spread* für den gc 9](#_Toc45894899)

[Simulator: Logging von Daten für Benchmarks und Tests 12](#_Toc45894900)

[Pipeline: Gitlab-ci Config für Docker 13](#_Toc45894901)

[Erstellung einer Objekterkennung 14](#_Toc45894902)

[Nao: Flashen der Robocup Firmware 19](#_Toc45894903)

[Vorbereiten des Flashers unter Ubuntu 18.04 19](#_Toc45894904)

[USB Stick zum Flashen vorbereiten 19](#_Toc45894905)

[Flashen des USB-Sticks 20](#_Toc45894906)

[Flashen der Software auf dem Nao 20](#_Toc45894907)

[Nao: Deployen der Software auf dem Roboter 22](#_Toc45894908)

[Den Code zum Deployen bauen 22](#_Toc45894909)

[Den Nao mit dem PC verbinden 22](#_Toc45894910)

[Den Nao vorbereiten 24](#_Toc45894911)

[Das Deployment vorbereiten 26](#_Toc45894912)

[Den eigentlichen Code deployen 27](#_Toc45894913)

[Pipeline: Grundgerüst für Regressionstests 28](#_Toc45894914)

# Dokumenthistorie

15.6.2020 Erste Fassung Jonas

16.6.2020 Überarbeitung Adrian: Eingefügt Abschnitte Team, Historie, Ziele dieses Laborbuches; Abschnitte Titel erweitert, Seitennummern, Verweise

17.6.2020 Überarbeitung Jonas: Eingefügt Screenshots und Zeilenangaben

29.6.2020 Überarbeitung Felix: Einfügen von Verbesserungsvorschläge für das Laborbuch

04.07.2020 Autor: Salome Schlemer, Ergänzung des Kapitels „Erstellung einer Objekterkennung“ Ausformulierung der ersten Versuche

7.7.2020 Autor Adrian: Neues Kapitel Strategie und allgemeine Information

14.7.2020 Autor Jonas: Neues Kapitel zum Flashen des Nao

17.7.2020 Autor Jonas: Neues Kapitel zum Einspielen des Codes auf dem Nao, Erweiterung des Kapitels für die Vorbereitung.

# Team HS-KL

Prof. Adrian Müller

David Kostka

Felix Mayer

Jonas Lambing

Unter Mitwirkung von Salome Schlemer

# Ansprechpartner

Prof. Adrian Müller - adrian.mueller@hs-kl.de

# Dokumentation

**Ziele dieses Laborbuches**: Dieses Laborbuch dokumentiert die Arbeiten für die Teilnahme an der Qualifikation der deutschen RoboCup SPL 2021 der HS-KL für neue und bestehende Team Mitglieder. Es ergänzt das GIT Repository <https://zwoogle3.informatik.hs-kl.de/root/hskl-robocupgermanopen-2021.git> um folgende Punkte:

* Hintergrundinformation, Installationshinweise
* Wie modifiziere bzw. erweitere ich den Simulator *SimRobot*?
* Wie arbeite ich remote mit dem Projekt-Server zwoogle4.ds.fh-kl.de?
* Erläuterungen zur Architektur, zu Modifikationen am Robot Code
* Welche automatisierten Tests gibt’s es, Performance Benchmarking
* (später): Bilderkennung mit Keras und Tensorflow: Testdaten, Modelle, Evaluierung

Die Arbeiten basieren aktuell auf dem Code Drop <https://github.com/bhuman/BHumanCodeRelease>. Die Dokumentation des Teams B-Human zum Bauen und anderen Prozessen befindet sich im Root der Repository in *CodeRelease2019.pdf*.

Wir verwenden die gleiche IDE „CodeLite“ (s. Unsere IDE CodeLite Workspace generieren , Seite 4) Das erforderliche Systemimage für die echten Naos ist das nao-2.8.5.11\_ROBOCUP\_ONLY und steht unter ?????

## Hinweise für Autoren, To-Do

Alle Verbesserungsvorschläge beziehen sich auf das Laborbuch, es sei denn es wird explizit etwas Anderes genannt.

* Wenn Dateien modifiziert oder neu erstellt werden, sollte auf den zugehörigen Ordner verwiesen werden, bzw. der Path mit hinterlegt werden, um die Datei schneller zu finden.
* Angabe nicht über Zeile, denn nach Erweiterungen stimmt diese nicht mehr überein. Vorschlag: im Kopf der Datei eine Notiz wie: Änderung bei Datum\*Autor\*Funktion

# Unsere Strategie

Input: unsere Erkenntnisse aus 6/2020 mit dem B-Human Code Release 2019, deren Simulator SimRobot, und ein Telefonat mit Rico Tilgner, Teamleiter Leipzig, Diskussion mit Jonas

## Hintergrundinformation

Stand: 07/2020

* Der Code Release 2019 von B-Human ist historisch gewachsener Code. Die einzelnen Module (TeamStrategie, WalkingEngine etc. sind über Jahre hin optimiert worden
* Die Bewegungen des Naos im *RoboCup* sind 2-3x schneller als im nativen „Kind“ Modus. Kaum ein Team benutzt NaoQi, da der aktuelle Release den Durchgriff auf seine WalkingEngine verbietet.
* Zugang zum Nao erfolgt über *LoLa* (s. LoLa RoboCupper Official Documentation.pdf). Es gibt einen LoLa Connector im github der HTWK Leipzig (<https://github.com/NaoHTWK> , dort liegt auch anderes). Unklar, wie *LoLa* in *B-Human* integriert ist (-> Praktikum Jonas)
* Alle Teams testen vorwiegend in der Realität, erzeugen Log-Dateien, speichern alle 1-2sec full frame Bilder. Das Erkennen von Robotern, und die Freund-Feind Unterscheidung sind schwierig. Es werden inoffizielle Testspiele organisiert.
* Für die reguläre TensorFlow Laufzeit ist die CPU des Nao zu schwach -> Verwenden von TensorFlow Lite. Dessen Kompilation ist „eine bitch“ (Zitat Rico 8-), er sendet uns die Library zu.
* Es gibt eine On-Board GPU; deren Einsatz macht aber Probleme (Zitat Rico).

