



Hochschule Darmstadt
Fachbereich Informatik

NAOs Gestenprogrammierung und Tracking vom Sphero

Projekt Systementwicklung Roboter-Roboter-Mensch-Mimik
Bachelor of Science (B.Sc.)

vorgelegt von R2M2 - Gruppe 2
Jörg Tran, Anil Alsac, Richard Fust, Manu Pumahualca,
Arthur Rieß, Selim Sinan

Version 1.0: 12.10.2018

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht. Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Jörg Tran, Anil Alsac, Richard Fust, Manu Pumahualca, Arthur Rieß, Selim Sinan
Darmstadt, den 12.10.2018

Inhaltsverzeichnis

Erklärung	iii
Abbildungsverzeichnis	vii
1 Einleitung	1
2 Zielsetzung	3
3 Grundlagen	5
3.1 Hardwareeigenschaften vom NAO	5
3.2 Hardwareeigenschaften vom Sphero	5
3.3 Choregraphie Suite	6
3.4 NAOqi und Module	7
3.5 Tracker	8
4 Technische Umsetzung im Choregraphie	11
4.1 Technische Umsetzung des Gestenportfolio	11
4.2 Technische Umsetzung des Redballtrackers	12
4.2.1 onStart	13
4.2.2 onFound	15
4.3 Technische Umsetzung des Spehros	19
5 Zusammenfassung	21
Literaturverzeichnis	xv

Abbildungsverzeichnis

2.1 NAO und Sphero	3
3.1 Timeline	6
4.1 Zustände	11
4.2 Gestenportfolio	12
4.3 Sphero Code	19

1 Einleitung

Roboter, auch Androiden oder kurz "Bots", sind in unserer heutigen Zeit allgegenwärtig. In vielen Bereichen des Menschen, zum Beispiel der Medizin, Industrie oder Automation sind diese fast nicht mehr wegzudenken. Sie werden immer schneller, stärker, besser. Roboter nehmen dem Menschen viel schwere körperliche Arbeit ab, erfüllen ihren Job mit unnachahmlicher Präzision und ohne Pausen. Doch solche automatisierten Geräte müssen nicht immer am Fließband schuftende Arbeitstiere sein, damit sie ihren Nutzen erfüllen - Mittlerweile werden Roboter erforscht und gebaut, die immer mehr dem Menschen ähneln und Aufgaben erfüllen, die in direkter sozialer Hinsicht ihren Erfindern helfen. Im Bachelormodul "Roboter-Roboter-Mensch-Mimik" des Informatik-Studiengangs an der Hochschule Darmstadt hatten wir die Möglichkeit, mit einem humanoiden Roboter, also einem Roboter mit menschlichem Aussehen und menschenähnlichen Interaktionen, zu arbeiten und diesen nach unseren Wünschen zu programmieren. Unsere Aufgabe war es, ihm die Möglichkeit zu geben, Gesten aus seiner unmittelbaren Umgebung zu erkennen; also durch Ausnutzen seiner visuellen und optischen Sensoren Gegenstände zu verfolgen und aufgrund derer Bewegung eine bestimmte Reaktion auszulösen. Wir wurden in sechsköpfige Teams gesteckt mit dem Ziel innerhalb drei Wochen unserem kleinen mechanischen Freund beizubringen, sinnvoll auf seine Umgebung zu reagieren. Nachfolgend finden Sie eine vollständige Dokumentation unserer Arbeit.

