Computational Engineering und Robotik

Prof. Jan Peters, J. Carvalho, P. Klink, P. Liu, S. Stark

Sommersemester 2020

1. Übung

Hinweise zu dieser Übung

- Für die Teilnahme an der Übung ist eine Anmeldung beim **Moodle** notwendig. Dort sind auf der Kursseite zusätzliche Informationen zur Veranstaltung und die Regelungen zur Anrechnung der Übungsleistungen in die Endnote aufgeführt:

  `https://moodle.tu-darmstadt.de/course/view.php?id=20214`

- Von dieser Übung werden die **2. Aufgabe** und die **Programmieraufgabe** bewertet. Für die Bearbeitung der Programmieraufgabe lädst du den letzten Code vom *git repository* der im Moodle verlinkt ist.
- **Abgabe der schriftlichen Lösungen** zu den Übungsaufgaben bis **Mittwoch, den 06.05.2019 um 23:59 Uhr** per Datei-Upload im Moodle.
- **Abgabe der Programmieraufgabe** bis **Mittwoch, den 06.05.2019 um 23:59 Uhr** per Datei-Upload im Moodle.
- Inhaltliche Fragen zu dieser Übung werden in den Tutoren-Sprechstunden und dem Frage- und Antwort-Forum im Moodle beantwortet.
- Organisatorische Fragen bitte im Moodle Forum stellen.

1 Homogene Transformationen

In der Vorlesung wurde gezeigt, dass jede affine Transformation der Form ${}^a\boldsymbol{p} = {}^a\boldsymbol{r}_b + {}^a\boldsymbol{R}_b\,{}^b\boldsymbol{p}$ mit Koordinatenvektoren ${}^i\boldsymbol{p} \in \mathbb{R}^3$ eines Punktes P bzgl. Koordinatensystem S_i, Translationsvektor ${}^a\boldsymbol{r}_b \in \mathbb{R}^3$ und Rotationsmatrix ${}^a\boldsymbol{R}_b \in \mathbb{R}^{3\times 3}$ auch als *homogene Transformation* in $\mathbb{R}^4$ darstellt werden kann:

$$ {}^a\hat{\boldsymbol{p}} = \begin{pmatrix} {}^a\boldsymbol{p} \\ 1 \end{pmatrix} = \begin{pmatrix} {}^a\boldsymbol{R}_b & {}^a\boldsymbol{r}_b \\ \boldsymbol{0} & 1 \end{pmatrix} \begin{pmatrix} {}^b\boldsymbol{p} \\ 1 \end{pmatrix} = {}^a\boldsymbol{T}_b \cdot {}^b\hat{\boldsymbol{p}} $$

Homogene Transformationsmatrizen ${}^a\boldsymbol{T}_b$ haben folgende Eigenschaften:

(1) $ {}^a\boldsymbol{T}_b^{-1} = \begin{pmatrix} {}^a\boldsymbol{R}_b & {}^a\boldsymbol{r}_b \\ \boldsymbol{0} & 1 \end{pmatrix}^{-1} = \begin{pmatrix} {}^a\boldsymbol{R}_b^T & -{}^a\boldsymbol{R}_b^T\,{}^a\boldsymbol{r}_b \\ \boldsymbol{0} & 1 \end{pmatrix}$,

(2) $ {}^a\boldsymbol{T}_b^{-1} = {}^b\boldsymbol{T}_a $,

(3) $ {}^a\boldsymbol{T}_b\,({}^b\boldsymbol{T}_c\,{}^c\boldsymbol{T}_d) = ({}^a\boldsymbol{T}_b\,{}^b\boldsymbol{T}_c)\,{}^c\boldsymbol{T}_d $ (assoziativ) , aber $ {}^a\boldsymbol{T}_b\,{}^b\boldsymbol{T}_c \neq {}^b\boldsymbol{T}_c\,{}^a\boldsymbol{T}_b $ (nicht kommutativ).

a) Es sei S_O ein festes Koordinatensystem und S_a ein weiteres Koordinatensystem, dessen x_a-Achse in Bezug auf S_O in die Richtung $(0, 0, 1)$ zeige. Die y_a-Achse zeige in Richtung $(-1, 0, 0)$. Der Ursprung von S_a liege bei $(3, 0, 0)$. Gib die Rotationsmatrix ${}^O\boldsymbol{R}_a$, den Translationsvektor ${}^O\boldsymbol{r}_a$ und die Transformationsmatrix ${}^O\boldsymbol{T}_a$ an.
Zeichne (qualitativ) das Koordinatensystem S_O und relativ dazu die Lage von S_a.

b) Ein weiteres Koordinatensystem S_{b_O} liege zunächst deckungsgleich auf S_O. Es werde dann in folgender Weise transformiert:

