

Technischer Bericht:  
**Inverse Kinematik - Roboterarm**

Gruppe C:  
Sven Müller, Yuliya Litvin, Marco Wieditz

06.07.2020

# Inhaltsverzeichnis

|   |           |
|---|-----------|
| <b>1 Einführung</b>   | <b>4</b>  |
| <b>2 Grundlagen</b>   | <b>4</b>  |
| 2.1 Inverse Kinematik . . . . .                                 | 4         |
| 2.2 Wissenschaftliche Arbeiten . . . . .                        | 4         |
| 2.2.1 Roboterarm . . . . .                                      | 4         |
| 2.2.2 Pfadfindung (Path Planning) . . . . .                     | 5         |
| 2.2.3 Aufgegliederte Ziele . . . . .                            | 5         |
| <b>3 Implementierung und Algorithmen</b>                        | <b>6</b>  |
| 3.1 Erzeugen eines Roboterarms mit variablen Gliedern . . . . . | 7         |
| 3.2 Erstellen des Umfelds . . . . .                             | 8         |
| 3.3 Bewegung des Roboterarms . . . . .                          | 10        |
| 3.4 Transport der Kisten . . . . .                              | 13        |
| <b>4 Numerische Beispiele</b>                                   | <b>13</b> |
| <b>5 Fazit und Ausblick</b>                                     | <b>15</b> |

# Codeverzeichnis

|  |    |
|--|----|
| 1 Aufruf der „roboterArm“-Funktion und Erstellung eines Roboterarms . . . . .                                | 7  |
| 2 Festlegen der Armgliedlängen . . . . .   | 7  |
| 3 Überprüfung: ob die Summe der Armgliedlängen ausreichend zum Erreichen der maximalen Distanz ist . . . . . | 7  |
| 4 Erstellung eines leeren Robotermanipulator-Modells . . . . .   | 7  |
| 5 Erzeugung des ersten Armglieds: Dieses bildet das Basisglied . . . . .                                     | 7  |
| 6 Generieren weiterer beliebig vieler Armglieder . . . . .   | 8  |
| 7 Endeffektor bekommt einen fixierten Körper . . . . .   | 8  |
| 8 Konsolenausgabe der Eigenschaften des Roboterarms . . . . .  | 8  |
| 9 Erschaffung des Grundbodens . . . . .  | 8  |
| 10 Eigenschaften der Ablegestellen . . . . .   | 8  |
| 11 Erstellung der Ablegestellen . . . . .  | 9  |
| 12 Größen und Koordinaten der Kisten . . . . .   | 9  |
| 13 Anlegen der Kisten . . . . .  | 9  |
| 14 Festlegung des Zeitschritts $t$ . . . . .   | 10 |

|    |  |    |
|----|--|----|
| 15 | Denifierung des Ausgangspunkts des Endeffektors und Berechnung aller nötigen Mitelpunkte zur Pfadfindung . . . . . | 11 |
| 16 | Auf dem Pfad liegende Punkte . . . . .   | 11 |
| 17 | Vordefinition von Konfigurationslösungen in einer Matrix . . . . .   | 11 |
| 18 | Berechner der inversen Kinematik . . . . .   | 11 |
| 19 | Berechnung der aktuellen Punkte für jeden Zeitschritt $t$ . . . . .  | 12 |
| 20 | Erstellung der Gelenkkonfiguration . . . . .   | 12 |
| 21 | Konfigurationslösungen animieren . . . . .   | 12 |
| 22 | Anpassen der Schnelligkeit der Roboterarmbewegung . . . . .  | 12 |
| 23 | Transport der Kisten . . . . .   | 13 |

## **Zusammenfassung**

Dieser Bericht behandelt die Implementation eines Roboterarms auf einer zweidimensionalen Ebene. Die Aufgabe des Roboterarms ist es, nach Kisten zu greifen und diese auf die gleichfarbigen Positionsmaerkierungen zu legen. Des Weiteren ist die GroÙe des Arms variabel, sodass mehrere Gelenke problemlos hinzugefügt werden können.

## **1 Einführung**

Die Inverse Kinematik wird vor allem in der Robotik und Animation eingesetzt. Das Thema ist umfangreich, weshalb ein Beispiel einer Anwendungsmöglichkeit einen hilfreichen Einstieg bereitstellt. In dieser Arbeit gilt zu erreichen, dass ein Roboterarm einen vordefinierten Pfad entlanggeht und dabei jeweils nacheinander farbige Kisten aufsammelt. Diese Kisten soll der Arm auf gleichfarbige Felder legen. Dabei soll es möglich sein, den Roboterarm um Glieder zu erweitern oder zu verkleinern und dessen einzelne Gliedlängen individuell festzulegen. Dies gilt zur Veranschaulichung der Bewegung des Arms und zum leichteren Verständnis der Thematik.

Außerdem werden in diesem Bericht drei wissenschaftliche Arbeiten beschrieben, die im Zusammenhang mit der inversen Kinematik und dem Roboterarm stehen. In der ersten wissenschaftlichen Arbeit geht es um das Erstellen eines Roboterarms im Allgemeinen. Eine weitere Arbeit erläutert die Pfadfindung bzw. Pfadplanung des Armes einem Ziel entgegen. Und zuletzt wird eine Arbeit untersucht, die sich mit vordefinierten Zielen eines Roboterarms befasst.