2 Zielsetzung



Abbildung 2.1: Nao betrachtet Sphero¹

Unser Projektziel ist, dass der Nao auf verschiedene Gesten des Sphero reagiert. Zunächst müssen wir Nao beibringen, seinen Fokus auf Sphero zu lenken. Unser Ziel besteht darin, dass Nao auf Gesten, Bewegungen und Farbeinstellungen des Sphero mit eigenen Bewegungen und Sprache reagiert - und dazu müssen wir erst NAOs Erkennung von Gesten implementieren. NAO muss also Bewegungsabläufe des Sphero kapseln, diese mit seiner Datenbank vergleichen und bei einem Match darauf reagieren. In dieser zweiten Phase möchten wir NAO zunächst nur beibringen, dass er seine

2 Zielsetzung

Hände hebt, wenn der Sphero ihm näher kommt. Um dies zu realisieren, muss NAO die Distanz zwischen ihm und Sphero in kurzen Intervallen bzw. in jedem Frame berechnen können und bei kontinuierlichem Abfall den Befehl "handsup()äusführen. Nachdem wir es geschafft haben, dass Nao spezielle Bewegungsmuster des Sphero erkennt, soll er darauf reagieren. Dafür bauen wir ein Gestenportfolio mit mindestens 5 Gesten auf:

1. Geste: Sphero rollt seitwärts herum

NAO soll sich über Spheros Gesellschaft freuen. Er fängt an zu klatschen, läuft auf der Stelle und sagt Dinge wie "You're great Sphero!"

2. Geste: Sphero kommt auf NAO zu

NAO kriegt Angst, hebt die Hände und sagt Sätze wie "Pls don't hurt me-öder "Hey, stay away!ÄLTERNATIV: Erkennt NAO dass der Sphero in seiner Bewegungsreichweite ist, könnte er ihm auf den "Kopf'tätscheln bzw. auf ihn zeigen und Dinge sagen wie "You're so small!ö.ä.

3. Geste: Sphero bewegt sich von NAO weg

NAO ruft Sphero zu sich zurück. Er hebt eine Hand und ruft Dinge wie "Come back!öder "Hey, dont leave me!"

4. Geste: Sphero verlässt NAOs Sichtweite

NAO kann Sphero nicht mehr tracken. In diesem Fall wird NAO traurig, dass Sphero ihn verlassen hat. Er nimmt eine traurige Pose ein, fängt ggf. an zu weinen und ruft SSo long, little friend..."

5. Geste: Sphero betritt NAOs Sichtweite

NAO fängt an Sphero zu tracken. Er begrüßt ihn freundlich mit einem Winken, steht auf einem Bein und sagt einen Satz wie "Hey my little friend sphero"

3 Grundlagen

3.1 Hardwareeigenschaften vom NAO

Nao ist ein humanoider Roboter des französischen Roboterherstellers Aldebaran Robotics. Nao reagiert auf Sprachbefehle, erkennt Gesichter und kann diverse Bewegungen mit Armen, Beinen, Fingern, Füßen und dem Kopf ausführen.

Hardwareeigenschaften:

- 58 cm groß und 5,6 kg schwer
- 1.91 GHz Intel Atom Prozessor
- Ethernet- und WLAN-fähig
- Zwei Kameras, vier Mikros und eine Vielzahl anderer Sensoren

3.2 Hardwareeigenschaften vom Sphero

Sphero ist ein kleiner kugelrunder Roboter des gleichnamigen Herstellers Sphero aus Amerika. Durch seine runde Form bewegt er sich durch Rollen über den Boden. Gesteuert wird der Sphero mit einer Smartphone-App oder einem "Force Band"(nur beim BB-8) über Bluetooth.

Hardwareeigenschaften:

- 170 Gramm schwer

3 Grundlagen

- 75 MHz ARM Cortex M4 Prozessor
- Bluetooth-Empfänger
- Beschleunigungssensor zur Messung der Geschwindigkeit
- Kreisstabilisator bzw. Gyroskop zur aktiven Lageregelung

