

Bearbeitungsbeginn: 01.09.2023

Vorgelegt am: 28.02.2024

## **Thesis**

zur Erlangung des Grades

**Bachelor of Arts**

im Studiengang Medienkonzeption

an der Fakultät Digitale Medien

**Simon Eugen Josef Daiber**

**Matrikelnummer: 265607**

**Erstellung eines prozeduralen Animationssystems für  
Charaktere mit Kettenlaufwerken, im 3 Dimensionalen Raum,  
am Beispiel von Wall-E.**

Erstbetreuer: Christoph Müller

Zweitbetreuer: David Lochmann

## **Inhaltsverzeichnis**

Abbildungsverzeichnis	3
Tabellenverzeichnis	3
1.Einführung in das Thema	4
1.1 Einleitung	4
1.2 Problemstellung	4
1.3 Zielsetzung	5
1.4 Struktur der Arbeit	5
2.Analyse des Charakters Wall-E	5
2.1 Bestandteile	5
2.2 Design und Eigenschaften	6
2.2.1 Körper und Beine	6
2.2.2 Räder und Ketten	6
3.Grundlagen der Animation	7
3.1 Framebasierte Animation	7
3.2 Prozedurale Animation	8
4.Entwicklung eines prozeduralen Animationssystems	9
4.1 Anforderungen an die Software	9
4.2 Auswahl der 3D-Software	9
4.3 Grundlegende Ansätze zur Umsetzung	10
4.3.1 Physik basiert	10
4.3.2 Constraint basiert	11
4.3.3 Verwendetes System	12
5.Animation und Rigging in Blender	12
5.1 Empty und Mesh Objekte	12
5.2 Modifier und Constraints	12
5.3 Hierarchy / Parenting	13
5.4 Lokale und Globale Koordinaten	13
5.5 Inverse Kinematics	13

6.Umsetzung des Animationssystems	14
6.1 Grundlegender Aufbau	14
6.2 Bewegungs und Rotationskontrolle des Rigs	14
6.3 Aufbau der Kettensysteme	17
6.3.1 Bodenerkennung	17
6.3.2 Unterer Teil	17
6.3.3 Oberer Teil	19
6.3.4 Ketten	20
6.3.5 Rotation von Rädern und Kette	21
6.4 Verbindung von Körper und Kettensystemen	22
7. Rig Control	23
7.1 Grundlagen	23
7.2 Bewegung und Rotation	24
7.3 Veränderung der Kettensysteme	25
7.4 Positionierung von Körper und Ketten	26
7.5 Federung	26
7.6 Kontrollparameter	27
8.Test und Evaluation	28
8.1 Ergebnisse	28
8.2 Limitationen und Herausforderungen	29
8.3 Weiterentwicklungsmöglichkeiten	29
9.Fazit	30
Literaturverzeichnis	31
Anhang	33
a.Nutzungshinweise ohne Python oder Controller	33
b.Installationshinweise Python Scripts	33
c.Quellcode Wall-E Script	33
Versicherung über redliches wissenschaftliches Arbeiten	39

## Abbildungsverzeichnis

Abbildung 1: Wall-E (Disney, 2008)	5
Abbildung 2: Wall-E Körper und Beine (Disney, 2008)	6
Abbildung 3: Wall-E Ketten und Räder (Disney, 2008)	7
Abbildung 4 : Aufbau Bewegung und Rotation 1 (Eigene Darstellung)	14
Abbildung 5 : Cube für Bewegung und Rotation (Eigene Darstellung)	15
Abbildung 6 : Aufbau Bewegung und Rotation 2 (Eigene Darstellung)	16
Abbildung 7: Bodenerkennung mit Shrinkwrap Modifier und Constraint (Eigene Darstellung)	17
Abbildung 8 : Konstruktion der unteren Räder 1 (Eigene Darstellung)	18
Abbildung 9 : Konstruktion der unteren Räder 2 (Eigene Darstellung)	18
Abbildung 10 : Konstruktion der oberen Räder (Eigene Darstellung)	19
Abbildung 11 : Konstruktion der Kette (Eigene Darstellung)	20
Abbildung 12 : Beine 1 (Disney, 2008)	22
Abbildung 13 : Armature (Eigene Darstellung)	23
Abbildung 14 : XBOX-One Controller (Evan Amos, 2014)	24
Abbildung 15 : Controller Input (Eigene Darstellung)	24
Abbildung 16 : Parameter (Eigene Darstellung)	24
Abbildung 17 : Code for Movement on X (Eigene Darstellung)	25
Abbildung 18 : Code for Sidewheels (Eigene Darstellung)	26
Abbildung 19 : Code for Up and Down (Eigene Darstellung)	26
Abbildung 20 : Code for moving Tracks (Eigene Darstellung)	26
Abbildung 21 : Code for Bounce (Eigene Darstellung)	27
Abbildung 22 : Code for Parameters (Eigene Darstellung)	28
Abbildung 23 : QR-Code für Video zu Funktionen (Eigene Darstellung)	28

## Tabellenverzeichnis

Tabelle 1: Berechnung Radrotationen	22
-------------------------------------	----

## 1. Einführung in das Thema

### 1.1 Einleitung

“Previz onset is an emerging discipline in film production to achieve live previsualization. It fills the gap between the storyboard or the full 3D preview sequence and the final image.” So beschreiben de Goussencourt et al. (2015) den Prozess der Live Previsualization. Hierbei werden geplante Filmszenen nicht nur in einem Storyboard mit Skizzen und Zeichnungen statisch festgehalten, sondern können in einer digitalen Umgebung anhand verschiedener Parameter und Quellen angepasst werden. Durch die Visualisierung komplexer Szenen können Regisseure den Ablauf von Szenen bereits vor der eigentlichen Umsetzung beurteilen und Änderungen vornehmen. Je nach Medium werden dafür 3D/2D Assets mit den Kameraaufnahmen kombiniert oder eine Szene komplett aus Assets erstellt. (vgl. de Goussencourt et al. 2015)

### 1.2 Problemstellung

Die zur Erstellung dieser Previsualizations werden je nach Szene verschiedene Assets benötigt. Statische Objekte können dabei als einfacher Blockout oder auch bereits fertiges Objekt in die Szene integriert werden. Bei der Erstellung von Charakteren müssen i.d.R. Rigs zur Steuerung dieser Charaktere erstellt werden. Je nach digitaler Umgebung gibt es dafür vorgefertigte Rigs, z.B. Meta Humans in Unreal Engine 5 oder das Rigify Addon für Blender.

Mit diesen Tools können häufig benötigte Rigs wie Menschen, Tiere oder Autos direkt auf ein Mesh angewandt werden, wodurch innerhalb von Minuten ein vollständig nutzbares Rig entstehen kann.

Wird ein individuelles Rig benötigt, müssen bestehende Rigs abgeändert oder ein komplett neues für das Objekt erstellt werden.

### 1.3 Zielsetzung

Ziel dieser Arbeit ist es, ein Rig zu erstellen, welches für eben solche Previsualizations in einer 3D Umgebung genutzt werden kann und nicht durch eine der oben genannten bereits vorhandenen Standardlösungen abgedeckt ist. Die Wahl des Objekts fiel dabei auf den Charakter Wall-E, wobei vor allem das komplexe Kettensystem interessant ist. Durch den beschränkten Umfang einer Bachelorarbeit

wird vor allem auf die Kettenysteme und Auswirkungen von Bewegung auf den Charakter eingegangen.

## 1.4 Struktur der Arbeit

In Kapitel 2 werden die relevanten Bestandteile des Charakters Wall-E vorgestellt. In Kapitel 3 wird auf die Unterschiede von Framebasierter und Prozeduraler Animation eingegangen. Kapitel 4 zeigt grundsätzliche Überlegungen zu Software und Umsetzung. In Kapitel 5 werden für Blender spezifische (und auch allgemeine) Begriffe und Konzepte für Animation und Rigging erklärt. In Kapitel 6 wird die Umsetzung / Konstruktion der einzelnen Teile des Animationssystems vorgestellt. Kapitel 7 beschreibt die Umsetzung der Kontrollmöglichkeiten des Rigs mithilfe eines Controllers und Python Script. In Kapitel 8 werden die Ergebnisse gezeigt und bewertet, sowie auf Limitationen eingegangen. Abschließend bildet Kapitel 9 das Fazit und gibt einen Ausblick in mögliche weitere Entwicklungen des Systems.

## 2. Analyse des Charakters Wall-E

### 2.1 Bestandteile

Diese Arbeit bezieht sich auf den Charakter Wall-E aus dem 2008 erschienenen Disney Film WALL-E. Der Charakter besteht aus den folgenden grundlegenden Teilen:

1. Körper
2. Hals + Augen
3. Arme + Hände
4. Beine
5. Radsysteme
6. Ketten

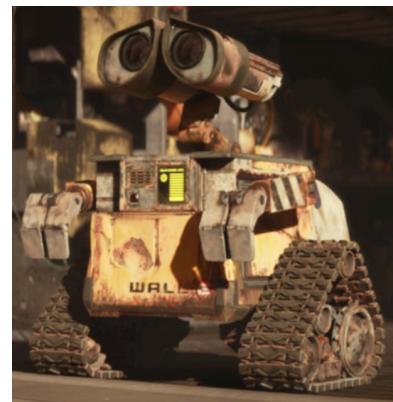


Abbildung 1: *Wall-E*  
(Disney, 2008)

Da der Fokus dieser Arbeit auf den Rad/Ketten Systemen liegt, werden diese nachfolgend näher analysiert.

## 2.2 Design und Eigenschaften

Grundsätzlich ist Wall-E ein Charakter in einem Animationsfilm und entsprechend sind seine Bewegungsabläufe und auch das Charakterdesign häufig überzeichnet und unrealistisch. So hat z.B. Fliehkraft, erzeugt durch Bewegungen, einen sehr starken Einfluss auf seinen Körper. Ein weiteres Beispiel findet sich auch in 2.2.1.

### 2.2.1 Körper und Beine

Der Körper von Wall-E ist ein grob rechteckiger Quader, welcher mittig zwischen den Rädern sitzt. Räder und Körper sind über mechanische Beine miteinander verbunden. Der Körper erfüllt im Film mehrere Aufgaben, welche rein optisch nicht möglich sein sollten. Zum einen befindet sich hier eine Müllpresse sowie die Elektronik und Solarpanele von Wall-E. Gleichzeitig kann Wall-E seine Arme, Augen, Beine und Ketten in den Körper ziehen und verwandelt sich so in einen Quader. Dieses Platzproblem wird später auch Auswirkungen auf die Animation von Beinen und Ketten haben.

Die Beine erlauben dem Körper, sich nach oben zu bewegen. Im Film bestehen diese aus mindestens 3 sichtbaren Teilen. Die Beine sind dabei sehr simpel gehalten und sind im Endeffekt abgerundete Quader ohne sichtbare mechanische Elemente.



Abbildung 2: Wall-E Körper und Beine (Disney, 2008)

### 2.2.2 Räder und Ketten

Die Radsysteme bestehen aus zwei Teilen. Unten hinten befindet sich ein großes Treibrad, welches die Kette antreibt. Vorne unten befindet sich ein mittleres Rad, über welches die Kette läuft. Dazwischen befindet sich eine Platte, welche über den Boden mitläuft und dahinter mit dem Bein verbunden ist. An der Verbindung mit den

Beinen befindet sich gleichzeitig eine Verbindung zu den beiden Rädern die den oberen Teil des Kettensystems bilden. Diese sind an einer rotierbaren Verbindung befestigt.

Die Verbindung nach oben kann von Wall-E aus oder eingefahren werden. außerdem können die oberen Räder zusammen mit der Verbindung rotiert werden um mehrere verschiedene Kettenzustände zu erzeugen.



