Dokumentation Softwareprojekt

19. Juni 2016

Inhaltsverzeichnis

1. Einleitung

1.1. Projektbeschreibung

Im Rahmen der Veranstaltung mussten wir uns, für die Bearbeitung von Softwareprojekten in Gruppen zusammenfinden. Anschließend musste sich jedes Team eine/n betreuende/n Professor/in suchen und mit dieser/m ein Projekt, zur Umsetzung innerhalb von 2 Semestern, planen und durchführen. Unser Team, bestehend aus Edgar Filipsen, Mika Katzwinkel, Jakob Kost, Hubert Richter und Sebastian Schreiter suchte sich Frau Prof. Dr. rer. nat. Brabender als betreuende Professorin aus. Da die Aufgabenstellung offen war, überlegten wir uns, dass der NAO Minigolf spielen soll. Dabei soll er Hindernisse selbstständig umspielen und schließlich ins Ziel treffen können.

2. Eingesetzte Tools/Werkzeuge

2.1. ROS Indigo

2.2. RViz

3. Eingesetzte Hardware

3.1. NAO Roboter

Im ersten halben Jahr nutzten wir einen NAO Roboter, welcher uns freundlicherweise von Frau Dederichs-Koch des Fachbereichs M zur Verfügung gestellt wurde. Bei diesem traten leider des öfteren Probleme mit überhitzten Motoren und einem schwächelndem Akku auf. Mit dem NAO der neueren Generation traten diese weitaus seltener auf. Auch hat sich die Griffestigkeit der Finger stark verbessert.

Bei dem NAO handelt es sich um einen humanoiden Roboter der Herstellers Aldebaran. Dieser ist etwa 60 cm groß und hat ein Gewicht von 5,2 kg. Er besitzt zwei Kameras, welche ihm eine Orientierung ermöglichen, mehrere Mikrofone, Druck- und Berührungssensoren zur Interaktion, sowie ein Sonarsystem auf der Brust. Die Verbindung kann über Ethernet oder WLAN erfolgen.

///// Beschreiben der Macken?, Grobe Umschreibung Technische Details? //////

3.2. Minigolf Ausrüstung

3.2.1. Die Bahn

Da der neue NAO Schwierigkeiten mit dem Teppichboden unseres Büros hatte und um gleichbleibende Bedingungen zu ermöglichen, entschieden wir uns für eine Bahn aus Holz mit den Maßen: 2,5 m x 1,35 m. Bis auf den Startbereich ist die Bahn von einem 5 cm hohen Rand umgeben, welcher als Bande dienen soll. Das Ziel hat einen Durchmesser von 15 cm und ist 45 cm vom hinteren Rand entfernt (siehe Anhang A1). Zudem entschieden wir uns für zwei kreisrunde Hindernisse in blau und grün mit einem Durchmesser von 18

cm. Um optimale Voraussetzungen für die Erkennung des Balls, sowie der Hindernisse zu ermöglichen wurde die Bahn weiß.

3.2.2. Der Schläger

Zu Beginn nutzten wir einen kleinen Minigolfschläger aus Plastik. Der NAO konnte diesen allerdings nicht immer fest halten, besonders beim Ausholen und Schlagen verrutschte der Schläger in der Hand des NAOs. Daraufhin ließen wir von der mechanischen Werkstatt der Hochschule Bochum einen Schläger aus Aluminium mit einem zusätzlichen Griff anfertigen(siehe Anhang A2).

3.2.3. Der Ball

Wir testeten über Monate verschiedene Bälle, sowohl in Hinblick auf Größe als auch auf Material. Die Wahl der Farbe fiel direkt auf rot, da der NAO bereits eine "Redball Detection"besitzt. Wir entschieden uns für einen massiven Gummiball mit einem Durchmesser von 6 cm.

4. Installation

4.1. Installation von ROS

Für das Projekt entschieden wir uns für ROS Indigo, die aktuellste Version mit Langzeitunterstützung. Die Installation erfolgte auf Ubuntu 14.04 Als Grundlage nutzten wir das offizielle Installationstutorial des ROS Wikis. Zunächst mussten Pakete von packages.ros.org akzeptiert werden. Der Befehl dazu lautet:

$$sudo\ sh\ -c\ 'echo\ "deb\ http://packages.ros.org/ros/ubuntu \tag{1}$$

$$\$(lsb\_release\ -sc)\ main"\ >\ /etc/apt/sources.list.d/ros-latest.list' \tag{2}$$

Außerdem muss noch die Schlüssel ID zur Authentifizierung der Pakete vom Server heruntergeladen werden:

$$sudo\ apt-key\ adv\ --keyserver\ hkp://ha.pool.sks-keyservers.net \tag{3}$$

$$--recv-key\ 0xB01FA116 \tag{4}$$

Mit dem Befehl

$$sudo\ apt-get\ install\ ros-indigo-desktop-full \tag{5}$$

wird die komplette Installation gestartet, die folgende Komponenten enthält: ROS, rqt, rviz, robot-generic libraries, 2D/3D Simulatoren, Navigation und Pakete für die 2D und 3D Wahrnehmung.