## **2 Grundlagen**

### **2.1 Inverse Kinematik**

Die Inverse Kinematik ist ein Prinzip der Robotik. Es ist dazu da, den sogenannten Endeffektor in eine gewünschte Position zu bringen. Der Endeffektor ist das letzte Stück einer kinematischen Kette, wie zum Beispiel das letzte Glied eines Roboterarms. Durch die Berechnung der Inversen Kinematik ist es möglich, dass sich ein Arm mit bestimmt vielen Gliedern einen Zielpunkt entgegen bewegt. Die anderen Glieder des Roboterarms werden so rotiert und positioniert, dass der Endeffektor das Ziel erreicht.[3]

### **2.2 Wissenschaftliche Arbeiten**

#### **2.2.1 Roboterarm**

Wie es im Foliensatz „Roboterkinematik“ von der Technischen Universität beschrieben ist, ist der Roboterarm einer der grundlegenden Elemente, bei dem die inverse Kinematik angewandt wird. Der Arm besteht aus mindestens zwei Gliedern, während jedes Glied genau zwei Gelenke besitzt. Die Gelenke verbinden fast immer zwei Glieder miteinander. Allerdings wird beim ersten und letzten Glied ein Gelenk nur mit dem jeweiligen Glied verbunden, deshalb bezeichnet man diese beiden Gelenke als „offen“. Hingegen dazu sind alle Gelenke geschlossen. Mit der inversen Kinematik wird anhand des Endeffektors die jeweilige Ausrichtung der Gelenke und Position der Glieder bestimmt, um den Endeffektor zu erreichen. Dabei bezieht sich das jeweils nächste Glied immer auf das vorherige

Glied. Ein Beispiel für einen zweigliedrigen Arm wäre:

$$f = \begin{bmatrix} \alpha_1 \cos \theta_1 + \alpha_2 \cos(\theta_1 + \theta_2) \\ \alpha_1 \sin \theta_1 + \alpha_2 \sin(\theta_1 + \theta_2) \end{bmatrix} \quad (1)$$

Für diese Projekt wurde jedoch eine Toolbox verwendet, um den Arm zu simulieren. [3, 2, 6, 7]

### 2.2.2 Pfadfindung (Path Planning)

Nachdem ein Roboterarm erstellt wurde, sollte dieser nun auch nach Objekten greifen und diese an gewünschter Stelle platzieren können. Um dies zu bewerkstelligen, muss der Roboterarm sich ein Zielobjekt und gegebenenfalls eine passende Ablagestelle auswählen und den Pfad zum Zielobjekt und zu dessen Ablagestelle berechnen.

Mit dieser Thematik befasst sich das Konferenz-Paper „An Integrated Approach to Inverse Kinematics and Path Planning for Redundant Manipulators“ [1], das in einer IEEE Internationalen Konferenz über Robotik und Automation im Januar 2006 entstand und von Dominik Bertram, James Kuffner, Rüdiger Dillmann und Tamim Asfour, die Universitäten aus Deutschland und den USA entstammen, verfasst wurde. In diesem Paper wurde ein Lösungsvorschlag zu dem Problem der inversen Kinematik für redundante Roboter manipulatoren für den Zweck des Wählens eines Zielobjekts und der dazugehörigen Pfadplanung bzw. Pfadfindung gegeben.

Der Lösungsvorschlag sieht vor, dass die Berechnung der Zielkonfiguration mit der Suche nach dem bestmöglichen Pfad kombiniert wird. So können Unsicherheit vermieden werden, die beim Auswählen einer Zielkonfiguration auftreten könnten, da diese sich eventuell für den Roboterarm als unerreichbar herausstellen können, wenn sie sich zum Beispiel im freien Konfigurationsraum befinden, welche mit der Ausgangskonfiguration nicht verbunden sind.

Die Funktionen, die bei der Umsetzung dieser Idee verwendet wurden, definieren Zielregionen des Konfigurationsraums und erweitern die Rapid-Exploring Random Trees (RRTs), die die Zielregionen, die vom Roboterarm angesteuert werden sollen, suchen.

Im Paper wurden zudem drei Probleme aufgeführt, auf die bei der Pfadplanung, unabhängig davon wie viele Zielobjekte vorhanden sind, gestoßen werden könnte: Zum einen könnten Kollisionen mit Barrikaden auf dem geplanten Pfad auftreten. Zum anderen könnte das gewählte Zielobjekt außerhalb der Reichweite der aktuell gewählten Roboterarmlänge liegen und wäre damit für den Roboterarm nicht zu erreichen. Zudem könnte das Zielobjekt für den Roboterarm trotz eines kollisionsfreien Pfades unerreichbar sein, wenn zum Beispiel Hindernisse auf der Pfadstrecke liegen, die nicht zu umgehen sind. Die Problematik, dass der Roboterarm eine passende Länge besitzen muss, um die Zielobjekte mit dessen Endeffektor erreichen zu können, haben wir auch in unserer Projektidee erkannt. Dies haben wir in der Implementierung anhand von aufgestellten Bedingungen einer Minimal- und Maximallänge des Roboterarms gelöst.

### 2.2.3 Aufgegliederte Ziele

Problematiken beim Erstellen von Roboterarmen sind mitunter, dass der Arm eine Reihe von Zielen hat, die er nur einmal besuchen darf. Der End-Effektor spielt hierbei eine große Rolle. Als weitere Komplikation kann jedes Ziel durch unterschiedliche Armpositionen erreicht werden, weshalb eine optimale Lösung manchmal nicht möglich ist. Für jedes dieser Ziele muss er den optimalen Pfad berechnen und sich daraufhin annähern.

Mit dem Thema der Aufgliederung von Zielen in Gruppen und dem Berechnen eines Pfads anhand eines Algorithmus befasst sich die wissenschaftliche Arbeit "Planning Tours of Robotik Arms among Partitioned Goals" von der Stanford Universität, USA, geschrieben von Mitul Saha, Tim Roughgarden, Jean-Claude Latombe und Gildaro Sánchez-Ante. [5]

Der Lösungsansatz sieht vor, Ziele in Gruppen aufzuteilen und jedes Gruppenfeld zu besuchen. So wird Rechenzeit eingespart, da nicht von jedem Ziel weitere Pfade berechnet werden müssen, sondern nur der Pfad von Gruppe zu Gruppe. Innerhalb der Gruppe kann der Arm sich mit einem Algorithmus die einzelnen Ziele greifen.