Abbildung 3: Wall-E Ketten und Räder (Disney, 2008)

### 3. Grundlagen der Animation

In diesem Kapitel werden die Unterschiede zwischen Framebasierte und Prozeduraler Animation erläutert.

#### 3.1 Framebasierte Animation

Framebasierte Animation beruht auf der Nutzung von Frames. Ein Frame ist hierbei die Zeiteinheit, in welcher die Animation abläuft. Um ein Objekt zu animieren, werden alle für die Animation benötigten Daten für jeden in die Animation integrierten Frame gespeichert. Zur Erleichterung werden Keyframes verwendet, welche an wichtigen Stellen gesetzt werden und die Veränderungen der Frames zwischen ihnen durch Interpolation berechnen. Das Konzept wird nachfolgend an einem einfachen Beispiel demonstriert.

Objekt A bewegt sich gleichmäßig von 0 zu 1 auf der x-Achse in 100 Frames, bei Frame 0 und 100 wird jeweils ein Keyframe gesetzt. Für jeden Frame bewegt sich das Objekt nun um 0.01 auf der x-Achse. (vgl. Paquette 2013 S.241)

Bei einer komplexen Animation mit vielen Veränderungen werden entsprechend viele Keyframes benötigt. Sind diese einmal gesetzt, ist es schwer, die Szene nachträglich zu verändern. Händische Animation, wie z.B. die Bewegung eines Fahrzeugs, kann außerdem unnatürlich wirken. Um solche Probleme zu beheben, sind möglicherweise mehrere Änderungen der Animation nötig. (vgl. Gowanlock 2021 S.4)

### 3.2 Prozedurale Animation

Prozedurale Animationen sind solche, die auf ihre Umgebung reagieren und, anders als die Framebasierte, keinen direkten Input eines Künstler benötigen, sondern auf mathematischen Berechnungen und Algorithmen beruhen. Ein häufiges Beispiel ist die Berechnung von Flüssigkeiten in Fluid Simulations. Hierbei bewegen sich Flüssigkeiten abhängig von Parametern wie Größe der Oberfläche oder Unruhe, welche Wellen erzeugen. Um eine realitätsnahe Simulation zu erzeugen, werden dabei i.d.R. Minimal- und Maximalwerte definiert, während zufällige Zahlen im Raum dazwischen die eigentlichen Werte angeben. Ein weiteres Beispiel sind Ragdoll Physics, welche realitätsnahe Körper durch die Verbindung von Objekten mit Gelenken und verschiedenen Beschränkungen erzeugen. Insbesondere bei mechanischen Animationen werden häufig Rigid Body Simulations verwendet, welche aus verschiedenen beweglichen Objekten ein Simulationssystem erstellen. (vgl. Gowanlock 2021 S.6-10, Zucconi 2017a)

Gegenüber der Framebasierten Animation zeichnet sich die Prozedurale mit einer flexibleren Arbeitsweise aus. Die Planung des genauen Ablaufs der Animation wird weniger wichtig, da diese ja prozedural generiert ist. Bevor eine Animation erstellt werden kann, muss jedoch zuerst das passende Programm oder Algorithmus programmiert werden.

## 4. Entwicklung eines prozeduralen Animationssystems

### 4.1 Anforderungen an die Software

Funktionale Anforderungen:

- "Klassische" Animationen mithilfe von Framebasierten Methoden
- Unterstützung von Rigid Body / Physics Simulation
- Scripting / Coding Interface

Support:

- durch Community

- durch Hersteller

Informationsressourcen

- Community

- Hersteller

### 4.2 Auswahl der 3D-Software

Für diese Arbeit wurde die 3D-Software "Blender" verwendet, welches ein OpenSource Programm ist. Der Hersteller "Blender Foundation" bezeichnet Blender als "The Free and Open Source 3D Creation Suite". Im nachfolgenden werden die oben genannten Kriterien anhand von 3 potenziellen Programmen erläutert.

Die funktionalen Anforderungen werden von mehreren bekannten Programmen erfüllt. Zu nennen sind hier vor allem die von Autodesk entwickelten Anwendungen Maya, sowie 3ds Max. Im Bereich Scripting verwendet Maya die Maya Embedded Language, 3ds Max verwendet MaxScript. Beide Skriptsprachen wurden speziell für diese Anwendungen entwickelt. Blender verwendet die allgemeine Programmiersprache Python, welche über eine API integriert ist. (vgl. Kumar 2022, Moioli 2022, Raju 2019)

Beim Support setzt Blender vor allem auf die Zusammenarbeit der Community auf Plattformen wie Reddit oder Discord. Max und Maya bieten ebenfalls dedizierte Foren, auf denen sich die Community austauschen kann. Je nach gekauftem Paket gibt es auch mindestens die Möglichkeit eines Telefonats mit einem Autodesk-Mitarbeiter. Bei Verwendung einer Educational License, entfällt diese Möglichkeit jedoch.

Bei Informationsressourcen unterscheiden sich Blender, Max und Maya vor allem zwischen der Bereitstellung durch Community und Hersteller. Während Blender ein kostenloses Open Source Projekt ist, sind Max und Maya kostenpflichtige Programme. Dies führt dazu, dass es zu Blender zahlreiche (inoffizielle) Videoanleitungen auf Plattformen wie Youtube oder Vimeo gibt. Autodesk hingegen bietet für Max und Maya unter anderem offizielle Videokurse sowie Seminare durch Industrieexperten.

Die 3 oben behandelten Programme erfüllen demnach die genannten Anforderungen. Die letztliche Entscheidung für Blender erfolgte daher vor allem durch die bereits bekannte Umgebung durch den ins Studium integrierten Kurs "Computergrafik und 3D-Modellierung". Die verwendete Blender Version ist dabei die 4.0.

### 4.3 Grundlegende Ansätze zur Umsetzung

#### 4.3.1 Physik basiert

Physik basierte Systeme verwenden Rigid Bodies, um durch z.B. Reibung mit einer Oberfläche ein Rad entsprechend zu bewegen, ohne dass weitere äußerliche Einflüsse notwendig sind.

##### Vorteile:

Für Szenen, die ein hohes Maß an physikalischem Realismus erfordern ( hier z.B. Bewegung über den Untergrund), bieten Rigid Bodies durch Collision Objekte und vorgefertigte Constraints, wie Spring Controller, einfache Möglichkeiten zur Umsetzung. Durch ihre zufällige Natur können diese Systeme je nach Einstellung der Werte für z.B. Federung sehr unterhaltsame oder auch realitätsnahe Ergebnisse produzieren.

##### Nachteile:

Systeme, die auf Rigid Bodies basieren, sind fehleranfällig und nicht vollständig vorhersehbar. Insbesondere bei ungewöhnlichen Kombinationen, die von Standard

Objekten, wie Fahrzeugen mit 4 Rädern oder auch Panzern mit fester Körperposition abweichen, wird das System unübersichtlich und fehleranfällig.

Darüberhinaus erfordert das Rigid Body System die zwingende Aktivierung von Keyframes und verringert die Performance von Blender stark. Real Time Anwendungen im Editor sind damit, zumindest bei mehreren Physic Objekten, nicht umsetzbar.

Das physikbasierte System ist also realistisch, aber fehleranfällig. (vgl. Guevarra 2020 S.174 & S.265-269)

#### 4.3.2 Constraint basiert

Constraint basierte Systeme verwenden Constraints, um Objekte durch feste Beziehungen und Verhältnisse zu verbinden. z.B. wird ein Rad bei der Bewegung eines anderen Objekts rotiert.

##### Vorteile:

Vorhersehbarkeit der Ergebnisse und volle Kontrolle sind durch Constraints gegeben. So können Constraints den minimalen Abstand, Rotation, maximalen Abstand usw. eines Objekts kontrollieren, um Ergebnisse außerhalb des gewünschten Rahmens auszuschließen.

Die Performance des Editors ist selbst bei sehr komplexen Systemen noch ausreichend für eine Real Time Anwendung im Editor.

##### Nachteile:

Der durch die zufälligen Elemente generierte Realismus entfällt hier vollständig, da es keine im GUI integrierte Möglichkeit von zufälligen Elementen gibt. Auch die Umsetzung von Objekten wie Federungen ist nur in einem begrenzten Umfang möglich.

Das Constraint-basierte System ist das Gegenteil des Physikbasierten und weist eine hohe Fehlerresistenz, aber auch wenig Realismus auf. (vgl. Belec 2022 S.221)

### 4.3.3 Verwendetes System

Das verwendete System ist ein Constraint basiertes, welches den mangelnden Realismus durch die Verwendung von auf Python basierenden Scripts auszugleichen versucht.

## 5.Animation und Rigging in Blender

In diesem Kapitel werden allgemeine Grundlagen für Themen wie Modifier und Constraints behandelt. Die genaue Anwendung dieser Themen wird in Kapitel 6 bei der Umsetzung des Rigs behandelt.

### 5.1 Empty und Mesh Objekte

Ein Empty ist ein Objekt, welches über keine Geometrie, also kein Mesh, verfügt. Diese Empties werden demnach auch nicht von der Render Engine erfasst und können platziert werden, ohne dass sie im gerenderten Bild auftauchen. Da sie kein Mesh haben, können auf sie auch keine Modifier angewendet werden. Diese Eigenschaften sind sehr hilfreich, um die Effekte von bestimmten Constraints zu kontrollieren. (vgl. Moioli 2022 S.86)

### 5.2 Modifier und Constraints

Modifier modifizieren insbesondere visuelle Objektdaten wie Mesh oder die Form einer Curve. Dies ist wichtig, da hierbei nicht der Ursprung des Objekts verändert wird, sondern nur das zugehörige Mesh. Modifier sind nondestruktiv und können zu einem Modifier Stack gestapelt werden, was in dieser Arbeit vor allem beim Aufbau der Ketten verwendet wurde. (vgl. Guevarra 2020 S.76)

Anders als Modifier wirken Constraints sich nicht auf das Mesh aus, sondern auf die Eigenschaften des Objekts. Diese beinhalten z.B. Rotation, Position oder Skalierung. Wie auch Modifier können Constraint zu einem Constraint Stack gestapelt werden. (vgl. Guevarra 2020 S.200 - 213)

Da Modifier und Constraints unterschiedliche Aufgaben ausführen, werden diese häufig kombiniert. So legt der Shrinkwrap Modifier das Mesh einer Ebene über das Terrain während der Shrinkwrap Constraint die Rotation an dieses Terrain anpasst.