3.3 Choregraphe Suite

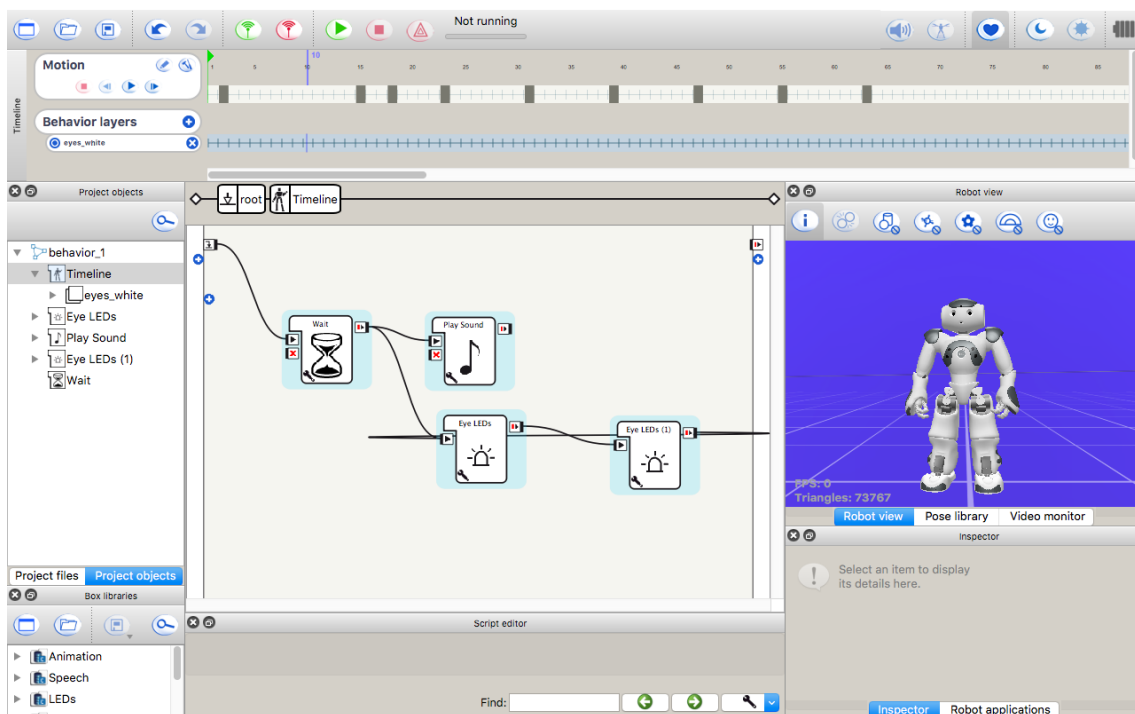


Abbildung 3.1: Timeline vom Choregraphe¹

Choregraphe ist eine Desktopapplikation, die erlaubt:

Animationen, Verhalten und Dialoge zu kreieren den Roboter zu überwachen und zu kontrollieren Python Code für das Roboterverhalten zu implementieren Im Timeline Modul von Choregraphe können wir Choreos erstellen. Dieser verbinden wir mit dem onStart und durch doppelklick kann man für einen bestimmten Zeitpunkt bzw. Frame dem Nao eine Haltung einnehmen lassen. Dieser wird in vier Bereiche eingeteilt:

- Whole Body
- Head
- Arms
- Legs

Den Abstand zwischen zwei Haltungen kann man bearbeiten, indem man die Framerate der Timeline erhöht bzw. erniedrigt. Der Default ist beim Choregraphe bei 25 fps, jedoch wird geraten diese zu erniedrigen auf 10 fps.

Um die Gliedmaßen zu bearbeiten klickt man auf einen bestimmten Bereich vom NAO im Simulator (Robot View) und kann diese im Inspector bewegen. Dazu sollte man zuerst autonomous life und dann wake up einschalten, da der NAO sonst nicht in seiner Position bleibt.

Um LEDs, Sprache, Musikdateien und weitere einzubinden, wählt man aus den Box Libraries die bestimmten Module und zieht diese mit in die Timeline.