Die Ziele in Gruppen aufzugliedern und den optimalen Lösungsweg abzulaufen ist im Fall dieses Roboterarms nicht notwendig. Auch nicht, die Ziele in Gruppen aufzuteilen, um niedrigere Computing-Time zu benötigen, ist in diesem Falle nicht wichtig. Eine einfache Annäherung an den Punkt genügt bereits zur Umsetzung der gewollten Funktion dieser Ausarbeitung.

### 3 Implementierung und Algorithmen

Ein rotierender und prismatischer Roboterarm, der eine gewisse ihm zugeteilte Aufgabe auszuführen hat, sollte erstellt werden. Die Aufgabe, die dem Roboterarm schließlich aufgetragen wurde, ist, unterschiedlich farbige Kisten nacheinander jeweils aufzusammeln und auf die entsprechend gleichfarbige Ablegestelle zu legen.

Um diese Aufgabe auszuführen, beginnt der Roboterarm in seinem vordefinierten Ausgangspunkt und bewegt sich zur ersten Kiste, die rot gefärbt ist. Diese Kiste sammelt er auf und trägt sie zur gleichfarbigen (roten) Ablegestelle. Dort stellt er die Kiste ab und bewegt sich anschließend zurück zu seinem Ausgangspunkt. Daraufhin fährt der Roboterarm die Strecke zur nächsten Kiste ab, die in gelber Farbe eingefärbt ist. Diese sammelt er wieder auf und transportiert sie zur gelben Ablagemarke. Er stellt die gelbe Kiste ab und bewegt sich wieder zum Ausgangspunkt. Als letztes fährt er die dritte grüne Kiste ab und zieht sie zur grünen Ablegestelle mit, wo er sie platziert und letztendlich wieder zurück zum Ausgangspunkt zurückkehrt. Seine Aufgabe ist hiermit abgeschlossen, sodass er in Ruhe an seinem Ausgangspunkt verweilt.

Dies ist anhand folgender Animation nachzuvollziehen:

---

<sup>1</sup>Zum Abspielen der Animation ist ein PDF-Reader(z.B. Acrobat Reader) notwendig

Die Aufgabenstellung wurde in Matlab anhand von zwei Skripten realisiert. Das Skript „roboterarm-Main.m“ dient als das Hauptprogramm, in dem alle nötigen Objekte erstellt werden und Berechnungen stattfinden. Ein zweites Skript, das den Namen „roboterArm.m“ trägt, beinhaltet die Erstellung des Roboterarms und wird wie folgt im Hauptprogramm aufgerufen:

```
1 roboterarm = roboterArm([armglied_L1 armglied_L2 armglied_L3], 0.6)
```

Quellcode 1: Aufruf der „roboterArm“-Funktion und Erstellung eines Roboterarms

Als Anregung haben wir uns an einem Beispiel „2-D Path Tracing With Inverse Kinematics“ [4] orientiert, das in der Matlab-Dokumentation auf der [mathworks.com](http://mathworks.com)-Webseite dokumentiert ist.

### 3.1 Erzeugen eines Roboterarms mit variablen Gliedern

Der Roboterarm wird im zweiten Skript „roboterArm.m“ erstellt. Der Funktion *roboterArm* werden zwei Parameter, die Längen der Armglieder und die maximale Distanz übergeben. Mit Hilfe dieser Parameter wird überprüft, ob die angegebenen Längen der Armglieder, die im Hauptprogramm definiert werden, zusammen ausreichen, um die gegebene maximale Distanz erreichen zu können.

```
1 % Armlängen angeben
2
3 armglied_L1 = 0.4;
4 armglied_L2 = 0.4;
5 armglied_L3 = 0.4;
```

Quellcode 2: Festlegen der Armgliedlängen

```
1 function robot = roboterArm(sizes,maxDist)
2 if sum(sizes)<=maxDist
3     error('arm_too_small')
4 end
```

Quellcode 3: Überprüfung: ob die Summe der Armgliedlängen ausreichend zum Erreichen der maximalen Distanz ist

Danach wird ein Objekt *roboter* der Klasse *rigidBodyTree* erstellt. Dabei ist das *rigidBodyTree* eine Darstellung der Konnektivität von *rigid bodies* (Starrkörpern) mit Gelenken. Somit kann mit Hilfe dieser Klasse ein leeres Robotermanipulator-Modell in Matlab kreiert werden, wobei unter einem Manipulator eine Kette aus Gelenken bzw. Bindungen verstanden wird.

```
1 robot = rigidBodyTree('DataFormat','column','MaxNumBodies',3);
```

Quellcode 4: Erstellung eines leeren Robotermanipulator-Modells

Nun werden die einzelnen Armglieder erzeugt.

Ein *rigidBodyTree* besteht aus mehreren *RigidBody*-Objekten. Dafür wird nun zunächst ein Objekt *body* der Klasse *rigidBody* generiert. Darauf folgt ein weiteres Objekt *joint* der Klasse *rigidBodyJoint*, das das Gelenk repräsentiert und dem Objekt *body* gegeben wird. Dieses Gelenk wird als das „Basisgelenk“ definiert, sodass dementsprechend ein weiteres dazukommendes Gelenk mit diesem verbunden werden muss.

```
1 body = rigidBody("link1");
2 joint = rigidBodyJoint('joint1', 'revolute');
3 setFixedTransform(joint,trvec2tform([0 0 0]));
4 joint.JointAxis = [0 0 1];
5 body.Joint = joint;
6 addBody(robot, body, 'base');
```