Die Anwendung der einzelnen Stacks erfolgt hierbei von oben nach unten. Das bedeutet, der unterste Modifier oder Constraint wird zuletzt angewendet und beeinflusst bei gleichem Einflussgebiet vorhergehende Constraints. Constraints verfügen jedoch auch über einen Influence Parameter, welcher den Einfluss mehrerer Constraints aufeinander ermöglicht. (vgl. Guevarra 2020 S.200 - 213, Moioli 2022 S.126)

### 5.3 Hierarchy / Parenting

Parenting ist der einfachste Weg, um eine Gruppe von Objekten zusammenzufassen. Hierbei übernehmen die Kinder die Transformationen (Rotation, Translation, Skalierung) des Parents. Die Kinder können sich hier unabhängig vom Parent bewegen. Bewegt sich das Parent, bewegen sich jedoch auch die Kinder mit einem entsprechenden Offset, um die veränderte Position mit einzukalkulieren. Die Effekte des Parenting können durch eine Verwendung von entsprechenden Transformation Constraints nachempfunden werden. Dies ist dann notwendig, wenn die Transformation eines Objekts von mehreren anderen gesteuert werden soll. (vgl. Raju 2019 S.51-75)

### 5.4 Lokale und Globale Koordinaten

Globale Koordinaten, oder auch Weltkoordinaten, bilden das grundlegende Koordinatensystem von Blender. Dieses kann nicht verändert werden. Lokale Koordinaten sind anpassbar und basieren auf einem Koordinaten Gizmo. Ist ein Objekt um 45° auf der globalen Z-Achse rotiert, rotiert auch der Gizmo und lokale Transformationen bleiben unverändert. (vgl. Blender Foundation 2024b)

### 5.5 Inverse Kinematics

Bei Inverse Kinematics ist ein Endpunkt und Startpunkt bekannt, für welchen nun passende Bewegungen eines Knochensystems erstellt werden müssen. Dies wird insbesondere bei mechanischen Armen eingesetzt. Zur Kontrolle des Systems werden Bone Constraints verwendet, welche ähnlich zu den in Kapitel 5.3 und 5.4 vorgestellten Constraints sind. Hierbei werden insbesondere Restriktionen von Transformationen und minimale sowie maximale Abstände verwendet, um unrealistische Ergebnisse wie Verdrehungen zu vermeiden. (vgl. Zucconi 2017b, Zucconi 2017a, Jazar 2007 S.263-296)

## 6.Umsetzung des Animationssystems

### 6.1 Grundlegender Aufbau

In den nachfolgenden Kapiteln wird der Aufbau der einzelnen Komponenten des Rigs näher vorgestellt. Der grundlegende Aufbau wird hier kurz dargestellt.

1. Ein Objekt wird auf x bewegt und auf z rotiert. Rechts und links von ihm sind Rechtecke die nach unten auf den Untergrund projiziert werden. Anhand deren Position erhält auch der Körper seine Position (Kapitel 6.2).
2. Auf diesen Rechtecken sind die Kettensysteme aufgebaut (Kapitel 6.3)
3. Die Verbindung zwischen Ketten und Körper erfolgt über eine Armature mit Inverse Kinematics (Kapitel 6.4)
4. Die Bewegungskontrolle und Reaktion auf die Umgebung wird durch ein Python-Script bestimmt. (Kapitel 6.5)

Wichtig ist hier, dass die lokale X-Achse, des später genannten HELP\_Main\_Movement, immer die Richtung von Wall-E anzeigt und auf dieser Vorwärts- und Rückwärtsbewegungen stattfinden. Für Rotationen wird die nach oben gehende globale Z-Achse verwendet.

### 6.2 Bewegungs und Rotationskontrolle des Rigs

Nachfolgend wird der Aufbau der Bewegungs- und Rotationskontrolle erklärt. Die Nummerierung in Abbildung 4 ist gleich mit der im Text verwendeten.

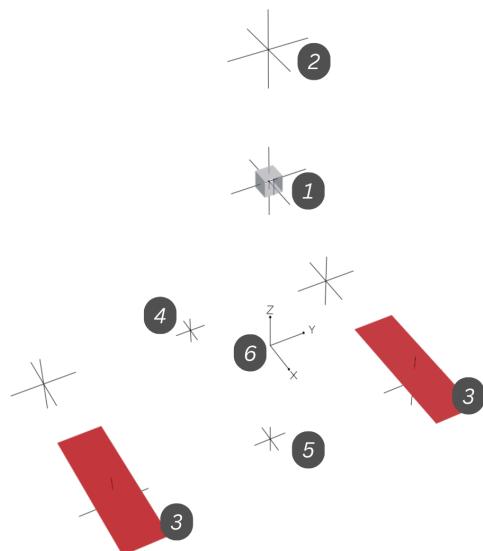


Abbildung 4: Aufbau Bewegung und Rotation 1 (Eigene Darstellung)

1.HELP\_Main\_Movement wird für Rotation und Translation des Rigs verwendet. Hierbei sind die in Kapitel 5 genannten Themen nützlich. Der Cube ist deshalb ein Mesh, weil ein Modifier für die Nutzung mit einer Curve angebracht werden kann. Da dabei aber nicht der Ursprung verändert wird, wird ein Empty verwendet, welches die Vertex Groups des Mesh nutzt, um die Position und Rotation des Cubes festzuhalten. Bei der Nutzung mit einem X-Box Controller kann der Cube weiterhin zur Positionsangabe verwendet werden.

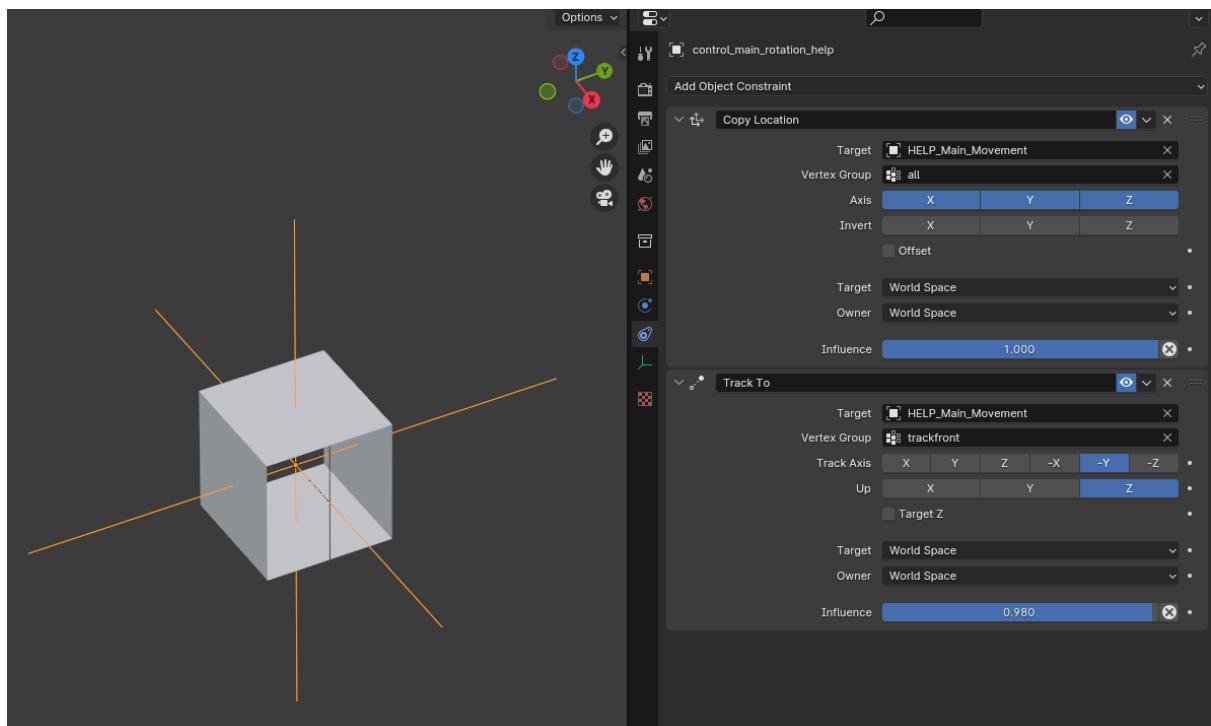


Abbildung 5: Cube für *Bewegung und Rotation* (Eigene Darstellung)

- 2.control\_main\_rotation erhält Position sowie Rotation auf Z von HELP\_Main\_Movement
- 3.shrinkwrap planes nutzen control\_main\_rotation als parent. Da diese mit dem Shrinkwrap Modifier auf der z achse projiziert werden würden rotationen auf x und y das ergebnis verzerrten, daher rotiert control\_main\_rotation nur auf der z-Achse.
- 4.Jedes der Shrinkwrap Rechtecke hat vorne und hinten ein Empty. Zwischen dem hinteren Empty rechts und dem links wird das Empty control\_rotation\_back erzeugt. Dieses übernimmt, durch zwei Constraints, von rechts und links die Position und erhält die Rotation der Verbindungsgeraden zwischen diesen durch einen TrackTo Constraint. (siehe Abb. 6)

5.Zwischen den vorderen Empties rechts und links liegt ein Empty control\_rotation\_front welches den selben Aufbau wie control\_rotation\_back hat. Der Hauptunterschied ist, dass dieses vordere einen Tracking constraint zum hinteren hat. Durch diesen wird verhindert, dass dieses bei Höhenunterschieden zwischen den Rechtecken rotiert wird. Dies wird in Abbildung 6 verdeutlicht.

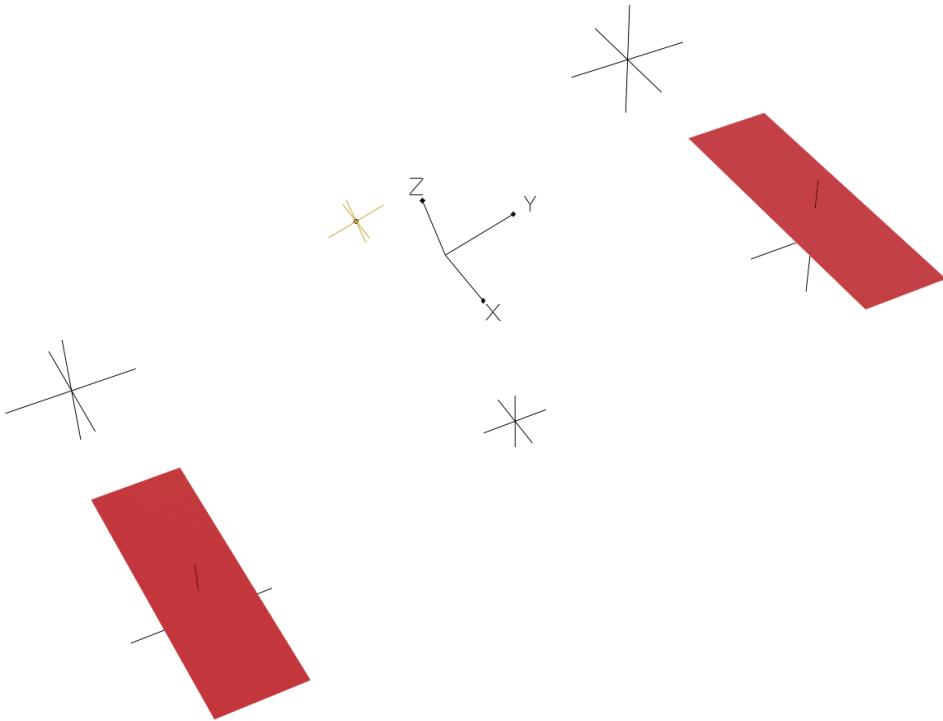


Abbildung 6: Aufbau Bewegung und Rotation 2 (Eigene Darstellung)

6.Durch diese Unterteilung von Rotationen und Translationen auf einzelne Empties kann nun das Empty control\_body, welches den Körper kontrolliert, diese unterteilte Transformationen mit Copy Location und Copy Rotation Constraints übernehmen.

Für die in Kapitel 7 beschriebene Steuerung wird später ein zusätzliches Empty auf der Position des control\_body sitzen um die Fliehkraft des Körpers zu simulieren. control\_body bleibt davon unberührt.

## 6.3 Aufbau der Kettensysteme

Im nachfolgenden wird der Aufbau der Kettensysteme beschrieben. Diese bestehen aus Bodenerkennung, unterer Radgruppe, oberer Radgruppe sowie der umspannenden Kette.

