Learning Robots Exercise 1

TECHNISCHE UNIVERSITÄT DARMSTADT

Dominik Marino - 2468378, Victor Jimenez - 2491031, Daniel Piendl - 2586991
December 1, 2020

Contents

1 Robotics in a Nutshell

1.1 Forward Kinematics

- **Pseudo-algorithm of DH-convention**[1]:
  1. a) Nummierung der Glieder von 0 bi n (0 ist die Basis, n die Anzahl der Gelenke) und der Gelenke von 1 bis n

     b) Festlegung der z_i-Achse als koinzident mit der Bewegungsachse des Gelenks $i + 1$

     Schubgelenk: z_i-Achse als Schugachse in Richtung weg vom Gelenk $i + 1$, d.h. in Richtung der Schubrichtung (in Nullstellung ist das Schubgelenk eingefahren)

     Drehgelenk: z_i-Achse als Rotationsachse in Richtung positiver Drehwinkel (wird festgelegt)
  2. Festlegung des Basis-Frames S_0 mit Ursprung auf der z_0-Achse:
     Wahl von x_0, y_0 als Rechtskoordinatensystem; oft x_0, y_0 parallel zu x, y-Achsen des Weltkoordinatensystem
  3. Festlegung des Ursprungs S_i:
     - Falls z_{i-1} und z_i sich schneiden: Schnittpunkt wird Urspung von S_i
     - Falls z_{i-1} und z_i parallel: Festlegung des Ursprungs auf z_i am Gelenk $i + 1$
     - Sonst (d.h z_{i-1} und z_i windschief): Beide gemeinsame Normale zu z_i und z_{i-1}, Ursprung wird der Schnittpunkt dieser mit z_i
  4. Festlegung der x_i-Achse:
     - Falls z_{i-1} und z_i sich schneiden: x_i in (positiver oder negativer) Richtung der Normalen der von z_{i-1} und z_i aufgespannten Ebene.
     - Falls z_{i-1} und z_i parallel oder windschief sind: x_i-Achse in Richtung der gemeinsamen Normalen von z_{i-1} und z_i durch Ursprung von S_i, so dass die x_i-Achse die z_{i-1}-Achse schneidet
       - Falls die Orientierung der x_i-Achse nicht eindeutig ist, soll diese, falls möglich, vom Aktuellen Frame entlang des Glieds hin zum nächsten Frame gerichtet sein
     - Sofern x_i-Achse durch vorstehende Angaben noch nicht eindeutig bestimmt ist, soll diese so gewählt werden, dass sich eine möglichst einfache DH-Tabelle ergibt. Beachte: Die DH-Eigenschaften müssen auch nach der Festlegung der x_i-Achse eingehalten werden. Dies kann insbesondere die Wahl der positiven x_i-Richtung beeinflussen.
  5. Festlegung der y_i-Achse, so dass x_i, y_i, z_i ein Rechtskoordinatensystem bilden.
  6. Festlegung des Endeffektor-Doordinatensystems S_n:

     Der Ursprung von S_n wird meist in den sogenannten Tool Center Point (TCP) gelegt.

     a) Wenn keine besonderen Anforderungen für die Orientierung von S_n vorliegen, kann für die Transformation von S_{n-1} nach S_n eine möglist einfache Transformation verwendet werden. Häufig reicht eine einfache Translation aus.

     b) Falls das n-te Gelenk ein Drehgelenk und das Werkzeug (tool ein einfacher Greifer ist, wird das Endeffektor-Frame x_n, y_n, z_n (tool frame) in der Regel wie folgt definiert, d.h. gegenüber der Variante (a) ist unter umständen noch eine zusätzliche Rotation notwendig, um diese Lage zu erhalten
  7. Erstellung einer Tabelle von Gliederparametern θ_i, d_i, a_i und α_i, mit $i = 0, 1, \ldots, n$

  θ_i = Winkel zwischen x_{i-1} und x_i gemessen um z_{i-1}
  θ_i ist variabel, falls Gelenk i Drehgelenk

  d_i = Entfernung vom S_{i-1}-Ursprung entlang z_{i-1}-Achse zum Schnittpunkt mit x_i-Achse;
  d_i ist variabel, falls Gelenk i Schubgelenk

  a_i = Entfernung vom Schnittpunkt der z_{i-1}-Achse mit der x_i-Achse zum ursprung von S_i entlang der x_iAchse (a_i kann auch negativ sein, je nach Orientierung von x_i)

  α_i = Winkel zwischen z_{i-1} und z_i gemessen um x_i

[1] Source: Lecture Grundlagen der Robotik Page 32-35, Prof. Stryk

- Tranformatonsmatrix:

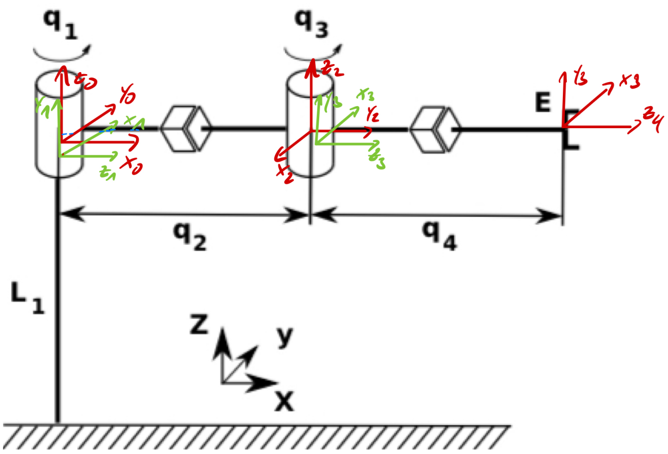
$$^{i-1}T_i = \begin{bmatrix} \cos\theta_i & -sin\theta_i cos\alpha_i & sin\theta_i sin\alpha_i & a_i cos\theta_i \\ sin\theta_i & cos\theta_i cos\alpha_i & -cos\theta_i sin\alpha_i & a_i sin\theta_i \\ 0 & sin\alpha_i & cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Useful Formulas:

$$sin(x + \frac{\pi}{2}) = cos(x) \quad \text{bzw.} \quad sin(x + 90^\circ) = cos(x)$$
$$cos(x + \frac{\pi}{2}) = -sin(x) \quad \text{bzw.} \quad cos(x + 90^\circ) = -sin(x)$$
$$cos(q_i - \pi) = -cos(q_i + \pi)$$
$$sin(q_i - \pi) = -sin$$
$$cos(q_i)^2 + sin(q_i)^2 = 1$$

- Additions theorem of sinus and cosinus:

$$sin(x \pm y) = sin(x)cos(y) \pm cos(x)sin(x)$$
$$cos(x \pm y) = cos(x)cos(y) \mp sin(x)sin(y)$$



- DH-Table:

Joint i	θ_i	d_i	a_i	α_i
1	$q_1 + \frac{\pi}{2}$	l_1	0	$\frac{\pi}{2}$
2	π	q_2	0	$\frac{\pi}{2}$
3	$q_3 - \pi$	0	0	$\frac{\pi}{2}$
4	0	q_4	0	0

- For simplicity we are using:

$$cos(q_i) = c_i \quad (1)$$
$$sin(q_i) = s_i \quad (2)$$
$$cos(q_1 + q_2) = c_{12} \quad (3)$$
$$sin(q_1 - q_2) = s_{1-2} \quad (4)$$

$$^0T_1 = \begin{bmatrix} cos(\theta_1 + \frac{\pi}{2}) & 0 & sin(\theta_1 + \frac{\pi}{2}) & 0 \\ sin(\theta_1 + \frac{\pi}{2}) & 0 & -cos(\theta_1 + \frac{\pi}{2}) & 0 \\ 0 & 1 & 0 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -s_1 & 0 & c_1 & 0 \\ c_1 & 0 & s_1 & 0 \\ 0 & 1 & 0 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^1T_2 = \begin{bmatrix} cos(\pi) & 0 & sin(\pi) & 0 \\ sin(\pi) & 0 & -cos(\pi) & 0 \\ 0 & 1 & 0 & q_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & q_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^2T_3 = \begin{bmatrix} cos(\theta_3 - \pi) & 0 & sin(\theta_3 - \pi) & 0 \\ sin(\theta_1 - \pi) & 0 & -cos(\theta_3 - \pi) & 0 \\ 0 & 1 & 0 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c_3 & 0 & -s_3 & 0 \\ s_3 & 0 & c_3 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^3T_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & q_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$
\begin{aligned}
{}^0T_4 &= {}^0T_1 * {}^1T_2 * {}^2T_3 * {}^3T_4 = \\
&= \begin{bmatrix} -s_1 & 0 & c_1 & 0 \\ c_1 & 0 & s_1 & 0 \\ 0 & 1 & 0 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & q_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} c_3 & 0 & -s_3 & 0 \\ s_3 & 0 & c_3 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & q_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \\
&= \begin{bmatrix} s_1 & c_1 & 0 & c_1q_2 \\ -c_1 & s_1 & 0 & s_1q_2 \\ 0 & 0 & 1 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} c_3 & 0 & -s_3 & 0 \\ s_3 & 0 & c_3 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & q_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \\
&= \begin{bmatrix} s_1c_3 + c_1s_3 & 0 & -s_1c_3 + c_1c_3 & c_qq_2 \\ -c_1c_3 + s_1s_3 & 0 & c_1s_3 + s_1c_3 & s_qq_2 \\ 0 & 1 & 0 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & q_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \\
&= \begin{bmatrix} s_1c_3 + c_1s_3 & 0 & -s_1s_3 + c_1c_3 & -s_1s_3q_4 + s_1s_3q_4 + c_1q_2 \\ -c_1c_3 + s_1s_3 & 0 & c_1s_3 + s_1c_3 & c_1s_3q_4 + s_1c_3q_4 + s_1q_2 \\ 0 & 1 & 0 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \\
&= \begin{bmatrix} s_{13} & 0 & c_{13} & q_4c_{13} + q_2c_1 \\ -c_{13} & 0 & s_{13} & q_4s_{13} + q_2s_1 \\ 0 & 1 & 0 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad => \quad {}^0r_4 = x_{end} = \begin{bmatrix} q_4c_{13} + q_2c_1 \\ q_4s_{13} + q_2s_1 \\ l_1 \end{bmatrix}
\end{aligned}
$$

1.2 Inverse Kinematics

- Given is the position ${}^0r_E \in \mathrm{R}^3$ and the orientation ${}^0R_E \in \mathrm{R}^{3x3}$ from the world coordinate system to the endeffector. The forward kinematic is given as:

$$
{}^0T_E(q) = \begin{bmatrix} {}^0R_E(q) & {}^0r_E(q) \\ \mathbf{0}^T & 1 \end{bmatrix} \stackrel{!}{=} \begin{bmatrix} {}^0R_E & {}^0r_E \\ \mathbf{0}^T & 1 \end{bmatrix} = {}^0T_E
$$