Dieses Jahr entfiel der RoboCup.

## Spielfeld, Testen in der Praxis

* Achtung: wechselnde Lichtverhältnisse. Auto-white balance erforderlich (Neon Licht)
* Unser Ansatz mit dem ¼ Feld könnte klappen, aber:
  + Mind. 0.5m Grün hinter den Außenlinien (oder an der Wand dahinter)
  + Der Anstoßkreis muss vollständig sichtbar sein
* Bezugsquellen
  + Fußball: s. Regelwerk
  + Tor: Plastikrohre (weiß) aus dem Baumarkt, Durchmesser 10cm
  + <https://messe-creation.de/kunstrasen-boston/>
* Wie starten? Rico sendet uns die Binaries der Weltmeister-Mannschaft (!), und ein Installation-Script
  + Darin ist u.a. die Spielfeldgröße wählbar (6x4, 6x9, andere)
* Ideen zur Testdaten Generierung
  + Ferngesteuerte Spiele
  + 2 gegen 3 Mannschaften (Code Leipzig, unser Code)
  + RC „plastic Naos“

## Strategie: Optionen für das Vorgehen

Zitate Rico

„Vorsicht vor dem B-Human Clone“ Effekt“

Wechsel des Frameworks tut weg

„Macht alles anders als B-Human“

*Das folgende ist eine Momentaufnahme, Stand 7.7.2020, und wird weiterentwickelt:*

Sinnvolle Strategien sind

a) Re-Engineering Framework B-Human oder rRUNSWIFt

b) Eigenes Framework + Module anderer Teams

Eigene Entwicklung bei

o TeamStrategie

o Pässe

o Dribbling, Gegner vermeiden

o Trade-Off Speed / Überhitzen Motoren / Stress auf Gelenke  Annecy

# SimRobot (ein Simulator für den RoboCup)

### Voraussetzungen für das Arbeiten mit dem Simulator: PC mit Ubuntu 18.04

Ein durschnittlicher PC genügt. Der Projekt Server zwoogle4.ds.fh-kl.de (im VPN der HS-KL zugreifbar) verfügt über ausreichend Performanz für flüssige Simulationen. Darauf sind folgende Pakete erforderlich

* Clang
* Make
* Qtbase5-dev
* Libqt5opengl5-dev
* Libqt5svg5-dev
* Libglew-dev
* Libasound-dev
* Ocl-icd-opencl-dev
* Net-tools
* Graphviz
* Xterm

### Benötigte Abhängigkeiten auf Deinem PC installieren

Den Befehl

*“sudo apt install clang make qtbase5-dev libqt5opengl5-dev libqt5svg5-devlibglew-dev libasound-dev ocl-icd-opencl-dev ne -tools graphviz xterm”*

im Terminal ausführen. Achtung: keine White-Spaces um die “-“ herum!

### Installieren der IDE

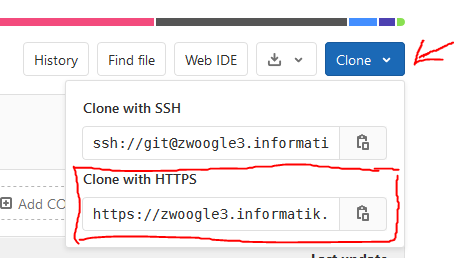
Die IDE kann mit dem Befehl *apt install CodeLite* über das Terminal installiert werden. Danach ist sie direkt startklar.

### Klonen der Codebase

Die Git-Repo für das Projekt ist unter <https://zwoogle3.informatik.hs-kl.de/root/hskl-robocupgermanopen-2021> zu finden.

Vor dem Klonen muss Git mit dem Befehl *apt install git* auf der Maschine installiert werden.

Danach kann man sich zur oben genannten Git-Repo begeben und mit einem Klick auf den blauen Button „Clone“ eine URL zum klonen kopieren.



Jetzt kann man einen beliebigen Ordner im Terminal öffnen (z.B. den home-Ordner) und die Codebase mit dem Befehl *git clone <URL>* klonen (Git erstellt automatisch einen Unterordner für die Repo).

### Für unsere IDE *CodeLite* einen Workspace generieren

Im Terminal nach *[Repo]/Make/LinuxCodeLite* navigieren („cd“) und lokal das Skript *generate* ausführen (./generate).

Danach kann der generierte Workspace mit *CodeLite* geöffnet und verändert werden. Starten von „codelite“, „Open Existing Workspace hskl-robocupgermanopen\_2021/Build/

Hinweis: alle Projekte sind unter „B-Human“ eingeordnet.

