

Grundkurs Matlab  
Abschlussprojekt SS2021

App zur Simulation der Bewegung eines  
2-Arm Roboters

Datum 31.07.2021

Gruppe I

Eugen Risling: Matr. 2017507939

Georg Kulodzik Matr. 2017507931

Paul Freynik Matr. 2017507935

Ronny Tolloch Matr. 185631

## Inhalt

Inhalt.....	2
1. Einführung.....	3
1.1 Beschreibung der Aufgabe (in eigenen Worten).....	3
1.2 Ziel des Projektes: Was ist das Ziel der App.....	3
1.3 Struktur der Dokumentation: Erläuterung des Inhaltes der Folgekapiteln.....	4
2. Beschreibung der App.....	5
2.1 Bild von der App.....	5
2.2 Beschreibung eurer App: Also was passiert, wenn man die Buttons betätigt usw.....	6
3. Beschreibung des Programms:.....	7
3.1 Programmstruktur:.....	8
3.2 Kinematik (P1):.....	9
3.3 Bahnplanung (P2):.....	12
3.4 Dynamik (P3):.....	15
3.5 App (P4):.....	22
4. Diskussion der Ergebnisse.....	36
4.1 Zeigt einen Beispielaufruf eurer App und präsentiert und diskutiert das Gesamtergebnis.....	36
4.2 Ergebnisse zu den Analysen zum Teil Kinematik (P1).....	39
4.3 Ergebnisse zu den Analysen Zum Teil Bahnplanung (P2).....	40
4.4 Ergebnisse zu den Analysen zum Teil Dynamik (P3).....	43
4.5 Ergebnisse zu den Analysen zum Teil App (P4).....	46
5. Zusammenfassung und Ausblick.....	48
5.1 Zusammenfassung:.....	48
5.2 Ausblick:.....	49

# 1. Einführung

## 1.1 Beschreibung der Aufgabe (in eigenen Worten)

Das Projekt App zur Simulation der Bewegung eines 2-Arm Roboters soll die Bewegung eines Roboters simulieren. Hierfür werden die Bewegungen der Roboterarme mit Verschiedenen Methoden simuliert. Dabei ist es möglich den Roboter über die Angabe von Winkeln, aber auch durch die Angabe von festen XY Koordinaten zu steuern. Der Weg von der Anfangs- zur Endposition wird über diese Matlab App errechnet und grafisch dargestellt.

Das Thema ist aus dem Bereich der Robotik, bei dem die Steuerung der Roboter im Vordergrund steht. Ohne eine solche Berechnung wäre es für Roboter nicht möglich eine Vielzahl von Positionen, ohne eine direkte Weg Beschreibung anzufahren. Da der Roboter aus zwei Armen und einen TCP (Tool Center Point) besteht, hat die Bewegung (Winkeländerung) von Arm 1 auch immer eine Auswirkung auf den Arm 2. Diese beiden Positionen (Winkel) zueinander müssen über die gesamte Strecke aufeinander abgestimmt (Berechnet) werden.

Das Projekt wird aufgrund des Umfangs von vier Personen durchgeführt. Hierfür sind die einzelnen Aufgaben schon vordefiniert und bestehen aus mehreren Funktionen, mit vorgegebener Übergabe und Rückgabe Variablen. Durch diese Struktur ist es möglich selbstständig an einer Funktion zu arbeiten, ohne dass dafür schon andere Teile des Projekts fertig sein müssen. Kern des Projekts ist die App "Robo", mit der die Funktionen sowie das Simulink Modell aufgerufen werden. Eine Übersicht wie auf die einzelnen Funktionen zugegriffen wird bietet die Übersicht unter Punkt 3.1.

## 1.2 Ziel des Projektes: Was ist das Ziel der App

Das Ziel des Projekts ist zum einen eine Funktionierende App, die die Bewegung eines zwei Arm Roboters simulieren kann. Dabei gibt es mehrere Wege, um ans Ziel zu kommen. Hier wird auf die Kinematik in vorwärts und rückwärts Bewegung eingegangen. Die Rückwärtsbewegung kann sowohl in der Numerischen- sowie in der Symbolischen Mathematik berechnet werden. Dabei kommt es zu unterschiedlichen Ergebnissen, was im Kapitel 3 näher erläutert wird. Die Endposition des Roboterarms kann über die Eingabe von Winkeln, sowie auch über die Eingabe einer genauen Position mit xy Koordinaten, vorgegeben werden.

Mit dieser App kann simuliert werden, welche Positionen für den Roboter erreichbar sind, sowie auch eine Zeitliche Vorstellung, wie lange der Roboterarm von einer Position zur anderen benötigt. Dieses ist möglich, da hier auch Die Masse, Trägheit und Reibung der Arme mitberücksichtigt werden.

Neben dem Programmieren ist auch das Arbeiten im Projekt ein wichtiger Teil der Aufgabe. Hierfür müssen die Aufgaben auf die Gruppenmitglieder verteilt werden, sowie auch Zeitliche Abstimmung mit festgesetzten Terminen und Absprache wie das Projekt zusammengeführt werden. Dafür ist auch das Schaubild in Kapitel 3.1 eine gute Unterstützung. Das Bild hilft beim Verstehen des Programms, sowie auch bei der Einteilung der einzelnen Aufgaben mit ihren Schnittstellen.

### 1.3 Struktur der Dokumentation: Erläuterung des Inhaltes der Folgekapiteln.

Die Dokumentation ist so aufgebaut, dass auf jeden Bereich des Projekts einzeln eingegangen werden soll. Auch wenn die Aufgaben klar einer einzelnen Person zugeordnet wurden, so wurde dennoch in Teilen zusammen an der Lösung gearbeitet.

Im Kapitel 2 ist die App, die ist der Kern des Programms. Mit dem Befehl Robo kann die App über das Command Window gestartet werden.

Im Kapitel 3 werden die einzelnen Funktionen beschrieben. Hierbei findet sich im Kapitel 3.1 ein Schaubild, was die Zusammenhänge des Programms beschreibt.

Im Kapitel 4 wird auf die fertige App, unser Ergebnis, eingegangen. Zusätzlich werden hier die Analysen zu einzelnen Fragestellungen erläutert. Anhand der Analysen lässt sich besser verstehen, welche Wirkungen einzelne Parameter auf das Ergebnis haben. Somit kann nochmal kritisch das erreichte beurteilt werden.

Kapitel 5 befasst sich mit dem erreichten Ziel, Einschätzung des Ergebnisses und einer Aussicht, was noch an Verbesserungen bei der App möglich wären.

## 2. Beschreibung der App

### 2.1 Bild von der App

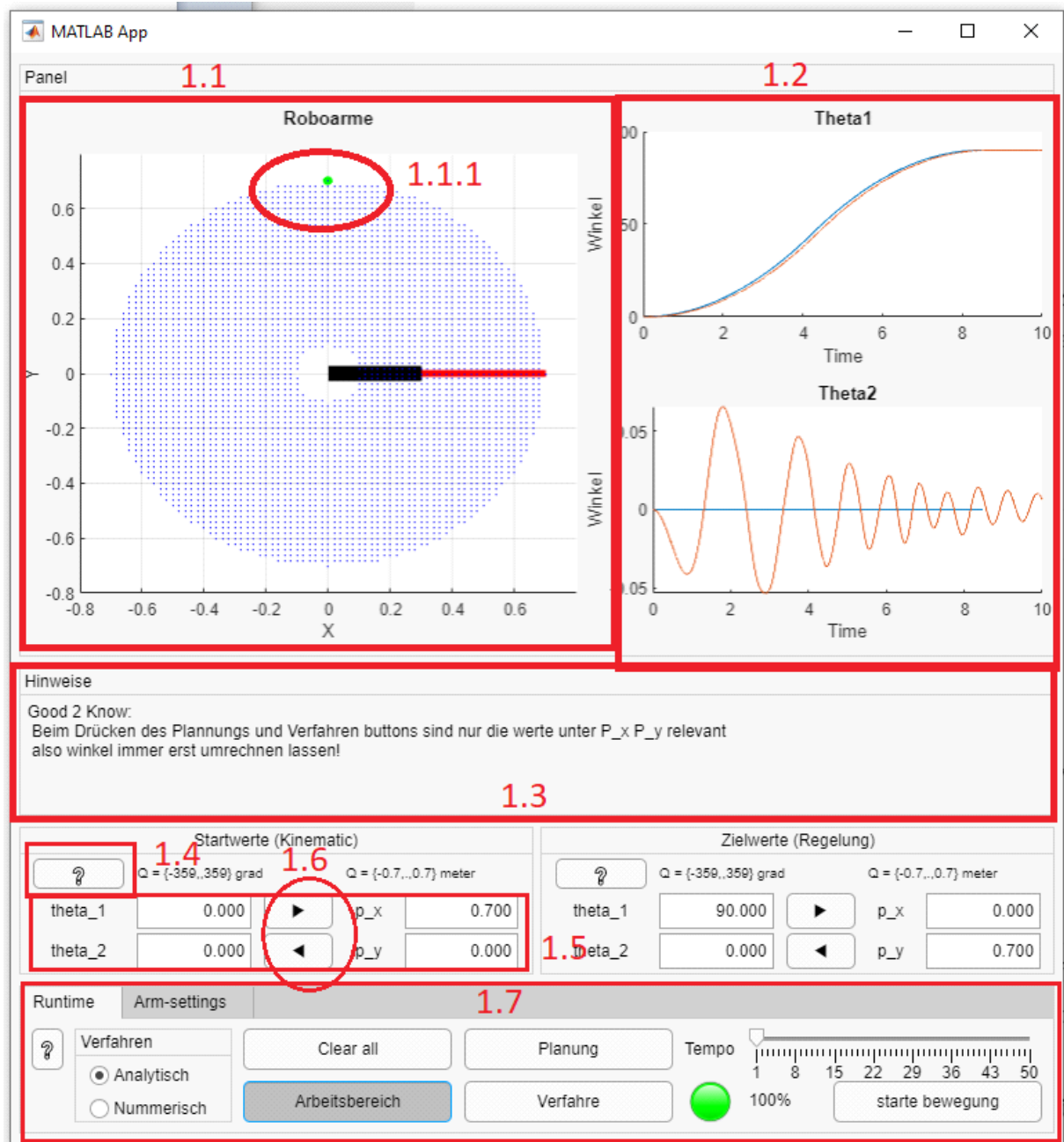


Abb. 2.1.1: Roboarme Plot

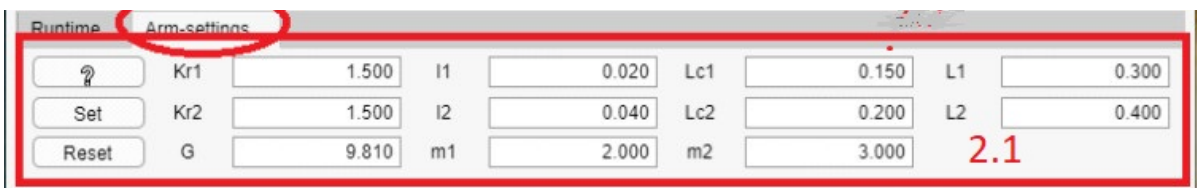




Abb. 2.2.1: Roboarme Plot Arm Setting

## 2.2 Beschreibung eurer App: Also was passiert, wenn man die Buttons betätigt usw.

Die Abbildung 2.2.1 zeigt die 2 Roboterarme (1.1 schwarz und rot) grafisch in einem Plot.

- der grüne Punkt (1.1.1) indiziert den Zielpunkt wohin der Roboterarm hinsteuern würde, der blau gepunktete Bereich zeigt den Arbeitsbereich des Arms.
- Theta1/theta2 Plot (1.2) zeigt in blau die Winkeländerung, die nötig ist um von den Koordinaten px/py Start zum Ziel zu gelangen.
- Hinweise Feld (1.3) zeigt Informationen, Probleme und Tipps, die sich auf die Start- und Zielwerte beziehen.
- Hilfsbutton (1.4) gibt kurze Erklärung des Bereiches wieder.
- theta\_i gibt die Winkelwerte und p\_xy die Positionswerte an (1.5)
-   wandelt den Winkelwert in Positionswert und umgekehrt um (1.6)
- Runtime Sheet (1.7) beinhaltet folgende Punkte
  - o Hilfsbutton - gibt kurze Erklärung des Bereiches wieder
  - o Verfahren – dient zur Änderung der xy zu Winkel Methode (analytisch oder numerisch)
  - o Clearall - setzt alle Werte zu einer 0 Position
  - o Planung - berechnet die Winkeländerung in Bezug auf max speed beider arme
  - o Verfahre - "fahre" die berechnete Bahn ab (hier mit einem Simulink Modell)
  - o Tempo - zum Ändern der Simulationsschrittweite (diese Funktion ist von „Verfahre“ getrennt, da man auf langsame Rechner Rücksicht nehmen muss)
- Arm-settings (Abb.2.2.1 nur Ausschnitt)) hier können alle Geometrischen Roboterarmdaten angepasst werden, dieses Sheet besteht aus folgenden Feldern
  - o Hilfsbutton - gibt kurze Erklärung des Bereiches wieder
  - o Reset - setze alle Werte auf Standard zurück
  - o Set - aktiviert die eingesetzten Werte für die App
  - o Editierbare Spezifische Eingabefelder

### 3. Beschreibung des Programms:

Das Programm wird über die unter Punkt 2 erläuterte App bedient. Dabei ist das Hauptprogramm in der Matlab App unter ansprechen diverser Funktionen und Reaktionen (Eventbasiert) programmiert. In diesem Programm werden die Variablen definiert und die vorgegebenen Parameter (Abb. 3.5.2) deklariert. Das Programm benutzt für sämtliche Berechnungen und die Ausgabe der Plots eigene Funktionen. Somit dient die App als Schnittstelle um die Funktionen und Ausgaben verarbeiten zu können. Die Abarbeitung muss immer durch eine Eingabe über die App bestätigt werden, um eine entsprechende Reaktion ausführen zu können. Die Struktur und die jeweiligen Abhängigkeiten sind unter Abb.3.1.1 des Programms dargestellt. Durch das Verwenden mehrerer Funktionen einer Schachtelung ist das Programm einfacher zu realisieren und eine unabhängige Programmierung mehrere Personen möglich, die Unterprogramme können durch Beispielaufufe getestet werden, das vereinfacht die Fehersuche und macht es einfacher später Veränderungen am Programm vor zu nehmen.

### 3.1 Programmstruktur:

Erläuterung der Struktur des Programms anhand eines Schaubildes und Textes. Beschreibt, wie euer Programm aufgebaut ist. Welche Funktion wird von der App und von den Funktionen selbst aufgerufen (sprich: welche Abhängigkeiten existieren)

Die Abb.3.1.1 zeigt den Aufbau des Programms als Schaubild. Dabei ist die App das Hauptprogramm, welches gestartet wird, um das Programm nutzen zu können. In der App gibt es 10 interne Funktionen, die zum Beispiel durch Buttons aufgerufen werden. Zudem gibt es 7 separat programmierte Funktionen, sowie ein Simulink Modell.

Die Rückwärtskinematik sowie die Bahnplanung ist nur möglich, wenn davor der Arbeitsbereich geprüft wurde und die Richtigkeit mit einer „Flag“ (in WS==1) bestätigt wird. Dieses ist im Schaubild mit den grünen Pfeilen gekennzeichnet. Diese Funktionen werden einzeln über die App aufgerufen und kommunizieren nicht direkt miteinander rufen aber in den einzelnen Funktionen weitere Funktionen auf.