  A formal system has a 3 * 3 + 3 = 12 nonlinear equations. For the inverse kinematic, there are a variety of possible solutions (INV Euler or INV RPY (Roll-Pitch-Yaw)). The easiest solution is, when we have the current joint position (orientation of ${}^0R_E$) and the target position (position of ${}^0r_E$). By that can we calculate the joint angles **q** very easily. We dont have to calculate 9 independent not linear equations. Now we have 3 equations. We also have to calculate all possible solution. For example to avoid a collision or of the technical descriptions of the joints.

$$
q_{i,min} <= q_i <= q_{i,max}
$$

  This can be really hard.

- Can we always accurately model the inverse kinematics of a robot with a function?
  - It can not always accurately modeled:
    - For n < 6 joints q_i: the inverse kinematic (INV KIN) has no general solution, only in special cases n-dimensional subsets of R^6)
    - For n = 6 joints q_i: the inverse kinematic (INV KIN) has just as many parameters as in equation
    - For n > 6 joints q_i: the inverse kinematic (INV KIN) has infinite solution of the manipulator

[2] Source: Lecture Grundlagen der Robotik Page 42

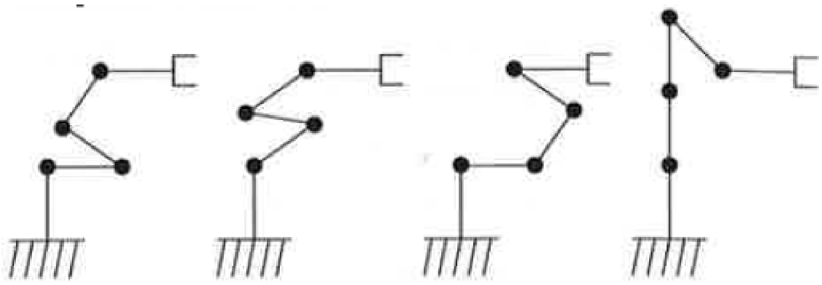


Figure 1: Manipulator with four DOF (degrees of freedom)[2]

1.3 Differential Kinematics

- Physiacal meaning of jacobian matrix:
  - The jacobian describes the velocity of the endeffector between the different joints. It is used for planning and execution, the determination of singularities or for the transformation of forces and moments.
- The Jacobian is given by:

$$\begin{aligned} {}^0v_n(t) &= {}^0J_{n,v}(q(t)) * \dot{q}(t) \quad \Longleftarrow \quad \text{linear velocity} \\ {}^0w_n(t) &= {}^0J_{n,w}(q(t)) * \dot{q}(t) \quad \Longleftarrow \quad \text{angle velocity} \end{aligned}$$

$$\underbrace{\begin{bmatrix} {}^0v_n(t) \\ {}^0w_n(t) \end{bmatrix}}_{\in R^3} = \underbrace{{}^0J_n(q(t))}_{(6\times n)-Matrix} * \underbrace{\dot{q}(t)}_{\in R^n} \quad \text{mit} \quad {}^0J_n = \begin{bmatrix} {}^0J_{n,v}(t) \\ {}^0J_{n,w}(t) \end{bmatrix}$$

- Calculation of the linear velocity ${}^0v_n(t)$:

$${}^0J_{n,v}(q(t)) = \begin{bmatrix} \frac{\partial {}^0r_n(q)}{\partial q_1} & \dots & \frac{\partial {}^0r_n(q)}{\partial q_i} & \dots & \frac{\partial {}^0r_n(q)}{\partial q_n} \end{bmatrix} \quad \longleftarrow \quad \text{3 rows}$$

$${}^0J_{4,v} = \begin{bmatrix} -(s_{13}q_4 + s_1q_2) & c_1 & -q_4s_{13} & c_{13} \\ c_{13}q_4 + c_1q_2 & s_1 & q_4c_{13} & s_{13} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

- Calculation of the angle velocity:
  1. Case: Joint $i, i \in \{1, 2, \dots, n\}$ is a revolute:

$${}^0R_{i-1}(q(t)) \cdot {}^{i-1}w_i() = \dot{q}_i \cdot \underbrace{{}^0R_{i-1}(q(t)) \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}}_{{}^0e_{z_{i-1}}} = \dot{q}_i \cdot {}^0e_{z_{i-1}}$$

  with

$${}^ie_{x_i} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad {}^ie_{y_i} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad {}^ie_{z_i} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

  2. Case: Joint $i, i \in \{1, 2, \dots, n\}$ is a prismatic:
     - Because of the DH-Convention $q_i = d_i$ its a translation therefore there is no angular velocity.