Bevor ROS nun genutzt werden kann, muss rosdep initialisiert werden. Dies erlaubt die einfache Eingabe von...

$$sudo\ rosdep\ init \tag{6}$$
$$rosdep\ update \tag{7}$$

Um die setup.bash Datei nicht bei jedem Öffnen einer neuen Shell sourcen zu müssen, empfiehlt es sich, diesen Befehl in die .bashrc-Datei zu kopieren. Dafür nutzten wir folgende Befehle:

$$echo\ \text{“}source\ /opt/ros/indigo/setup.bash\text{”}\ >>\sim /.bashrc \tag{8}$$
$$source\ \sim /.bashrc \tag{9}$$

4.2. Erstellen eines Workspaces

Nach der Installation ist es nötig, einen Workspace zu erstellen, in dem gearbeitet wird.

$$\$\ mkdir\ -p\ \sim /catkin\_ws/src \tag{10}$$
$$\$\ cd\ \sim /catkin\_ws/src \tag{11}$$
$$\$\ catkin\_init\_workspace \tag{12}$$

4.3. Tutorial

Auf der ROS-Wikiseite `http://wiki.ros.org/ROS/Tutorials` finden sich viele Tutorials, welche den Umgang mit dem ROS Filesystem, Packages und Nodes, sowie das Schreiben von Publishern und Subscribern in C++ oder Python beschreiben. Diese wurden von uns zur Einarbeitung bearbeitet.

4.4. Installation von RVIZ

Die Installation von RViz war bereits im Installationspaket von ROS enthalten und musste hierfür nicht noch einmal gesondert installiert werden. Allerdings musste noch zusätzlich das ganze NAO-Paket, wie z.B. Treiber u.Ä. installiert werden. Über die Seite `http://wiki.ros.org/nao/Tutorials/Installation` gelangt man zu einem Tutorial, in dem alles Nötige erklärt wird. Nachdem wir dieses Tutorial durchgearbeitet hatten, war die Installation aller wichtigen Komponenten fast fertiggestellt. Um ein schon fertiges Modell des NAO-Roboters benutzen zu können, mussten wir noch ein zusätzliches Paket herunterladen und dazu einer Lizenzbestimmung zustimmen. Dieses wird über den Befehl

$$sudo\ apt - get\ install\ ros - indigo - nao - meshes \tag{13}$$

heruntergeladen und installiert.

4.5. Konfiguration von RVIZ für den NAO

Bei der Konfiguration von ROS für den NAO-Roboter sahen wir uns einigen Problemen gegenübergestellt. Zunächst war das Tutorial, welches auf der ROS-Wikiseite zu finden ist, veraltet. Das Problem dabei war, dass dieses das einzige Tutorial war, das wir auffinden konnten. Deshalb mussten wir uns erst einmal eine ganze Weile mit RViz auseinander setzen. Nach dem Durchforsten einiger Foreneinträge und auch ein wenig Ausprobieren, fanden wir dann die richtigen Befehle, um eine ordentliche Simulation des NAO über RViz laufen zu lassen. Zunächst wird der Roscore gestartet:

$$cd\ \sim /catkin\_ws \tag{14}$$

$$roscore \tag{15}$$

Anschließend werden die Simulation und die dazugehörigen Treiber gestartet:

$$\sim /naoqi/naoqisdk2.1.4.13linux64/naoqi\ --verbose\ broker - ip\ 127.0.0.1 \tag{16}$$

$$roslaunch\ naoqi\_driver\_py\ naoqi\_driver.launch \tag{17}$$

Für den letzten Schritt bearbeiteten wir eine bereits vorhandene .launch-Datei so, dass RViz nach unseren Vorstellungen das Modell lädt. Das ganze startet man dann über folgenden Befehl:

$$roslaunch\ nao\_description\ display.launch \tag{18}$$

display.launch ist hierbei unsere editierte Datei. Damit haben wir einen simulierten NAO-Roboter, dem wir per Konsole Befehle erteilen können.