### 6.3.1 Bodenerkennung

Die grundlegende Idee ist ein Rechteck mithilfe des Shrinkwrap Modifiers und Constraints über eine, darunterliegende, Oberfläche zu legen. Hierfür wird ein Rechteck erstellt, welches dann in mehrere Abschnitte unterteilt wird. Um später die unteren Räder von Wall-E anzubringen, werden drei fixe Positionen auf diesem Rechteck bestimmt. Hierfür wurden drei Vertex Gruppen erstellt, welche jeweils eine kurze Kante beinhalten. Die Transformationen dieser Gruppen werden dann mithilfe eines Copy Transform Constraints an ein Empty übergeben. (Siehe Abb.7)

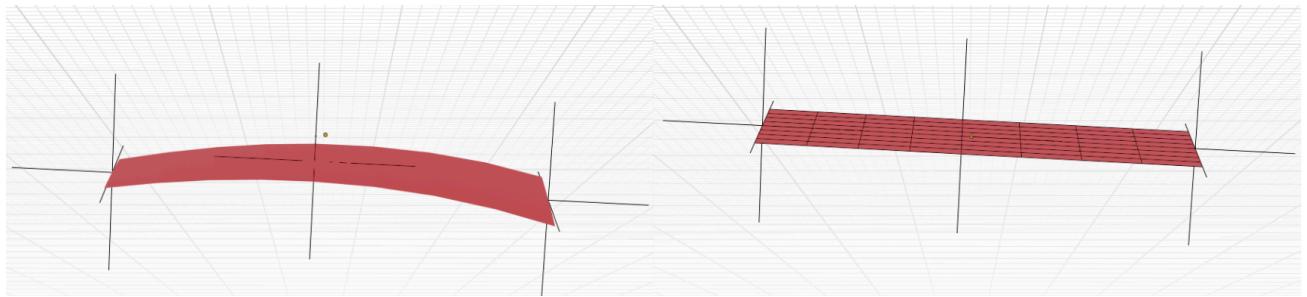


Abbildung 7: Bodenerkennung mit Shrinkwrap Modifier und Constraint (Eigene Darstellung)

Wichtig ist hier die in Kapitel 5.2 und 5.3 erwähnte Unterscheidung von Modifern und Constraints. Der Modifier projiziert das Rechteck, wie in der rechten Abbildung zu sehen, passgenau auf den Untergrund. Der Ursprung bleibt jedoch unverändert. Der Constraint hingegen projiziert den Ursprung auf den Untergrund. Das Mesh ist ohne Modifier jedoch weiterhin ein ebenes Rechteck.

### 6.3.2 Unterer Teil

Die unteren Räder verwenden die zwei Empties vorne und hinten auf dem Shrinkwrap-Rechteck, um ihre Position zu erhalten. Je nach Größe der Räder werden zwei Empties in einem festen Abstand zu den Shrinkwrap Empties platziert. Von einem der beiden Räder wird dann ein Pfeil-Empty mit dem Track To Constraint auf das andere Rad ausgerichtet. Dies entspricht der Situation in Abbildung 8.



Abbildung 8: Konstruktion der unteren Räder 1 (Eigene Darstellung)

Ein Verbindungsstück (in diesem Fall ein Cylinder) wird auf das rechte Rad gesetzt und der Ursprung zu diesem gesetzt. Mit einem Tracking Constraint wird dieser Cylinder nun auf das linke Rad ausgerichtet. Durch die Verschiebung des Ursprungs wird dieser Punkt nun als Basis für Transformationen verwendet. Ein weiteres Empty wird auf den Zylinder an einer passenden Stelle zwischen den Rädern gesetzt. Mithilfe dieses neuen Empties und dem bereits existierenden Empty auf der Shrinkwrap Platte wird nun die Bodenplatte bewegt und der darüber liegende Block bewegt. Der obere Block wird auf ein Pfeil Empty gesetzt welches das untere Empty trackt. Durch zusätzliche Constraints werden zu starke Bewegungen abgeschwächt.

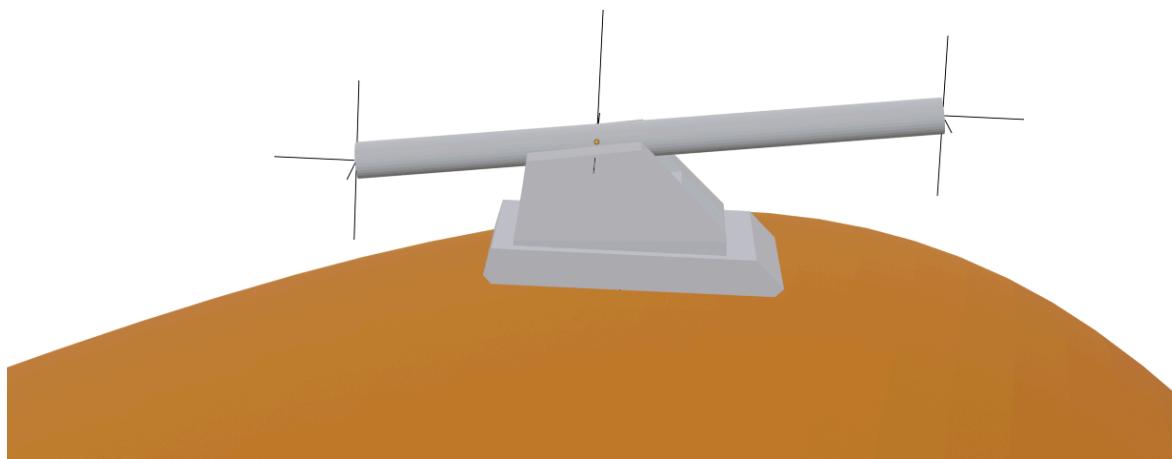


Abbildung 9: Konstruktion der unteren Räder 2 (Eigene Darstellung)

### 6.3.3 Oberer Teil

Der obere Teil des Kettensystems besteht aus zwei Rädern und einem Verbindungsstück zwischen diesen, welches als Parent dieser Räder dient. Das Verbindungsstück kann nach oben/unten sowie nach vorne/hinten bewegt werden. Außerdem kann es rotiert werden. Hinten am Verbindungsstück ist außerdem eine Verbindung zu den Rädern unten, welche beim Aus- und Einfahren die Bewegung eines Sliders simuliert. Zwei Constraints für maximale und minimale Distanz begrenzen die Bewegung des oberen Arms.

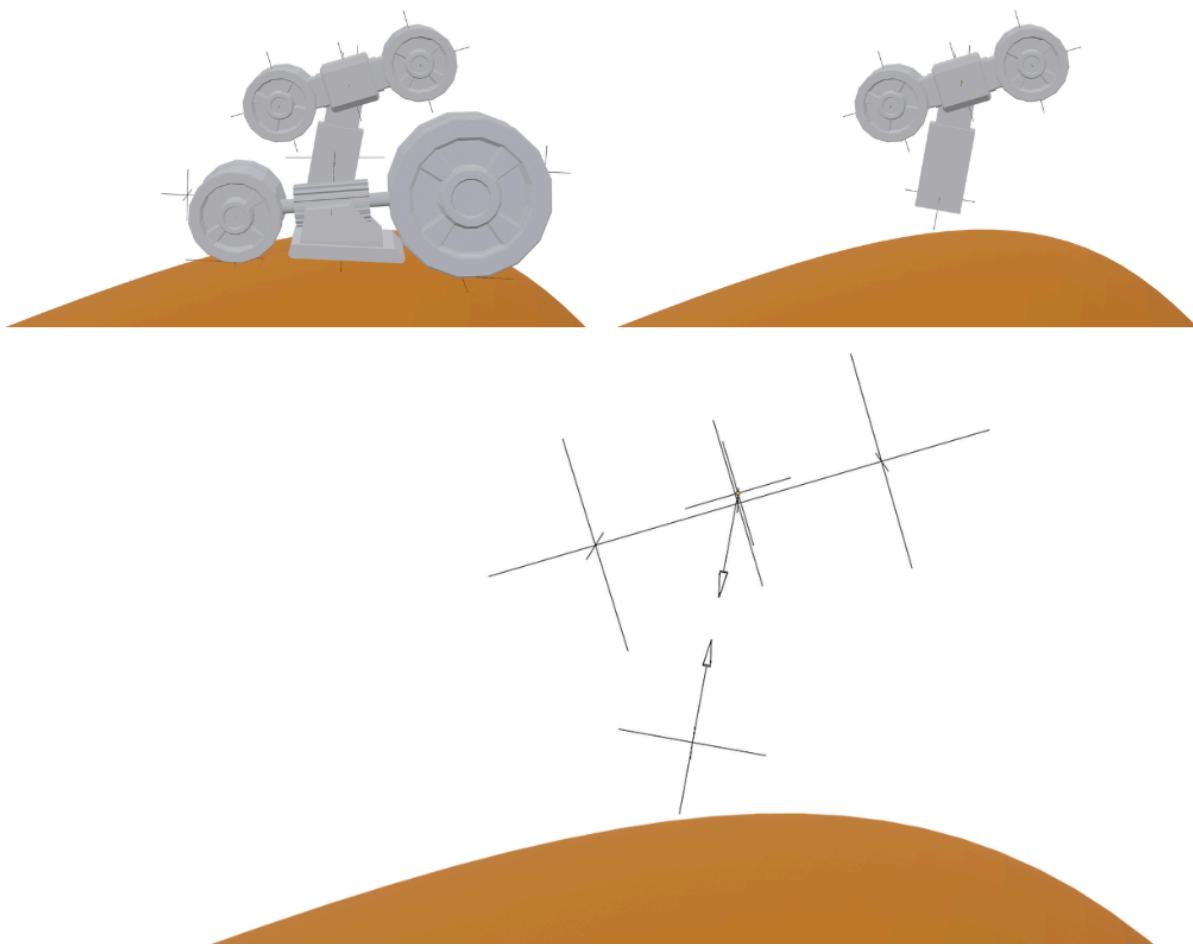


Abbildung 10: Konstruktion der oberen Räder (Eigene Darstellung)

### 6.3.4 Ketten

Für die Anbringung der Ketten werden die Empties verwendet, welche auch die Positionen der Räder bestimmen, sowie das dritte Empty, welches zwischen den unteren beiden Rädern liegt. Die Konstruktion der Kette wird nachfolgend beschrieben.

1. Erstellung zusätzlicher Empties in festem Abstand zu den Empties der Räder.
2. Erstellung einer Curve und eines Kettenstücks. Das Kettenstück wird mit einem Array und dann einem Curve Modifier versehen, wodurch sich Duplikate des Kettenstücks entlang der Curve aneinanderreihen.
3. Punkte der Curve werden mit Hook Modifiern an den neu erstellten Empties befestigt.

Die Abbildung 11 verdeutlicht die grobe Idee hinter der Konstruktion.