Der Roboterarm stellt den Verlauf der Vorwärts- und Rückwärtskinematik dar. Zwar wird dieses durch die App gestartet, dennoch ist es abhängig davon, dass davor eine der 3 Kinematik Funktionen ausgeführt wurden.

Die „fun\_moveRobo“ ist die einzige Funktion, die selbst direkt auf eine andere Funktion zugreift, in diesem Fall wird hier das Simulink Modell von der Funktion gestartet und ausgeführt.

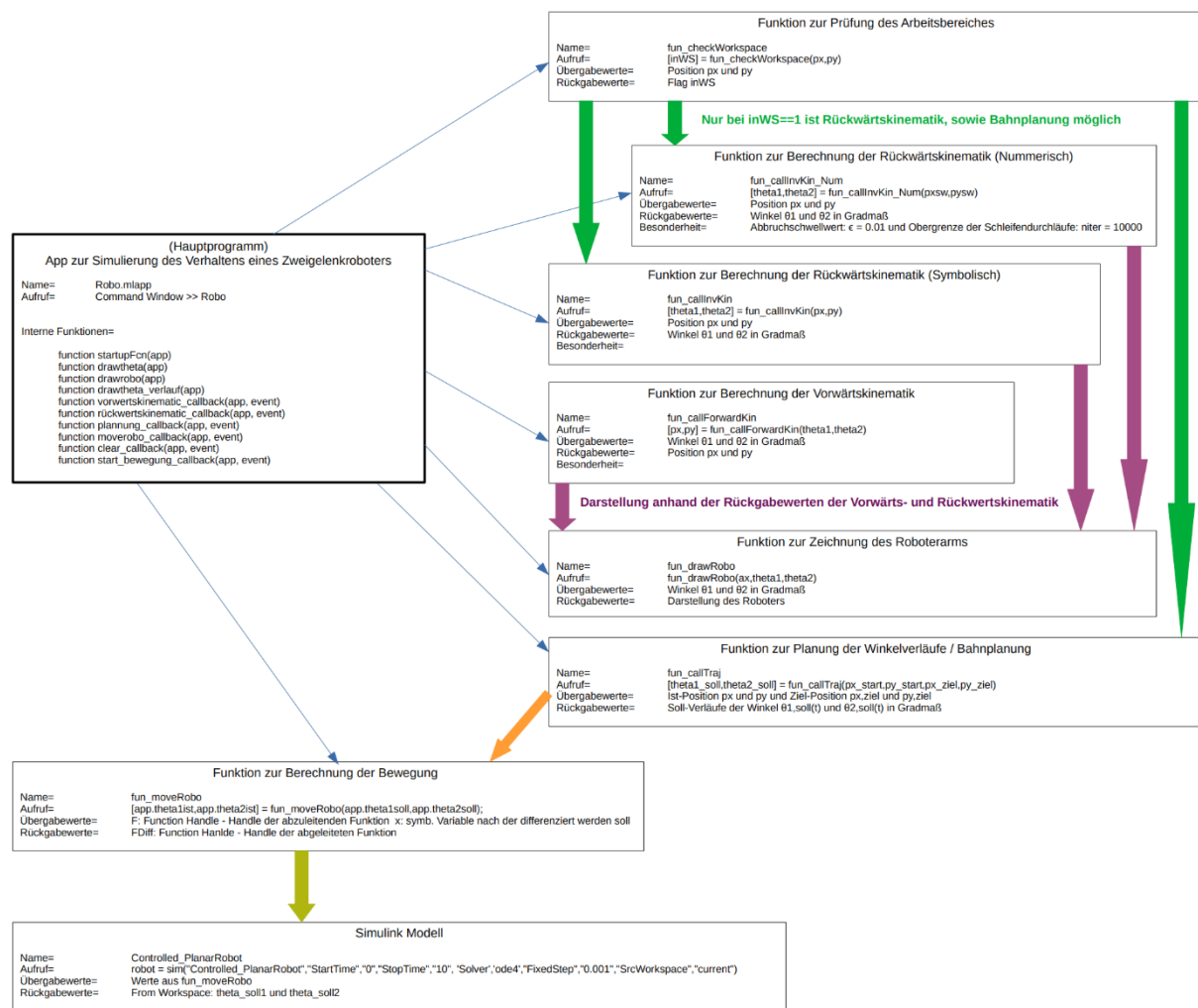


Abb. 3.1.1: Struktur Darstellung Abhängigkeiten der Funktionen



## 3.2 Kinematik (P1):

Autor: Eugen Risling: Matr.

2017507939

*Eventuelle Grundlagen und Vorüberlegungen zur Implementierung sind hier einzufügen, wie z.B. die Wiederholung der implementierten Gleichungen. Darstellung und Beschreibung des Codes.*

Bei der Kinematik, in unserem Fall der Vorwärtskinematik, handelt es sich um die Bewegung eines Körpers im Raum, diese lässt sich durch die Position, Bewegung und durch die Beschleunigung rein geometrisch beschreiben, in unserem Fall mit der Winkelfunktionen durch die Positionsbestimmungsformel (Abb. 3.2.1) wird die Position bestimmt.

Die Funktion hat folgende Aufgabe, Beschreibung Zeilenabschnittsweise:

- Zeile 1-10: Starten der Funktion und Initialisieren der statischen Werte
- Zeile 11-26: Vergleich des Arbeitsbereichs ob verfahren werden kann, wenn nicht wird ein Hinweis ausgegeben, wenn ja werden die Winkelwerte initialisiert

$$p_x = l_2 \cos(\theta_1 + \theta_2) + l_1 \cos(\theta_1) \quad (1)$$

$$p_y = l_2 \sin(\theta_1 + \theta_2) + l_1 \sin(\theta_1) \quad (2)$$

Abb. 3.2.1: Positionsbestimmungsgleichung

### fun\_callForwardKin

*Vorwärtskinematik.*

Die Funktion berechnet durch die vorgegebenen Gleichungen die Position des TCPs (px,py) im Raum.

Für die Winkel  $\theta_1$  und  $\theta_2$  müssen der Funktion dementsprechend jeweils ein Zahlenwert übergeben werden.

- Übergabeparameter: Winkel  $\theta_1$  und  $\theta_2$  in Gradmaß
- Rückgabewerte: Position px und py.

Beispielaufruf:

```
[px,py] = fun_callForwardKin(30,57)
```

Autor: Eugen Risling Matr.Nr. 2017507939

```
1 function [px,py] = fun_callForwardKin(theta1,theta2)
2     % Parameter für die Länge Armteil 1 und 2 des planarer Zweigelenkroboter
3     global l1 l2;
4
5     % Berechnung der TCP Position des Roboters mittels Gleichungen aus der Aufgabestellung für px und py
6     px = l2 * cosd(theta2+theta1) + l1 * cosd(theta1);
7     py = l2 * sind(theta2+theta1) + l1 * sind(theta1);
8 end
```

Abb. 3.2.2: Funktion „fun\_callForwardKin“

Bei der Inversen Kinetik (Rückwärtskinetik) versteht man die Berechnung der Winkel über die gegebene Lage des TCP's im Raum also umgekehrt (inverse) zu der Vorwärtskinetik.

Die Funktion „fun\_callInvKin“ hat folgende Aufgabe, Beschreibung Zeilenabschnittsweise:

- Zeile 3: Abrufen der Globalen Variablen
- Zeile 5-8: Umwandlung in Symbolische variable, berechnen der Koordinaten über die länge und der Winkelangabe
- Zeile 11-14: Auswahl der größeren Lösung
- Zeile 17-18: Ausgabe der Ergebnisse und umrechnen in Grad

### fun\_callInvKin

*Rückwärtskinematik.*

Durch die Lösung des Gleichungssystems wird die Funktion für die gegebene Lage (px,py) des TCPs, die Gelenkwinkel  $\theta_1$  und  $\theta_2$  berechnen.

Hier handelt es sich um symbolische Berechnung, die Ausgabe soll aber als reelle Zahlen zurückgegeben werden.

- Übergabeparameter: Position px und py
- Rückgabewerte: Winkel  $\theta_1$  und  $\theta_2$  in Gradmaß.

Beispielaufruf: [theta1,theta2] = fun\_callInvKin(0.3,0.5);

Autor: Eugen Risling Matr. Nr. 2017507939

```
1 function [theta1,theta2] = fun_callInvKin(px,py)
2 % Parameter für die Länge Armteil 1 und 2 des planarer Zweigelenkroboter
3 global l1 l2;
4
5 syms sym_theta1 sym_theta2;
6 px_func = l2 * cos(sym_theta1+sym_theta2) + l1 * cos(sym_theta1) == px;
7 py_func = l2 * sin(sym_theta1+sym_theta2) + l1 * sin(sym_theta1) == py;
8 [theta1,theta2] = solve([px_func,py_func],[sym_theta1,sym_theta2]);
9
10 % Es ist eine von zwei Lösungen für das Gleichungssystem auszuwählen
11 if size(theta1,1) == 2
12     theta1 = theta1(2);
13     theta2 = theta2(2);
14 end
15
16 % Umrechnen von rad zu grad
17 theta1 = double(rad2deg(theta1));
18 theta2 = double(rad2deg(theta2));
19 end
```

Abb. 3.2.2: Funktion „fun\_callInvKin“

Die Funktion „fun\_callInvKin\_Num“ hat folgende Aufgabe, Beschreibung Zeilenabschnittsweise:

- Zeile 3: Abrufen der Globalen Variablen
- Zeile 6-9: setzen der Startparameter
- Zeile 12-13: Füllen der Jacobi Matrix
- Zeile 15-26: for-Schleife mit 1000 Durchgängen, Berechnung der Position, Bildung der Differenz zur Sollposition, Abbruchbedingung entweder, wenn 1000 Durchläufe oder innerhalb der Toleranz um die gesuchte Position befindet
- Zeile 29: Erstellung der Matrix
- Zeile 32-33: Erstellen der neuen Winkel
- Zeile 39-40: Ausgabe der Winkel

$$\mathbf{J}(\theta_1, \theta_2) = \begin{pmatrix} -l_2 \sin(\theta_1 + \theta_2) - l_1 \sin(\theta_1) & -l_2 \sin(\theta_1 + \theta_2) \\ l_2 \cos(\theta_1 + \theta_2) + l_1 \cos(\theta_1) & -l_2 \cos(\theta_1 + \theta_2) \end{pmatrix}.$$

Abb. 3.2.3: Jacobi-Matrix

```

Funktion fun_callInvKin_Num
Rückwärtskinematik:
Die Funktion berechnet für eine gegebene Position (px,py) des TCPs, die Gelenkwinkel θ1 und θ2 durch eine Numerische Berechnung der inversen Kinematik..

• Übergabeparameter: Position px und py.
• Rückgabewerte: Winkel θ1 und θ2 in Gradmaß.

Beispielaufwurf:

[theta1, theta2] = fun_callInvKin_Num(0.3,0.5)

Autor: Eugen Rising Matr. Nr. 2017507939

1 function [theta1,theta2] = fun_callInvKin_Num(pxsw,pysw)
2 % Parameter für die Länge Armeil 1 und 2 des planarer Zweigelenkroboter
3 global l1 l2;
4
5 % startparameter
6 n = 100000; % max schleifen
7 epsilon = 0.001; % schwellwert
8 theta1 = 0; % startwinkel
9 theta2 = 0; % startwinkel
10
11 % Jacobi-Matrix zur berechnung der winkel
12 J = @(theta1p,theta2p) [-l2*sind(theta1p+theta2p) - l1 * sind(theta1p), - l2 * sind(theta1p+theta2p); ...
13                        l2*cosd(theta1p+theta2p) + l1 * cosd(theta1p), - l2 * cosd(theta1p+theta2p)];
14
15 for i = 1:n
16     % Berechnung der Position px und py des TCPs aus den Winkeln θ1 und θ2
17     [px,py] = fun_callForwardKin(theta1,theta2);
18
19     % Differenz zur Sollposition für dpx und dpy
20     dpx = pxsw - px;
21     dpy = pysw - py;
22
23     % Abbruchsbedingung für das Verfahren
24     if abs(dpx) <= epsilon && abs(dpy) <= epsilon
25         break;
26     end
27
28     % Winkeländerung
29     matrix = pinv(J(theta1,theta2))*[dpx;dpy];
30
31     % bestimmung der neuen winkel
32     theta1 = theta1 + matrix(1);
33     theta2 = theta2 + matrix(2);
34 end
35
36 % manchmal entstehen werte wie zum beispiel 400 das ist das selbe wie
37 % 40 grad + 1 volle umdrehung (360grad) , lesbarkeitshalber deswegen
38 % reduziert
39 theta1 = rem(theta1,360);
40 theta2 = rem(theta2,360);
41
42 end

```

Abb. 3.2.4: Funktion „fun\_callInvKin\_Num“

### 3.3 Bahnplanung (P2):

Autor: Paul Freynik Matr.

2017507935

*Eventuelle Grundlagen und Vorüberlegungen zur Implementierung sind hier einzufügen, wie z.B. die Begründung der Auswahl einer Lösung des Bahnplanungsalgorithmus. Darstellung und Beschreibung des Codes.*

Als Grundüberlegung der Methode zur Ermittlung der Bahnplanung war in erster Linie die Vorgabe der Aufgabenstellung unter Punkt 2.3, die besagt, dass die asynchrone PTP (Point-to-Point) Bahn ausgewählt werden sollte. Einige Bahnarten können aufgrund ihres Bahnverlaufs nicht verwendet werden, da diese durch den nicht möglichen Arbeitsbereich Abb.3.5.1 fahren würden.

Bei der Bahnplanung wird der Weg von dem Startpunkt zum Zielpunkt über die Winkelverläufe innerhalb der Endzeit berechnet. Hierbei ist es wünschenswert eine möglichst gerade Bahn abzufahren. Die Bahnplanung wird interpoliert, es werden Zwischenwerte berechnet, die zur Berechnung benötigten Werte bestehen aus der Winkelposition, der Winkelgeschwindigkeit und der Winkelbeschleunigung.

Die folgende Abbildung 3.3.1 stellt den gesamten Code der Funktion „fun\_callTraj“ dar.

Die Funktion besteht im Kopf aus einer Beschreibung der Funktion, welche Funktionalität hier erwartet wird, welche Übergabeparameter an diese Funktion übergeben werden müssen und ausgegeben werden, des Weiteren kann diese Funktion durch Eingabe einer Aufruffunktion getestet werden.

Die Funktion hat folgende Aufgabe, Beschreibung Zeilenabschnittsweise:

- Zeile 1-10: Starten der Funktion und Initialisieren der statischen Werte
- Zeile 11-26: Vergleich des Arbeitsbereichs ob verfahren werden kann, wenn nicht wird ein Hinweis ausgegeben, wenn ja werden die Winkelwerte initialisiert
- Zeile 27-36: Berechnen der Start und Zielwerte durch Aufruf der Funktion „fun\_callInv\_Kin“ und Rückgabe dieser an Theta1 und 2
- Zeile 37-47: Setzen der Beschleunigungsbegrenzung durch Implementierung der Gleichung „Gleichung Beschleunigungsbegrenzung“ Abb.2 des Funktionscodes
- Zeile 48-58: Berechnung der Endzeit und Übergangszeit durch Implementierung der Gleichung „Berechnung der Endzeit und Übergangszeit“ Abb.4 des Funktionscodes
- Zeile 59-84 Implementieren der Rechenvorschrift PTP-Bahn, Kernstück der Funktion, mit einer for-Schleife, die so oft wiederholt wird, wie die Länge der Zeit ist, hierbei wird von 0 bis  $t_e(i)$  Endzeit die Bedingungen geprüft
- Zeile 85-100 Hier wird das Resultat der gesamten Berechnung als Rückgabewert an die Funktion und entsprechend an die Visualisierung der App zurückgegeben!