4.6. Einarbeitung in NAO-API

Nachdem wir dem NAO-Roboter ein paar Befehle direkt über die Konsole erteilt hatten, ging es nun darum, ihn komplexere Befehlssätze ausführen zu lassen. Dazu mussten wir uns allerdings zunächst in die NAO-API einarbeiten, welche unter folgender Adresse zu erreichen ist:
`http://doc.aldebaran.com/1-14/naoqi/index.html`
Zusätzlich dazu schauten wir uns auch einige Code-Beispiele zu wichtigen Themen, wie z.B. der Ballerkennung an, um die Funktionsweise der einzelnen Befehle zu verstehen.

5. Programmierung

5.1. Erste Schritte

Nachdem wir die Installation und Konfiguration durchgeführt hatten, testeten wir zunächst einige Grundfunktionen anhand der Visualisierung mit RViz. Hierbei nutzten wir die Konsole um Befehle direkt zu übermitteln. Dabei stellten wir fest, dass einige Befehle in der Visualisierung nicht umgesetzt werden konnten, da das genutzte Modell in RViz nicht vollständig war. Im Anschluss führten wir die Kommandos am Roboter aus,

was weniger Probleme verursachte, als wir uns vorgestellt hatten. Dennoch mussten wir uns noch viel mit der NAO-API auseinander setzen, um die benötigten Schnittstellen zu finden.

Mit dem Befehl

$$rosrunrvizrviz \tag{19}$$

wird RViz gestartet und mit

$$rostopicpub - 1/cmd\_velgeometry\_msgs/Twist \tag{20}$$

$$'linear : x : 1.0, y : 0.0, z : 0.0, angular : x : 0.0, y : 0.0, z : 0.0' \tag{21}$$

beginnt der virtuelle NAO sich vorwärts zu bewegen. Die linearen Variabeln geben die Richtung, angular die ???? an.

5.2. Schlagbewegung

Um die Schlagbewegung durchzuführen testeten wir zunächst die ungefähren Bewegungen in der Simulation. Am Roboter führten wir danach die Feinabstimmung durch und nutzten dabei einen kleinen Spielzeuggolfschläger. Da sich in diesem Abschnitt herausstellte, dass das Ausführen unter ROS sehr viel Zeit beanspruchte, verzichteten wir im Folgenden temporär auf ROS und nutzten stattdessen einfache Python-Skripte, um Änderungen schneller austesten zu können.

5.3. Ballerkennung

Da der NAO bereits eine Funktion für die Erkennung eines roten Balls besitzt, nutzten wir diese direkt. Hierfür besorgten wir uns einen roten Plastikball und führten erste Versuche durch, um die Funktion für unsere Zwecke einsetzen zu können. Allerdings mussten wir hierbei einige Anpassungen vornehmen, weil die Kamera des NAO nur einen bestimmten Sichtbereich hat und der Bereich direkt vor dem Roboter nicht mehr sichtbar ist. Da wir deshalb das letzte Stück "blind" steuern mussten, kam es hierbei auch häufig zu Ungenauigkeiten beim Schlag. Daraus resultierte eine schwer zu kalkulierende Ballbewegung, für die wir auch erstmal keine Lösung fanden. Wir nahmen uns vor, diese Problemstellung zu einem späteren Zeitpunkt noch einmal zu bearbeiten und uns zunächst mal um die Erkennung des Zieles zu kümmern.

5.4. Zielen

Um auf das Ziel und um Hindernisse herum zu spielen, nahmen wir uns eine Funktion des NAOs zu Hilfe, die anhand eines farbigen Kreises die Koordinaten dieses Kreises ermittelt. Das Koordinatensystem bezieht sich hierbei auf den Sichtbereich des Roboters. Die X-Achse hat einen Bereich von 0-160, die Y-Achse von 0-120, der Nullpunkt liegt oben links. Da das Kamerabild vom NAO kommt, liegt sein Referenzpunkt unten in der Mitte, also bei (80,120). Hat man nun diese zwei Punkte gegeben, lässt sich ein Dreieck

bilden, dessen Hypotenuse die Verbindung zwischen NAO und Ziel darstellt. Da sich alle Längen des Dreiecks mithilfe der Längen der Achsen und den Koordinaten der Punkte errechnen lassen, kann auch der Winkel zwischen NAO und diesem Ziel berechnet werden. Nun musste nur noch eine Kreisbewegung programmiert werden, die den Roboter gleichmäßig um den Ball führt. Schließlich kann durch Messungen der Drehdauer eine Funktion bestimmt werden, die anhand eines Winkels die Zeit zurückgibt, die benötigt wird, damit der Roboter sich um diesen Winkel dreht.