Quellcode 5: Erzeugung des ersten Armglieds: Dieses bildet das Basisglied

Anhand einer *for*-Schleife können beliebig viele weitere Armglieder hinzugefügt werden. Wichtig dabei ist, dass jedes der Gelenke, mit dem vorherigen verbunden wird, damit eine Kette von Gelenken entsteht, die zusammen agiert.

```

1 for i = 1:length(sizes)-1
2   l = "link"+(i+1)
3   j = "joint"+(i+1)
4   l2 = "link"+i
5   body = rigidBody(l);
6   joint = rigidBodyJoint(j, 'revolute');
7   setFixedTransform(joint, trvec2tform([sizes(i) 0 0]));
8   joint.JointAxis = [0 0 1];
9   body.Joint = joint;
10  addBody(robot, body, l2);
11 end

```

Quellcode 6: Generieren weiterer beliebig vieler Armglieder

Nachdem alle gewünschten Armglieder bereits erstellt wurden, muss nun der Endeffektor *endEtool* mit dem Gelenk *fix1* verbunden werden, sodass der Endeffektor einen fixierten Körper bekommt.

```

1 body = rigidBody('endEtool');
2 joint = rigidBodyJoint('fix1','fixed');
3 setFixedTransform(joint, trvec2tform([sizes(end), 0, 0]));
4 body.Joint = joint;
5 addBody(robot, body, "link"+(length(sizes)));

```

Quellcode 7: Endeffektor bekommt einen fixierten Körper

Abschließend werden die Eigenschaften des Roboters zur Überprüfung mit dem Befehl *showdetails()* in der Konsole ausgegeben.

```
1 showdetails(robot)
```

Quellcode 8: Konsolenausgabe der Eigenschaften des Roboterarms

## 3.2 Erstellen des Umfelds

Im Hauptprogramm passieren die wesentlichen Dinge, wie die nötigen Berechnungen, aber auch die Erzeugung des Umfelds und der Objekte neben dem Roboterarm.

So wird in diesem Skript auch der Grundboden erschaffen, auf dem sich das Szenario abspielt. Dies wird mit Hilfe des *rectangle*-Befehls gemacht, dem zudem eine hellgraue Farbe verliehen wird.

```

1 % Grundboden - grau
2 boden = rectangle('Position', [-0.75 -0.75 1.5 1.5])
3 boden.FaceColor = [0.9 0.9 0.9];

```

Quellcode 9: Erschaffung des Grundbodens

Des Weiteren werden drei Ablegestellen als rechteckige Markierungen auf dem Grundboden erstellt. Dies wird ebenfalls mit dem *rectangle*-Befehl bewerkstelligt. Die Ablegemarkierungen werden zudem mit einer individuellen Farbe eingefärbt. Dabei sind die Maßen der Ablegestellen variabel.

```

1 % Eigenschaften der einzelnen Ablegemarkierungen
2 am1_x = -0.6; % x Position
3 am1_y = 0.5; % y Position
4 am1_b = 0.25; % b Breite
5 am1_h = 0.1; % h Hoehe
6

```

```

7 am2_x = -0.125;
8 am2_y = 0.5;
9 am2_b = 0.25;
10 am2_h = 0.1;
11
12 am3_x = 0.35;
13 am3_y = 0.5;
14 am3_b = 0.25;
15 am3_h = 0.1;

```

Quellcode 10: Eigenschaften der Ablegestellen

```

1 % Ablegemarkierung 1 - rot
2 ablege_markierung_1 = rectangle('Position', [am1_x am1_y am1_b am1_h])
3 ablege_markierung_1.FaceColor = 'red';
4 hold on
5
6 % Ablegemarkierung 2 - gelb
7 ablege_markierung_2 = rectangle('Position', [am2_x am2_y am2_b am2_h])
8 ablege_markierung_2.FaceColor = 'yellow';
9 hold on
10
11 % Ablegemarkierung 3 - gruen
12 ablege_markierung_3 = rectangle('Position', [am3_x am3_y am3_b am3_h])
13 ablege_markierung_3.FaceColor = 'green';

```

Quellcode 11: Erstellung der Ablegestellen

Nun fehlen nur noch die Kisten bzw. Boxen, die der Roboterarm transportieren soll. Diese werden in 3D wie folgt erzeugt, wobei jede Kiste variable Koordinaten zugewiesen bekommt und mit einer individuellen Größe charakterisiert wird:

```

1 % Eigenschaften der einzelnen Kisten
2 kiste1_groesse = 0.07;
3 kiste2_groesse = 0.07;
4 kiste3_groesse = 0.07;
5
6 k1_x = -0.25;
7 k1_y = -0.1;
8 k1_z = 0;
9 %k1_b = 0.07; % Breite -> fuer das Erstellen einer 2D-Kiste 1
10 %k1_h = 0.07; % Hoehe -> fuer das Erstellen einer 2D-Kiste 1
11
12 k2_x = 0;
13 k2_y = -0.5;
14 k2_z = 0;
15 %k2_b = 0.07; % Breite -> 2D-Kiste 2
16 %k2_h = 0.07; % Hoehe -> 2D-Kiste 2
17
18 k3_x = 0.5;
19 k3_y = -0.25;
20 k3_z = 0;
21 %k3_b = 0.07; % Breite -> 2D-Kiste 3
22 %k3_h = 0.07; % Hoehe -> 2D-Kiste 3

```

Quellcode 12: Größen und Koordinaten der Kisten

```

1 % Kiste 1 - rot
2 % in 2D
3 %kiste_1 = rectangle('Position', [k1_x k1_y k1_b k1_h])
4 %kiste_1.FaceColor = 'red';
5
6 % in 3D