## fun\_callTraj

### Bahnplanung:

Diese Funktion berechnet den Verlauf von einer aktuellen Position TCPs (Tool Center Point) (px, py) um zu einer gegebenen Zielposition (px\_ziel, py\_ziel) des TCPs zu gelangen. Hierfür wird der Algorithmus 2 (Abb. 1) verwendet. Die Schrittweite beträgt  $\delta t = 10$  ms. Kann der Verlauf der Bahn nicht abgefahren werden, erfolgt eine Ausgabe einer Fehlermeldung und eine Variation der Parameter  $\bar{a}$  und  $\bar{v}$  werden solange geändert bis diese eine gültige Bahn erhält. Es wird mit folgenden Werten:  $\bar{v}_1 = \bar{v}_2 = 100$  m/s und  $\bar{a} = 5$  m/s<sup>2</sup> gestartet.

#### Algorithmus 2:

**Gegeben:** Startpunkt  $(p_{x,start}, p_{y,start})$ ,  
Endpunkt  $(p_{x,ziel}, p_{y,ziel})$ ,  
Grenzen  $\bar{v}_1, \bar{v}_2, \bar{a}$   
und Schrittweite  $\delta t$

1. Überprüfe, ob sich der Start- und Endpunkt im Arbeitsraum des Roboters befindet.
2. **for**  $i = 1 : 2$
3. Berechne den Startwinkel  $\theta_{i,start}$  und den Endwinkel  $\theta_{i,ziel}$  aus den gegebenen Positionen.
4. **If**  $\bar{v}_i > \sqrt{\bar{a}|\theta_{i,ziel} - \theta_{i,start}|}$  **then**  $\bar{v}_i = \sqrt{\bar{a}|\theta_{i,ziel} - \theta_{i,start}|}$
5. Berechne die Endzeit  $t_{i,e}$  und die Übergangszeit  $t_{i,b}$  gemäß Gleichung (4)
6. Berechne den Verlauf gemäß der Gleichung (3).
7. **end**

**Resultat:** Verlauf  $\theta_{1,soil}(t)$  und  $\theta_{2,soil}(t)$

Abb. 1: Verlaufberechnung Algorithmus 2

#### Übergabewerte:

- Ist-Position px und py und Ziel-Position px\_ziel und py\_ziel

#### Rückgabewerte:

- Soil-Verläufe der Winkel  $\theta_1, soil(t)$  und  $\theta_2, soil(t)$  in Gradmaß

#### Erläuterung zum Rückgabewert:

- $\theta_{1,soil}(:,1)$ : Zeitwerte t für den Verlauf von  $\theta_1, soil(t)$
- $\theta_{1,soil}(:,2)$ : Berechnete Winkelwerte  $\theta_1$
- $\theta_{2,soil}(:,1)$ : Zeitwerte t für den Verlauf von  $\theta_2$
- $\theta_{2,soil}(:,2)$ : Berechneten Winkelwerte  $\theta_2, soil(t)$

#### Aufruf:

```
fun_callTraj(px_start,py_start,px_ziel,py_ziel)
```

#### Beispiel:

```
global t1 t2;  
t1 = 0.3;  
t2 = 0.4;  
[theta1_soil,theta2_soil] = fun_callTraj(0.3,0.6,0.5,0.8)
```

Autor: Paul Freynik Matr.Nr. 2017507935

```
1 % Start der Funktion  
2 function [theta1_soil,theta2_soil] = fun_callTraj(px_start,py_start,px_ziel,py_ziel)  
3 % Beschleunigungsgrenze Startwerte  
4 a = 5;  
5 % Geschwindigkeitbegrenzung Startwerte  
6 v = [100,100];  
7 % Schrittweite  
8 delta_t = 0.01;  
9  
10
```

1. Überprüfung, ob sich der Start- und Endpunkt im Arbeitsraum des Roboters befindet.
2. und 3. Berechne den Startwinkel  $\theta_{i,start}$  und den Endwinkel  $\theta_{i,ziel}$  aus den gegebenen Positionen.

```
11 % Startbedingungen überprüfen  
12 % Berechne den Startwinkel  $\theta_{1,start}$  und den Endwinkel  $\theta_{1,ziel}$   
13 % aus den gegebenen Positionen  
14 % if-Schleife Vergleich Arbeitsbereich  
15 if ~fun_checkWorkspace(px_start,py_start) || ~fun_checkWorkspace(px_ziel,py_ziel)  
16 % Ausgabe Fehlermeldung  
17 disp('error start oder zielpunkt liegen ausserhalb des arbeitsbereichs!')  
18 % Initialisierung Sollwert  
19 theta1_soil = 0;  
20 % Initialisierung Sollwert  
21 theta2_soil = 0;  
22 % Ausgabe  
23 return;  
24 % Ende if Schleife Zeile 16  
25 end  
26
```

```
27  
28 % for Schleife  
29 for i = 1:2  
30 % Start und Endwinkel berechnen  
31 % Startwerte berechnen  
32 [theta_start(i),theta_start(2)] = fun_callInvKin(px_start,py_start);  
33 % Zielwerte berechnen  
34 [theta_ziel(i),theta_ziel(2)] = fun_callInvKin(px_ziel,py_ziel);  
35  
36
```

#### Gleichung Beschleunigungsbegrenzung Abb.2:

$$0 \leq \bar{v}_i \leq \sqrt{\bar{a}|\theta_{i,ziel} - \theta_{i,start}|}$$

Abb. 2: Gleichung zur Berechnung Beschleunigungsbegrenzung

#### 4. Setzen der Beschleunigungsbegrenzung

```
37  
38 % setzen der Beschleunigungsbegrenzung  
39 % Berechnung  
40 agrenz = sqrt(a*abs(theta_ziel(i)-theta_start(i)));  
41 % if Schleife  
42 if v(i) > agrenz  
43 % Ersetzen der Beschleunigungsbegrenzung  
44 v(i) = agrenz;  
45 % Ende if Schleife Zeile 42  
46 end  
47
```

Gleichung zur Berechnung der Endzeit  $t_{e,i}$  und der Übergangszeit  $t_{b,i}$  gemäß Gleichung Abb.4:

$$t_{e,i} = \frac{|\theta_{i,ziel} - \theta_{i,start}|}{\dot{\theta}_i} + \frac{\dot{\theta}_i}{a}, \quad t_{b,i} = \frac{\dot{\theta}_i}{a}.$$

Abb 4: Gleichung zur Berechnung der Endzeit  $t_{e,i}$  und der Übergangszeit  $t_{b,i}$ .

5. Berechnung der Endzeit  $t_{e,i}$  und die Übergangszeit  $t_{b,i}$  gemäß Gleichung (Abb.4)

```

48 % Berechnen der Endzeit te und Übergangszeit tb
49 % Berechnung der Übergangszeit
50 tb = v(1)/a;
51 % Berechnung der Endzeit
52 te = abs(theta_ziel(1)-theta_start(1))/v(1) + v(1) / a;
53 % Bestimmung der Schrittmenge
54 time = 0:delta_t:(abs(theta_ziel(1))-theta_start(1))/v(1) + v(1) / a);
55 % Erstellen einer Matrix mit nullen
56 thetasoll = zeros(1,length(time));
57
58

```

Rechenvorschrift asynchronen PTP-Bahn:

Die Winkelverläufe berechnen sich bei der asynchronen PTP-Bahn unabhängig von einander. Die Rechenvorschrift (Abb. 3) lautet wie folgt:

$$\theta_{i,soll}(t) = \begin{cases} \theta_{i,start} + \text{sign}(\theta_{i,ziel} - \theta_{i,start}) \frac{1}{2} a t^2 & 0 \leq t \leq t_{b,i} \\ \theta_{i,start} + \text{sign}(\theta_{i,ziel} - \theta_{i,start}) \left( v_i t - \frac{v_i^2}{2a} \right) & t_{b,i} < t \leq t_{e,i} - t_{b,i} \\ \theta_{i,start} + \text{sign}(\theta_{i,ziel} - \theta_{i,start}) \left( v_i (t_{e,i} - t_{b,i}) - \frac{1}{2} (t_{e,i} - t)^2 \right) & t_{e,i} - t_{b,i} < t \leq t_{e,i} \end{cases}$$

Abb 3: Rechenvorschrift

6. Berechnung des Verlaufs gemäß der Gleichung (Abb.3)

```

59 % Verlaufs Berechnung
60 % for Schleife
61 for j = 1:length(time)
62     % Zeitwert
63     t = time(j);
64     % if-Schleife
65     if 0 <= t && t <= tb
66         % Verlaufs Berechnung
67         thetasoll(j) = theta_start(1) + sign(theta_ziel(1) ...
68             -theta_start(1))*((1/2)*a*t^2);
69     % elseif-Bedingung
70     elseif tb < t && t <= (te - tb)
71         % Verlaufs Berechnung
72         thetasoll(j) = theta_start(1) + sign(theta_ziel(1) ...
73             -theta_start(1))*((v(1)*t) - (v(1)^2)/(2*a));
74     % elseif-Bedingung
75     elseif (te - tb) < t && t <= te
76         % Verlaufs Berechnung
77         thetasoll(j) = theta_start(1) + sign(theta_ziel(1) ...
78             -theta_start(1))*(v(1)*(te-tb) - (a/2)*(te-t)^2);
79     % Ende if-Schleife Zeile 66
80     end
81 % Ende for-Schleife Zeile 62
82 end
83
84

```

Resultat: Verlauf  $\theta_1, \text{soll}(t)$  und  $\theta_2, \text{soll}(t)$

```

85 % Werte speichern und zurückgeben
86 % if Schleife
87 if i == 1
88     % Rückgabewert Winkelwert
89     theta1_soll = [time(:), thetasoll(:)];
90 % else Bedingung
91 % Rückgabewert Winkelwert
92     theta2_soll = [time(:), thetasoll(:)];
93 % Ende if Schleife Zeile 88
94 end
95 % Ende for Schleife Zeile 29
96 end
97 % Ende Funktion Zeile 2
98 end
99
100

```

Abb. 3.3.1: Funktionscode „fun\_callTraj“

### 3.4 Dynamik (P3):

Autor: Georg Kulodzik Matr. 2017507931

Eventuelle Grundlagen und Vorüberlegungen zur Implementierung sind hier einzufügen, wie z.B. geeignete Struktur für die übersichtliche Umsetzung des Simulink Modells. Darstellung und Beschreibung des Simulink-Modells und des zugehörigen Codes. Die Beschreibung des Simulink-Modells umfasst die Erläuterung anhand eines oder mehrerer (erkennbarer!) Bilder. Tipp: Zerlege das Modell und beschreibe die Einzelelemente.

In der Dynamik wird das Verhalten des Roboters beschrieben. Dabei werden alle Kräfte die als Parameter vom Programm vorgegeben wurden mit einberechnet, eine Übersicht über die Parameter ist in der Funktion `fun_moveRobo` aufgeführt. Mit dieser Funktion wird auch das Simulink Modell „Controlled\_PlanarRobot“ gestartet und initialisiert.

Das Simulink Modell besteht aus einem Modell mit zwei Subsystemen, dem Regler und der Regelstrecke, wobei der Teil des Reglers schon als fertiges Subsystem für dieses Projekt zur Verfügung gestellt wird. Das Simulink Modell ist notwendig, damit der Roboter den Soll-verläufen der Winkel folgen kann. Diese Ist-Werte bekommt er von der Bahnplanung. Für die Ansteuerung der Roboter Motoren ist der Regler zuständig, der sicherstellt, dass der tatsächlich gemessene Winkel auch wirklich dem Soll-Winkel entspricht. Das Bild 3.4.1 Zeigt das Simulink Modell mit den zwei Subsystemen.

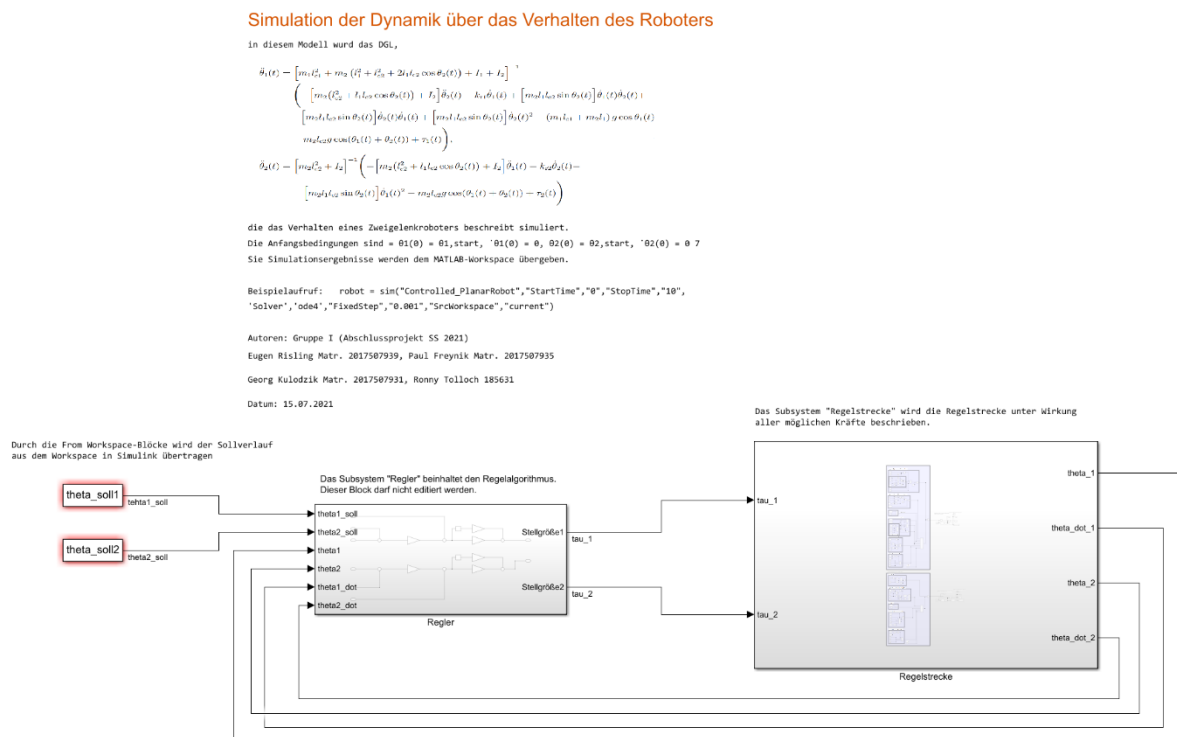


Abb. 3.4.1: Simulink Model

In dem Subsystem mit der Regelstrecke werden die beiden Differenzialgleichungen (Abb.3.4.3+4) umgesetzt. Da es sich hierbei um eine nichtlineare verkoppelte Differenzialgleichung handelt, die sehr komplex ist, wird diese schon für das Projekt zur Verfügung gestellt. Die Beiden Gleichungen (Abb.3.4.2) lauten:

$$\ddot{\theta}_1(t) = \left[ m_1 l_{c1}^2 + m_2 (l_1^2 + l_{c2}^2 + 2l_1 l_{c2} \cos \theta_2(t)) + I_1 + I_2 \right]^{-1} \left( - \left[ m_2 (l_{c2}^2 + l_1 l_{c2} \cos \theta_2(t)) + I_2 \right] \ddot{\theta}_2(t) - k_{r1} \dot{\theta}_1(t) + \left[ m_2 l_1 l_{c2} \sin \theta_2(t) \right] \dot{\theta}_1(t) \dot{\theta}_2(t) + \left[ m_2 l_1 l_{c2} \sin \theta_2(t) \right] \dot{\theta}_2(t) \dot{\theta}_1(t) + \left[ m_2 l_1 l_{c2} \sin \theta_2(t) \right] \dot{\theta}_2(t)^2 - (m_1 l_{c1} + m_2 l_1) g \cos \theta_1(t) - m_2 l_{c2} g \cos(\theta_1(t) + \theta_2(t)) + \tau_1(t) \right),$$

$$\ddot{\theta}_2(t) = \left[ m_2 l_{c2}^2 + I_2 \right]^{-1} \left( - \left[ m_2 (l_{c2}^2 + l_1 l_{c2} \cos \theta_2(t)) + I_2 \right] \ddot{\theta}_1(t) - k_{r2} \dot{\theta}_2(t) - \left[ m_2 l_1 l_{c2} \sin \theta_2(t) \right] \dot{\theta}_1(t)^2 - m_2 l_{c2} g \cos(\theta_1(t) + \theta_2(t)) + \tau_2(t) \right)$$