Starten des Simulators <project>/Build/Linux/SimRobot/Develop

Simulation auswählen: File:Open Comfig/Scenes “BH Testing.ros2”

Doppelclick auf <robo<cup -> Simulator öffnet

Doppelclick auf console: Console öffnet

Mehr zur Bedienung Simulator: s. Handbuch B-Human

# Simulator: Neue Befehle *randomize* und *spread* für den gc

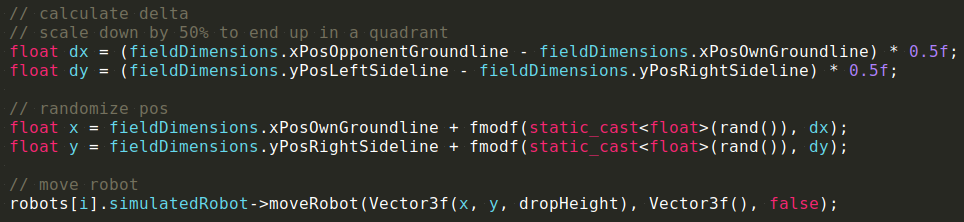
**15-06-2020, Hash e38e6e51, Author: Jonas Lambing**

**GIT Message**

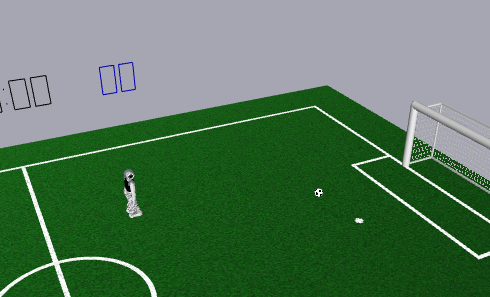
added basic benchmarking terminal output (first ballcontact, time until goal scored), added *gc randomize* + *gc spread* commands

**Inhalt**

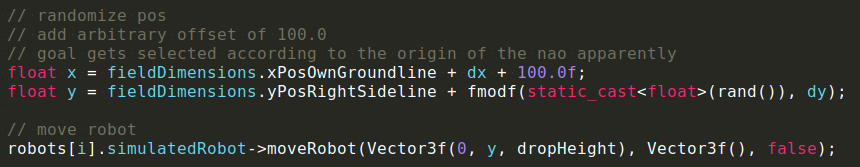
* Ziel: Positionierung erfolgt im ¼ Feld, Random-Positionierung eines Nao’s und eines Balls, Random-Positionierung eines Nao’s mit einem Winkel zwischen 0° und 45° zum Ball, das Prefix gc steht für *GameController* – also alles was mit der eigentlichen Spielsimulation zu tun hat.
* Eine Testszene mit einem Ball und einem simulierten Nao
  + *Config/Scenes/BH Testing.ros2*
  + *Config/Scenes/BH Testing.con*
* Ein Include, das von der Testszene benutzt wird
  + Config/Scenes/Includes/Single.rsi2
* Die Konsolenbefehle *gc randomize und gc spread* im Simulator
  + Der Befehl *gc randomize* platziert den simulierten Nao und den Ball in der Testszene jeweils an zufallsgenerierten Positionen im rechten, oberen Viertel des Spielfelds (GameController.cpp, Z. 328):



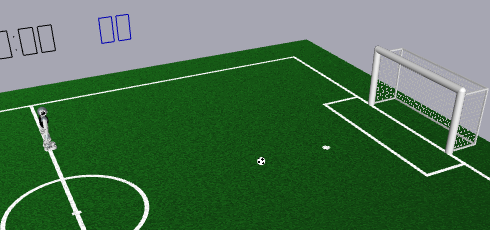
Resultat:



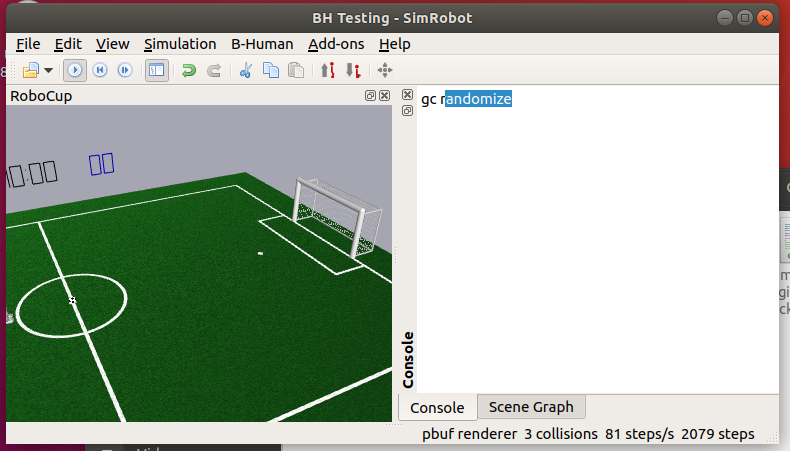
* + Der Befehl *gc spread* platziert den Nao an einem zufallsgenerierten Punkt entlang der Mittellinie am rechten, oberen Viertel. Der Ball wird statisch in der Mitte des unteren Rands des Viertels platziert (GameController.cpp, Z. 360):



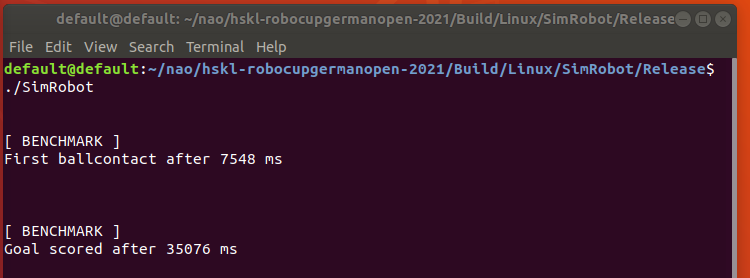
Resultat:



* + Die beiden Befehle wurden im Projekt *Controller* in der Datei *GameController.cpp* in der Funktion *GameController::handleGlobalCommand()* hinzugefügt.
  + Des Weiteren wurden die Befehle auch zur Autocompletion in *GameController::addCompletion()* in *GameController.cpp* (Z. 888) und *ConsoleRoboCupCtrl::createCompletion()* in *ConsoleRoboCupCtrl.cpp* (Z. 712) hinzugefügt:



* Terminal output für Benchmarking
  + Logging für geschossene Tore und vergangene Zeit seit Spielbeginn in *GameController::handleGoalCommand()* in *GameController.cpp (Z. 166)*
  + Logging für die Zeit vom Beginn des Spiels bis zum ersten Ballkontakt in *GameController::setLastBallContact()* in *GameController.cpp (Z. 813)*



# Simulator: Logging von Daten für Benchmarks und Tests

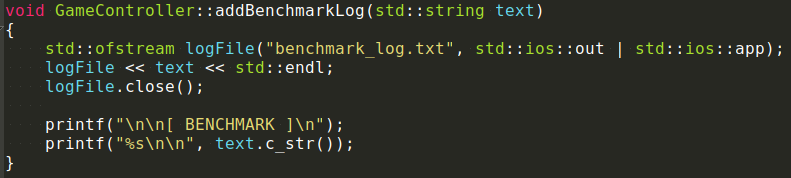
***18-06-2020, Hash 89153b78, Author: Jonas Lambing***

**GIT Message**

added benchmark logging to file

**Inhalt**

* Ziel: Zeitmessung bis zum ersten Ballkontakt und bis zum Tor nach Rundenanfang.
* Logging wird jetzt auch in einer Datei gespeichert, die in */Config/Scenes/benchmark\_log.txt* liegt.
* Zum Loggen habe ich den alten Loggingcode etwas verändert und das Ganze in die Methode GameController::addBenchmarkLog() in GameController.cpp verpackt.Diese loggt nun ins Terminal und in die Logdatei.



# Pipeline: Gitlab-ci Config für Docker

***18-06-2020, Hash 9fb8f0a2, Author: Jonas Lambing***

**GIT Message**

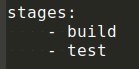
Update .gitlab-ci.yml

**Inhalt**

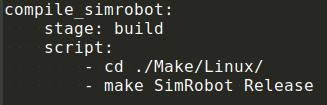
* Ziel: Automatisiertes Bauen und Testen / Benchmarken des Projekts mit Hilfe eines Docker Containers und der Gitlab CI CD Pipeline. Dafür wird eine Configdatei benötigt die als Bauplan („Blueprint“) verwendet wird.
* Dieser Bauplan gibt u.A. an, welches Image für das Betriebssystem des Containers verwendet werden soll. In unserem Fall Ubuntu 18.04.



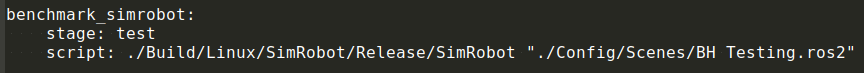
* Der Plan ist auch in 2 Stages unterteilt: Build und Test.



* Die Buildstage besteht darin, in den *./Make/Linux* zu navigieren („cd“) und den Buildbefehl auszuführen.



* Wenn die Buildstage vollendet ist, wird der Simulator mit der Testmap ausgeführt.



# Erstellung einer Objekterkennung

Der NAO Roboter soll mithilfe von künstlicher Intelligenz lernen, einen anderen NAO, ein Tor oder einen Ball zu erkennen. Die Erkennung erfolgt über die Aufnahmen der drei eingebauten Kameras links und rechts, sowie die nach unten gerichtete Kamera. Mithilfe dieser Aufnahmen kann der NAO seine Umgebung wahrnehmen, allerdings fehlt hier noch die Interpretation des Gesehenen.

Dazu soll die Erstellung eines Modells zur Objekterkennung beitragen. Dieses wurde im Folgenden mithilfe von Tensorflow[[1]](#footnote-1) und Keras[[2]](#footnote-2) konzipiert.

Die Abbildung 1 zeigt den Ausgangspunkt des Modells, von dem aus es weiter verbessert werden soll.

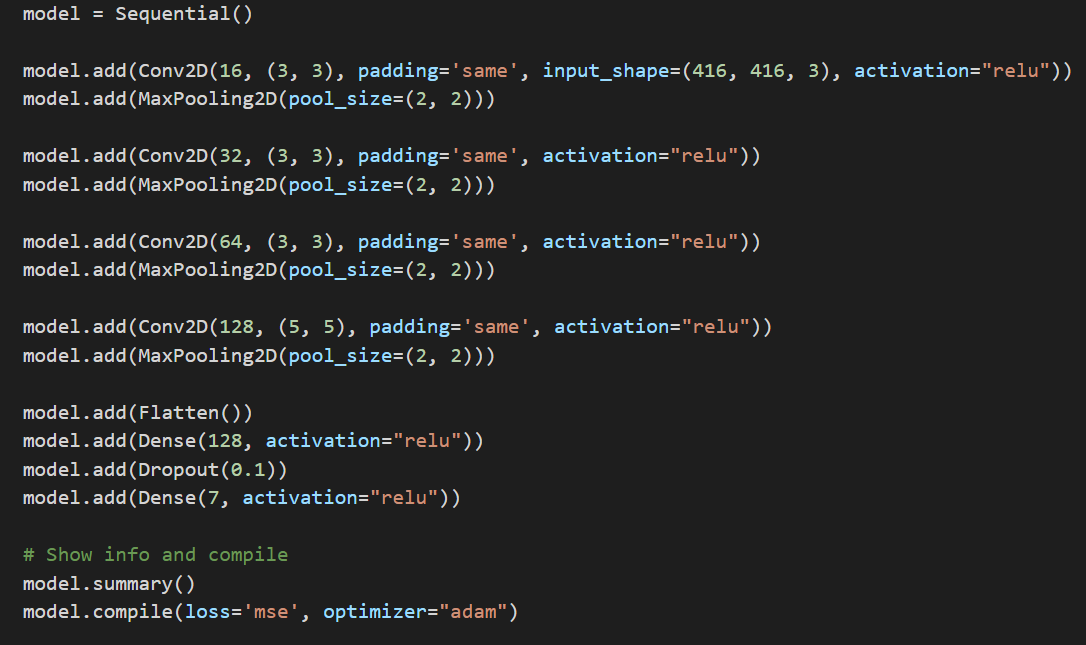


Abbildung 1 Ausgangsmodell zur Objekterkennung

Mit diesem Aufbau wurden die ersten Versuche zur Verbesserung durchgeführt. Diese sind in Tabelle 1dokumentiert und anschließend mit Begründungen zu den jeweiligen Änderungen versehen.