```

```

7      vert = [0 0 0; kiste1_groesse 0 0; kiste1_groesse kiste1_groesse 0; 0 kiste1_groesse 0; 0 0
8          kiste1_groesse; kiste1_groesse 0 kiste1_groesse; kiste1_groesse kiste1_groesse
9          kiste1_groesse; 0 kiste1_groesse kiste1_groesse];
10     fac = [1 2 6 5;2 3 7 6;3 4 8 7;4 1 5 8;1 2 3 4;5 6 7 8];
11     kiste_1 = hgtransform;
12     patch('Vertices',vert,'Faces',fac, 'FaceVertexCData',hsv(6),'FaceColor','red', 'Parent', kiste_1
13         );
14     kiste_1.Matrix = makehgform('translate',[k1_x, k1_y, k1_z]);
15     hold on
16
17 % Kiste 2 - gelb
18 % in 2D
19 %kiste_2 = rectangle('Position', [k2_x k2_y k2_b k2_h])
20 %kiste_2.FaceColor = 'yellow';
21
22 % in 3D
23 vert = [0 0 0; kiste2_groesse 0 0; kiste2_groesse kiste2_groesse 0; 0 kiste2_groesse 0; 0 0
24     kiste2_groesse; kiste2_groesse 0 kiste2_groesse; kiste2_groesse kiste2_groesse
25     kiste2_groesse; 0 kiste2_groesse kiste2_groesse];
26     fac = [1 2 6 5;2 3 7 6;3 4 8 7;4 1 5 8;1 2 3 4;5 6 7 8];
27     kiste_2 = hgtransform;
28     patch('Vertices',vert,'Faces',fac, 'FaceVertexCData',hsv(6),'FaceColor','yellow', 'Parent',
29         kiste_2);
30     kiste_2.Matrix = makehgform('translate',[k2_x, k2_y, k2_z]);
31     hold on
32
33 % Kiste 3 - gruen
34 % in 2D
35 %kiste_3 = rectangle('Position', [k3_x k3_y k3_b k3_h])
36 %kiste_3.FaceColor = 'green';
37
38 % in 3D
39 vert = [0 0 0; kiste3_groesse 0 0; kiste3_groesse kiste3_groesse 0; 0 kiste3_groesse 0; 0 0
40     kiste3_groesse; kiste3_groesse 0 kiste3_groesse; kiste3_groesse kiste3_groesse
41     kiste3_groesse; 0 kiste3_groesse kiste3_groesse];
42     fac = [1 2 6 5;2 3 7 6;3 4 8 7;4 1 5 8;1 2 3 4;5 6 7 8];
43     kiste_3 = hgtransform;
44     patch('Vertices',vert,'Faces',fac, 'FaceVertexCData',hsv(6),'FaceColor','green', 'Parent',
45         kiste_3);
46     kiste_3.Matrix = makehgform('translate',[k3_x, k3_y, k3_z]);

```

Quellcode 13: Anlegen der Kisten

### 3.3 Bewegung des Roboterarms

Damit der Roboterarm weiß, wie er sich zu verhalten hat und welche Strecken er abfahren muss, müssen Pfade berechnet werden.

Dementsprechend wird ein Pfad generiert. Dafür wird zunächst ein Zeitschritt  $t$  mit Hilfe des Befehls *linspace()* festgelegt, wobei eine hohe Zahl an Schritten festgelegt wird, sodass die Bewegung des Roboterarms flüssig aussieht. Zusätzlich wird ein *zaehler* anhand der Länge des Zeitschritts  $t$  definiert, der später beim Erstellen der Konfigurationslösungen weitere Verwendung findet.

```

1 t = linspace(0,1,50); % Zeitschritt
2 zaehler = length(t);

```

Quellcode 14: Festlegung des Zeitschritts  $t$

Um einen Pfad im Weiteren berechnen zu können, müssen zunächst die Punkte kalkuliert werden, die der Roboterarm auf seiner Strecke anfahren muss. Hierfür wird jeweils der Mittelpunkt jeder gegebenen Kiste und Ablegestelle berechnet. Die allgemeine Formel zur Berechnung eines Mittelpunktes

lautet:

$$[x, y, z] = [(x_1 + x_2)/2 \quad (y_1 + y_2)/2 \quad (z_1 + z_2)/2] \quad (2)$$

Da auf der 2D-Ebene gearbeitet wird, behält die Koordinate  $z$  immer den Wert 0. Zudem wird der Ausgangspunkt des Endeffektors definiert.

```

1  ursprungspunkt_endeffektor = [0 0 0];
2
3  % Mittelpunkte der Ablegemarkierungen
4  mittelpunkt_a_m_1 = [(am1_x + (am1_x + am1_b))/2 (am1_y + (am1_y + am1_h))/2 0]; % Mittelpunkt von
   Ablegemarkierung 1
5  mittelpunkt_a_m_2 = [(am2_x + (am2_x + am2_b))/2 (am2_y + (am2_y + am2_h))/2 0]; % Mittelpunkt von
   Ablegemarkierung 2
6  mittelpunkt_a_m_3 = [(am3_x + (am3_x + am3_b))/2 (am3_y + (am3_y + am3_h))/2 0]; % Mittelpunkt von
   Ablegemarkierung 3
7
8  % Mittelpunkte der Kisten
9  mittelpunkt_k_1 = [(k1_x + (k1_x + k1_b))/2 (k1_y + (k1_y + k1_h))/2 0]; % Mittelpunkt von Kiste 1
10 mittelpunkt_k_2 = [(k2_x + (k2_x + k2_b))/2 (k2_y + (k2_y + k2_h))/2 0]; % Mittelpunkt von Kiste 2
11 mittelpunkt_k_3 = [(k3_x + (k3_x + k3_b))/2 (k3_y + (k3_y + k3_h))/2 0]; % Mittelpunkt von Kiste 3

```

Quellcode 15: Denierung des Ausgangspunkts des Endeffektors und Berechnung aller nötigen Mittelpunkte zur Pfadfindung