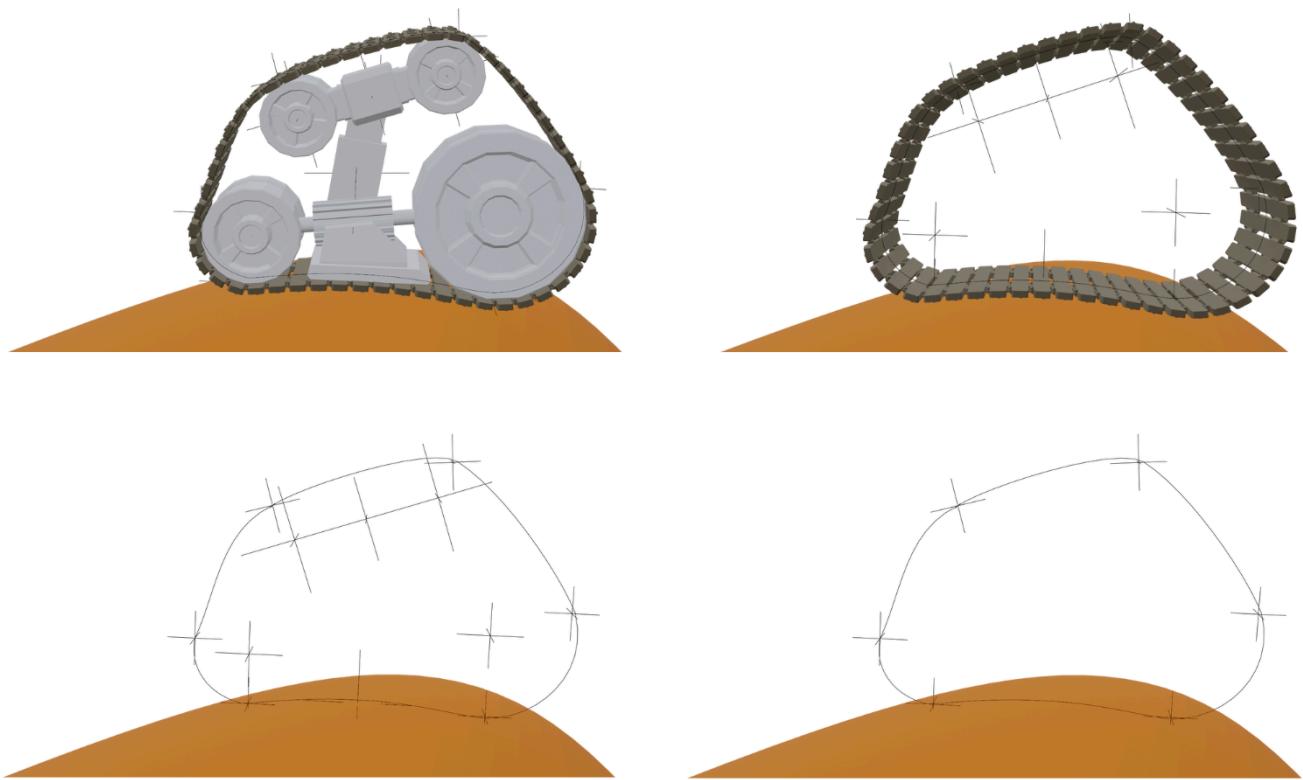


Abbildung 11: *Konstruktion der Kette* (Eigene Darstellung)

#### Probleme:

Ein Problem bei der Konstruktion ist die Tatsache, dass sich die Kette durch die mögliche Bewegung der oberen zwei Räder um mehrere Meter strecken können

muss. Bei der daraus resultierenden Verlängerung der Curve werden neue Stücke durch den Array Modifier neue Kettenstücke entlang der Curve hinzugefügt.

Die hier gezeigte Methode ist dann geeignet, wenn keine detaillierten Aufnahmen der Kette gemacht werden, oder der Stand der Kette so verändert werden kann, dass die Verlängerung an einer Stelle erfolgt, welche für die Kamera nicht sichtbar ist.

Ein Ansatz zur Konstruktion eines realistischeren Verhaltens wäre zum Beispiel die Verwendung einer physik-basierten Kette, bei welcher mehrere Stücke zusammengefasst und dann mit dehnbaren Verbindungsstücken eine Kette erstellt wird.

### 6.3.5 Rotation von Rädern und Kette

Das Kettenlaufwerk besteht aus vier Rädern und einer Kette, die um diese gelegt ist. Die Rotation dieser wird von einem Empty kontrolliert. Bei einer Aktivierung des Joysticks für die Translationskontrolle dreht sich das Empty "help\_Sphere\_left" auf der lokalen X-Achse. Hierfür wird der Transformation Constraint verwendet. Durch Extrapolation werden auch kleinere oder größere Werte linear zu den vorgegebenen Werten verarbeitet.

#### Kette

Bei einer Rotation des Empty um  $1^\circ$  bewegt sich die Kette um 0.1m auf ihrer lokalen Y-Achse. Diese Distanz ist Grundlage der Berechnung der Räder.

#### Räder

Da die Kette bei einer Rotation des Empty 0.1m zurücklegt und zugleich die Räder um einen bestimmten Wert rotiert werden, wird im folgenden die Rotation der einzelnen Räder für eine Distanz von 0.1 Metern berechnet.

1. Kreisumfang berechnen:  $U = \pi * \text{Durchmesser(Rad)}$
2. Achsumdrehungen für 0.1 Meter:  $A = 0.1 / U$
3. Benötigte Rotation für 0.1 Meter:  $R = A * 360$

Beispielhafte Berechnung für das große Rad:

1.  $U = \pi * 3,04 = 9,5504$
2.  $A = 0.1 / 9,5504 = 0,0105$

$$3. R = 0,0105 * 360 = 3,78^\circ$$

	Durchmesser	Kreisumfang	Achsumdrehungen	Rotation
Großes Rad	3,04	9,5504	0,0105	3,78°
Mittleres Rad	1,85	5,8119	0,0172	6,192°
Kleines Rad	1,35	4,241	0,0236	8,496°

Tabelle 1: Berechnung Radrotationen

Bei einer Rotation des Empty um 1° auf der lokalen X-Achse rotieren die Räder um den Wert der berechneten Rotation auf ihrer lokalen Y-Achse.

#### 6.4 Verbindung von Körper und Kettensystemen

Die “Beine” von Wall-E bestehen aus mindestens 3 einzelnen Segmenten welche an einer festen Position an der Unterseite des Körpers befestigt sind (siehe Abb. 12). In einzelnen Segmenten ist zu sehen, dass Wall-E sich vollständig in seinen Körper ziehen kann, die Beine müssen also ebenfalls vollständig im Körper untergebracht werden. An den Kettensystemen sind die Beine an einem festen Punkt der Schiene zwischen oberen und unteren Rädern angebracht. Die Grundaufgabe der Beine ist also auf alle Veränderungen von Körper und Beinen zu reagieren.



Abbildung 12: Beine 1 (Disney, 2008)

Diese Anforderungen werden mithilfe einer Armature und Inverse Kinematics umgesetzt. Der Grundaufbau ist in Abbildung 13 zu sehen. Das rechteckige Empty stellt dabei den Körper dar, die Empties oben rechts und links nutzen diesen als Parent. Auf der Innenseite des Kettensystems sitzt ebenfalls ein Empty welches als Parent der Armature dient. Die Knochen zwischen Körper und Kette werden über eine IK-Chain vom obersten Knochen aus gesteuert. Die Knochen haben dabei Constraints für Rotationen auf x und z, welche solche Rotationen verhindern. Um Überrotationen auf beim aus und einfahren zu verhindern, sind auch, für jeden Knochen individuell, Rotationen auf x zu einem gewissen Maße eingeschränkt.

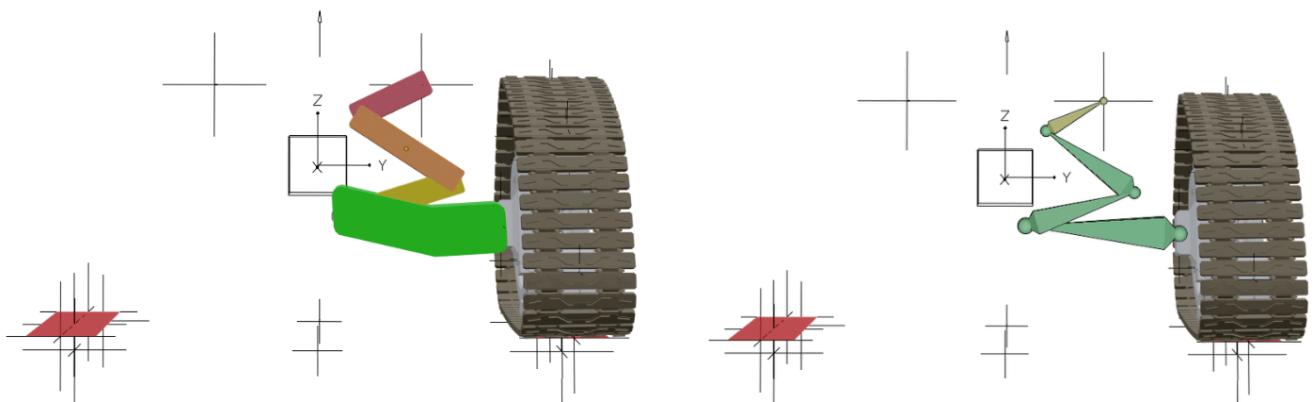


Abbildung 13: *Armature* (Eigene Darstellung)

## 7. Rig Control

Für die Steuerung des Rigs gibt es zwei grundlegende Ansätze. Zum einen das Erstellen einer Curve, welcher der Charakter folgt. Zum anderen die Steuerung mit einem Controller. Der Vorteil einer Steuerung mit dem Controller ist, dass der Input sehr dynamisch erfolgt, was für die Preivisualization ein großer Vorteil ist, da Szenen verändert werden können, ohne dass der Weg verändert werden muss. Der Controller Support erfordert dabei die Verwendung eines Python Scripts, welches in diesem Kapitel, mit Hilfe von Screenshots, näher dargestellt wird. Das gesamte Python-Script liegt auch dem Anhang bei.

### 7.1 Grundlagen

Zur Bedienung wird ein XBOX-One Controller (siehe Abb.14) verwendet. Dieser verfügt über 11 Buttons mit Eingabewerten 0 und 1, zwei Trigger mit Eingabewert 0-1, sowie zwei Joysticks mit Eingabewerten von -1 bis 1. Die Eingabewerte werden

dabei mit der Python Library XInput abgerufen und sind in Abbildung 15 dargestellt (vgl. PyPi 2024). Das Script verwendet dabei einen Timer Operator, welcher das Script mehrmals pro Sekunde aufruft.



Abbildung 14: XBOX-One Controller (Evan Amos, 2014)

```
state = XInput.get_state(0)
thumbL = XInput.get_thumb_values(state)[0][0]
thumbR = XInput.get_thumb_values(state)[1][1]
backBottomL = XInput.get_trigger_values(state)[0]
backBottomR = XInput.get_trigger_values(state)[1]
backTopL = XInput.get_button_values(state)[ "LEFT_SHOULDER"]*1
backTopR = XInput.get_button_values(state)[ "RIGHT_SHOULDER"]*1
dPadUp = XInput.get_button_values(state)[ "DPAD_UP"]*1
dPadDown = XInput.get_button_values(state)[ "DPAD_DOWN"]*1
dPadRight = XInput.get_button_values(state)[ "DPAD_RIGHT"]*1
dPadLeft = XInput.get_button_values(state)[ "DPAD_LEFT"]*1
X = XInput.get_button_values(state)[ "X"]
Y = XInput.get_button_values(state)[ "Y"]
A = XInput.get_button_values(state)[ "A"]
B = XInput.get_button_values(state)[ "B"]
start = XInput.get_button_values(state)[ "START"]
```

Abbildung 15: Controller Input (Eigene Darstellung)

Die in den nachfolgenden Kapiteln genannten Abläufe können anhand von einigen im Code definierten Parametern angepasst werden, um unterschiedliche Ergebnisse zu erzielen. (siehe Abb.16)

```
41 #values for user to change
42 translation_max = 1 #max translation on X (standard 1)
43 translation_maxY = 0.8 #max translation on Y (standard 0.8)
44 rotation_max = 90 #maximum rotation for body
45 speedXmultiplier = 1 #multiply with Joystick Input 1 to -1 (standard 1.5)
46 speedZmultiplier = 0.05 #multiply with Joystick Input 1 to -1 (standard 0.1)
47 resistance = 0.3 #force for translating back to the origin (standard 0.3)
48 resistanceR = 1.3 #force for rotating back to the origin (standard 1.3)
49 bounce_height_multiplier = 30 #multiplier for how high the body bounces on movement, decrease for stronger bounce (standard 30)
50 bounce_frequency_multiplier = 10 #multiplier for how fast the body bounces on movement (standard 10)
```