3.4 NAOqi und Module

NAOqi

Die NAOqi, die auf dem Roboter ausgeführt wird, ist ein Broker. Wenn es gestartet wird, wird eine Voreinstellungsdatei namens autoloading.ini geladen, die definiert, welche Bibliotheken geladen werden sollen. Jede Bibliothek enthält ein oder mehrere Module, die den Broker verwenden, um ihre Methoden zur Verfügung zu stellen.

Modul

Ein Modul ist eine Klasse innerhalb einer Library. Wenn eine Library durch die autoloading.ini geladen startet sie die instanziierte Modul Klasse. Ein Modul kann entweder ein Remote Modul oder ein Local Modul sein.

Remote:

3 Grundlagen

Wird es als ausführbare Datei kompiliert und kann außerhalb des Roboters ausgeführt werden. Remote-Module sind einfacher zu bedienen und können leicht von außen getestet werden, sind jedoch in Bezug auf Geschwindigkeit und Speichernutzung weniger effizient.

Local:

Wird als local library kompiliert und kann nur auf dem Roboter verwendet werden. Sie ist jedoch effizienter als ein Remote-Modul.

Proxy

Ein Proxy ist Objekt das ein Modul repräsentiert. Es verhält sich auch dementsprechend wie das Modul. Damit lassen sich alle Methoden des repräsentierenden Moduls ausführen.

```
bewegung = ALProxy("ALMotion")  
bewegung.walkTo()
```

Will man extern zB. auf einen Rechner ein Proxy ausführen muss man die IP und den Port als Parameter übergeben. Wichtig dabei ist die Naoqi zu importieren.

```
import naoqi  
bewegung = ALProxy("ALMotion", IP, PORT)  
bewegung.walkTo()
```

Will man den Proxy intern im Roboter ausführen muss man einen Service in seiner Session starten.

```
self.bewegung=self.session().service("ALMotion")  
self.bewegung.walkTo()
```

3.5 Tracker

Das Gesamtprogramm wurde über Choregraphie realisiert, einer Entwicklungsumgebung für NAO, welches die Programmierung der Animationen über Bausteine vereinfacht. Die Outputs dieser Bausteine kann man mit den Inputs anderer Bausteine

verbinden. Ein Baustein ist wie eine Klasse. Ein Input ist eine Funktion dieser Klasse und eine Output ist ein Funktionsaufruf mit oder ohne Parameter einer fremden Funktion. Mehrere Bausteine kann man auch in einen Baustein zusammenfassen. Innerhalb des Bausteins kann man den Python-Code bearbeiten. Der Tracker wurde in einem Baustein von uns programmiert und ist eine Art Mainfunktion die nach jeder Animation wieder aufgerufen wird. Der Baustein *Tracker* gibt einen String-Output an einen Switch-Case-Baustein, welche die Animationen *left*, *near*, *stop*, *hello*, *lost*, *far*, *right* aufruft. Nach der ausgeführten Animation geht der Pfad zurück in die Tracker-Box.