Alle nötigen Punkte, die sich im Pfad des Roboterarms befinden sollen, sind nun bekannt. Somit werden sie in ein Array *punkte* in einer bestimmten Reihenfolge eingetragen.

```

1  % Punkte, die der Roboterarm ansteuert
2  punkte = [ursprungspunkt_endeffektor; mittelpunkt_k_1; mittelpunkt_a_m_1; ursprungspunkt_endeffektor
   ; mittelpunkt_k_2; mittelpunkt_a_m_2; ursprungspunkt_endeffektor; mittelpunkt_k_3;
   mittelpunkt_a_m_3; ursprungspunkt_endeffektor]

```

Quellcode 16: Auf dem Pfad liegende Punkte

Nun muss ein Pfad gefunden werden, an dem sich der Endeffektor entlang bewegen kann. Hierzu wird ein Objekt der Klasse „inverseKinematics“ erzeugt, das dem Roboterarm eine Konfigurationslösung, d. h. einen Pfad, finden soll. Ebenso wird eine Ausgangskonfiguration *ursprungsKonfiguration* definiert, wie auch Konfigurationslösungen als Matrix *matrixKonfigurationsloesungen* vordefiniert.

```

1  ursprungsKonfiguration = homeConfiguration(roboterarm);
2  ndof = length(ursprungsKonfiguration);
3  matrixKonfigurationsloesungen = zeros(zahler, ndof);

```

Quellcode 17: Vordefinition von Konfigurationslösungen in einer Matrix

Anschließend wird ein Berechner generiert, der die inverse Kinematik berechnen soll.

```

1  % Kreieren des Berechners fuer die inverse Kinematik
2  inverseKinematik = inverseKinematics('RigidBodyTree', roboterarm);
3  gewichtung = [0, 0, 0, 1, 1, 0]; % weights
4  endeffektor = 'endEtool';

```

Quellcode 18: Berechner der inversen Kinematik

Mit Hilfe einer *for*-Schleife werden die Punkte, die zuvor in das Array *punkte* eingetragen wurden, im Zusammenhang mit dem Zeitschritt  $t$  für jeden Schritt neu berechnet, um den Pfad mit ergänzenden Punkten zwischen den gegebenen Zielpunkten zu füllen und damit die Bewegung des Roboterarms flüssiger zu gestalten. Daraus lässt sich folgende mathematische Formel herleiten:

$$P = p_1 + (p_2 - p_1) * t \quad (3)$$

```

1 altePunkte=punkte;
2 for i = 1:length(punkte)-1
3     diff=altePunkte(i+1,:)-altePunkte(i,:);
4     for j=1:length(t)
5         punkte((i-1)*length(t)+j,:)=altePunkte(i,:)+diff*t(j);
6     end
7 end

```

Quellcode 19: Berechnung der aktuellen Punkte für jeden Zeitschritt  $t$

Daraufhin wird das Objekt *inverseKinematik* für jeden Punkt erneut aufgerufen, um die Gelenkkonfiguration zu generieren, sodass mit dieser die Endeffektorposition erreicht werden kann.

```

1 initiale_konfiguration = ursprungsKonfiguration; % Initialisierung mithilfe der
    Ursprungskonfiguration
2 for i = 1:length(punkte)-1
3     % Loesung fuer die Konfiguration, die die gewuenschte
    % Endeffektorposition erfuellt
4     punkt = punkte(i,:);
5     konfigurations_loesung = inverseKinematik(endeffektor, trvec2tform(punkt), gewichtung,
        initiale_konfiguration);
    % Konfiguration speichern
8     matrixKonfigurationsloesungen(i,:) = konfigurations_loesung;
9     % mit vorheriger Loesung starten
10    initiale_konfiguration = konfigurations_loesung;
11 end

```

Quellcode 20: Erstellung der Gelenkkonfiguration

Nun muss die Roboterarmbewegung bzw. die Konfigurationslösungen zum Erstellen eines Pfades noch animiert werden. Dafür wird mit dem Befehl *figure* ein Fenster generiert und mit *show()* der Roboterarm für jede Konfigurationslösung geplottet. Mit dem Befehl *view()* kann die Perspektive ausgewählt werden, aus welcher das Szenario gezeigt werden soll. Außerdem wird die Ansicht der Achsen angepasst.

```

1 % Loesung animieren
2 figure
3 show(roboterarm,matrixKonfigurationsloesungen(1,:)); % Roboter fuer jeden Rahmen der Lesung unter
    Verwendung spezifischer Roboterkonfiguration plotten
4 view(2) % Perspektive --> frontview
5 % view(3) % Perspektive --> sideview
6 ax = gca; % gibt die aktuellen Achsen fuer die aktuelle Figure zurueck
7 ax.Projection = 'orthographic'; % die Art wie man die Achsen anzeigen laesst --> Perspektive, hier:
        senkrecht
8 hold on

```

Quellcode 21: Konfigurationslösungen animieren

Die Schnelligkeit des Pfadabfahrens des Roboterarms kann auch wie folgt angepasst werden:

```

1 % Schnelligkeit des Pfadabfahrens vom Roboterarm einstellen
2 frames_pro_sekunde = 30;
3 rate_kontrolle = rateControl(frames_pro_sekunde);

```

Quellcode 22: Anpassen der Schnelligkeit der Roboterarmbewegung

Zudem wird der Roboterarm bei jedem Zeitschritt mit Hilfe des Befehls *show()* auf der aktuellen Position angezeigt, indem eine *for*-Schleife durchlaufen wird.