Abbildung 16: Parameter (Eigene Darstellung)

## 7.2 Bewegung und Rotation

Die Grundidee bei der Bewegung von Wall-E ist simpel. Bewegen sich die Ketten nach vorne, entsteht Fliehkraft/Zentrifugalkraft, welche den Körper nach hinten schiebt und auch entsprechend rotiert. Endet die Bewegung der Ketten, bewegt sich der Körper zurück in seine ursprüngliche Position, erhält dafür aber eine entgegengesetzte Fliehkraft, welche ihn zuerst über die ursprüngliche Position hinaus bewegt, bevor sie ihn endgültig dorthin zurückbewegt. Im Code wurde dies über die verschiedenen if else statements umgesetzt, welche zuerst auf Bewegung des Joysticks bzw. Geschwindigkeit prüfen, dann auf Erreichen der Maximalwerte,

und dann die vorgesehenen Veränderungen (oder Nichtveränderungen) durchführen. Endet die Bewegung wird überprüft ob der Körper aktuell vor oder hinter der eigentlichen Ursprungsposition steht. Der Wert für die Bewegung wird dann mithilfe des frei wählbaren Parameters resistance, bzw. für Rotationen resistanceR berechnet.

```

108     #values for body
109     speedX_previous = 0
110     speedZ_previous = 0
111
112     speedX = thumbR*speedXmultiplier
113     speedZ = thumbL*speedZmultiplier
114
115     #body Move and rotate when moving forward
116     bodyT = body.matrix_local.translation
117     distanceX = checkDistanceX(bodyT, body_base_position)
118
119     if(thumbR > 0.01 or thumbR < -0.01):
120         if(distanceX < translation_max):
121             accelerationX = (speedX-speedX_previous)*(-1)
122         else:
123             accelerationX = 0
124         elif(thumbR < 0.01 and thumbR > -0.01):
125             if(checkPositiveX(bodyT, body_base_position) == True):
126                 if(distanceX > resistance):
127                     accelerationX = -resistance
128                 else:
129                     accelerationX = 0
130             elif(checkPositiveX(bodyT, body_base_position) == False):
131                 if(distanceX > resistance):
132                     accelerationX = resistance
133                 else:
134                     accelerationX = 0
135
136     #rotate body on Y
137     y_degrees = math.degrees(body.rotation_euler[1])
138     if(speedX > 0.02 or speedX < -0.02):
139         if(y_degrees < rotation_max and y_degrees > rotation_max*(-1)):
140             rotationY = (accelerationX * 5)*(-1)
141         else:
142             rotationY = 0
143         elif(speedX < 0.01 and speedX > -0.01):
144             if(y_degrees < resistanceR and y_degrees > -resistanceR):
145                 rotationY = 0
146
147             elif(y_degrees >= 1):
148                 if(y_degrees > resistanceR):
149                     rotationY -= resistanceR*0.5
150                 else:
151                     rotationY = 0
152             elif(y_degrees <= -1):
153                 if(y_degrees < resistanceR*(-1)):
154                     rotationY += resistanceR*0.5
155                 else:
156                     rotationY = 0
157

```

Abbildung 17: Code for Movement on X [a](Eigene Darstellung)

Für die Bewegung und die gleichzeitige Rotation gilt eine ähnliche Grundidee. Hinzu kommt nun dass der Körper sich bei einer geschwindigkeit auf x die größer als 0 ist, nach rechts oder links bewegt und entsprechend rotiert.

### 7.3 Veränderung der Kettensysteme

Für die Veränderung von Höhe und Rotation der oberen beiden Räder werden die 4 Rückentasten sowie der Wert von Acceleration X aus den vorherigen Kapiteln. Die Verschiebung nach oben wird direkt über den kombinierten Wert der unteren beiden Trigger ermittelt. Die Bewegung auf der lokalen y-Achse, nach vorne und hinten, wird

über eine Subtraktion des Werts accelerationX erreicht. Dadurch erhalten auch die oberen Räder die in Kapitel 7.2 aufgezeigte Fliehkraft bei Bewegungen.

Rotationen auf x werden über das direkte Addieren bzw. Subtrahieren der oberen beiden Schultertasten erreicht.

```
213 #sidewheels
214     rotation_angle_degrees = 0 + backTopL - backTopR
215     wheelLeft = bpy.data.objects["control_position_topwheels"]
216     wheelLeftT = wheelLeft.matrix_local.translation
217     wheelLeft.matrix_local.translation=(wheelLeftT.x, wheelLeftT.y+accelerationX*0.2, wheelLeftT.z-backBottomL*0.1+backBottomR*0.1)
218     wheelLeft.rotation_euler[0] += math.radians(rotation_angle_degrees*2)
```

Abbildung 18: *Code for Sidewheels* (Eigene Darstellung)

## 7.4 Positionierung von Körper und Ketten

Um den Körper nach oben und unten zu verschieben wird der Wert der oberen Pfeiltaste auf die lokale Translation des control\_body addiert. Der Wert der unteren Pfeiltaste wird subtrahiert (siehe Abb.19). Dasselbe Prinzip wird bei der Bewegung der Kettensysteme verwendet, jedoch mit der rechten und linken Kreuztaste dargestellt in Abbildung 20. Durch die Wahl der Kreuztasten können die Abläufe des Absenkens des Körpers auf die Ketten und gleichzeitiges Einfahren dieser mit nur einem Finger durchgeführt werden.

```
220 #main up and down
221     emptyMain = bpy.data.objects["control_body"]
222     MainT = emptyMain.matrix_local.translation
223     emptyMain.matrix_local.translation=(MainT.x, MainT.y, MainT.z-dPadDown*0.5+dPadUp*0.5)
```

Abbildung 19: *Code for Up and Down* (Eigene Darstellung)

```
248 #shrinkwrap move
249     planeRight = bpy.data.objects["plane_groundDeform_right"]
250     planeLeft = bpy.data.objects["plane_groundDeform_left"]
251
252     movePlaneRight = (0 - dPadRight + dPadLeft)*0.1
253     movePlaneLeft = (0 + dPadRight - dPadLeft)*0.1
254
255     planeRight.location.x += movePlaneRight
256     planeLeft.location.x += movePlaneLeft
```

Abbildung 20: *Code for moving Tracks* (Eigene Darstellung)

## 7.5 Federung

Die Idee der Federung ist, dass bei einer höheren Geschwindigkeit der Körper von Wall-E entsprechend auf diese Geschwindigkeit und den Untergrund reagiert. Würde er z.B. über ein Feld fahren, hätte er eine stärkere Reaktion als wenn er mit derselben Geschwindigkeit auf einer Schnellstraße fährt. Zur Umsetzung wird hier das Frame System von Blender verwendet.

Liegt die Geschwindigkeit auf x bei mehr oder weniger als 0, werden die Werte bounce\_height und bounce\_frequency anhand der Geschwindigkeit und frei wählbaren Modifikatoren gesetzt. Für die eigentliche Bewegung wird dann eine Sinus Funktion verwendet, welche diese Werte mit einem Wert für Time kombiniert. Da die Sinusfunktion eine Kurve darstellt, wird der Wert für bounce trotz dauerhafter positiver Eingabewerte in regelmäßigen Abständen positiv bzw. negativ. Die Multiplikatoren können von Benutzern verändert werden um unterschiedliches Terrain zu simulieren.

```

225     #bounce on z while driving
226     _timer = None
227     bounce_height = 0
228     bounce_frequency = 0
229
230     if(speedX > 0.01):
231         bounce_height = speedX/bounce_height_multiplier
232         bounce_frequency = speedX*bounce_frequency_multiplier
233     elif(speedX < -0.01):
234         bounce_frequency = speedX*-bounce_frequency_multiplier
235         bounce_height = speedX/-bounce_height_multiplier
236     else:
237         bounce_frequency = 0
238
239     time = context.scene.frame_current / context.scene.render.fps
240     bounce = math.sin(time * bounce_frequency) * bounce_height
241
242     #update transformations of body
243     body.matrix_local.translation=(bodyT.x+accelerationX,bodyT.y+accelerationY,bodyT.z+bounce)

```

Abbildung 21: Code for Bounce (Eigene Darstellung)

## 7.6 Kontrollparameter

Für die Benutzer des Rigs sind einfach einzustellende Parameter ein wichtiger Bestandteil der Nutzung. Die in Abbildung 22 dargestellten werden daher hier kurz mit Fokus auf Funktion und Bedeutung erklärt.

*translation\_max, translation\_maxY, rotation\_max*: Diese beschränken die maximale Wirkung der Fliehkraft bei Bewegung. Insbesondere die Beschränkung auf Y ist wichtig, um eine optische Überschneidung mit den Kettensystemen zu vermeiden. Hier fließt auch die rotation\_max mit ein welche die maximale Rotation des Körpers beschränkt.

*speedXmultiplier, speedZmultiplier*: Multiplikatoren, welche die Geschwindigkeit der Bewegungen erhöhen.

*resistance, resistanceR*: resistance beeinflusst wie schnell sich der Körper beim Bremsen zurück in die Ursprungsposition bewegt. resistance R bestimmt dies bei Rotationen.

*bounce\_height\_multiplier* und *bounce\_frequency\_multiplier*: Diese Werte verändern wie schnell und hoch die Sinuskurve ist, welche die Federung bestimmt.

```
41 #values for user to change
42 translation_max = 1 #max translation on X (standard 1)
43 translation_maxY = 0.8 #max translation on Y (standard 0.8)
44 rotation_max = 40 #maximum rotation for body (standard 40)
45 speedXmultiplier = 1 #multiply with Joystick Input 1 to -1 (standard 1.5)
46 speedZmultiplier = 0.05 #multiply with Joystick Input 1 to -1 (standard 0.1)
47 resistance = 0.3 #force for translating back to the origin (standard 0.3)
48 resistanceR = 1.3 #force for rotating back to the origin (standard 1.3)
49 bounce_height_multiplier = 30 #multiplier for bounces upon movement (standard 30)
50 bounce_frequency_multiplier = 10 #multiplier for bouncefrequency upon movement (standard 10)
```

Abbildung 22: *Code for Parameters* (Eigene Darstellung)

Die Veränderung dieser Parameter bestimmt wie realitätsnah oder fern die Bewegungen des Charakters sind. So kann je nach gewünschten emotionalen Effekt z.B. die *translation\_max* erhöht werden um die Bewegungen überspitzter und comichaft zu gestalten. Mit den *bounce multipliern* kann dagegen auch unterschiedliches Terrain simuliert werden, was wiederum Realitätsnähe erzeugen kann.

## 8. Test und Evaluation

### 8.1 Ergebnisse

In diesem Kapitel wird der Test des Rigs anhand eines Videos gezeigt. Das Video wurde auf Youtube hochgeladen und kann über den Link <https://youtu.be/5gwSkgWDpf0> erreicht werden. Alternativ kann der in Abbildung 23 gezeigte QR Code verwendet werden.



Abbildung 23: *QR-Code für Video zu Funktionen* (Eigene Darstellung)

Die im Video gezeigten Funktionen demonstrieren die, durch den in Kapitel 7 erläuterten Code, bereitgestellten Funktionen in einer Live Blender Anwendung mit einem XBOX One Controller.

## 8.2 Limitationen und Herausforderungen

Die in Kapitel 4.2 besprochenen Auswahlkriterien für die verwendete 3D Software hätten mehr Fokus auf für die Animation relevante Themen legen müssen. So wären mehrere Funktionen der Software 3ds-Max für diese Arbeit hilfreich gewesen. Beispiele sind der Spring Controller, welcher einen Großteil des hier gezeigten Codes ersetzen könnte. Blender bietet einen solchen zwar auch, jedoch nur in Verbindung mit dem Rigid Body / Physics System welches mit einzelnen Teilen des Rigs nicht kompatibel ist.