$$\Longrightarrow \quad {}^{i-1}w_i(t) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Therefore from case 1 and case 2:

$$^0J_{n,w}(q(t)) = \begin{bmatrix} p_1 *^0 e_{z_0} & \dots & p_i *^0 e_{z_{i-1}} & \dots & p_n *^0 e_{z_{n-1}} \end{bmatrix} \quad \longleftarrow \quad \text{3 rows}$$

$$p_i = \begin{cases} 0, \text{if } q_i = \theta_i \text{ (prismatic)} \\ 1, \text{if } q_i = \theta_i \text{ (revolute)} \end{cases}$$

$$^0J_{4,w} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

The Jacobi-Matrix $^0J_4$ is:

$$^0J_4 = \begin{bmatrix} \frac{^0r_1(q)}{\partial q_1} & \frac{^0r_2(q)}{\partial q_2} & \frac{^0r_3(q)}{\partial q_3} & \frac{^0r_4(q)}{\partial q_4} \\ ^0e_{z_0} & \mathbf{0} & ^0e_{z_2} & \mathbf{0} \end{bmatrix} = \begin{bmatrix} -(s_{13}q_4 + s_1q_2) & c_1 & -q_4s_{13} & c_{13} \\ c_{13}q_4 + c_1q_2 & s_1 & q_4c_{13} & s_{13} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

1.4 Singularities

- We can detect singularities with $det(^0J_n) = 0$. Prismatic joints cannot move into singularity. Singularities describes the loss of rank of the jacobian-matrix (loss one or more degrees of freedom). We have to check certains rows and column of the jacobian matrix.

$$J_{revolute} = \begin{bmatrix} -(s_{13}q_4 + s_1q_2) & -q_4s_{13} \\ c_{13}q_4 + c_1q_2 & q_4c_{13} \end{bmatrix}$$

$$\begin{aligned} det(J_{revolute}) &= -(s_{13}q_4 + s_1q_2) * q_4c_{13} - [(q_4c_{13} + q_2c_1) * -q_4s_{13}] = \\ &= -q_4^2s_{13}c_{13} - q_2s_1q_4c_{13} + q_4^2c_{13}s_{13} + q_2c_1q_4s_{13} = \\ &= -q_2s_1q_4c_{13} - q_2s_1q_4c_{13} = \\ &= -s_1c_{13} + c_1s_{13} \\ &= -s_1(c_1c_3 - s_1s_3) + c_1(s_1c_3 + c_1s_3) = \\ &= -s_1c_1c_3 + s_1^2s_3 + c_1s_1c_3 + c_1^2c_3 = \\ &= s_1^2s_3 + c_1^2c_3 = \\ &= s_3(s_1^2 + c_1^2) = \\ &= s_3 = 0 \end{aligned}$$

- The robot hits a singularity when $q_3 = 0$ or π. We reach the singularity because of the technical limit. We cannot rotate q_3 by 180 degrees when $q_1 = 0$, because the links are overlapping.
- We can avoid a kinematic singularity by adding another revolute joint. Alternative we can drive around the singularity, i.e exclude critic areas in the workspace with a corresponding trajectory planning.

1.5 Workspace

- For sorting items on a table is this robot not suitable. It is impossible for lifting items. This would make the sorting easier. Maybe the robot will throw the items on the floor while sorting. We would not recommend to buy this robot, because it is ineffective to sort items.

2 Control

2.1 PID Controller

The form of a PID controller is the following:

$$u(t) = K_P e(t) + K_I \int_{-\infty}^{t} e(\tau)\, d\tau + K_D \frac{de(t)}{dt}, \quad e(t) = q_d(t) - q(t)$$

With this controller it is possible to control the joints position and their velocities. The PID controller has three different components. The proportional part with its gain K_P scales the error between the desired and the measured state. The integral part sums the error over the time. This part of the controller eliminates the residual steady-state error which was accumulated over the time. The derivative term is determined by calculating the derivative of the error. The (ideal) D-Controller improves the settling time and the stability of the system.

The advantage of the PID-Controller is that it eliminates the residual steady state error. With the other linear controllers (P, PD) it is not possible to eliminate the steady-state error. The PID-controller comes shorthanded as soon as the system state continuously changes, because the accumulated error over time will increase the integral part of the controller. This will cause that the system turns unstable. A solution for this problem would be an anti wind-up PID controller.

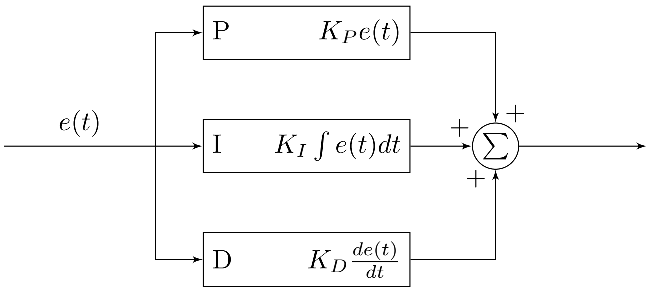


Figure 2: Block diagram PID Controller

2.2 Gravity Compensation and Inverse Dynamic Control

Using the model for the double inverted pendulum we can have a control law for the control vector u as a function of the system state.

$$u = M(q)\ddot{q}_{ref} + c(\dot{q}, q) + g(q)$$

Setting $\ddot{q}_{ref}$ as

$$\ddot{q}_{ref} = \ddot{q}_d + K_P(q_d - q) + K_D(\dot{q}_d - \dot{q}) + g(q),$$

it is possible to plug the control law in the forward dynamics model:

$$\ddot{q} = M^{-1}(q)(u - c(\dot{q}, q) - g(q)).$$

After plugin u in the previous equation we obtain the homogeneous second-order differential equation:

$$\ddot{q}_d - \ddot{q} + K_D(\dot{q}_d - \dot{q}) + K_P(q_d - q) = 0.$$

This yields to a system that behaves like a linear decoupled system.