Zuvor erfolgt aber, zum besseren Verständnis, eine kurze Übersicht über die verwendeten Begriffe. Aufgrund der besseren Wiedererkennbarkeit im Code werden für die Dokumentation der Werte die englischen Begriffe verwendet.

**Erläuterungen zum Modell**

**Tensor**

Ein Tensor ist eine mathematische Funktion, die eine bestimmte Anzahl von Vektoren auf einen Zahlenwert abbildet.[[3]](#footnote-3)

Ableitung des Begriffs Tensorflow

**Class Sequential**[[4]](#footnote-4)

Ein sequenzielles Modell eignet sich für einen einfachen Stapel von Schichten, bei dem jede Schicht genau einen Eingangstensor und einen Ausgangstensor hat.

Die Klasse Sequentiell gruppiert daher einen linearen Stapel von Schichten zu einem tf.keras.Modell.

**Class Model**

Model gruppiert Schichten zu einem Objekt mit Trainings- und Inferenzmerkmalen.

Inferenz ist der Prozess, durch den die Modelle mit den Daten verglichen werden. Dazu gehört normalerweise, das Modell mathematisch zu gießen und die Wahrscheinlichkeitsprinzipien anzuwenden, um die Qualität der Übereinstimmung zu quantifizieren.

**Conv2D**

Übersetzt: 2D Convolutional Layer = Faltungsschicht

Hier geschieht eine räumliche Faltung über die Bilder nach den angegebenen Werten.

**MaxPooling2D**

Verringert die Eingabedarstellung, indem der Maximalwert über das durch pool\_size definierte Fenster für jede Dimension entlang der Feature-Achse genommen wird. Das Fenster wird in jeder Dimension um eine Schrittweite verschoben.

**Flatten**

Verflacht die Eingabe. Hat keinen Einfluss auf die batch\_size.

**Dense**

Dient dazu die Matrizen auf eine einzelne Zahl als Ausgabe zu reduzieren.

**Dropout**

Die Dropout-Schicht setzt die Eingabeeinheiten nach dem Zufallsprinzip auf 0 bei jedem Schritt während der Trainingszeit, was dazu beiträgt, eine Überanpassung zu verhindern. Eingaben, die nicht auf 0 gesetzt werden, werden um 1/(1 - Rate) hochskaliert, so dass die Summe über alle Eingaben unverändert bleibt.

**Erläuterungen zu den Trainingsparametern**

**Epochs[[5]](#footnote-5)**

In eine Epoche läuft der gesamte Datensatz einmal vorwärts und rückwärts durch das neuronale Netzwerk.

**Batch\_Size**

Gesamtzahl der Trainingsdaten die in einem Stapel vorhanden sind. Das bedeutet die Aufteilung der Daten, welche in einer Epoche durch das Netz laufen. Damit werden in jeder Epoche nur dieser Teil der Trainingsdaten gelernt.

Batch\_Size = 1 entspricht daher allen Trainingsdaten.

**Validation Split**

Aufteilung der Daten in Trainingsdaten und Testdaten. Mithilfe der Trainingsdaten wird anhand der Labels gelernt was z.B. ein NAO ist. Anhand der Überprüfung mithilfe der Testdaten passt das Netz die jeweiligen Verknüpfungen an.

Validation Split = 0,3 entspricht bei 100 Daten = 70 Trainingsdaten und 30 Testdaten

**Loss**

Je geringer der Verlust (Loss), desto besser ein Modell (es sei denn, das Modell hat sich zu sehr an die Trainingsdaten angepasst). Der Verlust wird auf der Grundlage von Training und Validierung berechnet, und seine Interpretation gibt an, wie gut das Modell für diese beiden Sätze abschneidet. Im Gegensatz zur Genauigkeit (Accuracy) ist der Verlust kein Prozentsatz. Er ist eine Summierung der Fehler, die für jedes Beispiel in Trainings- oder Validierungssätzen gemacht wurden.

**Accuracy**

Die Genauigkeit für z.B. eines Klassifikationsalgorithmus für maschinelles Lernen, ist eine Möglichkeit zu messen, wie oft der Algorithmus einen Datenpunkt korrekt klassifiziert. Die Genauigkeit ist die Anzahl der korrekt vorhergesagten Datenpunkte von allen Datenpunkten.

**Val\_Loss**

Validation Loss = Validierte Verlustfunktion

Val\_loss gibt die Gesamtzahl der Fehler der Testdaten aus. Das bedeutet der validierten Daten des Netzes. Das Ziel ist es den loss und val\_loss zu minimieren auch um Overfitting zu vermeiden.