Abb. 3.4.2: Differenzialgleichung

Die für die Differenzialgleichungen sind die Anfangsbedingungen vorgegeben und werden dem Simulink Modell mit übergeben:

$$\theta_1(0) = \theta_{1,\text{start}}, \quad \dot{\theta}_1(0) = 0, \quad \theta_2(0) = \theta_{2,\text{start}}, \quad \dot{\theta}_2(0) = 0$$

Die Abbildung 3.4.3 (für  $\theta_1$ ) und Abb. 3.4.4 (für  $\theta_2$ ) zeigen den Aufbau des Subsystems der Regelstrecke. Im lila Kasten wird jeweils die Differenzialgleichung umgesetzt. Für eine bessere Übersicht ist die Differenzialgleichung jeweils in einzelne Summanden aufgeteilt. Das ist als Beispiel im Bild 3.4.5 zu sehen. Diese Summanden sind in einzelne Kästen aufgeteilt und haben den jeweiligen Teil der Mathematischen Formel als Überschrift (Ausschnitt der DGL). Rechts Außerhalb der lila Kästen sind die beiden Integrationen für  $\theta_1$  und  $\theta_2$  durchgeführt. Die roten Kästen weisen darauf hin, dass hierfür keine Variable vom Workspace zur Verfügung gestellt wird. Diese Variablen werden erst beim Aufruf der App erstellt.



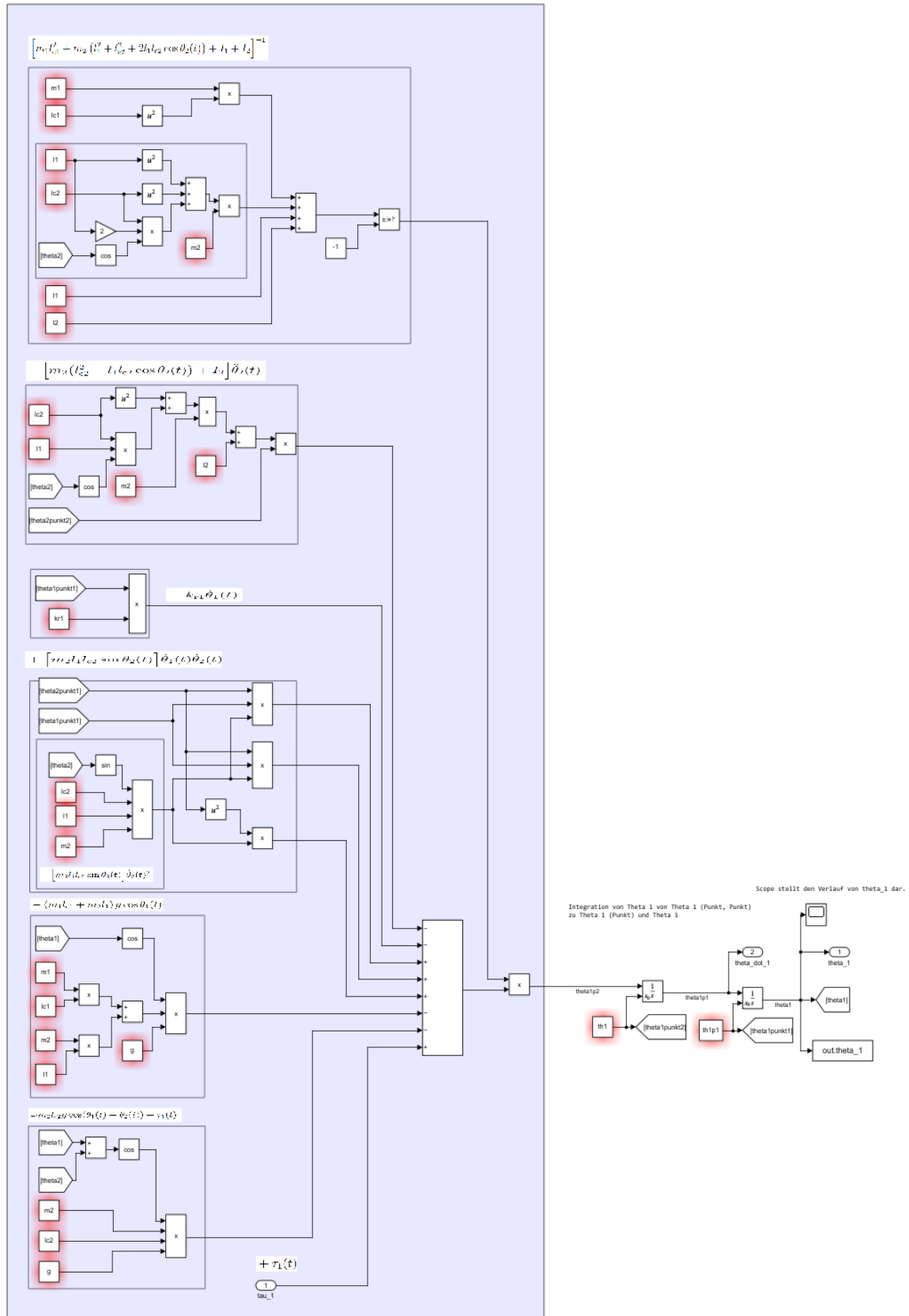


Abb. 3.4.3: Subsystemaufbau für  $\theta_1$

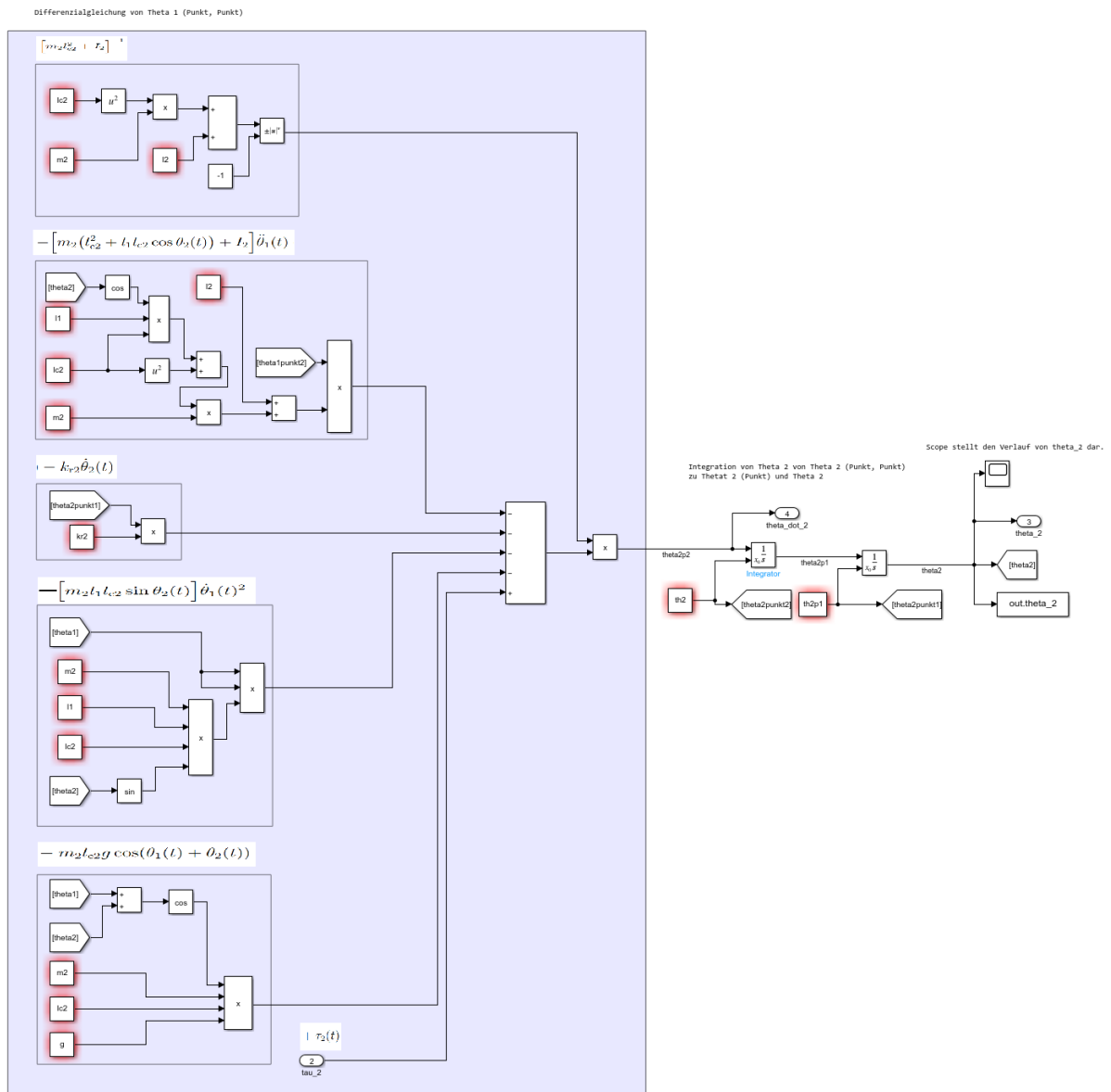


Abb. 3.4.4: Subsystemaufbau für  $\theta_2$



## fun\_moveRobo

### Roboter Bewegung

Diese Funktion ist für die Berechnung des Ist-Winkelverlaufs zuständig. Dieses wird anhand der Soll-Winkelverlaufs aus der Bahnplanung berechnet. Zu dieser Berechnung ruft diese Funktion das Simulink Modell Controlled\_PlanarRobot auf.

- Übergabeparameter: Soll-Verlauf der Winkel  $\theta_{1,soll}(t)$  und  $\theta_{2,soll}(t)$
- Rückgabewerte: Ist-Verlauf der Winkel  $\theta_{1,ist}(t)$  und  $\theta_{2,ist}(t)$  und der Zeitvektor  $t$

Beispielaufruf:

clear

```
global l1 l2 lc1 lc2 m1 m2 I1 I2 kr1 kr2 g th1 th2 th1p1 th2p1;
```

```
l1 = 0.3;      % Länge Armteil 1 [m]
l2 = 0.4;      % Länge Armteil 2 [m]
lc1 = 0.15;    % Abstand Gelenk 1 Schwerpunkt Armteil 1 [m]
lc2 = 0.2;     % Abstand Gelenk 2 Schwerpunkt Armteil 2 [m]
m1 = 2;        % Masse Armteil 1 [kg]
m2 = 3;        % Masse Armteil 2 [kg]
I1 = 0.02;     % Trägheitssensor Armteil 1 [kgm²]
I2 = 0.04 ;    % Trägheitssensor Armteil 2 [kgm²]
kr1 = 1.5;     % Reibungskoeffizient Armteil 1 [Nms]
kr2 = 1.5;     % Reibungskoeffizient Armteil 2 [Nms]
g = 9.81;      % Fallbeschleunigung [m/s²]
th1p1=0;       % Theta 1
th2p1=0;       % Theta 2
th1=0;         % Theta 1 Punkt 1, erste Ableitung mit Anfangswert 0
th2=0;         % Theta 2 Punkt 1, erste Ableitung mit Anfangswert 0
```

```
[theta1_soll,theta2_soll] = fun_callTraj(0.3,0.6,0.5,0.3);
[theta1_soll,theta2_soll] = fun_moveRobo(theta1_soll,theta2_soll);

plot(theta1_soll.Time,theta1_soll.data)

plot(theta2_soll.Time,theta2_soll.data)
```

```
function [theta1,theta2] = fun_moveRobo(thetasoll1,thetasoll2)
    % verknüpfen der Variablen zum Simulink Modell
    global l1 l2 lc1 lc2 m1 m2 I1 I2 kr1 kr2 g th1 th2 th1p1 th2p1 theta_soll1
    theta_soll2;
    % Sollwerte aus der Bahnplanung
    theta_soll1 = thetasoll1;
    theta_soll2 = thetasoll2;

    % Berechnung der Simulationszeit anhand der Ist-Werte
    stoptime = 1.2*max(theta_soll1(end,1),theta_soll2(end,1));

    % Aufruf des Simulink Modells
    robot =
    sim("Controlled_PlanarRobot","StartTime","0","StopTime",num2str(stoptime),"FixedStep","0.001","SrcWorkspace","current");

    % Ausgabe der Ergebnisse (Ist-Werte)
    theta1 = robot.theta_1;
    theta2 = robot.theta_2;
end
```

Der Programmcode zeigt die Übergabe- und Rückgabewerte  $\theta_1, \text{soll}(t)$ ,  $\theta_2, \text{soll}(t)$  sowie  $\theta_1, \text{ist}(t)$  und  $\theta_2, \text{ist}(t)$ . Für den Aufruf des Simulink Modells wird die in der Bahnplanung errechnete Zeit mit dem Faktor 1,2 multipliziert um ausreichend Zeit für eine vollständige Berechnung zu haben, dieser Wert wird in der "stoptime"-Variable gespeichert um hier Variabel auf verschiedene Berechnungen reagieren zu können und nicht unnötig immer von der maximal möglichen Zeit auszugehen. Die Hauptaufgabe der Funktion ist das Starten des Simulink Modells und die Übergabe der Werte. Anhand dieser Werte wird dann die Bewegung des Roboters über die fun\_drawRobo Funktion dargestellt, was aber von der App gesteuert wird.

### 3.5 App (P4):

Autor: Ronny Tolloch Matr.