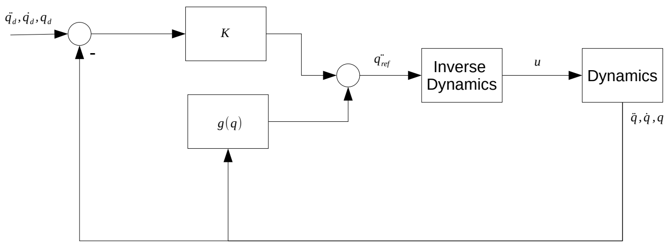


Figure 3: Block diagram Model-based control

2.3 Comparison of Different Control Strategies

The P-Controller yields the worst result compared to all the other controllers. The state of both joints oscillates continuously with high over-shooting and the steady-state is unreachable. This behavior is expected to happen because of the strong nonlinear nature of this system. The additional D-Controller stabilizes the behavior of the pendulum. Even though the joints' position is stable after 2 seconds and the end velocity equals zero, the desired state is not achieved. Nevertheless, the system behaves better with the PD-Controller.

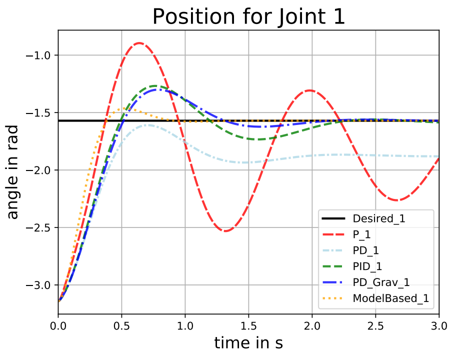
Intuitively one would assume that adding an I-Controller eliminates the residual error steady-state error. Instead of using the PID-Controller, because of the previously mentioned problems, the PD-Controller with gravity compensation is a good option for this kind of task. Both controllers reach the desired steady-state even though the PD-Controller with gravity compensation is slightly faster with significantly lower over-shooting. The joints' velocities over time using both controllers resemble one another reaching at some point the desired steady-state.

The Model-based Controller shows the fastest, most stable behavior of all the possible controllers. Because of the nonlinear nature of the problem, the proposed nonlinear controller is perfect for the task at hand. Nevertheless, the velocity of the first joint increases and decreases faster than all of the other controllers. These abrupt changes in the velocity result in high torques generated by the motors. Depending on the real system, the motors and joints may get damaged.

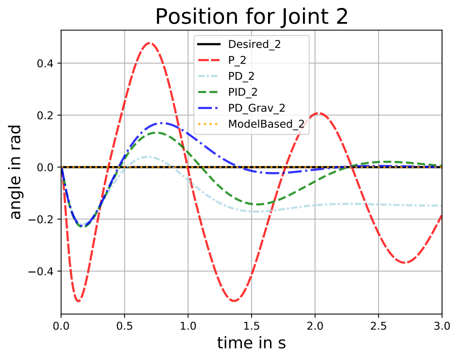
Judging only from the simulations, the time to reach the desired state, and the small overshooting, the best option is the model-based controller.

```python
gainMultiplier = 1

kp = np.array((60, 30)) * gainMultiplier
kd = np.array((10, 6)) * gainMultiplier
ki = np.array((0.1, 0.1)) * gainMultiplier


def my_ctl(ctl, q, qd, q_des, qd_des, qdd_des, q_hist, q_deshist, gravity, coriolis, M):
    if ctl == 'P':
        u = kp * (q_des - q)
    elif ctl == 'PD':
        u = kp * (q_des - q) + kd * (qd_des - qd)
    elif ctl == 'PID':
        if q_hist.shape[0] == 0:
            u = kp * (q_des - q) + kd * (qd_des - qd)
        else:
            u = kp * (q_des - q) + kd * (qd_des - qd) + ki * np.sum(q_deshist - q_hist, axis=0)
    elif ctl == 'PD_Grav':
        u = kp * (q_des - q) + kd * (qd_des - qd) + gravity
    elif ctl == 'ModelBased':
        u = M @ (qdd_des + kp * (q_des - q) + kd * (qd_des - qd)) + coriolis + gravity
    else:
        raise Warning(f"wrong definition for ctl: {ctl}")
    u = np.mat(u).T
    return u
```