**Val\_Accuracy**

Validation Accuracy = Validierte Genauigkeit

Die Testdaten werden auf die Genauigkeit hin überprüft. Stimmt beispielsweise die Klasse Nao mit dem Bild Nao überein, geht die Genauigkeit hoch.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Änderung** | **Parameter Training** | | | **Ergebnis**  **(Angabe letzte Epoche)** | | | |
|  |  | Epochs | Batch Size | Val\_  Split | Loss  (Fehler) | Accuracy  (Prozent) | Val\_ Loss | Val\_ Accuracy |
| 1 | Keine Änderung, Test Ausgangsmodell | 50 | 1 | 0,3 | 522,2696 | 0,8654 | 1376,1442 | 0,7510 |
| 2 | Ergänzung eines weiteren Layers | 10 | 1 | 0,3 | 1474,1371 | 0,6704 | 1533,0802 | 0,6570 |
| 3 | Erhöhung der Epochen | 50 | 1 | 0,3 | 1459,6091 | 0,6704 | 1503,2593 | 0,6570 |
| 4 | Loss = categorical crossentropy | 20 | 1 | 0,3 | 486,4404 | 0,8388 | 530,9670 | 0,7704 |
| 5 | Loss bleibt, letzter Layer wird wieder entfernt (Ausgangsmodell) | 20 | 1 | 0,3 | 485,5236 | 0,8483 | 545,8169 | 0,7372 |

Tabelle 1 Durchführung der ersten Versuche zur Verbesserung

**Versuch 1**

Allgemeines Testen des Ausgangsmodells, daher keine Veränderungen.

**Versuch 2**

Ergänzung um einen Layer wie folgt:

model.add(Conv2D(256, (5, 5), padding='same', activation="relu"))

model.add(MaxPooling2D(pool\_size=(2, 2)))

Änderung der Epoche auf 10 für eine schnelle Überprüfung, ob eine Verbesserung eingetreten ist.

**Versuch 3**

Erhöhung der Anzahl der Epochen aufgrund, der Verschlechterung des vorherigen Versuchs.

**Versuch 4**

Änderung der Loss-funktion von „mse“ auf „categorical crossentropy“

Mse: „Mean Square Error“ ist die Summe der quadrierten Abstände zwischen der Zielvariablen und den vorhergesagten Werten.

Categorical Crossentropy[[6]](#footnote-6): Berechnet den Kreuzentropieverlust zwischen den Labels und Vorhersagen. Verwendung dieser Funktion des Kreuzentropieverlusts, wenn es zwei oder mehr Label-Klassen gibt.

Idee ist es, durch diese Anpassung andere Gewichte zu trainieren, da das Netz anhand des loss angepasst wird.

Änderung der Epochen auf 20, da 10 für eine Aussage zu wenig ist und 50 erheblich länger dauert.

**Versuch 5**

Änderung auf das Originalmodell, Loss bleibt bei Categorical crossentropy.

**Fazit der ersten Durchläufe**

Durch die Einstellung des Models auf 7 Ausgabeparameter wurde das Netz vermutlich falsch trainiert.

model.add(Dense(7, activation="relu"))

Die 7 steht an dieser Stelle für die 4 Koordinaten, welche die Bounding Box aufspannen, sowie die 3 Klassen „NAO, Ball und Tor“. Aufgrund der erwarteten Ausgabe von 7 möglichen Klassifizierungen, kann das Netz keine Unterscheidung zwischen den Klassen und den Bounding Boxen treffen. Daher müssen an dieser Stelle Anpassungen vorgenommen werden.

# Nao: Flashen der Robocup Firmware

**Hinweis**: Das Image *nao-2.8.5.11\_ROBOCUP\_ONLY\_with\_root.opn* und eine funktionsfähige Kopie des *NaoFlashers* befinden sich im Downloads-Ordner auf der *zwoogle4*.

## Vorbereiten des Flashers unter Ubuntu 18.04

Zuerst muss die Flashersoftware heruntergeladen werden, aktuelle Releases gibt es unter https://www.softbankrobotics.com/emea/en/support/nao-6/downloads-softwares.

Der Flasher funktioniert unter Ubuntu 18.04 standardmäßig nicht, da der eine veraltete Version des ZLIB-Moduls verwendet. Dies kann aber relativ einfach behoben werden.

Nach dem Extrahieren des Flashers muss man in den Unterordner */lib/* im Flasherverzeichnis navigieren und diesen im Terminal öffnen.

Dort benennt man das alte ZLIB-Modul mit dem Befehl *mv libz.so.1 libz.so.1.old* um und erstellt mit dem Befehl *ln -s /lib/x86\_64-linux-gnu/libz.so.1* einen Link zur neueren Version des ZLIB-Moduls im System. Jetzt kann man den Flasher Problemlos ausführen (Immer als Root starten!).

## USB Stick zum Flashen vorbereiten

Der USB Stick darf kein Datei- oder Partitionssystem enthalten! Dieses kann mit einem Tool wie Gparted unter Linux entfernt werden.

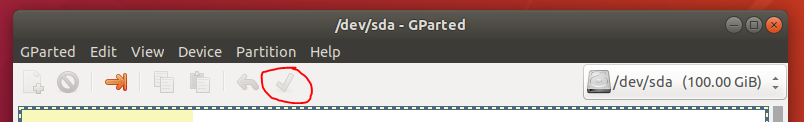
Man installiert und startet Gparted und wählt der USB Stick rechts oben aus (ACHTUNG! Immer darauf achten das richtige Speichermedium auswählen!)

Diesen muss man jetzt per Rechtsklick auf die Partitionstabelle zuerst Unmounten.



Danach kann man entweder *Entf* auf der Tastatur drücken, oder die Partitionen per Rechtsklick und Delete auf die Tabelle entfernen.

Im letzten Schritt muss man noch die Änderungen anwenden. Dazu klickt man auf den Haken oben im Programm.