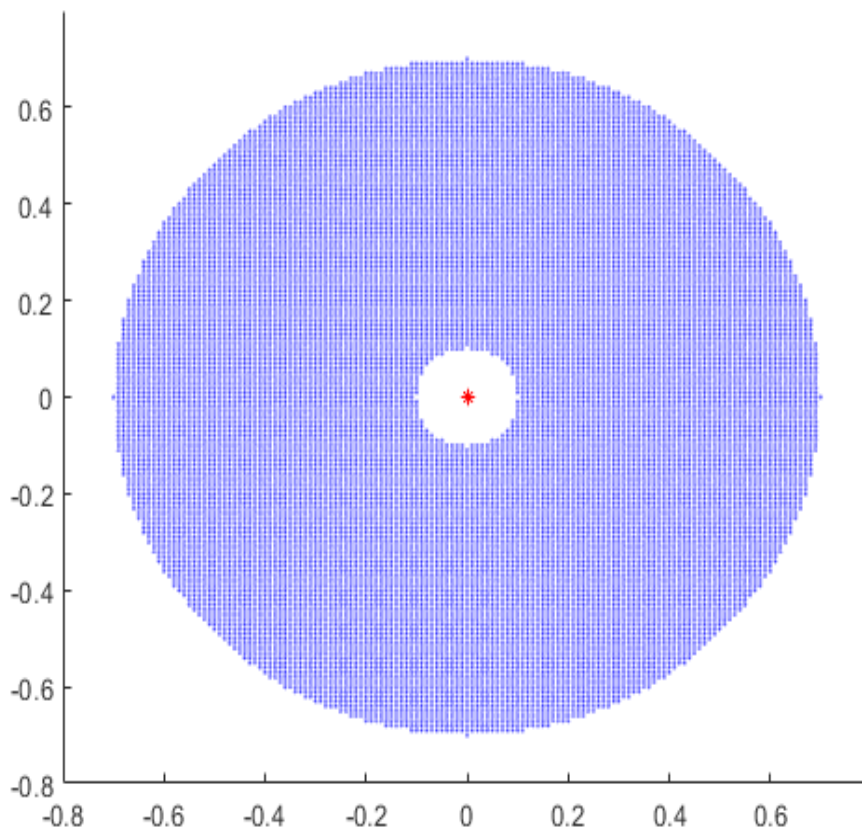
185631

*Eventuelle Grundlagen und Vorüberlegungen zur Implementierung sind hier einzufügen, wie z.B. ein Bild des Arbeitsraums des Roboters. Darstellung und Beschreibung des Code-Views, des Design-Views und des weiteren Codes.*

Als Basis für den Arbeitsraum des Roboters ist der TCP als Greifarm zu sehen. Nur Positionen in der sich der TCP hinbewegen kann können als Arbeitsraum gesehen werden. Der Arbeitsraum des Roboters ist nach Außen und zur Mitte hin begrenzt. Die Roboterarme/-gelenke können in unserem Fall nur in einem Gesamtradius von 0,7 Metern arbeiten. Dieses ergibt sich durch die Summe der beiden Arme (Arm 1 = 0,3m , Arm 2 = 0,4m). Da der Arm 1 in der Mitte des Arbeitsraums fixiert ist, muss auch hier berücksichtigt werden, dass ein bestimmter Bereich der Mitte, aufgrund der unterschiedlichen Armlängen nie erreicht werden kann. Hier ergibt sich ein toter Raum mit dem Radius von 0,1 Metern ( $|Armlänge\ 1 - Armlänge\ 2|$ ). Dieses ist in der Abb. 3.5.1 zu sehen.

Die Anwendungsoberfläche des Programmes hat als Ziel die verschiedenen im Vorfeld erstellten Funktionen in eine Anwendung zusammen zu bringen und die Ergebnisse Visuell (Arm Bewegung) und grafisch darzustellen.

*Darstellung des Arbeitsbereichs:*



*Abb. 3.5.1: Darstellung Arbeitsbereich*

Die App greift auf Funktionen zu, um die Funktionalität des Gesamten zu ermöglichen, dafür ist es für den Programmieren notwendig diese zu wissen und einzubringen, dieses interessiert den Bediener wenig, da er nur die Funktionalität und das Ergebnis sehen möchte. Im folgenden Code View (Abb.3.5.2) lassen sich alle Funktionen nachvollziehen, hierbei gibt es Editierbare Bereiche (weißer Hintergrund) und nicht Editierbare Bereiche (hellgrau).

properties:

```
properties (Access = public)

    % variablen für ergebnisse zum darstellen
    theta1soll=0;
    theta2soll=0;
    theta1ist=0;
    theta2ist=0;
end
```

funktion:

```
methods (Access = private)

function drawtheta(app,string)

    if string == "planung"

        % zeichnet winkel in die winkel plots ein
        plot( app.theta1verlauf, app.theta1soll(:,1), app.theta1soll(:,2) );
        plot( app.theta2verlauf, app.theta2soll(:,1), app.theta2soll(:,2) );
        app.theta1verlauf.XLim = [0, (app.theta1soll(end,1)+1/10)];
        app.theta2verlauf.XLim = [0, (app.theta2soll(end,1)+1/10)];

    elseif string == "verlauf"

        % zeichnet winkel in die winkel plots ein
        hold(app.theta1verlauf, "on");
        plot(app.theta1verlauf,app.theta1ist.time,app.theta1ist.data);
        hold(app.theta1verlauf, "off");

        hold(app.theta2verlauf, "on");
        plot(app.theta2verlauf,app.theta2ist.time,app.theta2ist.data);
        hold(app.theta2verlauf, "off");

    else

        % ansonsten leert es beide plots
        app.theta1soll=0;
        app.theta2soll=0;
        app.theta1ist=0;
        app.theta2ist=0;
        cla(app.theta1verlauf);
        cla(app.theta2verlauf);

    end
end

function arbeitsbereich(~,ax)

    % berechnet für jeden punkt ob diesr im arbeitsbereich ist
    time = -1:0.02:1;
    p = zeros(length(time),length(time));
    for x = 1:length(time)
        for y = 1:length(time)
            xa=time(x);
            ya=time(y);
            p(x,y) = fun_checkWorkspace(xa,ya);% prüft alle x + y positionen
        end
    end

    % fügt alle makierung in den plot ein
    hold(ax,"on")
    for i=1:length(time)
        p(p==0) = NaN; % entfernt alle 0 so das sie nicht angezeigt werden
        plot(ax,time,time(i)*p(i,:), "b.", 'MarkerSize',1)
    end
    hold(ax,"off")
end
```

## Callbacks 1:

<pre> % Button pushed function: vorwaerts, vorwaerts_2 function vorwaertskinematic_callback(app, event)      if event.Source.Parent.Parent.Tag == "Startwert"          [px,py] = fun_callForwardKin(app.theta_1.Value,app.theta_2.Value); % berechnet werte         app.p_x.Value = round(px,3); % rundet ergebnisse auf 3 stellen und tut sie in die entsprechenden felder         app.p_y.Value = round(py,3);      elseif event.Source.Parent.Parent.Tag == "Zielwert"          [px,py] = fun_callForwardKin(app.theta_1_2.Value,app.theta_2_2.Value); % berechnet werte         app.p_x_2.Value = round(px,3); % rundet ergebnisse auf 3 stellen und tut sie in die entsprechenden felder         app.p_y_2.Value = round(py,3);      end      app.drawrobo_callback; % zeichnet Roboterarm end </pre>	
<pre> % Button pushed function: ruckwaerts, ruckwaerts_2 function ruckwaertskinematic_callback(app, event)      if event.Source.Parent.Parent.Tag == "Startwert"         xvalue = app.p_x.Value;         yvalue = app.p_y.Value;     elseif event.Source.Parent.Parent.Tag == "Zielwert"         xvalue = app.p_x_2.Value;         yvalue = app.p_y_2.Value;     end      % prüft ob wert sich im arbeitsbereich befindet     if fun_checkWorkspace(xvalue,yvalue) == 0         infotext(app,201); % warnung         return;     else         infotext(app,500); % info text     end      set_werte_callback(app); % speichert gesetzte werte zum nutzen     % entscheidet ob das analytische oder numerische verfahren     % benutzt werden soll     if app.Analytisch.Value == true         [theta1,theta2] = fun_callInvKin(xvalue,yvalue);     else         [theta1,theta2] = fun_callInvKin_Num(xvalue,yvalue);     end      % speichert die werte in die entsprechenden felder und rundet     % auf 3 stellen     if event.Source.Parent.Parent.Tag == "Startwert"         app.theta_1.Value = round(theta1,3);         app.theta_2.Value = round(theta2,3);     elseif event.Source.Parent.Parent.Tag == "Zielwert"         app.theta_1_2.Value = round(theta1,3);         app.theta_2_2.Value = round(theta2,3);     end      app.drawrobo_callback; % zeichnet Roboterarm end </pre>	



## callbacks 2:

<pre> % Button pushed function: Planung function plannung_callback(app, event)  % prüft ob wert sich im arbeitsbereich befindet if fun_checkWorkspace(app.p_x_2.Value,app.p_y_2.Value) == 0     infotext(app,201); % warnung     return; else     infotext(app,500); % info text end  drawtheta(app,"clear");% löscht wenn vorhanden alte werte [app.theta1soll,app.theta2soll] = fun_callTraj(app.p_x.Value,app.p_y.Value,app.p_x_2.Value,app.p_y_2.Value); % berechnet Winkelverlauf drawtheta(app,"planung"); % zeichnet verlauf  end </pre>	
<pre> % Button pushed function: Verfahren function moverobo_callback(app, event)  % speichert startwerte für roboter movement global th1 th2 th1p1 th2p1; th1p1 = 0; th2p1 = 0; th1 = app.theta_1.Value; th2 = app.theta_2.Value;  planung_callback(app); % da die werte von plannung benötigt werden führt das programm die plannung einfach selbst durch um fehler zu mitigrieren [app.theta1ist,app.theta2ist] = fun_moveRobo(app.theta1soll,app.theta2soll); % Berechnet winkelverlauf über symulink drawtheta(app,"verlauf"); % zeichnet verlauf  end </pre>	
<pre> % Button pushed function: Clear function clear_callback(app, event)  % setzt alle werte auf 0 % und leert die plots app.p_x.Value = app.L1.Value+app.L2.Value; % 0 grad app.p_y.Value = 0; app.p_x_2.Value = 0; app.p_y_2.Value = app.L1.Value+app.L2.Value; % 90 grad  app.theta_1.Value = 0; % 0 grad app.theta_2.Value = 0; app.theta_1_2.Value = 90; % 90 grad app.theta_2_2.Value = 0;  drawtheta(app,"clear"); app.drawrobo_callback; % zeichnet verlauf  end </pre>	

## callbacks 3:

<pre> % Value changed function: Startrobot function start_bewegung_callback(app, event)  % checkt ob verfahren schon benutzt wurde if app.theta1ist == 0     app.Startrobot.Value = false;     return; end  % funktion aus Verfahren ausgelagert , weil das neu zeichnen % des Roboters ältere Computer / Laptops belasten kann app.Lamp.Color = [1 0 0]; % lampe rot  % berechnet eine zahl die bestimmt in wie großen schritten % gegangen wird wichtig da das verfahren ein array mit 100000 % einheiten produzieren kann , jeder schritt wird sonst gezeichnet % und 50 ms gewartet len = min(length(app.theta1ist.time),length(app.theta2ist.time)); i = 1;  while i &lt; len     if app.Startrobot.Value == false % ermöglicht die bewegung zu beenden         break;     end      fun_drawRobo(app.Robotarm,app.theta1ist.data(i),app.theta2ist.data(i)); % zeichnet den jeweiligen schritt     app.Statuspercentlabel.Text = sprintf('%2f',(i/len)*100) + "%"; % zeigt fortschritt in gerundet xxx.xx% an     pause(50/1000); % wartet 50 ms bei 0 könnte der computer freezeen     i = i+round(app.TempoSlider.Value); end  app.Statuspercentlabel.Text = "100 %"; % 100 % prozent status app.Lamp.Color = [0 1 0]; % lampe grün app.Startrobot.Value = false;  end </pre>	
<pre> % Button pushed function: Helpbutton_1, Helpbutton_2, % Helpbutton_3, Helpbutton_4 function help_callback(app, event)  if event.Source.Parent.Parent.Tag == "Startwert" % button texte     infotext(app,301);  elseif event.Source.Parent.Parent.Tag == "Zielwert" % button texte     infotext(app,302);  elseif event.Source.Parent.Parent.Tag == "Runtime" % button texte     infotext(app,303);  elseif event.Source.Parent.Parent.Tag == "Armsettings" % button texte     infotext(app,304); end  end </pre>	
<pre> % Button pushed function: Reset </pre>	

callbacks 4:

<pre>% Button pushed function: Reset function reset_callback(app, event)      % setzt vorgebene standartwerte ein     app.L1.Value = 0.3; % länge     app.L2.Value = 0.4;     app.Lc1.Value = 0.15; % abstand     app.Lc2.Value = 0.2;     app.m1.Value = 2; % gewicht     app.m2.Value = 3;     app.I1.Value = 0.02; % sensor     app.I2.Value = 0.04;     app.Kr1.Value = 1.5; % Reibung     app.Kr2.Value = 1.5;     app.G.Value = 9.81; % Fallbeschleunigung</pre>	
<pre>end  % Button pushed function: Set function set_werte_callback(app, event)</pre>	
<pre>    % setz ein paar globale variablen aufdie externe funktionen     % zugreifen können     global l1 l2 lc1 lc2 m1 m2 I1 I2 kr1 kr2 g;     l1 = app.L1.Value; % länge     l2 = app.L2.Value;     lc1 = app.Lc1.Value; % abstand     lc2 = app.Lc2.Value;     m1 = app.m1.Value; % gewicht     m2 = app.m2.Value;     I1 = app.I1.Value; % sensor     I2 = app.I2.Value;     kr1 = app.Kr1.Value; % Reibung     kr2 = app.Kr2.Value;     g = app.G.Value; % Fallbeschleunigung</pre>	
<pre>end  % Value changed function: Arbeitsbereich function drawrobo_callback(app, event)</pre>	
<pre>    % prüft ob wert sich im arbeitsbereich befindet     if fun_checkWorkspace(app.p_x.Value,app.p_y.Value) == 0         infotext(app,201); % warnung         return;     else         infotext(app,500); % info text     end      % zeichnet den roboterarm     fun_drawRobo(app.Robotarm,app.theta_1.Value,app.theta_2.Value );      % fügt ziel makierung hinzu     hold(app.Robotarm, "on");     plot(app.Robotarm,app.p_x_2.Value,app.p_y_2.Value,"g.", 'MarkerSize',15)     hold(app.Robotarm, "off");      % fügt arbeitsbereich hinzu     if app.Arbeitsbereich.Value == true         arbeitsbereich(app,app.Robotarm);     end</pre>	
<pre>end end</pre>	

Abb. 3.5.2: Roboterarm Code

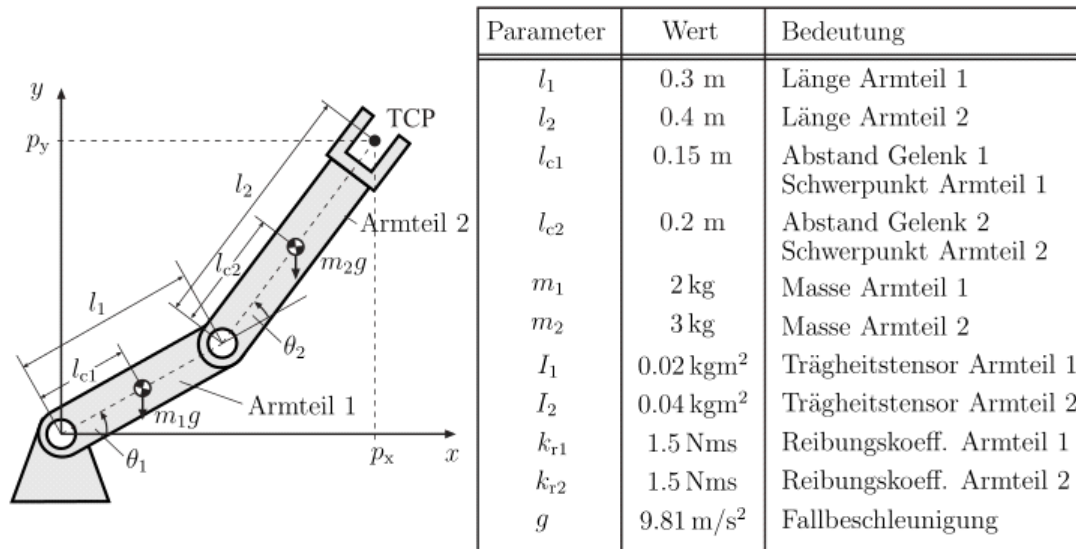


Abb. 3.5.3: Parameter für Roboterarm

Nun wird alles in eine design umgewandelt und dieses dann im App Designer von Matlab unter dem design tab gebaut (Abb.2.2.1 + .2 unter Punkt 2 Beschreibung der App).

