| Achse | $a_{i-1}$ | $\alpha_{i-1}$ | $d_i$ | $\theta_i$ | Art |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | $-90°$ | Rotation |
| 2 | $l_1 = 0.16$ m | 0 | 0 | $\alpha$ | Translation |
| 3 | $l_2 = 0.128$ m | 0 | 0 | $\beta$ | Both |

Tabelle 1: DH-Parameter des Knickarmroboters

# 1 Weihnachtsprojekt: Simulation und Regelung eines Knickarmroboters

## Aufgabe 1 - Bestimmung der Kinematik und DH-Parameter

Zur Bestimmung der Kinematik ist es notwendig herauszufinden, wie die einzelnen Gelenke des Roboters miteinander verbunden sind. Die Denavit-Hartenberg-Notation wird verwendet, um die 3D-Transformation zum nächsten Gelenk mithilfe von vier Parameter zu beschreiben. Im Fall des Knickarmroboters ist dies sehr einfach, da es sich lediglich um eine Kette von drei Gelenken handelt. Die Koordinatentransformation ist jeweils eine Rotation um die $z$-Achse mit $\theta_i$ und eine Translation in der $xy$-Ebene, um die Länge $a_i = l_i$. Wir können somit das erste Gelenk in den Ursprung legen und mithilfe von zwei Transformationen $T_{12}$ und $T_{23}$ alles beschreiben. Um mit einem nicht gedrehten Koordinatensystem anzufangen, benötigen wir zusätzlich $T_{01}$. Die DH-Parameter sind in Tabelle **??** aufgeführt.

Daraus resultieren die Transformationsmatrizen

$$T_{12}(\alpha, l_1) = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & l_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & \cos(\alpha) l_1 \\ \sin(\alpha) & \cos(\alpha) & 0 & \sin(\alpha) l_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

und

$$T_{23}(\beta, l_2) = \begin{pmatrix} \cos(\beta) & -\sin(\beta) & 0 & 0 \\ \sin(\beta) & \cos(\beta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & l_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos(\beta) & -\sin(\beta) & 0 & \cos(\beta) l_2 \\ \sin(\beta) & \cos(\beta) & 0 & \sin(\beta) l_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Die Gesamttransformation ergibt sich aus der Multiplikation der beiden Matrizen, welche mithilfe von trigonometrischen Additionstheoremen vereinfacht werden kann.

$$T_{13} = T_{12} \cdot T_{23} = \begin{pmatrix} \cos(\alpha+\beta) & -\sin(\alpha+\beta) & 0 & \cos(\alpha) l_1 + \cos(\alpha+\beta) l_2 \\ \sin(\alpha+\beta) & \cos(\alpha+\beta) & 0 & \sin(\alpha) l_1 + \sin(\alpha+\beta) l_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T_{03} = \begin{pmatrix} \cos(\alpha+\beta) & -\sin(\alpha+\beta) & 0 & \cos(\alpha) l_1 + \cos(\alpha+\beta) l_2 \\ \sin(\alpha+\beta) & \cos(\alpha+\beta) & 0 & \sin(\alpha) l_1 + \sin(\alpha+\beta) l_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Somit ergibt sich die Position von Gelenk 2 im Weltkoordinatensystem zu

$$\begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \begin{pmatrix} \cos(\alpha) l_1 \\ \sin(\alpha) l_1 \\ z_2 \end{pmatrix}$$

und die Position von Gelenk 3 bzw. des Endeffektors zu

$$\begin{pmatrix} x_3 \\ y_3 \\ z_3 \end{pmatrix} = \begin{pmatrix} \cos(\alpha) l_1 + \cos(\alpha + \beta) l_2 \\ \sin(\alpha) l_1 + \sin(\alpha + \beta) l_2 \\ z_3 \end{pmatrix}$$

*Anmerkung*: Da die Parameter $\alpha_{i-1}$ und $d_i$ sind immer null sind, ist eine 2D Betrachtung ausreichend, wie man auch an der *Identitäts-Zeile-Spalte* für $z$ erkennen kann.

## Aufgabe 2 - Bestimmung der Bewegungsgleichung

### Generalisierte Koordinaten

Die Wahl fällt zu $q_1 = \alpha$ und $q_2 = \beta$. Die generalisierten Koordinaten sind somit die Winkel der Gelenke 2 und 3.