## Flashen des USB-Sticks

Man öffnet den Ordner des Flashers im Terminal und startet ihn mit dem Befehl *sudo ./flasher.*

Über den Button *Browse* neben *Image to flash* kann eine Firmwaredatei ausgewählt werden (in unserem Fall *nao-2.8.5.11\_ROBOCUP\_ONLY\_with\_root.opn*).

Unter *Choose your usb stick* wird der zu flashende USB Stick ausgewählt (ACHTUNG! Immer sicherstellen, dass der richtige Stick ausgewählt ist, der Flasher kann sonst Schäden an den Partitionen des ausgewählten Speichermediums verursachen!).

Jetzt muss man nur noch die Option *Factory Reset* aktivieren und auf den Button *Write* klicken.

## Flashen der Software auf dem Nao

Zuerst muss der Nao komplett ausgeschaltet werden (Chestbutton so lange drücken, bis Nao ein akustisches Signal ausgibt).

Dann wird der Stick in die USB-Buchse am Hinterkopf des Naos eingesteckt und man hält den Chestbutton ca. 5s gedrückt, bis er anfängt blau zu leuchten.

Das Update dauert eine Weile und der Roboter sollte nach Abschluss automatisch rebooten.

Näheres zum Flashingprozess findet man hier: [*http://doc.aldebaran.com/2-1/software/naoflasher/naoflasher.html*](http://doc.aldebaran.com/2-1/software/naoflasher/naoflasher.html)und hier: *http://doc.aldebaran.com/2-1/nao/boot\_process\_nao.html#upgrading-process-nao.*

# Nao: Deployen der Software auf dem Roboter

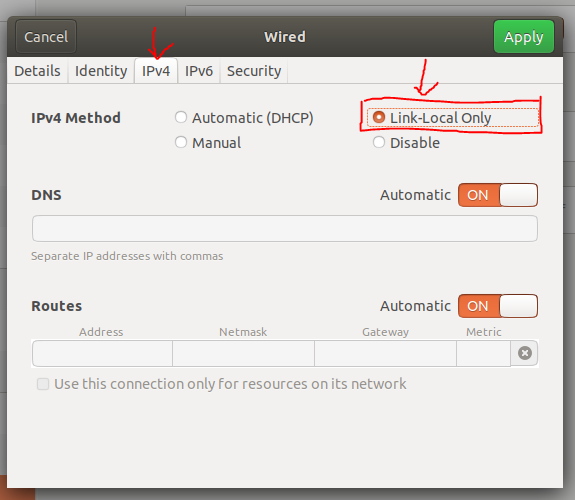
## Den Code zum Deployen bauen

Man öffnet den Ordner *<Git-Repo>/Make/Linux* und führt den Befehl „make all“ aus (dieser baut alle Projekte, wenn nichts anderes spezifiziert ist wird die Develop Buildconfig benutzt, weitere Optionen findet man in der *CodeRelease2019.pdf* im Punkt 2.2).

## Den Nao mit dem PC verbinden

Zuerst muss der Nao mit einem Patchkabel an die LAN-Buchse des PC’s angeschlossen werden.

Danach öffnet man die Netzwerkeinstellungen, setzt einen Haken bei *Link- Local Only* im Tab IPv4 und klickt auf *Apply*.



Nun kann man den Chestbutton des Nao drücken, und der Roboter sollte seine IP-Adresse verraten (diese sollte man sich für später notieren!).

## Den Nao vorbereiten

Wenn der Nao erfolgreich mit dem PC verbunden wurde und auf *ping* (ping <IP>) reagiert, kann man mit dem Prozess fortfahren.

Jetzt begibt man sich im Terminal in den Ordner *<Git Repo>/Install/* und führt das Skript *createRobot* aus. Dieses Skript hat noch ein paar Parameter, es ist folgendermaßen aufgebaut:

*./createRobot -t <Teamnummer> -r <Nummer des Roboters> -i <IP-Adresse> <Name>*

Der Parameter *-t <Teamnummer>* ist **optional**, falls nicht angegeben wird die Teamnummer aus *Config/settings.cfg* genommen (standardmäßig 89).

Es wird mit hoher Wahrscheinlichkeit 2x nach einem Passwort gefragt, dies ist normalerweise *nao*.

Weitere Infos gibt es in der Datei *CodeRelease2019.pdf* unter dem Punkt *2.3*.

Wenn dieses Skript erfolgreich ausgeführt wurde kann man mit dem nächsten Schritt beginnen.

Im gleichen Ordner befindet sich ein weiteres Skript mit dem Namen *installRobot*. Dieses wird mit *./installRobot <IP-Adresse>* ausgeführt.

Es wird sehr wahrscheinlich nach dem root-Passwort gefragt, dieses ist im Normalfall auch *root*.

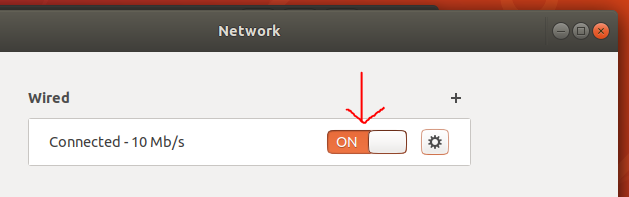
Wenn der Befehl erfolgreich ausgeführt wurde, sollte der Roboter von selbst Neustarten. Nach dem Neustart wird er eine neue IP-Adresse nach dem Muster 192.168.<Teamnummer>.<Nummer des Roboters> haben (ACHTUNG! Die Nummern 0 und 1 sind für die Nummer des Roboters nicht zulässig).