Es fehlt aber noch die Funktion der Anwendung da alles nur ein design ist, diese wird über Callback Funktionen in ein Event mit entsprechender Reaktion realisiert.

Hier wurden folgenden callbacks genutzt:

- startupFCN -> Standard Ablauf zum Start des Programmes
- Vorwaertskinematic\_callback -> Umwandlung Winkel zu xy-Koordinate
- Rueckwaertskinematic\_callback -> Umwandlung xy-Koordinate zu Winkel
- Planung\_callback -> Wegplanung
- moverobo\_callback -> zeichnen der Robobewegung (hier getrennt mit echter Bewegung)
- clear\_callback -> clear alle Felder (setzt alles zu 0 werten)
- Start\_bewegung\_callback -> vorher erwähnt echte Bewegung
- Help\_callback -> Infos für neue Nutzer
- Reset\_callback -> setzt Werte auf die Standardwerte
- Set\_werte\_callback -> setzt die Werte effektiv für die App
- drawrobo\_callback -> zum zeichnen des Roboterarmes

## 4. Diskussion der Ergebnisse

### 4.1 Zeigt einen Beispielaufruf eurer App und präsentiert und diskutiert das Gesamtergebnis

Durch Eingeben des Befehls Robo im Command Window oder durch das Öffnen der Robo.mlapp Datei erscheint diese Grafische Darstellung (Abb.4.1.1) der App.

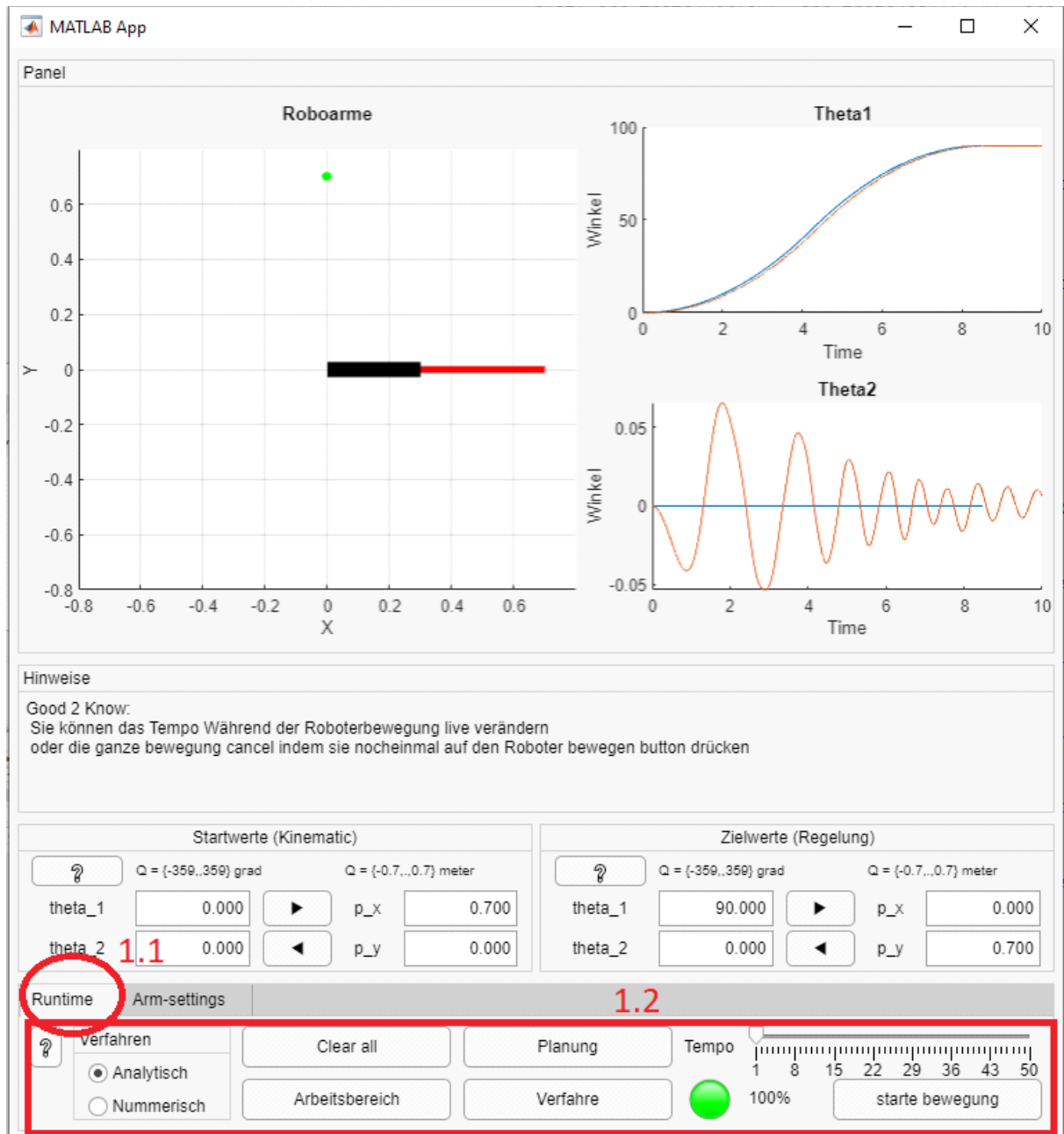


Abb. 4.1.1: App Darstellung Runtime

Man sieht, dass viele Standard Werte enthalten sind, die dem Bediener die Anwendung durch die Visualisierung möglichst einfach machen, hier können Einstellungen sehr schnell geändert werden.

Unter dem Reiter Runtime (1.1) lassen sich alle relevanten Einstellungen (1.2) vornehmen, die bezogen auf das Verhalten der Arm Bewegung wichtig sind oder das Löschen vorinitialisierter Daten oder eingegebener Werte können hier durchgeführt werden

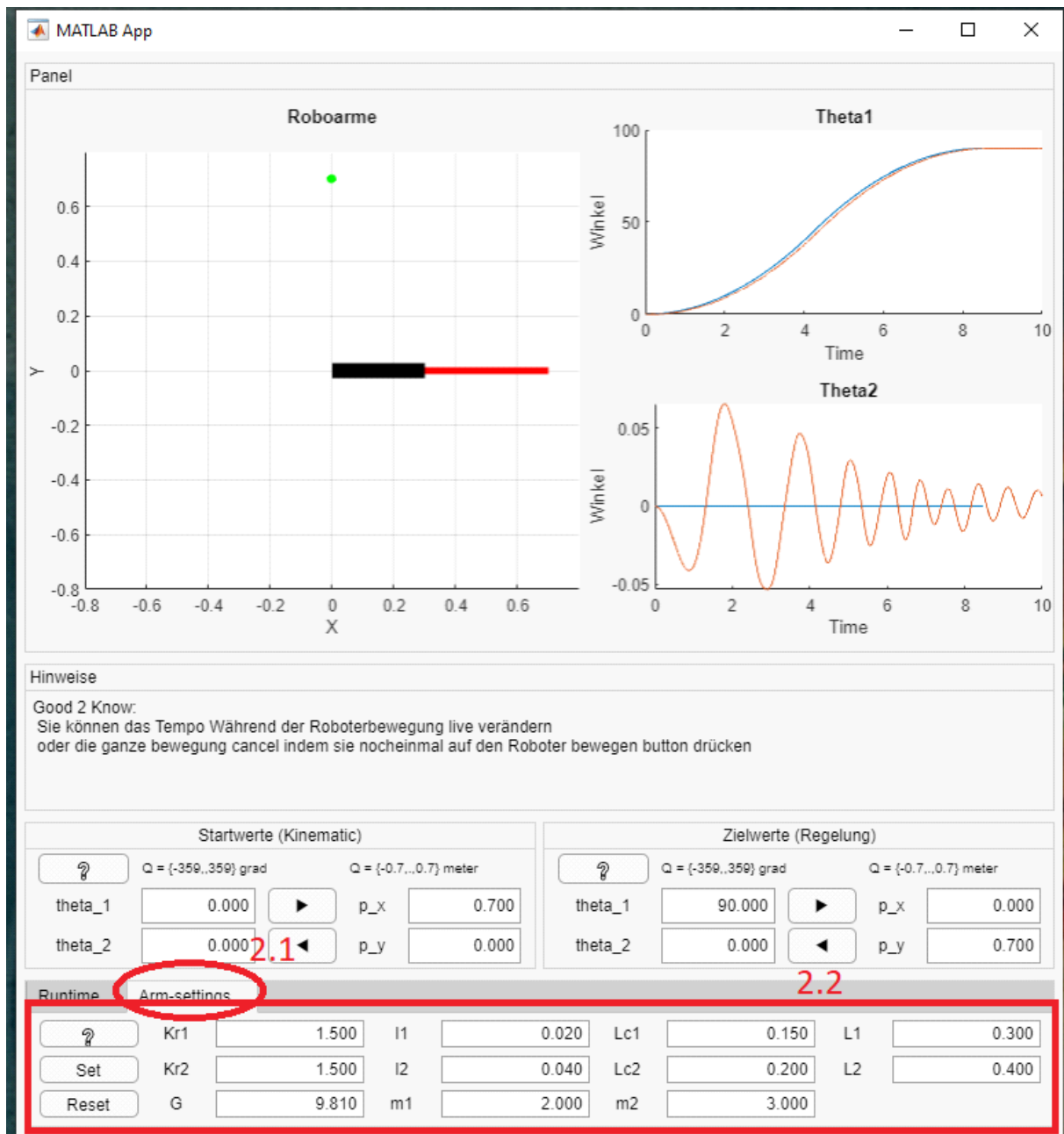


Abb. 4.1.2: App Darstellung Arm Settings

Bei dem nächste Reiter Arm Settings (2.1) lassen sich alle Geometrischen Daten der Roboterarme/-gelenke durch einfaches Ändern der Zahlenwerte und durch das Bestätigen der Set Taste (2.2) angepasst werden, die wiederum das Verhalten der Armbewegung ändern. Hier könnte man individuelle Roboterarme entsprechend ihrer Geometrischen Abmaße konfigurieren und dadurch für verschiedene Typen universell einsetzen!

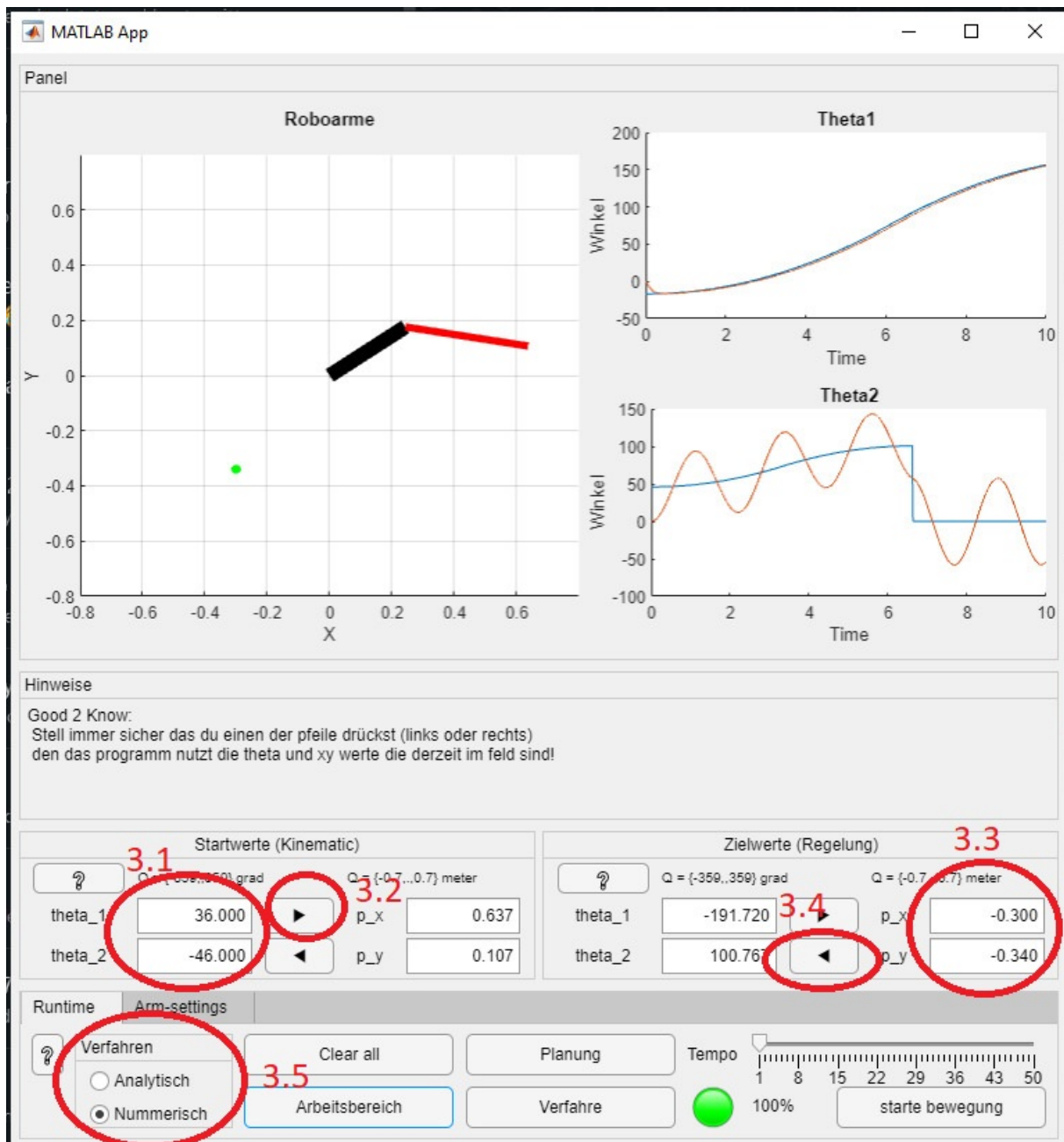


Abb. 4.1.3: App Darstellung Position

Um den Roboterarm entweder in eine Position zu verfahren müssen vorher Startwerte (3.1) in Winkelmaß eingegeben werden diese werden durch Betätigen der rechtstaste (3.2) in die entsprechende Koordinate umgerechnet. Das gleich kann mit den Zielwerten durchgeführt werden. Sind aber nur die Koordinaten bekannt wird ähnlich vorgegangen, man trägt die Zielkoordinaten (3.3) ein und lässt durch betätigen der linkstaste (3.4) die Werte in das entsprechende Winkelmaß umrechnen. Das Berechnungsverfahren kann unter Punkt 3.5 verändert werden.

## 4.2 Ergebnisse zu den Analysen zum Teil Kinematik (P1)

Eugen Risling: Matr. 2017507939

**Analyse 1:** *Verändere die Werte für die Abbruchschwelle  $\epsilon$  und die Obergrenze der Schleifendurchläufen  $n_{\text{iter}}$ . Welchen Einfluss haben diese Werte auf das Ergebnis?*

Durch Verändern von  $n_{\text{iter}}$  erhöht man die Wahrscheinlichkeit die Lösung zu erreichen. Im Falle, wenn  $n_{\text{iter}}$  durchläuft und nicht unter dem  $\epsilon$ -Wert fällt, dann ergibt sich kein Ergebnis bzw. keine Lösung.