4 Technische Umsetzung im Choregraphie

4.1 Technische Umsetzung des Gestenportfolio

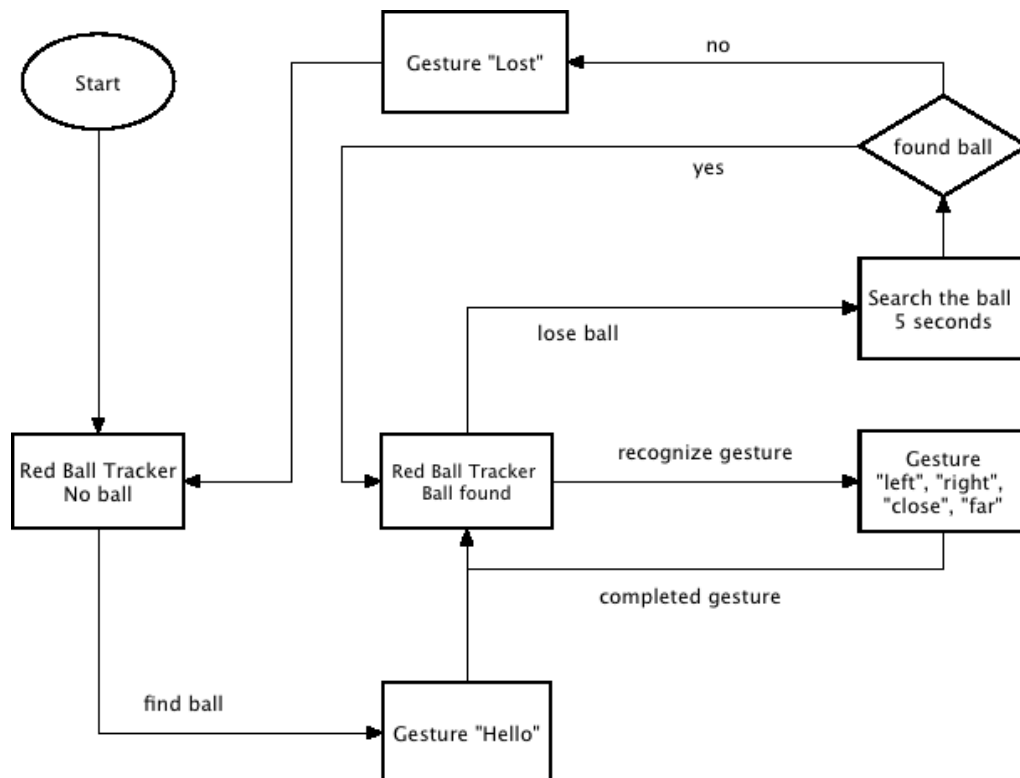


Abbildung 4.1: Zustände des Programms¹

Nach dem Programmstart gehen wir in den Projektzustand "Redballtracker no Ball". In dem Zustand sucht NAO den roten Ball Sphero mit der Position und macht dann die Geste "Hello", nachdem er diesen gefunden hat. Daraufhin sind wir im Zustand "Redballtracker Ball found", in der NAO den ball im Blick hat und auf eine Aktion

4.2.1 onStart

```
def onInput_onStart(self):

    # Initialisierung
    self.motion=self.session().service( "ALMotion" )
    self.posture=self.session().service( "ALRobotPosture" )
    self.tracker=self.session().service( "ALTracker" )
    self.tts=self.session().service( "ALTextToSpeech" )
```

Es wird eine Proxy zu den Modulen vom Nao aufgebaut, um deren Funktionen aufrufen zu können.

```
#Tracker resettet
self.tracker.stopTracker()
self.tracker.unregisterAllTargets()
self.tracker.toggleSearch(False)
```

Das Trackermodul wird explizit gestoppt und alle vorigen Ziele gelöscht, bevor unser Programm den Tracker neu initialisiert.

```
#Tracker Parameter
self.fractionMaxSpeed = 0.8
self.targetName = "RedBall"
self.diameterOfBall = 0.06
self.mode = "Head"
self.hasTargetDetected = None
self.n = []
self.position = [0,0,0]
print self.position
self.ynew = 0
self.yold = 0
self.xpos = 0
self.hasTargetDetected = None
```

Variablen für den Redballtracker werden initialisiert. Hierbei ist anzumerken, dass in *self.targetName* Ziele aus vordefinierten Trackingfunktionen gewählt werden. Diesen

4 Technische Umsetzung im Choregraphe

kann man auf Gesichter oder wie in unserem Fall auf rote Bälle konfigurieren, so dass wir keine eigene Bilderkennung benutzen müssen. Demnach haben wir unseren Sphero mit einer roten Hülle ausgestattet, um diese bereits programmierte Funktion aus der Library benutzen zu können.

```
#Tracker starten
self.tracker.registerTarget(self.targetName,
self.diameterOfBall)
self.tracker.setMode(self.mode)
self.tracker.track(self.targetName)
self.tts.say("Hey, where is my friend Sphero...")
```