### 3.4 Transport der Kisten

Das Transportieren der Kisten wird im letzten Schritt der Implementierung kreiert. Drei boolesche Werte werden erzeugt und als *false* initialisiert. Eines der booleschen Werte bezieht sich genau auf eine der Kisten und überprüft, ob der Roboterarm im aktuellen Moment die Kiste festhält oder nicht. Dies wird anhand der aktuellen Position des Endeffektors und des Mittelpunktes der einzelnen Kisten überprüft, die miteinander verglichen werden. Sobald beide Werte aufeinander zutreffen, wird der boolesche Wert der jeweiligen Kiste auf *true* gesetzt, wobei diese Kiste sich mit dem Roboterarm zur entsprechenden Ablegestelle bewegt.

```

1 % Transport der Kisten
2 holdBox1=false;
3 holdBox2=false;
4 holdBox3=false;
5 for i = 1:length(punkte)-1
6     show(roboterarm, matrixKonfigurationsloesungen(i,:)', 'PreservePlot', false); % Roboterarm in
7         jedem Durchlauf auf der aktuellen Position angezeigt
8     if(punkte(i,:) == mittelpunkt_k_1)
9         holdBox1=true;
10    elseif(punkte(i,:) == mittelpunkt_k_2)
11        holdBox2=true;
12    elseif(punkte(i,:) == mittelpunkt_k_3)
13        holdBox3=true;
14    elseif(punkte(i,:) == mittelpunkt_a_m_1)
15        holdBox1=false;
16    elseif(punkte(i,:) == mittelpunkt_a_m_2)
17        holdBox2=false;
18    elseif(punkte(i,:) == mittelpunkt_a_m_3)
19        holdBox3=false;
20    end
21    if(holdBox1)
22        kiste_1.Matrix = makehgtform('translate',[punkte(i,1)-kiste1_groesse/2,punkte(i,2)-
23            kiste1_groesse/2,0]);
24    elseif(holdBox2)
25        kiste_2.Matrix = makehgtform('translate',[punkte(i,1)-kiste2_groesse/2,punkte(i,2)-
26            kiste2_groesse/2,0]);
27    elseif(holdBox3)
28        kiste_3.Matrix = makehgtform('translate',[punkte(i,1)-kiste3_groesse/2,punkte(i,2)-
            kiste3_groesse/2,0]);
29    end
30    drawnow
31    waitfor(rate_kontrolle); % Pausieren der Ausfuehrung bis der Code die gewuenschte
32        Ausfuehrungsrate erreicht

```

Quellcode 23: Transport der Kisten

## 4 Numerische Beispiele

Folgende Beispieldaten wurden verwendet:

Grundboden (grau):  $x = -0.75, y = -0.75, b = 1.5, h = 1.5$

Ablegemarkierung 1 (rot):  $x = -0.6, y = 0.5, b = 0.25, h = 0.1$

Ablegemarkierung 2 (gelb):  $x = -0.125, y = 0.5, b = 0.25, h = 0.1$

Ablegemarkierung 3 (grün):  $x = 0.35, y = 0.5, b = 0.25, h = 0.1$

Kiste 1 (rot):  $x = -0.25, y = -0.1, z = 0, grösse = 0.07$

Kiste 2 (gelb):  $x = 0, y = -0.5, z = 0, \text{ grösse} = 0.07$   
Kiste 3 (grün):  $x = 0.5, y = -0.25, z = 0, \text{ grösse} = 0.07$

Armglied 1: längte = 0.4

Armglied 2: längte = 0.4

Armglied 3: längte = 0.4

Maximale Distanz: längte = 0.6

Anhand der oben genannten Beispieldaten konnte der Roboterarm und seine Aufgabendurchführung gut dargestellt werden. Die Armlänge des Roboterarms ist ausreichend, um an jedes Zielobjekt zu gelangen, das heißt die definierte maximale Distanz kann dementsprechend erreicht werden. Außerdem besitzen die Kisten eine passende Größe, sodass sie in die Ablegemarkierungen bzw. Ablegestellen reinpassen. Dies ist in Abbildung 1 und Abbildung 2 ersichtlich.