Durch Verändern von  $\epsilon$ , verändert sich die Toleranz somit erhöht sich die Wahrscheinlichkeit, dass das Ergebnis akzeptiert wird.

**Analyse 2:** *Nicht immer stimmen die ermittelten Winkel mit denen aus der Funktion `fun_callInvKin` überein. Woran könnte das liegen?*

Das liegt daran, dass das Ergebnis numerisch ist und somit nicht exakt. Die Funktion `fun_callInvKin` ist exakt, weil die analytisch ist.

## 4.3 Ergebnisse zu den Analysen Zum Teil Bahnplanung (P2)

Paul Freynik Matr.

2017507935

*Analyse: Variiere diese Werte für  $\bar{v}_1$ ,  $\bar{v}_2$  und  $\bar{a}$  und beschreibe, wie sich der Verlauf verändert  $\theta_1, \text{soll}(t)$  und  $\theta_2, \text{soll}(t)$  verändert.*

Variiert man die Geschwindigkeit  $v_1$  und  $v_2$  der Bewegung ergeben sich die Verläufe in der Abb.4.3.1 bis .4, hierbei kann man sagen, je höher die Geschwindigkeitsgrenzen desto weniger Zeit ist nötig, dieses gilt aber nur bis das Beschleunigungslimit bei  $v_1$  erreicht ist. Hierbei wird der Roboterarm schneller. Das Beschleunigungslimit ist bei  $v_2$  schon direkt erreicht, somit ergeben sich keine Änderungen. Als Ursache ist die asynchrone Bewegung der Arme, erst der eine dann der andere Arm, festzustellen.

Variiert man die Beschleunigung Abb.4.3.5 und .6 so ist festzustellen, dass diese sich entsprechend verändert und annähernd gleich bei unterschiedlichen Theta  $\theta_1$  und  $\theta_2$  verlaufen!

Um die Kennlinien darstellen zu können ist in Abb. 4.3.7 der Code dargestellt!

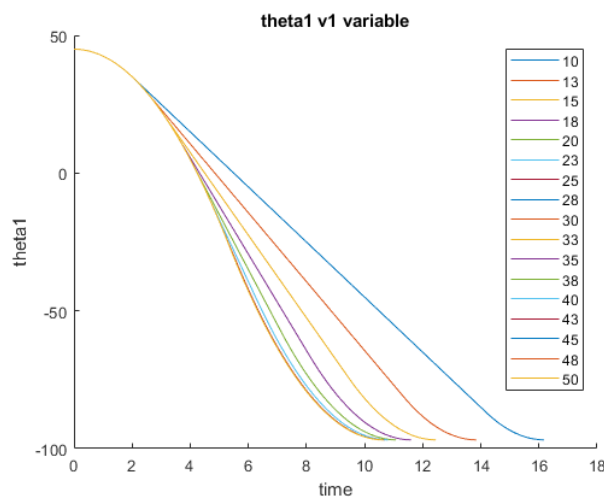


Abb.4.3.1: Variation von  $v_1$  bei  $\theta_1$

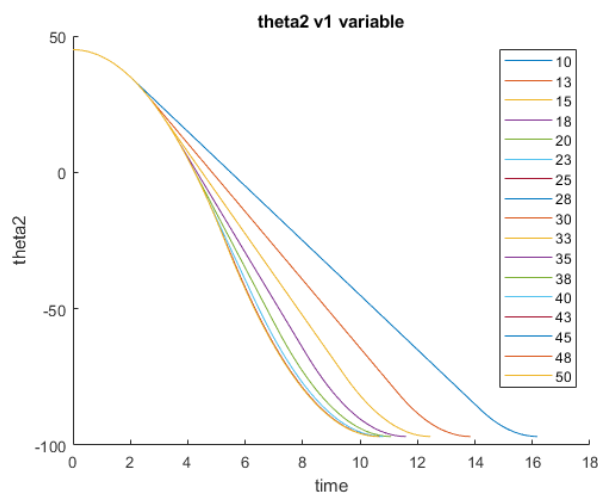


Abb.4.3.2: Variation von  $v_1$  bei  $\theta_2$



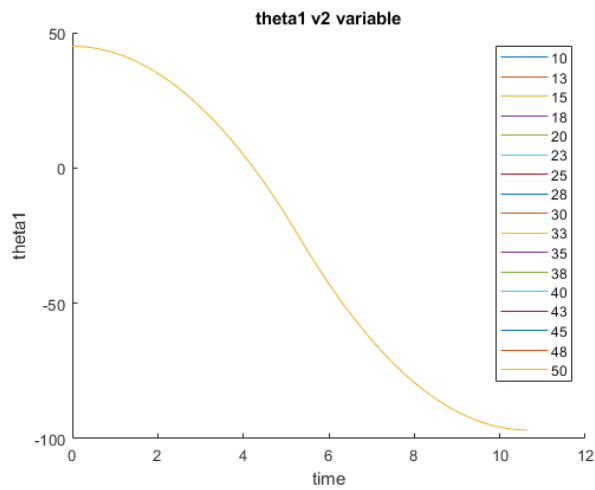


Abb.4.3.3: Variation von  $v_2$  bei  $\Theta_1$

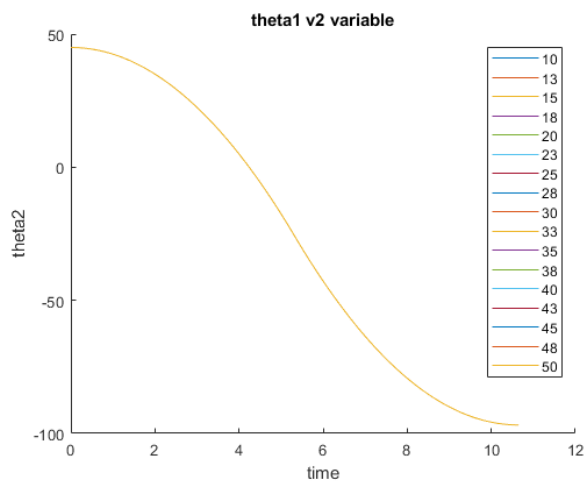


Abb.4.3.4: Variation von  $v_2$  bei  $\Theta_2$

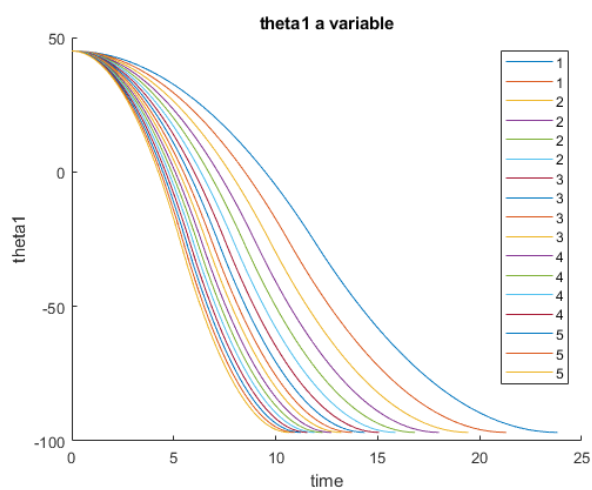


Abb.4.3.5: Variation von  $a$  bei  $\Theta_1$

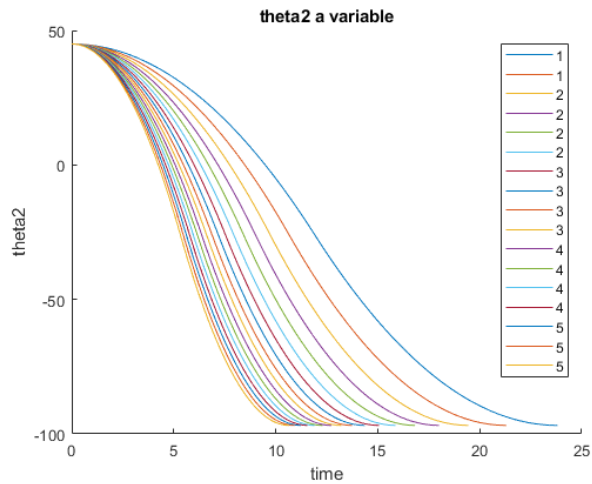


Abb.4.3.6: Variation von  $a$  bei  $\Theta_2$

Referenzcode:

```
clear
global l1 l2;
l1= 0.3;l2=0.4;

ax=axes('Parent',figure);
title(ax,'theta1 v1 variable')
legend(ax,'show')
xlabel(ax,'time')
ylabel(ax,'theta1')

bx=axes('Parent',figure);
title(bx,'theta2 v1 variable')
legend(bx,'show')
xlabel(bx,'time')
ylabel(bx,'theta2')

cx=axes('Parent',figure);
title(cx,'theta1 v2 variable')
legend(cx,'show')
xlabel(cx,'time')
ylabel(cx,'theta1')

dx=axes('Parent',figure);
title(dx,'theta2 v2 variable')
legend(dx,'show')
xlabel(dx,'time')
ylabel(dx,'theta2')

ex=axes('Parent',figure);
title(ex,'theta1 a variable')
legend(ex,'show')
xlabel(ex,'time')
ylabel(ex,'theta1')

fx=axes('Parent',figure);
title(fx,'theta2 a variable')
legend(fx,'show')
xlabel(fx,'time')
ylabel(fx,'theta2')

hold(ax,'on');
hold(bx,'on');
hold(cx,'on');
hold(dx,'on');
hold(ex,'on');
hold(fx,'on');

for i = 1:25:5
    [a,b] = fun_calitraj(0.212,0.612,0.312,-0.612,10^i,100,5); % funktion veränderung um v1 , v2 , a zu verändern
    plot(ax,a(:,1),a(:,2),'DisplayNone',int2str(i*10));
    plot(bx,b(:,1),b(:,2),'DisplayNone',int2str(i*10));
end

for i = 1:25:5
    [a,b] = fun_calitraj(0.212,0.612,0.312,-0.612,100,10^i,5);
    plot(cx,a(:,1),a(:,2),'DisplayNone',int2str(i*10));
    plot(dx,b(:,1),b(:,2),'DisplayNone',int2str(i*10));
end

for i = 1:25:5
    [a,b] = fun_calitraj(0.212,0.612,0.312,-0.612,100,100,1^i);
    plot(ex,a(:,1),a(:,2),'DisplayNone',int2str(i*10));
    plot(fx,b(:,1),b(:,2),'DisplayNone',int2str(i*10));
end

hold(ax,'off');
hold(bx,'off');
hold(cx,'off');
hold(dx,'off');
hold(ex,'off');
hold(fx,'off');
```

Abb.4.3.7: Code zur Darstellung der Kennlinien Abb.4.4.1-6

## 4.4 Ergebnisse zu den Analysen zum Teil Dynamik (P3)

Georg Kulodzik Matr. 2017507931

*Analyse1: Vergrößere die Schrittweite. Ab welcher Schrittweite versagt der numerische Löser ode4? Welche minimalen Schrittweiten benötigt man für andere Solver?*

Bis zum Erreichen der maximalen Schrittweite von 0.02 ändert sich der Ausgang, ab dem Wert versagt die Rechnung. Zu sehen ist dieses an der Abb. 4.4.1 für eine Schrittweite von 0.01 und in Abb. 4.4.2 mit einer Schrittweite von 0.02.

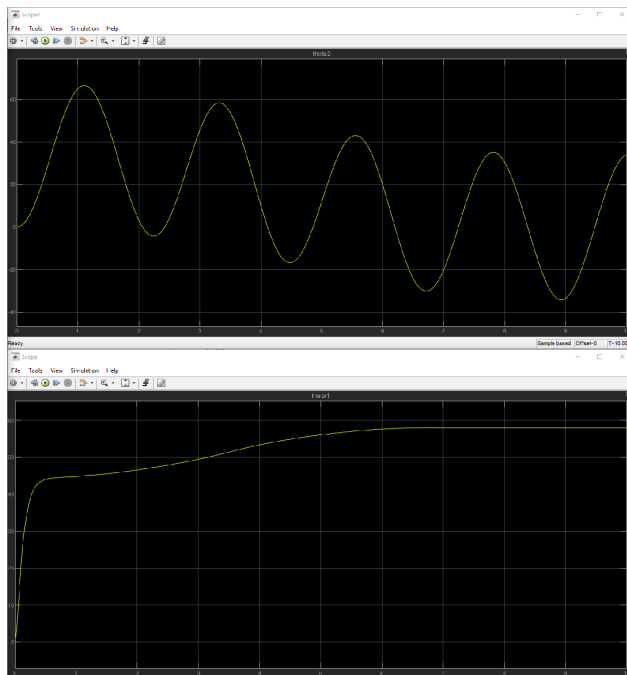


Abb.4.4.1: Schrittweite 0.01

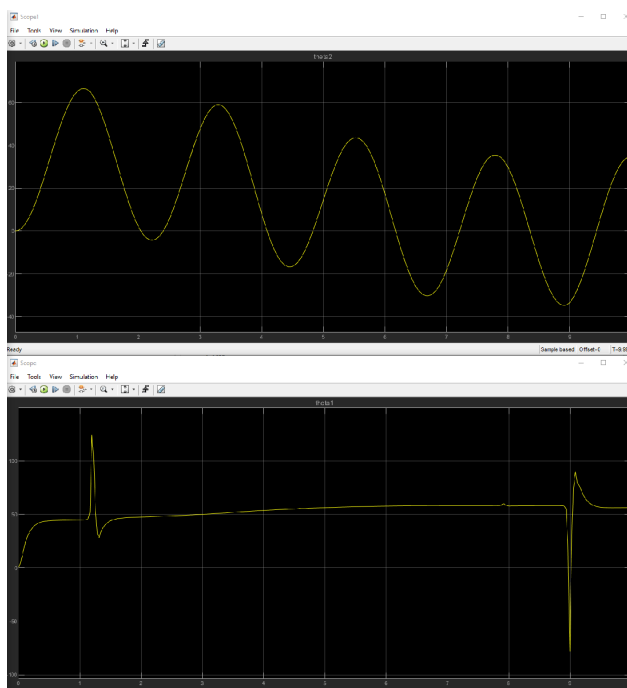


Abb.4.4.2: Schrittweite 0.02

Bei einem Test mit ode1(Euler) kann man bis zu einer Fixed-step Size von 0,004 sec einen zufriedenstellenden Verlauf sehen, wie im Bild 4.4.3, zum Vergleich ist ein Ausschnitt der Verfahrnung bei einer Zeit von 0,04 sec im Bild 4.4.4 zu sehen. Der gleiche Versuch wurde auch mit dem ode2 (Heun) Solver gemacht. Hierbei ist bis zu einer Fixed-step Size von 0,008 sec eine zufriedenstellende Bewegung zu sehen. Bis zu diesen Werten wird das Ziel erreicht. Bei Werten über diese Zeit wird das Ziel teilweise deutlich bei der Bewegung verfehlt. Zusätzlich kann man sagen, dass die Berechnungszeit deutlich zulegt, je kleiner die Schrittweite (Zeit) ist.