Der Tracker wird über die Proxy initialisiert und gestartet. Die Parameter hierfür wurden schon initialisiert. Über *tts.say* spricht Nao und gibt bekannt, dass er Sphero sucht. Der ALTracker aus der Library läuft parallel während unser Programm in eine Schleife springt.

```
while True:
    print("HasTarget_onStart: ",
self.hasTargetDetected)
    if self.tracker.isNewTargetDetected()
and self.hasTargetDetected == None:
        self.hasTargetDetected = True
        self.movementDetected("hello")
        break
    time.sleep(0.2)
pass
```

Die Schleife wird fünf Mal die Sekunde aufgerufen, bis die Bedingung in der If-Klausel eintritt. Die If-Klausel prüft, ob der parallel laufende Tracker den Status *.isNewTargetDetected()* gesetzt hat und ob unsere Kontrollvariabel false ist. Wenn die If-Klausel eintritt, wird unsere Kontrollvariable auf true gesetzt, ein Output aufgerufen mit *self.movementDetected* und über den Parameter "hello" die Begrüßungsanimation abgespielt. Die Schleife wird an diesem Punkt abgebrochen. Nach jeder Animation wird nicht onStart aufgerufen sondern onFound, da bereits ein Ziel existiert.

4.2.2 onFound

Bis zur While-Schleife ist der Code identisch mit onStart, da die gleiche Initialisierung erforderlich ist. Der Tracker wird im Hintergrund neugestartet, da durch die Kopfbewegungen in der Animation das Ziel oft verloren geht.

```
def onInput_onFound(self):

    self.motion=self.session().service( "ALMotion" )
    self.posture=self.session().service( "ALRobotPosture" )
    self.tracker=self.session().service( "ALTracker" )
    self.tts=self.session().service( "ALTextToSpeech" )

    self.tracker.stopTracker()
    self.tracker.unregisterAllTargets()
    self.tracker.toggleSearch(False)

    self.fractionMaxSpeed = 0.8
    self.targetName = "RedBall"
    self.diameterOfBall = 0.06
    self.mode = "Head"
    self.n = 0

    self.hasTargetDetected = True
    self.tracker.registerTarget(self.targetName, //
                               self.diameterOfBall)
    self.tracker.setMode(self.mode)
    self.tracker.track(self.targetName)
```

Die folgende Schleife wird ebenfalls fünf Mal die Sekunde aufgerufen.

```
while True:
```

```
print("HasTarget_onFound: ", self.hasTargetDetected)
print("i: ", self.i)
#Wenn ein Ziel existiert
    if self.hasTargetDetected:
        #Kontrolle ob Ziel verloren
            if self.tracker.isTargetLost():
                self.n = 0
                self.i = self.i + 1
            if self.i == 5:
                self.targetLost("right");
                self.tts.say("Sphero?")
            if self.i == 20:
                self.targetLost("left");
            if self.i >= 35:
                self.movementDetected("lost")
                break
        print("break lost")
```

Diese Verschachtelung von If-Klauseln stellt im Grunde eine Kontrolle an den Tracker-modul, ob das verfolgte Ziel noch gültig ist. Wenn das Ziel verloren ist dreht der Roboter den Kopf nach links und rechts und die Prüfung findet so lange noch einmal statt bis der Zähler i=35 erreicht hat. Wenn diese Kontrollbedingung immer noch gültig ist wird in der nächsten Iteration der Output "lost" aufgerufen und die Schleife abgebrochen. Nach der Lost-Animation wird anders als in anderen Animationen nicht wieder onFound aufgerufen sondern onStart, sodass das Programm in den Startzustand wieder geht.

```
#Wenn Ziel nicht verloren
    else:
        self.position = self.tracker.getTargetPosition(0)
        self.xpos = self.position[0]
        self.i = 0
        print "n =", self.n
        print "Target Coordinates: ", self.position
```



```

if self.n == 3:
    self.yold = self.position[1]
    print("yold: ", self.yold)