Listing 1: my_ctl.py different controll strategies

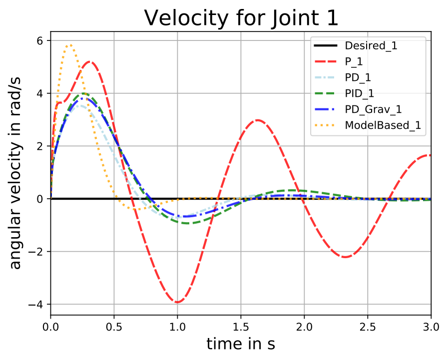


(a) Plot for the angular position of joint 1 for the steady-state trajectory

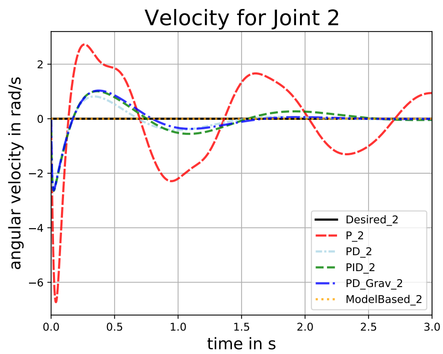


(b) Plot for the angular position of joint 2 for the steady-state trajectory

Figure 4: Compare position with controllers



(a) Plot for the angular velocity of joint 1 for the steady-state trajectory



(b) Plot for the angular velocity of joint 2 for the steady-state trajectory

Figure 5: Compare velocity with controllers

2.4 Tracking Trajectories

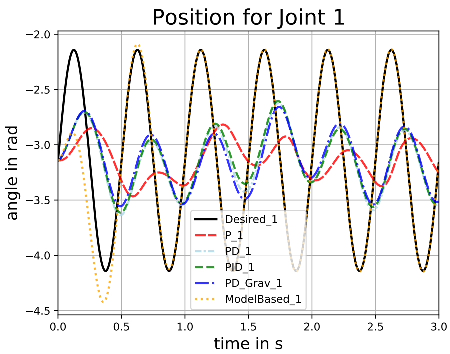
Figure 6a to Figure 7b show the plots for the position and velocity for both joints with an initial state of [-pi, 0] and a time-varying target depicted in black.

From all five controllers only the model-based controller is able to achieve an error of approx. zero within the first seconds and tracks the desired trajectory accurately for both joints. All other controllers produce unsatisfactory results.

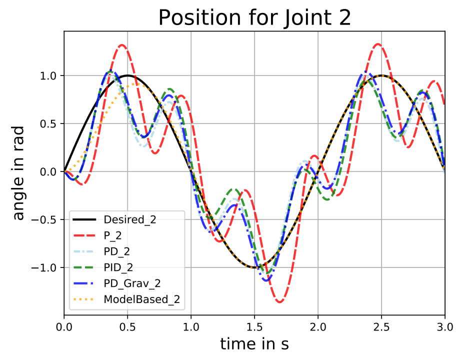
The PD, PID and PD with gravity compensation controllers all follow a similar trajectory. For the first joint the amplitude of the resulting trajectory for the position and the velocity is too low and is phase-delayed by approx. $\frac{\pi}{2}$. While the resulting trajectory for the second joint does in general follow the desired frequency of $0.5\,\text{Hz}$, the control for the first joint with a frequency of $2\,\text{Hz}$ heavily distorts the signal. In comparison to the position of joint one, the controllers track the desired trajectory better while still not as well as the model-based controller. The trajectory for the velocity of the second joint is dominated by the control for the first joint and only tracks the desired trajectory slightly.

The P-controller has the worst performance of all five controllers. Its trajectory for the first joint is a lot more "wobbly" and the phase-delay is higher and the error for the control of the second joint's position and velocity is a larger.

Based on these results, the model-based controller is the only one whose performance is satisfactory. While some of the other controllers perform better than the remaining ones, the results are still not sufficient. The model-based controller requires a detailed and exhaustive understanding and modeling of the physical system but given the defined control parameters it is the only viable solution in this case.

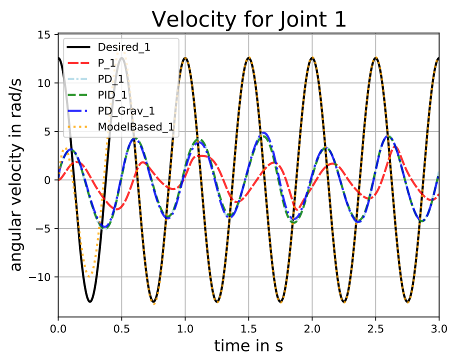


(a) Plot for the angular position of joint 1 for the tracked trajectory with normal gains. The PD, PID and PD_Grav trajectory overlap in a lot of places and are therefore not always visible.

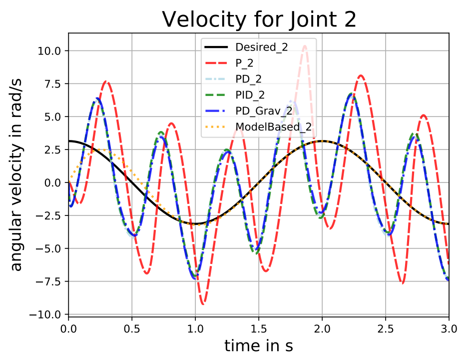


(b) Plot for the angular position of joint 2 for the tracked trajectory with normal gains. The PD, PID and PD_Grav trajectory overlap in a lot of places and are therefore not always visible.

Figure 6: Compare position with tracking behavior



(a) Plot for the angular velocity of joint 1 for the tracked trajectory with normal gains. The PD, PID and PD_Grav trajectory overlap in a lot of places and are therefore not always visible.



(b) Plot for the angular velocity of joint 2 for the tracked trajectory with normal gains. The PD, PID and PD_Grav trajectory overlap in a lot of places and are therefore not always visible.

Figure 7: Compare tracking velocity behavior

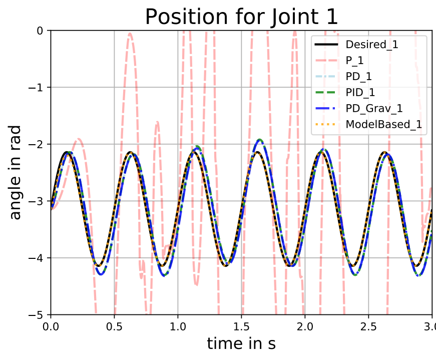
2.5 Tracking Trajectories - High Gains

Figure 8a to Figure 9b show the position and velocity of both joints when increasing the gains by a factor of ten in comparison to task 2 d). When increasing the gains, the model-based still performs the best. The desired trajectory is achieved within the first second.