Eine weitere Limitation ist die Tatsache, dass Blender Open Source ist und daher insbesondere häufig verwendete inoffizielle Python Libraries, welche zusätzliche Funktionen bereitstellen, zuerst von der Community für die neueste Version veröffentlicht werden müssen. Ein Beispiel ist die häufig verwendete MathUtils Library, welche einfachen Zugang zu ansonsten umfangreichen Matrix Berechnungen gibt.

Dies führt dazu, dass einfache, aber umfangreiche Funktionen im Code selbst geschrieben werden müssen, was dessen Übersichtlichkeit einschränkt.

## 8.3 Weiterentwicklungsmöglichkeiten

1.Für die Benutzung ist ein gewisses Grundwissen des Codes sowie direkte Änderungen an diesem notwendig, um Parameter anzupassen. Im Zweifelsfall könnten Nutzer daher den Code negativ verändern und die Funktion des Programms unterbinden. Um dieses Problem zu beheben, könnte ein neues UI Element in Blender integriert werden, welches die Veränderung der Parameter ohne direkte Eingriffe in den Code ermöglicht.

2.Arme und Kopf könnten durch die Benutzung von hier erstellten Berechnungen der Fliehkraft über Armatures und Bones integriert werden, um ein vollständiges Rig von Wall-E zu erstellen.

3.Wenn weitere Elemente zum Rig hinzugefügt werden, müssten auch neue Möglichkeiten zur Kontrolle geschaffen werden. Eine Möglichkeit wäre das

Umschalten zwischen einzelnen Gruppen, wie Augen, Beinen, Armen, mit jeweils neuer Buttonbelegung auf dem Controller.

4. Das aktuelle System wurde so programmiert, dass jede manuelle Bewegung nur ein Objekt anspricht. Für Bewegungen wie das Einfahren der Beine und gleichzeitiges Ablassen des Körpers ist dies unintuitiv, da zwei Knöpfe gleichzeitig betätigt werden müssen. Hier wäre das Angebot von kombinierten Animationen sinnvoll über z.B. Tastenkombinationen.

5. Das Rig wird vor allem durch die Codestruktur eingeschränkt. Diese sollte überarbeitet werden, um weitere Anpassungen zu vereinfachen. Dies führt auch zu visuellen Fehlern wie dem dauerhaften Ausführen der durch Fliehkraft erzeugten Bewegung. Im Video ist dies z.B. bei Sekunde 48 sichtbar. Hier würde sich anbieten, die Fliehkräfte für Rotation und Translation einmalig zu berechnen und dann auf alle Objekte anwendbar zu machen.

## 9. Fazit

In Kapitel 1 wurde als Ziel dieser Arbeit die Erstellung eines Rigs für den Charakter Wall-E, mit Fokus auf den Kettensystemen, für die Verwendung bei der Previsualization, definiert. Dafür wurden in dieser Arbeit die verschiedenen benötigten Systeme geplant, konstruiert und vorgestellt. Die benötigten theoretischen Ansätze wurden aufgelistet und dem Umfang der Arbeit entsprechend erklärt.

Das in Kapitel 8 gezeigte Video stellt die Ergebnisse der Arbeit in anschaulicher und praxisnaher Weise dar. Die wichtigsten Weiterentwicklungsmöglichkeiten werden in Kapitel 8.3 genannt.

Das zu Beginn genannte Ziel wurde somit erfüllt und auch Ansätze für Verbesserungen und Weiterentwicklungsmöglichkeiten bereitgestellt.

## Literaturverzeichnis

Belec, Arijan (2022): *Blender 3D Incredible Models*, Packt

Blender Foundation (2024a): [blender.org](https://blender.org), Blender Foundation, [online] [blender.org](https://blender.org).

Blender Foundation (2024b): [blender.org](https://blender.org), Blender Foundation, [online]  
<https://docs.blender.org/manual/en/latest/editors/3dview/controls/orientation.html>.

Disney (2008): Wall-E, Disney, [online]  
<https://www.disneyplus.com/en-gb/movies/wall-e/5G1wpZC2Lb6I>.

Evan-Amos (2014): [wikipedia.org](https://commons.wikimedia.org/wiki/File:Microsoft-Xbox-One-Console-Set-wKinect.jpg), Wikimedia Foundation, [online]  
<https://commons.wikimedia.org/wiki/File:Microsoft-Xbox-One-Console-Set-wKinect.jpg>.

de Goussencourt, Timothee/Dellac, Jean/Bertolino, Pascal (2015): *A Game Engine as a Generic Platform for Real-Time Previz-on-Set in Cinema Visual Effects*, in:  
Advanced Concepts for Intelligent Vision Systems: 16th International Conference,  
ACIVS 2015 Catania, Italy, October 26–29, 2015 Proceedings, Springer.

Gowanlock, Jordan (2021): *Animating Unpredictable Effects: Nonlinearity in Hollywood's R&D Complex*, Palgrave.

Guevarra, M., T., Ezra (2020): *Modeling and Animation Using Blender: Blender 2.80: The Rise of Eevee*, Springer.

Jazar, N., Reza (2007): *Theory of Applied Robotics: Kinematics, Dynamics, and Control*, Springer.

Kumar, Abhishek (2022): *Beginning VFX with Autodesk Maya: Create Industry-Standard Visual Effects from Scratch*, Springer.

Moioli, Gianpiero (2022): *Introduction to Blender 3.0: Learn Organic and Architectural Modeling, Lighting, Materials, Painting, Rendering, and Compositing with Blender*, Springer.

Paquette, Andrew (2013): *An Introduction to Computer Graphics for Artists*, 2.Aufl., Springer.

PyPi (2024): XInput-Python, pypi.org, [online]  
<https://pypi.org/project/XInput-Python/#description>.

Zucconi, Alan (2017a): An Introduction to procedural Animations, alanzucconi.com, [online] <https://www.alanzucconi.com/2017/04/17/procedural-animations/>.

Zucconi, Alan (2017b): Inverse Kinematics for Robotic Arms, alanzucconi.com, [online] <https://www.alanzucconi.com/2017/04/10/robotic-arms/>.

## Anhang

### a.Nutzungshinweise ohne Python oder Controller

Wenn die Datei "Bachelorarbeit Wall-E Blender 4.0" geöffnet wird, sollte das Objekt "HELP\_Main\_Movement" bereits ausgewählt sein. Dieses kann auf der lokalen x achse bewegt werden um die Veränderung der Kettenysteme zu sehen. Die manuelle Anpassung der Rigeigenschaften erfolgt über die Veränderung der folgenden Empties.

"control\_position\_topWheels" verändert die oberen Räder

"control\_body" gibt die ursprungsposition des körpers wieder, diese sollte nur auf der z-achse bewegt werden.

"HELP\_Body\_Control" simuliert die fliehkraft und sollte nur auf x und y bewegt werden. Der ursprung liegt dabei in "control\_body"

Der Shrinkwrap kann über die armature "Armature\_shrinkwrap" deaktiviert werden. Dafür muss in den Pose Mode gewechselt werden. Der Knochen kann dann einfach nach oben oder unten bewegt werden.

### b.Installationshinweise Python Scripts

<https://www.youtube.com/watch?v=Q6Ewy0r0RQ> tutorial für installation von python packages zu blender

Im Quellcode muss dann die zweite Zeil durch den entsprechenden lokalen Installationsordner von python ersetzt werden.

### c.Quellcode Wall-E Script

Hinweis: Der Quellcode ist in der Blender Datei im Bereich Scripting einsehbar. In der Liste mit Viewports direkt unter dem Blender Name ist der Bereich bereits hinterlegt.

```
import site
site.addsitedir('C:\\Users\\daibe\\Desktop\\New folder\\venv\\Lib\\site-packages')

import win32api
import bpy
import XInput
import math

from math import radians
from bpy.props import BoolProperty, PointerProperty
from bpy.types import Operator

#define global variables once
#main controlblock
mainControl = bpy.data.objects["HELP_Main_Movement"]
mainControl.location = (0,0,0)
main_positionOG = mainControl.matrix_local.translation
main_position = mainControl.matrix_local.translation

camera = bpy.context.scene.camera

#body
body = bpy.data.objects["HELP_Body_Control"]
body_base_position = (0,0,0)
body.matrix_local.translation = (0,0,0)
current_position = body.matrix_local.translation

#Rotation Values
body.rotation_euler[1] = 0
```

```

body.rotation_euler[0] = 0
rotationY = 0
rotationX = 0

obj = mainControl
euler = obj.rotation_euler
local_translation = (0, 0, 0)
rotated_translation = 0
accelerationX = 0
accelerationY = 0

#values for user to change
translation_max = 1 #max translation on X (standard 1)
translation_maxY = 0.8 #max translation on Y (standard 0.8)
rotation_max = 40 #maximum rotation for body (standard 40)
speedXmultiplier = 1 #multiply with Joystick Input 1 to -1 (standard 1.5)
speedZmultiplier = 0.05 #multiply with Joystick Input 1 to -1 (standard 0.1)
resistance = 0.3 #force for translating back to the origin (standard 0.3)
resistanceR = 1.3 #force for rotating back to the origin (standard 1.3)
bounce_height_multiplier = 30 #multiplier for bounces upon movement (standard 30)
bounce_frequency_multiplier = 10 #multiplier for bouncefrequency upon movement (standard 10)

#reset timeline
bpy.context.scene.frame_set(0)
bpy.ops.screen.animation_play()

class ModalTimerOperator(bpy.types.Operator):
    #Operator which runs itself from a Timer
    bl_idname = "wm.modal_timer_operator"
    bl_label = "Modal Timer Operator"
    _timer = None

    def modal(self, context, event):
        global body
        global body_base_position
        global current_position
        global main_position
        global main_positionOG
        global accelerationX
        global accelerationY
        global rotationY
        global rotationX

        ##activate controller##

        state = Xinput.get_state(0)
        thumbL = Xinput.get_thumb_values(state)[0][0]
        thumbR = Xinput.get_thumb_values(state)[1][1]
        backBottomL = Xinput.get_trigger_values(state)[0]
        backBottomR = Xinput.get_trigger_values(state)[1]
        backTopL = Xinput.get_button_values(state)["LEFT_SHOULDER"]*1
        backTopR = Xinput.get_button_values(state)["RIGHT_SHOULDER"]*1
        dPadUp = Xinput.get_button_values(state)["DPAD_UP"]*1
        dPadDown = Xinput.get_button_values(state)["DPAD_DOWN"]*1
        dPadRight = Xinput.get_button_values(state)["DPAD_RIGHT"]*1
        dPadLeft = Xinput.get_button_values(state)["DPAD_LEFT"]*1
        X = Xinput.get_button_values(state)["X"]
        Y = Xinput.get_button_values(state)["Y"]
        A = Xinput.get_button_values(state)["A"]
        B = Xinput.get_button_values(state)["B"]
        start = Xinput.get_button_values(state)["START"]

        #print(Xinput.get_button_values(state))
        #####
        #change Camera
        if(X == True):
            bpy.context.scene.camera = bpy.data.objects["cam3"]
        if(A == True):
            bpy.context.scene.camera = bpy.data.objects["cam1"]
        if(B == True):
            bpy.context.scene.camera = bpy.data.objects["cam2"]
        if(Y == True):
            bpy.context.scene.camera = bpy.data.objects["cam4"]

        #values for body
        speedX_previous = 0
        speedZ_previous = 0