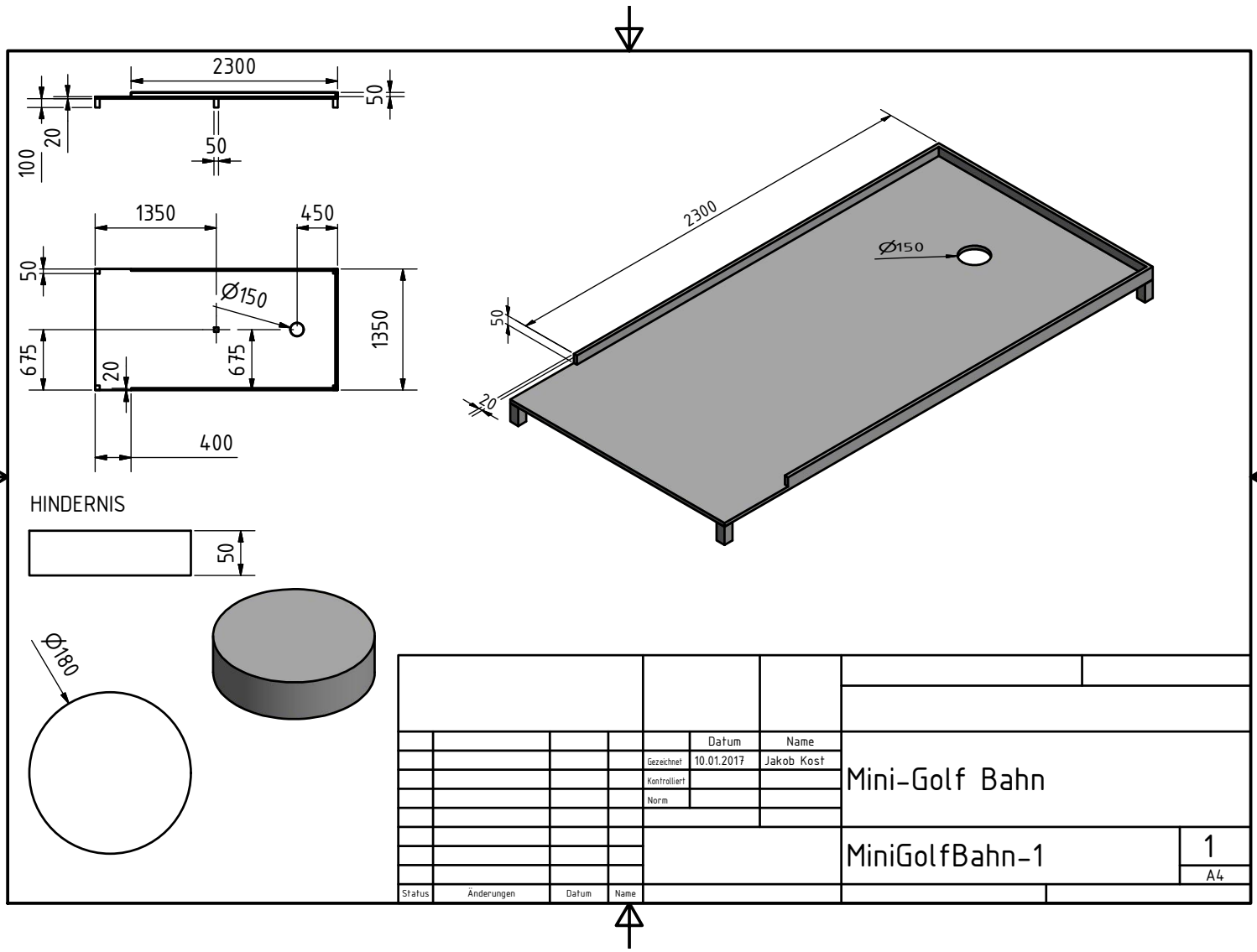
```
def turnToTarget(target):

    # Calculate angle to Target
    if(target[0] <= 80):
        degree = math.degrees(math.atan(
                (120-target[1])/(80-target[0])))
    else:
        degree = 90 + math.degrees(math.atan(
        (target[0]-80)/(120-target[1])))

    # Calculate time to turn
    global turnTime
    turnTime = 0.079318*degree + 7.901573

    # Turn
    motion.move(0,0.02,-0.084)
    time.sleep(turnTime)
    motion.move(0,0,0)
```