Nun muss man seine Netzwerkeinstellungen verändern, um auf den Nao zugreifen zu können. Man geht wieder in die Einstellungen und setzt im Tab *IPv4* einen Haken bei *Manual*.

Unter *Address* gibt man eine IP im neuen Bereich des Roboters ein, also z.B. *192.168.<Teamnummer>.0* (ACHTUNG! Die letzte Ziffer darf nicht gleich mit der Roboternummer sein!).

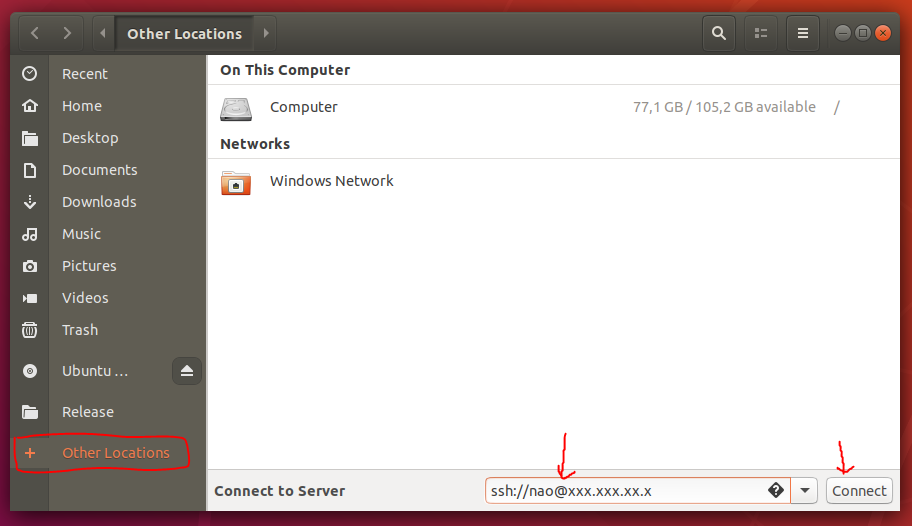
Unter Netmask gibt man *255.255.255.0* ein. Danach klickt man auf den grünen Button mit der Aufschrift *Apply.*

Jetzt muss die Verbindung mit 2 Klicks auf den On / Off Switch neugestartet werden, und man sollte sich wieder mit der neuen IP mit dem Nao verbinden können.



## Das Deployment vorbereiten

Zuerst muss man sich mit dem Dateimanager auf den Roboter verbinden. Dies funktioniert, indem man auf *Other Locations* auf der linken Seite des Dateimanagers klickt und dann unten bei Connect to Server die Adresse *ssh://nao@<IP-Adresse* eingibt. Danach kann man sich mit dem Connect-button verbinden.



Nach erfolgreicher Verbindung navigiert nach */home/nao/.*

Nun öffnet man sich noch ein zweites Dateibrowserfenster und navigiert zu */<Git-Repo>/Config* undkopiert alle Inhalte (ohne die Ordner /Keys, /Logs und /Scenes) nach */home/nao/Config* auf dem Roboter.

Jetzt kopiert man noch die Datei *bhuman* aus dem Ordner */<Git-Repo>/Install/Files/bin/* nach */home/nao/Config* auf dem Roboter.

Außerdem kopiert man noch die Inhalte aus */<Git-Repo>/Install/Network/Profiles/* nach */home/nao/Profiles* auf dem Roboter.

## Den eigentlichen Code deployen

Man muss sich zuerst im Terminal in den Ordner */<Git-Repo>/Make/Common/* begeben, dort befindet sich ein Skript namens *copyfiles*.

Auch dieses Skript hat mehrere Parameter.

Der Aufbau ist *./copyfiles <Build, z.B. Develop> -nc -r <Nummer des Roboters> <IP-Adresse>*.

Das Skript hält den bhuman Dienst auf dem Nao selbständig an, setzt den Code auf dem Roboter auf und started den Dienst wieder.

Dieser ist der einzige Schritt, der gemacht werden muss, wenn man den Code updaten will und die Skripte *createRobot* und *installRobot* schon auf dem Roboter gelaufen sind.

Nach einem Neustart des Roboters started sich der Dienst selbstständig und der Roboter kann benutzt werden.

Durch drücken des Chestbuttons kann man durch die verschiedenen Modi der Software gehen.

Näheres gibt es in der Datei *CodeRelease2019.pdf* im Punkt *2.5*.

# Pipeline: Grundgerüst für Regressionstests

Zum Benchmarken habe ich eine Methode im *GameController* namens *addBenchmarkLog()* hinzugefügt (Weitere Informationen auf Seite 12 dieses Dokuments).

Diese erzeugt eine Datei namens *benchmark\_log.txt* in */<Git-Repo>/Config/Scenes/.*

Momentan wird die Zeit vom Starten des Matches bis zum ersten Ballkontakt und bis zum ersten Tor mitgeloggt.

Mein Vorschlag wäre, einen *GameController* Befehl zu Implementieren, der Global eine Variable setzt um das Benchmark ein- / auszuschalten (Mehr Informationen dazu auf den Seiten 10-11 dieses Dokuments).

1. <https://www.tensorflow.org/> [↑](#footnote-ref-1)
2. <https://www.tensorflow.org/api_docs/python/tf/keras> [↑](#footnote-ref-2)
3. <https://de.wikipedia.org/wiki/Tensor> [↑](#footnote-ref-3)
4. <https://www.tensorflow.org/api_docs/python/tf/keras/Sequential> [↑](#footnote-ref-4)
5. <https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9> [↑](#footnote-ref-5)
6. <https://keras.io/api/losses/probabilistic_losses/#categoricalcrossentropy-class> [↑](#footnote-ref-6)