```

```

speedX = thumbR*speedXmultiplier
speedZ = thumbL*speedZmultiplier

#body Move and rotate when moving forward
bodyT = body.matrix_local.translation
distanceX = checkDistanceX(bodyT,body_base_position)

if(thumbR > 0.01 or thumbR < -0.01):
    if(distanceX < translation_max):
        accelerationX = (speedX-speedX_previous)*(-1)
    else:
        accelerationX = 0
    elif(thumbR < 0.01 and thumbR > -0.01):
        if(checkPositiveX(bodyT,body_base_position) == True):
            if(distanceX > resistance):
                accelerationX = -resistance
            else:
                accelerationX = 0
        elif(checkPositiveX(bodyT,body_base_position) == False):
            if(distanceX > resistance):
                accelerationX = resistance
            else:
                accelerationX = 0

#rotate body on Y
y_degrees = math.degrees(body.rotation_euler[1])
if(speedX > 0.02 or speedX < -0.02):
    if(y_degrees < rotation_max and y_degrees > rotation_max*(-1)):
        rotationY = (accelerationX * 5)*(-1)
    else:
        rotationY = 0
    elif(speedX < 0.01 and speedX > -0.01):
        if(y_degrees < resistanceR and y_degrees > -resistanceR):
            rotationY = 0
        elif(y_degrees >= 1):
            if(y_degrees > resistanceR):
                rotationY -= resistanceR*0.5
            else:
                rotationY = 0
            elif(y_degrees <=-1):
                if(y_degrees < resistanceR*(-1)):
                    rotationY += resistanceR*0.5
                else:
                    rotationY = 0

#body Move and rotate when moving forward and rotating on Z
#get body and distance to OG Point
distanceY = checkDistanceY(bodyT,body_base_position)
#get value for acceleration

if(thumbL > 0.01 or thumbL < -0.01):
    if(speedX > 0.01):
        if(distanceY < translation_maxY):
            accelerationY = (speedZ-speedZ_previous)*5
        else:
            accelerationY = 0
    elif(speedX < -0.01):
        if(distanceY < translation_maxY):
            accelerationY = (speedZ-speedZ_previous)*-5
        else:
            accelerationY = 0

    elif(thumbL < 0.01 and thumbL > -0.01):
        if(checkPositiveY(bodyT,body_base_position) == True):
            if(distanceY > resistance):
                accelerationY = -resistance*0.9
            else:
                accelerationY = 0
        elif(checkPositiveY(bodyT,body_base_position) == False):
            if(distanceY > resistance):
                accelerationY = resistance*0.9
            else:
                accelerationY = 0

#rotate body on X
x_degrees = math.degrees(body.rotation_euler[0])

if(thumbL > 0.02 or thumbL < -0.02):
    if(speedX > 0.05 or speedX <-0.05):

```

```

if(x_degrees < rotation_max and x_degrees > rotation_max*(-1)):
    rotationX = (accelerationY * 10)*(-1)
else:
    rotationX = 0
else:
    rotationX = 0
elif(thumbL < 0.02 and thumbL > -0.02):
    if(x_degrees < resistanceR and x_degrees > -resistanceR):
        rotationX = 0

    elif(x_degrees >= 1):
        if(x_degrees > resistanceR):
            rotationX = resistanceR*0.9
        else:
            rotationX = 0
    elif(x_degrees <= -1):
        if(x_degrees < resistanceR*(-1)):
            rotationX -= resistanceR*0.9
        else:
            rotationX = 0

rotationX = 0

#sidewheels
rotation_angle_degrees = 0 + backTopL - backTopR
wheelLeft = bpy.data.objects["control_position_topWheels"]
wheelLeftT = wheelLeft.matrix_local.translation
wheelLeft.matrix_local.translation=(wheelLeftT.x,wheelLeftT.y+accelerationX*0.2,wheelLeftT.z-backBottomL*0.1+backBottomR*0.1)
wheelLeft.rotation_euler[0] += math.radians(rotation_angle_degrees*2)

#main up and down
emptyMain = bpy.data.objects["control_body"]
MainT = emptyMain.matrix_local.translation
emptyMain.matrix_local.translation=(MainT.x,MainT.y,MainT.z-dPadDown*0.5+dPadUp*0.5)

#bounce on z while driving
_timer = None
bounce_height = 0
bounce_frequency = 0

if(speedX > 0.01):
    bounce_height = speedX/bounce_height_multiplier
    bounce_frequency = speedX*bounce_frequency_multiplier
elif(speedX < -0.01):
    bounce_frequency = speedX*-bounce_frequency_multiplier
    bounce_height = speedX*-bounce_height_multiplier
else:
    bounce_frequency = 0

time = context.scene.frame_current / context.scene.render.fps
bounce = math.sin(time * bounce_frequency) * bounce_height

#update transformations of body
body.matrix_local.translation=(bodyT.x+accelerationX,bodyT.y+accelerationY,bodyT.z+bounce)

body.rotation_euler[0] += math.radians(-rotationX)
body.rotation_euler[1] += math.radians(rotationY)

#shrinkwrap move
planeRight = bpy.data.objects["plane_groundDeform_right"]
planeLeft = bpy.data.objects["plane_groundDeform_left"]

movePlaneRight = (0 - dPadRight + dPadLeft)*0.1
movePlaneLeft = (0 + dPadRight - dPadLeft)*0.1

planeRight.location.x += movePlaneRight
planeLeft.location.x += movePlaneLeft

#main
#move main on local x axis
local_translation = (speedX,0,0)
rotated_translation = rotate_vector(local_translation, euler.x, euler.y, euler.z)
obj.location.x += rotated_translation[0]
obj.location.y += rotated_translation[1]
obj.location.z += rotated_translation[2]
bpy.data.objects["help_Sphere_left"].rotation_euler[0] -= speedX*0.1

#rotate on global z axis
bpy.data.objects["HELP_Main_Movement"].rotation_euler[2] -= speedZ
bpy.data.objects["help_Sphere_left"].rotation_euler[0] -= thumbL*0.1

speedX_previous = speedX
speedZ_previous = speedZ

```

```

#reset
bodyR = bpy.data.objects["HELP_Body_Control"]
mainR = bpy.data.objects["control_body"]

if(start == True):
    bodyR.matrix_local.translation=(0,0,0)
    bodyR.rotation_euler[0] = math.radians(0)
    bodyR.rotation_euler[1] = math.radians(0)
    mainR.matrix_local.translation=(0,0,15)
    planeRight.location.x = -5
    planeLeft.location.x = 5
    bpy.data.objects["help_Sphere_left"].rotation_euler[0] = 0

if event.type in {'RIGHTMOUSE', 'ESC'}:
    #mainControl.matrix_local.translation = main_positionOG
    wheelLeft.matrix_local.translation=(0,wheelLeft.matrix_local.translation.y,wheelLeft.matrix_local.translation.z)
    self.cancel(context)
    bpy.ops.screen.animation_cancel()
    return {'CANCELLED'}

return{'PASS_THROUGH'}
```

```

def execute(self,context):
    wm = context.window_manager
    self._timer = wm.event_timer_add(0.1, window=context.window)
    wm.modal_handler_add(self)
    return {'RUNNING_MODAL'}
```

```

def cancel(self, context):
    wm = context.window_manager
    wm.event_timer_remove(self._timer)
```

```

def menu_func(self, context):
    self.layout.operator(ModalTimerOperator.bl_idname, text=ModalTimerOperator.bl_label)
```

```

def register():
    bpy.utils.register_class(ModalTimerOperator)
    bpy.types.VIEW3D_MT_view.append(menu_func)
```

```

def unregister():
    bpy.utils.unregister_class(ModalTimerOperator)
    bpy.types.VIEW3D_MT_view.remove(menu_func)
```

```

def checkPositiveX(v1, v2):
    conX = v2[0] - v1[0]
    if(conX <= 0):
        return(True)
    else:
        return(False)
```

```

def checkPositiveY(v1, v2):
    conY = v2[1] - v1[1]
    if(conY <= 0):
        return(True)
    else:
        return(False)
```

```

def checkDistanceX(dist1, dist2):
    conX = dist1[0] - dist2[0]
    conY = 0
    conZ = 0
    dist = float(round(math.sqrt(conX**2+conY**2+conZ**2),3))
    return(dist)
```

```

def checkDistanceY(dist1, dist2):
    conX = 0
    conY = dist1[1] - dist2[1]
    conZ = 0
    dist = float(round(math.sqrt(conX**2+conY**2+conZ**2),3))
    return(dist)
```

```

def rotate_vector(v, roll, pitch, yaw):
    """Rotate vector 'v' by Euler angles (roll, pitch, yaw)"""
    # Calculate cosines and sines
    cr = math.cos(roll)
    sr = math.sin(roll)
    cp = math.cos(pitch)
    sp = math.sin(pitch)
    cy = math.cos(yaw)
    sy = math.sin(yaw)
```

```

# Rotation matrices around x, y, and z axes
rx = [1, 0, 0], [0, cr, -sr], [0, sr, cr]
ry = [cp, 0, sp], [0, 1, 0], [-sp, 0, cp]
rz = [cy, -sy, 0], [sy, cy, 0], [0, 0, 1]

# Combined rotation matrix: Rz * Ry * Rx
r = [[sum(a*b for a, b in zip(R_row, C_col)) for C_col in zip(*rx)] for R_row in rz]
r = [[sum(a*b for a, b in zip(R_row, C_col)) for C_col in zip(*ry)] for R_row in r]

# Rotate vector
return [sum(a*b for a, b in zip(R_row, v)) for R_row in r]

if __name__ == "__main__":
    register()
    bpy.ops.wm.modal_timer_operator()

```

## Versicherung über redliches wissenschaftliches Arbeiten

Hiermit versichere ich, Simon Eugen Josef Daiber, dass ich die vorliegende Arbeit selbstständig verfasst und erstellt habe. Ich versichere, dass ich nur zugelassene Hilfsmittel und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ferner versichere ich, dass ich alle wörtlich oder sinngemäß übernommenen Stellen in der Arbeit gemäß gängiger wissenschaftlicher Zitierregeln korrekt zitiert und als solche gekennzeichnet habe. Darüber hinaus versichere ich, dass alle verwendeten Hilfsmittel, wie KI-basierte Chatbots (bspw. ChatGPT), Übersetzungs- (bspw. Deepl), Paraphrasier- (bspw. Quillbot) oder Programmier-Applikationen (bspw. Github Copilot) vollumfänglich deklariert und ihre Verwendung an den entsprechenden Stellen angegeben und gekennzeichnet habe.

Ich bin mir bewusst, dass die Nutzung maschinell generierter Texte keine Garantie für die Qualität von Inhalten und Text gewährleistet. Ich versichere, dass ich mich textgenerierender KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Ich verantworte die Übernahme jeglicher von mir verwendetener maschinell generierter Textpassagen vollumfänglich selbst.

Auch versichere ich, die „Satzung der Hochschule Furtwangen (HFU) zur Sicherung guter wissenschaftlicher Praxis“ vom 27. Oktober 2022 zur Kenntnis genommen zu haben und mich an den dortigen Ausführungen zu orientieren.

Mir ist bewusst, dass meine Arbeit auf die Benutzung nicht zugelassener Hilfsmittel oder Plagiate überprüft werden kann. Auch habe ich zur Kenntnis genommen, dass ein Verstoß gegen § 10 bzw. § 11 Absatz 4 und 5 der Allgemeinen Teile der HFU-SPOen zu einer Bewertung der betroffenen Arbeit mit der Note 5 oder mit «nicht ausreichend» und/oder zum Ausschluss von der Erbringung aller weiteren Prüfungsleistungen führen kann.



28.02.2024

Ort, Datum

Unterschrift