5.5. Erkennen der Hindernisse

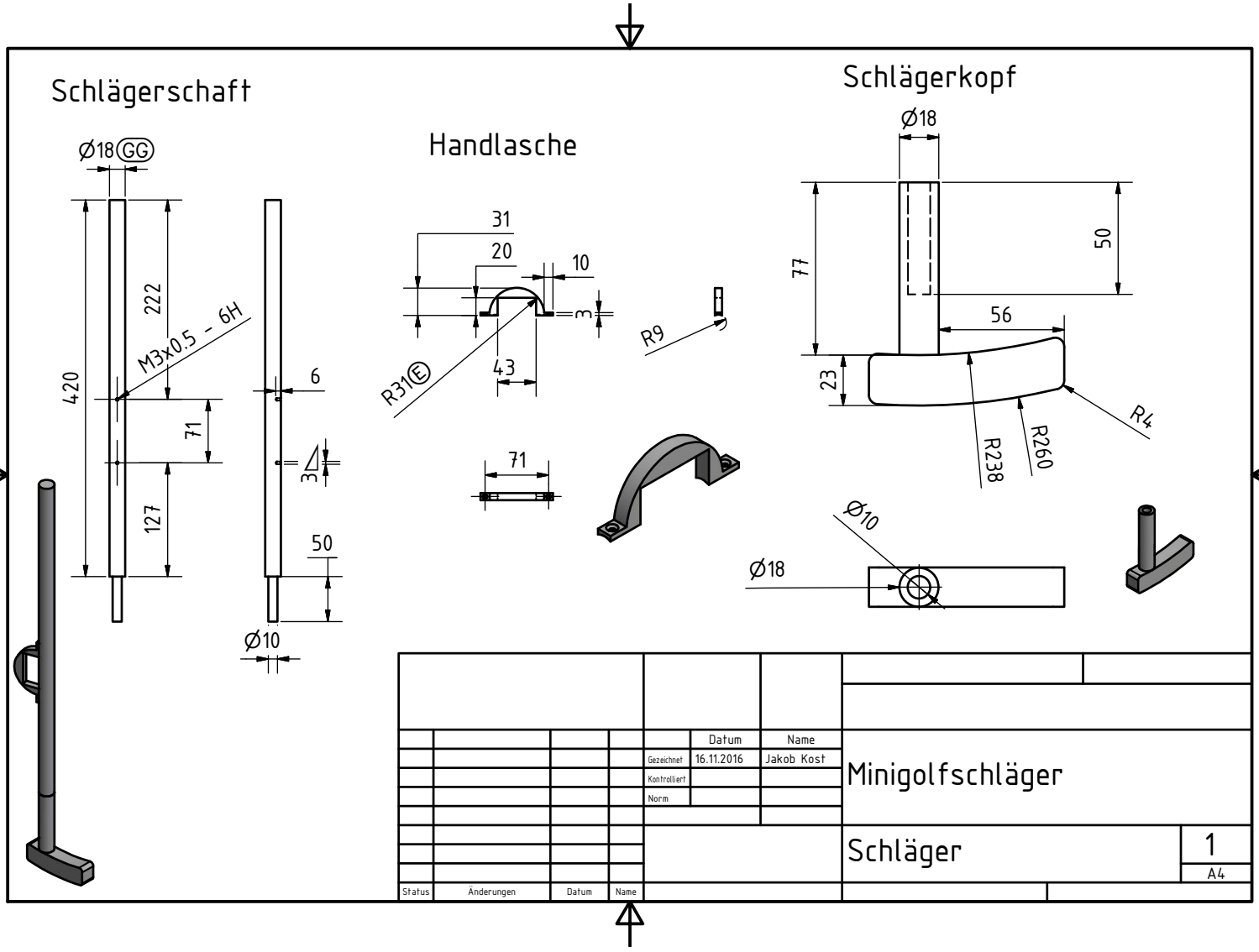
Für die Erkennung der Hindernisse nutzen wir die selbe Methode wie für die Zielerkennung. Für die Ermittlung der Positionen wurde die Bahn in drei Abschnitte geteilt: rechts, mitte und links. Da sich das Sichtfeld des NAOs wie beim Menschen auch nach hinten hin verzerrt, musste zunächst eine Funktion zur Ermittlung der Abschnitte erstellt werden.

A. Anhang

A.1. Minigolfbahn



A.2. Schläger


Schlägerschaft
Ø18 GG
222
M3x0.5 - 6H
420
71
127
6
3
50
Ø10
Handlasche
31
20
10
3
R9
R31
43
71
Schlägerkopf
Ø18
77
50
56
23
R4
R238
R260
Ø10
Ø18
Datum
Name
Gezeichnet
16.11.2016
Jakob Kost
Kontrolliert
Norm
Minigolfschläger
Schläger
1
A4
Status
Änderungen
Datum
Name