```

In den Array *self.position*[] wird in jedem Durchgang an erster Stelle die X-Koordinate des Ziels gespeichert und nach drei Durchgängen die erste Y-Koordinate. Die X-Koordinate ist relativ zum Roboter die Entfernung. Die Y-Koordinate ist relativ zum Roboter die horizontale Bewegung.

```

if self.n >= 15:
    self.ynew = self.position[1]
    self.difference = 0
    self.difference = self.yold - self.ynew
    self.tolerance = 0.2
    self.minDist = 0.7
    self.maxDist = 2.8

```

Da wir bei der horizontalen Bewegung einen Vergleich anstellen müssen wird ein neuer Y-Punkt gemessen und die Differenz berechnet. Die Toleranz zwischen der Bewegung und die minimum und maximum Werte sind Erfahrungsbedingt definiert worden.

```

if self.xpos <= self.minDist:
    self.movementDetected("near")
    print("near")
    #time.sleep(10)
    break
    print("break movement")
elif self.xpos >= self.maxDist:
    self.movementDetected("far")
    print("far")
    #time.sleep(10)
    break
    print("break movement")
elif self.difference <= -self.tolerance:
    self.n = 0

```

4 Technische Umsetzung im Choregraphe

```
        self.movementDetected("left")
        print("left")
        break
        print("break movement")
    elif self.difference >= self.tolerance:
        self.n = 0
        self.movementDetected("right")
        print("right")
        break
        print("break movement")

    self.n += 1

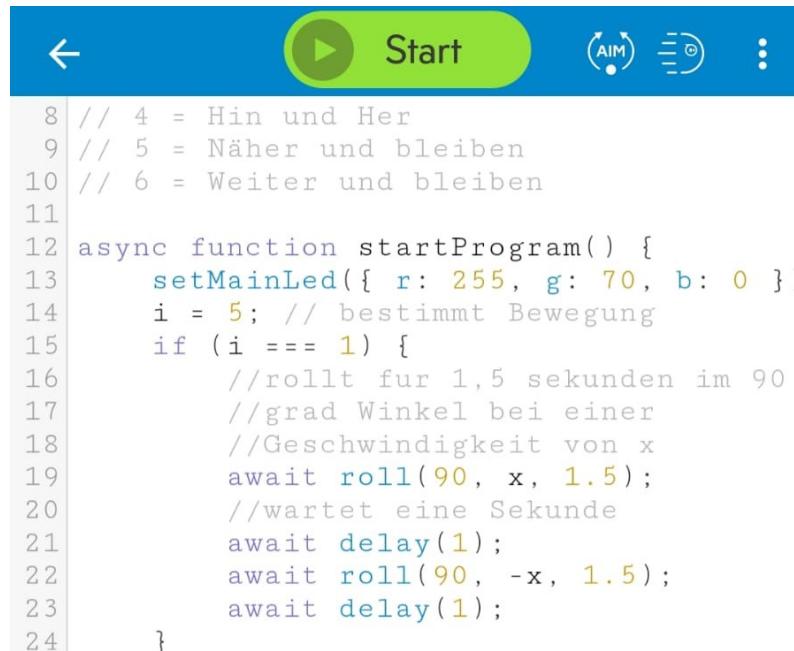
    time.sleep(0.2)
pass
```

Wenn die Entfernung kleiner oder Größer als die Minimum/Maximum Distanz ist, wird die Animation "closeöder "faräusgeführt. Bei horizontalen Bewegungen des Ziels überhalb der Toleranz wird je nachdem ob die Differenz negativ oder positiv ist die Animation "leftöder "rightäusgeführt. Die Animationen werden über Output-Aufrufe umgesetzt und nach jedem Aufruf wird die Schleife abgebrochen. Nach diesen Animationen wird wieder onFound ausgeführt.