### Lagrang'sche Gleichungen

### Jacobi-Matrizen

### potentielle Energie $U$

## Aufgabe 4 - Bewegungsgleichung in Matlab

Unsere Gruppe hat sich dafür entschieden die Bewegungsgleichung und ihre Lösung direkt in MatLab zu implementieren. Wie bereits in Aufgabenteil 2 erläutert wurde, haben wir die `Symbolic Toolbox von Matlab` benutzt, um die Bewegungsgleichung aufzustellen.

Unser Ziel ist es nun gewesen die aufgestellte Ordinary Differential Equations (ODE) mithilfe eines Solvers wie Euler-Vorwärts zu lösen. Dafür haben wir die Funktion `ode45` benutzt, welche die ODE numerisch löst. Die Funktion `ode45` benötigt als Eingabe die ODE, die Anfangsbedingungen und den Zeitbereich, in dem die ODE gelöst werden soll. Als Ausgabe liefert die Funktion die Lösung der ODE.

Wie vorgegeben haben wir als Optionen des Solvers `ode45` die `RelTol` auf $10^{-4}$ und `AbsTol` auf $10^{-7}$ gesetzt. Die `RelTol` gibt die relative Toleranz an, die die Lösung der ODE haben darf. Die `AbsTol` gibt die absolute Toleranz an, die die Lösung der ODE haben darf. Die Toleranzen sind wichtig, da die Lösung der ODE numerisch berechnet wird und somit nicht exakt ist. Die Toleranzen geben an, wie genau die Lösung der ODE sein muss. Außerdem haben wir eine maximale Schrittweite gesetzt mithilfe von `MaxStep` $= 3 \cdot 10^{-3}$. Das setzen dieser initiailen Werte geschieht in Listing **??**.

Listing 1: Aufruf der Funktion `ode45`

```
% Syntax: y_0 = [alpha; alpha_dot; beta; beta_dot, err_alpha, err_beta]
y_0 = [pi/2; 0.5; -pi/5; -0.1; 0; 0];
tspan = [0, 1];
opts = odeset('RelTol', 1e-4, ...
       'AbsTol', 1e-7, ...
       'MaxStep', 3*1e3);

[t, y] = ode45(odefun, tspan, y0, opts);
```

### Aufstellen der rechten Seite

Nun benötigen wir noch eine rechte Seite in der Form einer `odefun`, die abhängig ist von der Zeit und der Lösung der ODE. Die rechte Seite der ODE ist die Ableitung der Lösung der ODE. Haben wir diese können wir die ODE numerisch lösen und visualisieren.

Vorerst beschäftigen wir uns nicht mit der Implementierung des Reglers, welcher die Trajektorienplanung umsetzt, sondern schauen ganz grundlegend auf das Problem. Die Bewegungsgleichung erhalten wir als Symbolic Equation aus der von uns implementierten Funktion `bewegungsgl` und schreiben mithilfe des Befehls diese als Matlab-M-Funktion in den workspace. Aus Effizienzgründen machen wir das nur, wenn die Funktion noch nicht existiert oder wir explizit wollen, dass sie neu berechnet wird, siehe Listing: **??**. Zuerst hatten wir es mit `subs` probiert, aber dies ist langsamer und weniger elegant.

Listing 2: Aufruf der Funktion `ode45`

```
%% Bewegungsgleichung
if ~exist('func_y_ddot.m', 'file') || berechne_doppelt
    symbolic_y_ddot = bewegungsgl();
    matlabFunction(symbolic_y_ddot, 'file', 'func_y_ddot.m');
end
```

Nun schreiben wir einen *Wrapper* für diese Funktion, da wir aus einer ODE 2. Ordnung eine ODE 1. Ordnung machen müssen. Dies geschieht in Listing **??**.

Listing 3: Definition der rechten Seite

```
function dy = assemble_odefun(t, y, func, reg)
% assuming y -> [alpha; alpha_dot; beta; beta_dot; err_alpha; err_beta]
    dy = zeros(6,1);
    u = reg.pid(t, y);

    % limit u values
    u_min = -1;
    u_max = 1;
    u = min(u_max, max(u_min, u));

    % Noise in friction can also be added to u
    noise = reg.noise_amp *2*(rand(2,1)-0.5);
    u = u + noise;

    % Bewegungsgleichung from symbolic toolbox
    y_ddot = func(y(1), y(2), y(3), y(4), u(1), u(2));

    dy(1) = y(2);
    dy(2) = double(y_ddot(1));
    dy(3) = y(4);
    dy(4) = double(y_ddot(2));
    % error from soll-value
    dy(5) = y(1) - reg.r_alpha(t);
    dy(6) = y(2) - reg.r_beta(t);
end
```

Das wichtigste was hier passiert ist das die Ableitung des Winkels einfach die Winkelgeschwindigkeit ist, welches ein Eingabewert ist. Erst die rechte Seite der Winkelgeschwindigkeit ist dann die Bewegungsgleichung. Hier fügen wir unseren Regler hinzu und das Rauschen, welches wir in der Aufgabenstellung vorgegeben bekommen haben.