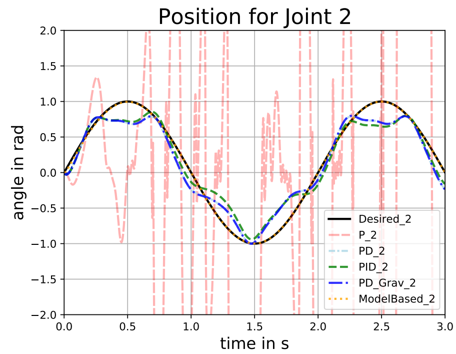
The PD, PID and PD with gravity compensation controller still perform similar to each other but a lot better than with the lower gains. The desired joint position trajectories are almost achieved. There is still a slight phase-delay but it is a lot smaller than in the previous task. The velocity of both joints is not as well traced as for the positions but also better than in the previous task, with larger errors for the second joint's velocity than for the first one. Only for the velocity for the second joint, all three controllers[3] exhibit a high-frequency oscillation within the first quarter second, probably due to the large initial error.

The P-controller performs worse than in the previous case. The large gains cause huge control inputs, resulting in a chaotic rotation of all joints. Thus, this controller should not be implemented in any way.

The huge gains make the use of the PD, PID and PD with gravity compensation controller more viable and a model-based controller is not necessarily needed anymore. On the other hand, the behavior of the P-controller shows that the controller needs to be carefully selected, otherwise the system can become unstable. Furthermore, huge gains can very easily lead to large control inputs that can potentially damage the system if no precautions are implemented.

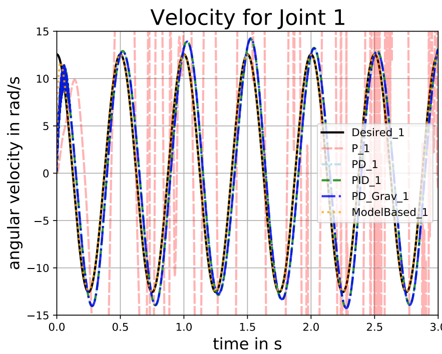


(a) Plot for the angular position of joint 1 for the tracked trajectory with high gains

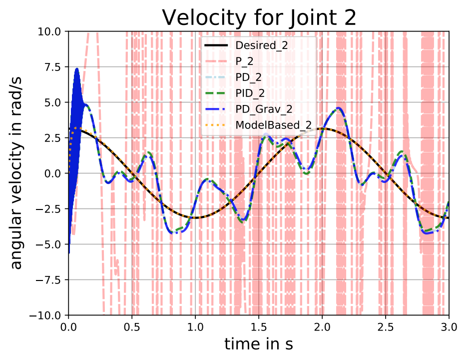


(b) Plot for the angular position of joint 2 for the tracked trajectory with high gains

Figure 8: Compare position tracking behavior with high gain



(a) Plot for the angular velocity of joint 1 for the tracked trajectory with high gains



(b) Plot for the angular velocity of joint 2 for the tracked trajectory with high gains

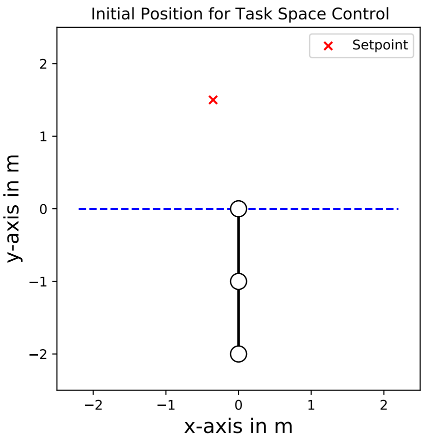
Figure 9: Compare velocity tracking behavior with high gain

[3] All three controllers follow the same trend at the start, but only the one for PD_Grav is visible in the plot

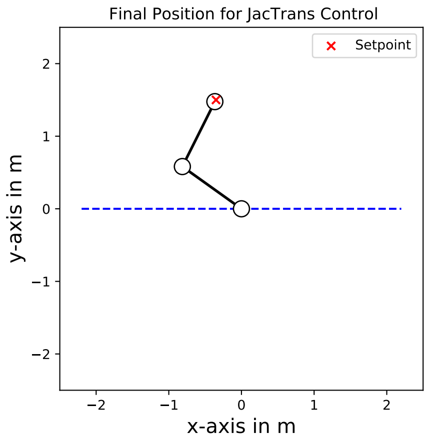
2.6 Task Space Control

Figure 10a to Figure 12b show the initial and final pose of the robot when utilizing different task space controllers and resting positions. For the controllers without null-space-movements, the JacTrans and the JacDPseudo achieve the final position but with different joint angles, while the JacPseudo controller fails to reach the setpoint. This could be because the JacDPseudo controller is more numerically stable than the JacPseudo controller due to the damped solution.

For the JacNullSpace Controller and for both resting positions the robot is able to reach the desired setpoint[4]. In none of the cases the resting position is actually achieved. The angle of the second joint is in both cases almost at π or $-\pi$, respectively. For the first resting position the angle of the first joint is closer to the desired 0° than in the second case. Because this robot operates in 2D-space and only has two degrees of freedom, there is a maximum of two different poses[5] that result in the desired setpoint. Thus, utilizing a resting position for this robot generally only influences which of the two different poses is ultimatly achieved.