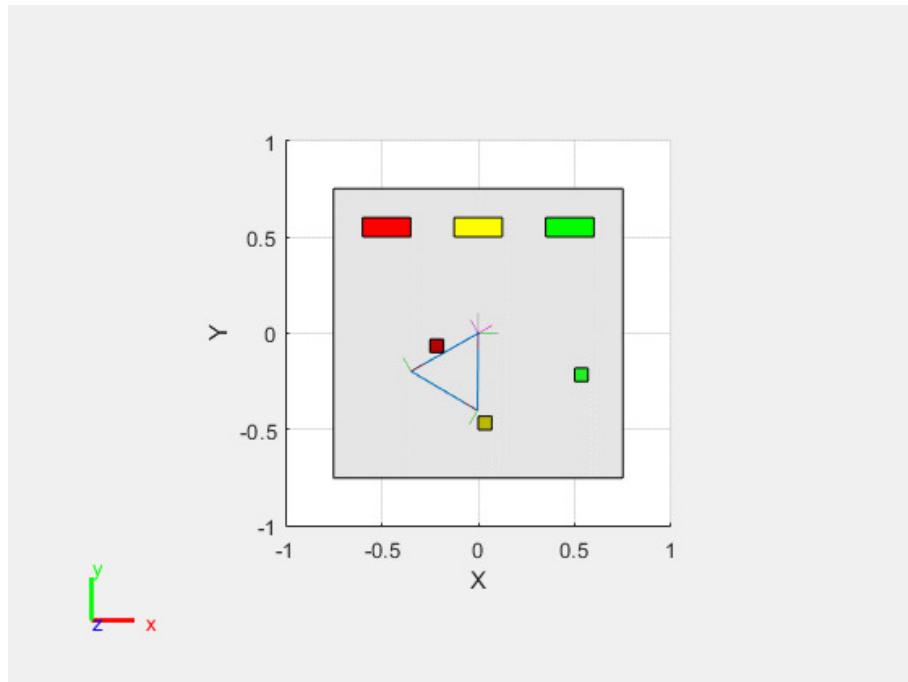


Abbildung 1: Roboterarm - Perspektive: Vorderseite - Anfangsposition

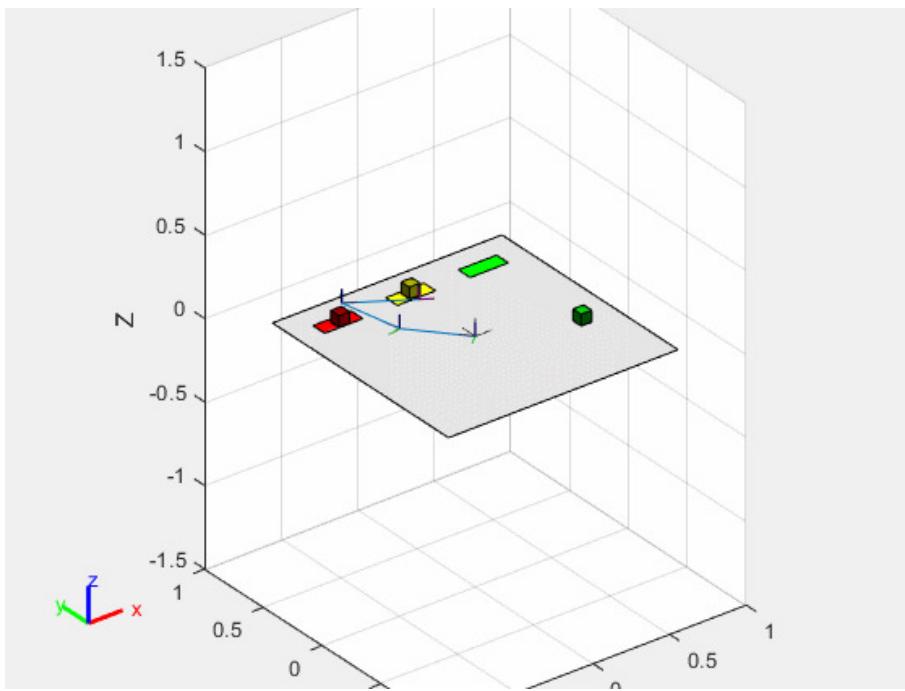


Abbildung 2: Roboterarm - Perspektive: Seitenansicht - während der Durchführung

## 5 Fazit und Ausblick

Der Roboterarm und seine Bewegung mit Hilfe der Pfadplanung, wie auch das Transportieren der Kisten auf die zugehörigen Ablegestellen, konnte erfolgreich implementiert werden.

Nichtsdestotrotz könnte das Programm in einer weiteren Ausarbeitung mit zusätzlichen Features erweitert werden. So könnte zum Beispiel eine Kollisionserkennung eingebaut werden, die Hindernisse erkennen würde und demnach den Pfad neu berechnen lassen würde, um den Hindernissen auszuweichen.

Des Weiteren könnte die Farberkennung hinzukommen, welche bei einer größeren Zahl von Kisten sinnvoll wäre, womit die Kisten vom Roboterarm auf die richtigen gleichfarbigen Ablegestellen sortiert werden könnten. Außerdem könnte die Darstellung aller Elemente, wie auch des Roboterarms, von 2D auf 3D ausgebaut werden.

## Literatur

- [1] Ruediger Dillmann und Tamim Asfour Dominik Bertram James Kuffner. *An Integrated Approach to Inverse Kinematics and Path Planning for Redundant Manipulators*. URL: [https://www.researchgate.net/publication/221076968\\_An\\_Integrated\\_Approach\\_to\\_Inverse\\_Kinematics\\_and\\_Path\\_Planning\\_for\\_Redundant\\_Manipulators](https://www.researchgate.net/publication/221076968_An_Integrated_Approach_to_Inverse_Kinematics_and_Path_Planning_for_Redundant_Manipulators). (zuletzt besucht: 01.07.2020).
- [2] Prof. Martin Hering-Bertram. *Kapitel 2: Simulation von Robotern*. URL: [https://aulis.hs-bremen.de/goto.php?target=file\\_1159349\\_download&client\\_id=hsbremen](https://aulis.hs-bremen.de/goto.php?target=file_1159349_download&client_id=hsbremen). (zuletzt besucht: 06.07.2020).
- [3] Univ. Prof. Dr. Manfred L. Husty. *Implementierung eines neuartigen, effizienten Algorithmus zur Berechnung der inversen Kinematik von seriellen Robotern mit Drehgelenken*. URL: <http://geometrie.uibk.ac.at/cms/datastore/husty/husty-linz.pdf>. (zuletzt besucht: 01.07.2020).
- [4] mathworks.com. *2-D Path Tracing With Inverse Kinematics*. URL: <https://de.mathworks.com/help/robotics/ug/2d-inverse-kinematics-example.html>. (zuletzt besucht: 01.07.2020).
- [5] Jean-Claude Latombe Mitul Saha Tim Roughgarden. *Planning Tours of Robotic Arms among Partitioned Goals*. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.833.3961&rep=rep1&type=pdf>. (zuletzt besucht: 01.07.2020).
- [6] *Roboterkinematik: Roboterarm und Gelenke*. URL: [https://www.tu-chemnitz.de/informatik/KI/edu/robotik/ws2012/robotik\\_4.pdf](https://www.tu-chemnitz.de/informatik/KI/edu/robotik/ws2012/robotik_4.pdf). (zuletzt besucht: 01.07.2020).
- [7] MathWorks Toolbox. *Robotics System Toolbox*. URL: <https://de.mathworks.com/products/robotics.html>. (zuletzt besucht: 06.07.2020).