## Resultat und Visualisierung

Animiert wird die Bewegung des Knickarmroboters mithilfe des `plot` Befehls. Hierfür werden die bereits in Aufgabe 1 erläuterten homogenen Transformationsmatrizen genutzt, um die generalisierten Koordinaten $\alpha$ und $\beta$ auf kartesische Koordinaten zu übersetzen. Das erste Gelenk

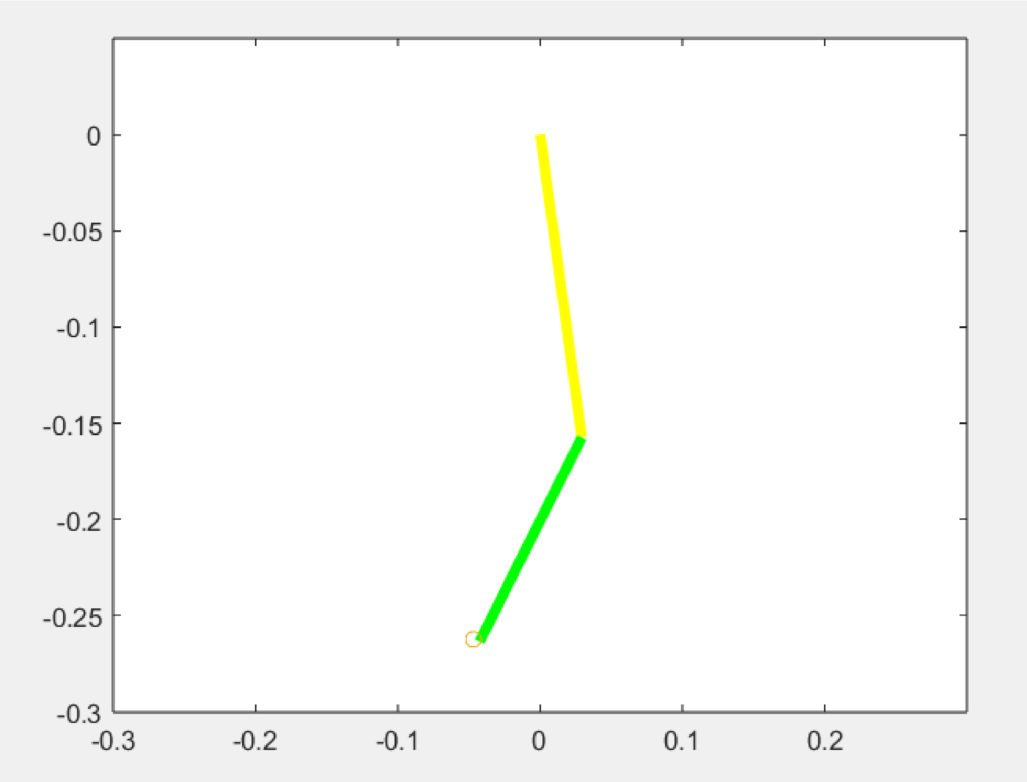


Abbildung 1: Visualisierung der Simulation

Listing 4: Definition der rechten Seite

```
    %% Visualization
if plot_Ergebnis
    alphas = y(:, 1);
    betas = y(:, 3);
    [T_02, T_03] = dh_trafo();
    orig = [0 0 0 1]';
    time = [diff(t); 0];

    % Animation
    figure(1);
    % plots maximal 250 frames
    max_frame = 250;
    n_frame = min(size(y,1), max_frame);
    for frame=1:n_frame
        i = floor(frame/n_frame .* numel(t));
        G2 = T_02(alphas(i), betas(i))*orig;
        G3 = T_03(alphas(i), betas(i))*orig;
        plot_robot(G2, G3)
        hold on
        plot(y_end_glob(1), y_end_glob(2), "o")
        hold off
        drawnow;
        %wait such that the animation is as long as the simulated time
        pause(time(i));
    end
```

end

## Diskussion