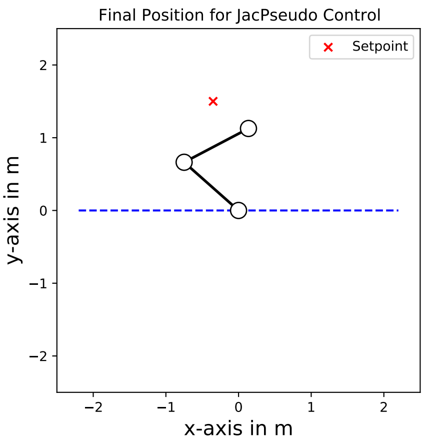


(a) Initial pose for task space control

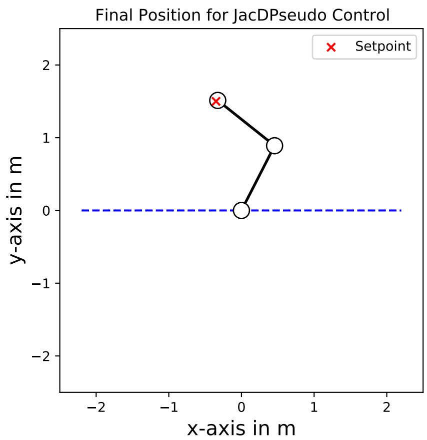


(b) Final pose for JacTrans Controller

Figure 10: Comparison of different task space controllers 1/3



(a) Final pose for JacPseudo Controller

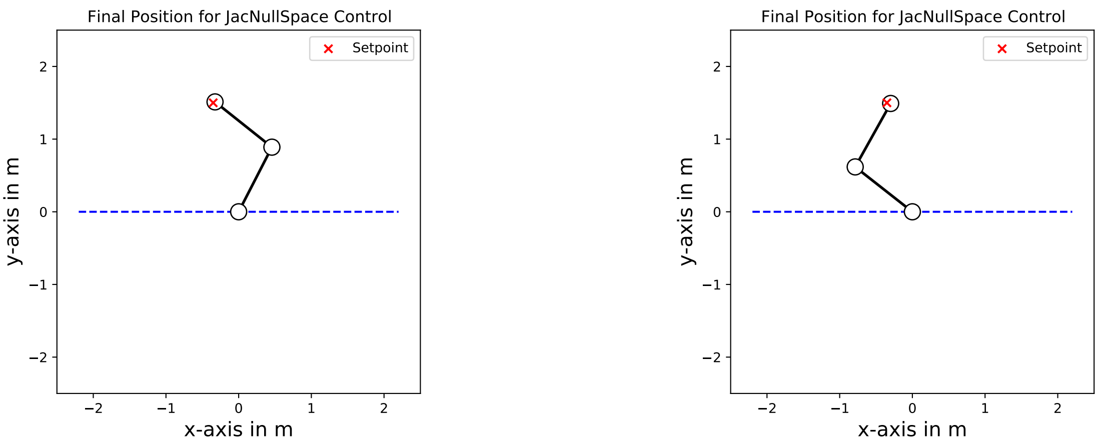


(b) Final pose for JacDPseudo Controller

Figure 11: Comparison of different task space controllers 2/3

[4] It seems that for the first resting position the setpoint is reached more precisely than for the second one. However this difference is very small and difficult to judge when only utilizing the plots

[5] Except for x=0 and y=0 with equally long joints.



(a) Final pose for JacNullSpace Controller with resting position (0, pi)

(b) Final pose for JacNullSpace Controller with resting position (0, -pi)

Figure 12: Comparison of different task space controllers 3/3

```python
def my_taskSpace_ctl(ctl, dt, q, qd, gravity, coriolis, M, J, cart, desCart, resting_pos=None):
    KP = np.diag([60, 30])
    KD = np.diag([10, 6])
    gamma = 0.6
    dFact = 1e-6

    if ctl == 'JacTrans':
        qd_des = gamma * J.T * (desCart - cart)
        error = q + qd_des * dt - q
        errord = qd_des - qd
        u = M * np.vstack(np.hstack([KP,KD])) * np.vstack([error,errord]) + coriolis + gravity
    elif ctl == 'JacPseudo':
        qd_des = gamma * J.T * np.linalg.pinv(J * J.T) * (desCart - cart)
        error = q + qd_des * dt - q
        errord = qd_des - qd
        u = M * np.vstack(np.hstack([KP,KD])) * np.vstack([error,errord]) + coriolis + gravity
    elif ctl == 'JacDPseudo':
        qd_des = J.T * np.linalg.pinv(J * J.T + dFact * np.eye(2)) * (desCart - cart)
        error = q + qd_des * dt - q
        errord = qd_des - qd
        u = M * np.vstack(np.hstack([KP,KD])) * np.vstack([error,errord]) + coriolis + gravity
    elif ctl == 'JacNullSpace':
        assert resting_pos is not None
        J_pseudoInverse = J.T * np.linalg.pinv(J * J.T + dFact * np.eye(2))
        q0 = KP * (resting_pos - q)
        qd_des = J_pseudoInverse * (desCart - cart) + (np.eye(2) - np.matmul(J_pseudoInverse, J)) * q0
        error = q + qd_des * dt - q
        errord = qd_des - qd
        u = M * np.vstack(np.hstack([KP,KD])) * np.vstack([error,errord]) + coriolis + gravity
    else:
        raise Warning(f"wrong definition for ctl: {ctl}")

    return u
```

Listing 2: Code for task space control