4.3 Technische Umsetzung des Spehros

Man kann den Sphero mittels der Sphero EDU app programmieren. Die genutzte Programmiersprache ist JavaScript.

Hier ein Codestück von unserm Programm um mit dem Sphero verschiedene Bewegungsabläufe auszuführen:



```

8 // 4 = Hin und Her
9 // 5 = Näher und bleiben
10 // 6 = Weiter und bleiben
11
12 async function startProgram() {
13     setMainLed({ r: 255, g: 70, b: 0 })
14     i = 5; // bestimmt Bewegung
15     if (i === 1) {
16         //rollt für 1,5 sekunden im 90
17         //grad Winkel bei einer
18         //Geschwindigkeit von x
19         await roll(90, x, 1.5);
20         //wartet eine Sekunde
21         await delay(1);
22         await roll(90, -x, 1.5);
23         await delay(1);
24     }

```

Abbildung 4.3: Ausschnitt des Spherocodes³

5 Zusammenfassung

Nachdem wir es geschafft haben, die Schwierigkeit zu meistern, den rohen Python-code des Red-Ball-Trackers in Choregraph zu implementieren und mit den Gesten zu verknüpfen, konnten wir den Erfolg unserer Arbeit bestaunen: Je nachdem in welche Richtung wir Sphero rollen ließen, verfolgte ihn Nao und wartete nur darauf, seine vorprogrammierten Reaktion auszuführen. Wir sind jedoch auch hier auf viele Probleme gestoßen: Zum Einen wurde Nao oft durch andere rote Objekt abgelenkt - zum Beispiel durch eine Jacke, einmal sogar durch das Gesicht eines Projektmitglieds. Dadurch hörte er auf Sphero zu tracken und wir mussten die Störung manuell auf seiner Sicht entfernen. Als Zweites schwankten auch oft der erkannte Abstand und die Koordinaten des Sphero - dadurch kam es vor, dass er Reaktionen zeigte, die er hätte nicht zeigen sollen. Vor allem war die x-Koordinate stark davon betroffen; dadurch passierte es oft, dass Nao Sphero wieder zu sich zurückrief, wobei Sphero eigentlich in seiner normalen Sichtweite war. Ein drittes großes Problem war die Tatsache, dass wir Nao in unregelmäßigen Abständen neustarten mussten, da er durch mehrmaliges Ausführen unseres Codes ein unnatürliches Verhalten zeigte. All diese Probleme haben uns die Arbeit erschwert - und aufgrund unseres knappen Zeitfensters von drei Wochen konnten wir viele davon nicht beheben. Doch wir hoffen sehr, dass unsere Arbeit und unser Code von anderen Studenten aufgegriffen, verbessert und erweitert wird. Denn Nao hat ein unglaubliches Potenzial - und uns hat es sehr viel Spaß gemacht, immerhin die Oberfläche des Möglichen angekratzt zu haben.

Literaturverzeichnis

- [NaoPdf] RobotikLaborNaoAufgaben. Available online at http://northernstars-wiki.wdfiles.com/local--files/projects:nao/RobotikLabor_NaoAufgaben.pdf, checked on 10/14/2018.
- [NaoDoku] Overview — NAO Software 1.14.5 documentation (2017). Available online at <http://doc.aldebaran.com/1-14/index.html>, updated on 10/23/2017, checked on 10/14/2018.
- [SpheroDev] Blocks 1: Intro & Loops. Available online at <https://edu.sphero.com/>, checked on 10/14/2018.
- [NaoDevGuide] Wikipedia (Ed.) (2018): NAOqi - Developer guide — Aldebaran 2.8.3.54.c documentation (2018). Available online at http://doc.aldebaran.com/2-8/index_dev_guide.html, updated on 9/12/2018, checked on 10/14/2018.