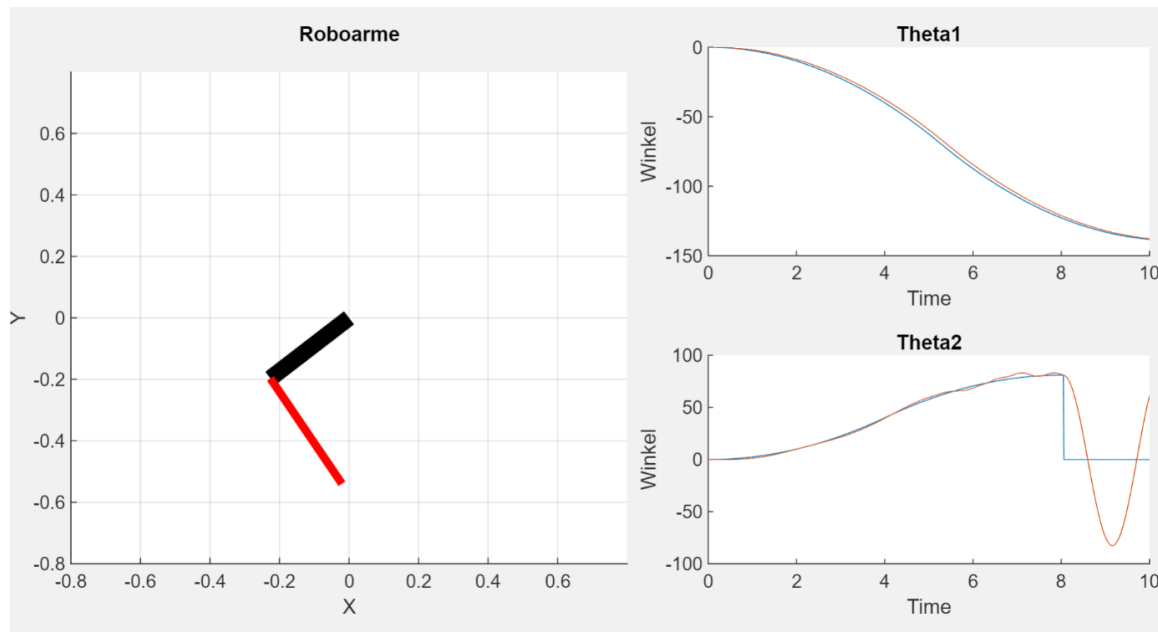


Abb.4.4.3: Schrittweite 0.004 ode1

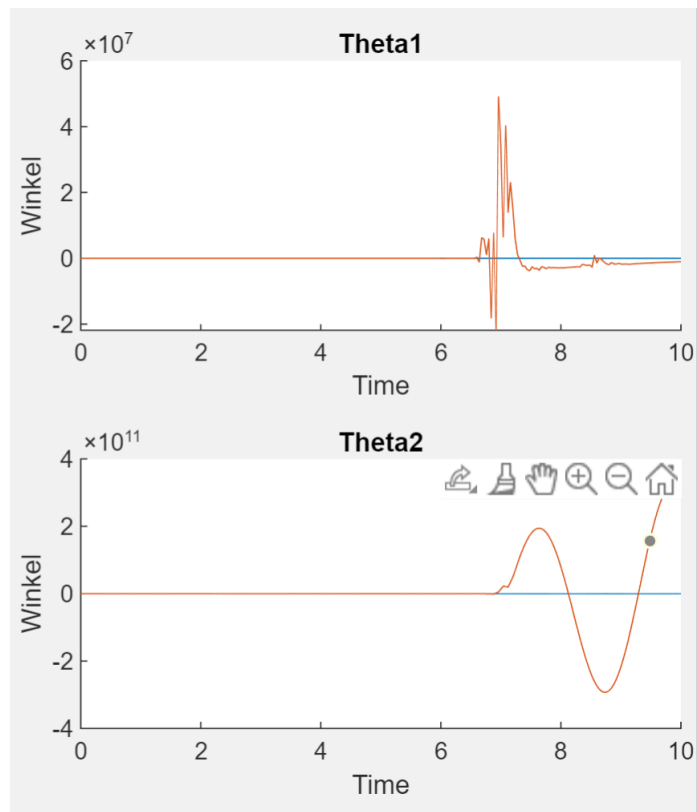


Abb.4.4.4: Schrittweite 0.04 ode1

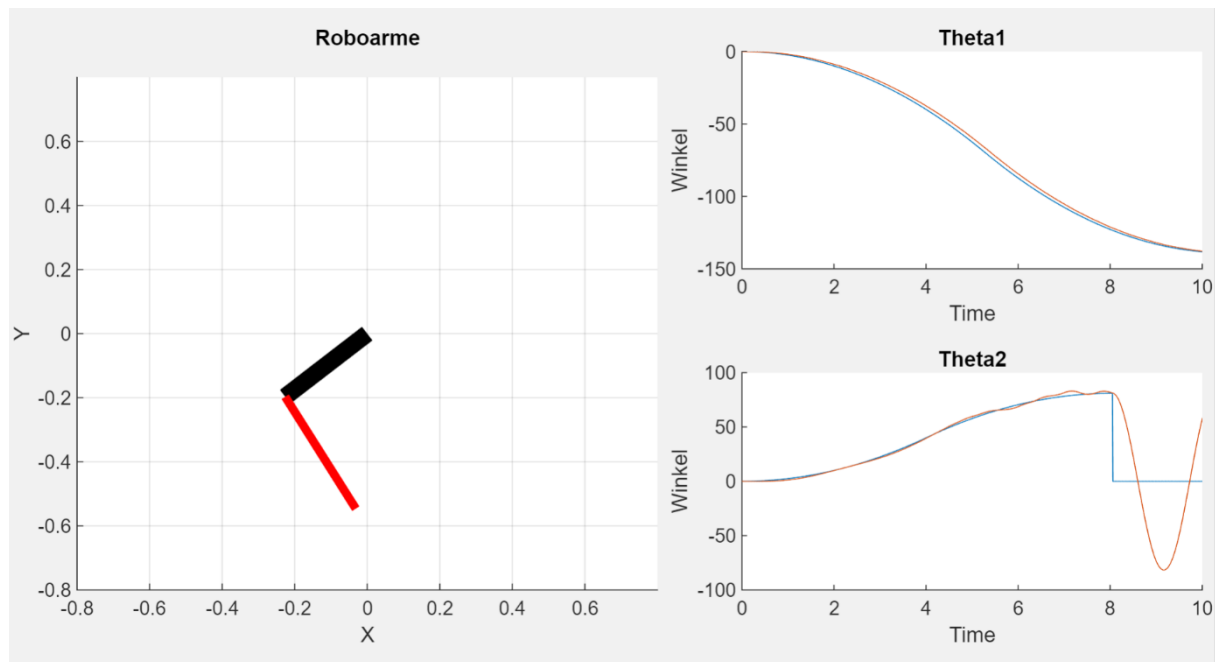


Abb.4.4.5: Schrittweite 0.005 ode2

*Analyse2: Bei der Simulation des Modells kommt es zu einer Warnung mit dem Schlagwort Algebraic Loop (Algebraische Schleife). Was bedeutet diese Warnung?*

Wenn die Warnung Algebraische Schleife auftritt, hat zur Bedeutung, dass die Funktion kein echtes ende erreicht, wie es im Gegensatz zu einer „for-Schleife“ besitzt diese ein ende, um diese Warnung/Fehler zu vermeiden muss eine eigene Abbruchbedingung oder ein Zeitlimit eingebettet werden.

## 4.5 Ergebnisse zu den Analysen zum Teil App (P4)

Autor: Ronny Tolloch Matr.

185631

*Analyse: Stelle den Arbeitsraum des Roboters grafisch dar.*

Grafische Darstellung des Roboter Arbeitsraumes.

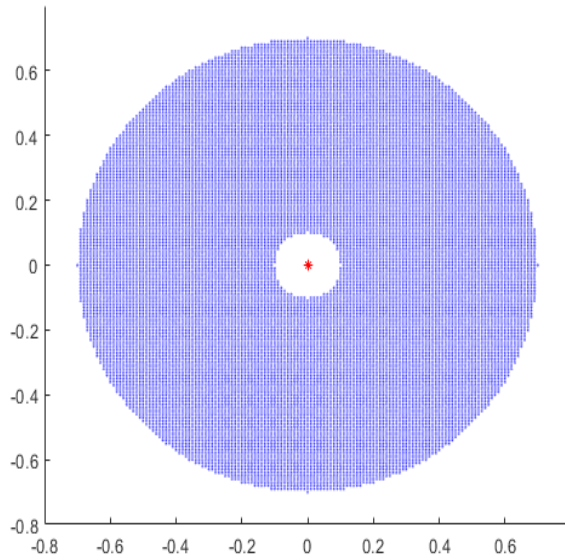


Abb.4.5.1: Roboter Arbeitsraum

Um diesen Arbeitsraum Grafisch in einem Plot darstellen zu können wird folgender Referenz Code verwendet!

*code Referenz:*

```
clear;
global l1 l2;
l1 = 0.3;
l2 = 0.4;
time = -1:0.01:1;

ax=axes('Parent',figure);
ax.XLim = [-(l1+l2+0.1),(l1+l2+0.1)]; % +0.1 für ein wenig Abstand zum Rand
ax.YLim = [-(l1+l2+0.1),(l1+l2+0.1)];

for x = 1:length(time)
    for y = 1:length(time)
        xa=time(x);
        ya=time(y);
        p(x,y) = fun_checkWorkspace(xa,ya); % prüft alle x + y Positionen
    end
end

hold(ax,"on")
for i=1:length(time)
    p(p==0) = NaN; % entfernt alle 0 so dass sie nicht angezeigt werden

    plot(ax,time,time(i)*p(i,:), "b.", 'MarkerSize',1)
end
```

```
plot(ax,0,0,"r*",'MarkerSize',5); % Mittelpunkt
hold(ax,"off")
```

Um einen Punkt in dem Arbeitsraum erstellen zu können, und zwar überall wo die Funktion „fun\_checkWorkspace“ sagt das dort kein Zugriff ist! Wird die folgende Funktion verwendet.

```
if (px^2+py^2) <= (l1+l2)^2 && ~((px^2+py^2) <= abs(l1-l2)^2)
```

Um diese Funktion etwas besser erklären zu können wird Schritt für Schritt drauf eingegangen

If %if-Bedingung, wenn Inhalt wahr ist, hat die App Zugriff auf die xy-Werte

$px^2+py^2 \leq (l1+l2)^2$  % benutzt Pythagoras zum Feststellen ob beide Arme zusammen zu weit weg sind

$\sim((px^2+py^2) \leq \text{abs}(l1-l2)^2)$  % macht das gleiche wie oben aber im negativen falls ein arm länger ist als ein anderer entsteht ein arbeitsfreier Raum, dieser wird hiermit erkannt

Man wiederholt dies nun für jede Zeile x und für jede Zeile y bis sich am Schluss im Plot ein Muster ergibt, was hier einem Donut ähnelt mit einem Radius von 0,7 Metern was der gemeinsamen Länge beider armen einspricht und einem inneren Radius von 0.1 Meter das sich aus  $0.4 \text{ m} - 0.3 \text{ m}$  bildet.

## 5. Zusammenfassung und Ausblick

### 5.1 Zusammenfassung:

*Wurde das in Kapitel 1 beschriebene Ziel erreicht? Sind alle Anforderungen erfüllt? Kritische Bewertung der Umsetzung (von jeder bearbeitenden Person)*

Paul Freynik Matr.Nr. 2017507935:

Alle die in dem Kapitel 1 beschriebenen Ziele wurden erreicht, die Umsetzung aller Formeln in die Funktionen wurden etabliert, die Grafische Darstellung der App sowie die sehr gute Unterstützung wurde durch Ronny Tolloch mehr als übertroffen. Im Großen und Ganzen ein gelungenes Projekt, was die Zusammenarbeit im Team gestärkt und im Grunde nach Fertigstellung jeder einzelnen Komponenten zu einem Ganzen sehr viel Spaß gemacht hat.

Eugen Risling: Matr. 2017507939:

Also das ganze Projekt insgesamt hat sehr viel Spaß bereitet. Um die Ziele und Anforderungen zu erreichen war nicht alles so einfach, denn man musste schon ein oder anderen Überlegung machen. Die geschriebenen Funktionen haben nicht immer sofort alle einwandfrei funktioniert, dabei entstanden immer wieder die Fehler die beseitigt werden mussten. Aber durch gute Unterstützung von den anderen Leuten aus der Gruppe ist man letztendlich alle Anforderungen und die Ziele erreicht hat.

Georg Kulodzik Matr. 2017507931

Die Ziele des Projekts sind erfolgreich umgesetzt worden. Die App kann alle geforderten Funktionen umsetzen. Die einzelnen Punkte der Aufgabe sind geprüft worden und die Grafiken und Analysen sind vorhanden. Es war eine neue Herausforderung so ein komplexes Thema in einer Gruppe durchzuführen und sich zu viert gut abzusprechen. Die Schnittstellen wurden durch das Schaubild in Kapitel 3.1 erst richtig greifbar und verständlich. Da die einzelnen Funktionen abhängig von der App bzw. von vorherigen Funktionen abhängig sind, ist hier eine enge Zusammenarbeit notwendig und hat bei uns gut funktioniert. Allerdings war es dadurch auch nicht immer sofort möglich zu prüfen, ob das programmierte auch fehlerfrei ist. Das Simulink Modell hat viel Konzentration für die Umsetzung gefordert und ist aufgrund des Umfangs auch nicht sofort fehlerfrei gewesen. Der in der Dynamik erstellte Verlauf der Roboterbewegung kann mich nicht zu 100% überzeugen. Der TCP läuft nach meiner Meinung zu unruhig zum Ziel, allerdings sehe ich auch keine Möglichkeit, das mit unserem Fachwissen noch weiter zu verbessern. Dennoch sehe ich das Projekt als einen Erfolg und das auch aufgrund der guten Unterstützung der Dozenten und Tutoren.

Ronny Tolloch Matr. 185631

Die App ermöglicht es einen unerfahren Endnutzer, der beispielsweise in einer Firma sitzt und die Winkel für einen echten Roboterarm braucht, diese einfach zu bekommen und somit ist das Ziel des Projektes erfüllt. Das in diesem Projekt genutzte Beispiel (Robotik) kann nun auf alle möglichen verschiedenen Bereiche angewendet werden und Probleme auch größere können mithilfe einer Gruppe bewältigt werden, Durch eine gute Aufgabenteilung und koordinierter Planung war es allen möglich deren Freizeit zu genießen sei es zum Lernen der anderen Klausuren oder Urlaub und trotzdem für andere Gruppen Mitglieder da zu sein. Wir waren sehr gut in der Lage unsere Schwächen gegenseitig zu ergänzen und so erfolgreich ein nutzbares Projekt herzustellen.



## 5.2 Ausblick:

*Wie könnte die App weiterentwickelt werden? Welche weiteren Features könnte die App noch besitzen?*

Da die App schon sehr gut vorbereitet wurde und eine Erweiterung dieser durch die Einstellung der Arm Geometrien im Sheet „Arm Setting“ umgesetzt wurde, könnte man diese noch zusätzlich durch externe Sensoren erweitern, um Hindernisse oder Werkstückerkennung zu detektieren. Eine mögliche Weiterentwicklung der App wäre eine Umstellung der Maßeinheit. Im Maschinenbau ist die Bemaßung in Millimetern üblich, was vielen Bedienern somit entgegenkommen würde. Eine weitere Verbesserung wäre die Darstellung des Arbeitsraums bei der Bewegung des Roboters. Zusätzlich wäre es denkbar das Gewicht was der TCP Trägt in die Rechnung mit einfließen zu lassen und dadurch die Bewegungsgeschwindigkeiten zu verlangsamen oder anzupassen, damit keine Schäden am Roboterarm entstehen. Bei wiederkehrenden anzufahrenden Positionen könnte man eine Art Speicher entwickeln, der die Bahn optimiert und später einfach nur abruft, ohne die Berechnung immer neu ausführen zu müssen.