(i) Drehung um π um die y_O-Achse

(ii) Translation in Richtung $(0, 2, 0)$

(iii) Drehung um $-\frac{\pi}{2}$ um die z_O-Achse

Abbildung 1: Rechte-Faust-Regel

Trage in deiner Zeichnung aus a) die Koordinatensysteme S_{b_i}, $S_{b_{ii}}$ und $S_b = S_{b_{iii}}$ entsprechend den Teiltransformationen (i), (ii) und (iii) ein.
Hinweis: Beachte dabei die Rechte-Faust-Regel aus Abbildung 1: Wenn der Daumen in Richtung der Drehachse zeigt, dann zeigen die gekrümmten Finger in Richtung der positiven Drehrichtung um diese Achse.
Gib die Transformationsmatrizen $\text{Rot}(y_O, \pi)$, $\text{Trans}(0, 2, 0)$ und $\text{Rot}(z_O, -\frac{\pi}{2})$ an und berechne damit ${}^O\boldsymbol{T}_b$. Überprüfe dein Ergebnis anhand der Zeichnung.

c) Bestimme ${}^O\boldsymbol{T}_a^{-1}$ ohne explizit eine Matrixinverse zu berechnen. Überprüfe dein Ergebnis anhand der Zeichnung.

d) Bestimme aus ${}^O\boldsymbol{T}_a$ und ${}^O\boldsymbol{T}_b$ die Transformationsmatrix ${}^a\boldsymbol{T}_b$ ohne explizit eine Matrixinverse zu berechnen. Überprüfe dein Ergebnis anhand der Zeichnung.

e) Es sei ${}^O\boldsymbol{p} = (2, 1, 0)^T$ gegeben. Berechne $\hat{\boldsymbol{p}}_1 = {}^O\boldsymbol{T}_a \, {}^O\hat{\boldsymbol{p}}$ und $\hat{\boldsymbol{p}}_2 = {}^O\boldsymbol{T}_a^{-1} \, {}^O\hat{\boldsymbol{p}}$. Entscheide und begründe für beide Fälle, ob der Punkt fest bleibt und sich nur das Bezugssystem ändert oder andersherum.

2 Direkte und inverse Kinematik (8 Punkte)

Den in Abbildung 2 dargestellten Roboter kannst du dich als Modell eines Baggers vorstellen: Das erste Gelenk stellt die horizontal drehbare Basis des Fahrzeugs dar, an dem in einigem Abstand der jeweils vertikal drehbare Ausleger und Löffelstiel angebracht sind.

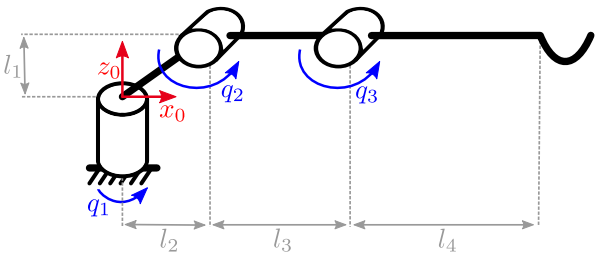


Abbildung 2: Modell des Baggers

Gelenk i	θ_i	d_i	a_i	α_i
1	q_1	l_1=1m	l_2=1m	$\frac{\pi}{2}$
2	q_2	0	l_3=2m	0
3	q_3	0	l_4=3m	π

Tabelle 1: Denavit-Hartenberg-Parameter

a) Zeichne in der Vorlage am Ende dieser Aufgabe alle lokalen Roboterkoordinatensysteme ein. Beachte dafür die zwei bereits eingezeichneten Achsen und die Parameter aus Tabelle 1 und halt dich an die Denavit-Hartenberg-Konvention:

- Die z_i-Achse liegt entlang der Bewegungsachse des $i+1$-ten Gelenks.
  Hinweis: Lass dich davon nicht verwirren: Die Aufzählung der Koordinatensysteme beginnt mit 0, wir sprechen aber vom 1. Gelenk.
- Die x_i-Achse steht senkrecht zur z_{i-1}- und z_i-Achse, zeigt von diesen weg und hat einen Schnittpunkt mit der z_{i-1}-Achse.
- Mit der y_i-Achse ergibt x_i, y_i, z_i ein Rechts-Koordinatensystem.

b) Berechne die Transformationen ${}^0\boldsymbol{T}_1$, ${}^1\boldsymbol{T}_2$ und stelle dann das Vorwärtskinematikmodell ${}^0\boldsymbol{T}_2$ auf. Berechne in Hinblick auf die Aufgabenteile c) und d) auch den Positionsvektor ${}^0\boldsymbol{r}_3$ des Endeffektors. Beachte hierbei, dass es dafür nicht notwendig ist ${}^0\boldsymbol{T}_3$ komplett zu berechnen!
Fasse trigonometrische Ausdrücke so weit wie möglich zusammen! z.B., $\sin(\alpha+\beta) = \sin\alpha\cos\beta + \cos\alpha\sin\beta$

c) Bestimme jetzt die Endeffektorposition des Baggerlöffels für $q_1 = 0$, $q_2 = -\frac{\pi}{4}$, $q_3 = \frac{\pi}{8}$. Gib das Ergebnis auf fünf Dezimalstellen genau an.

d) Es sei $q_2 = 0$ gegeben. Bestimme gültige Gelenkparameter q_1 und q_3 um die Endeffektorposition

$$ {}^0\boldsymbol{r}_3 = \begin{pmatrix} 0m \\ 6m \\ 1m \end{pmatrix} $$

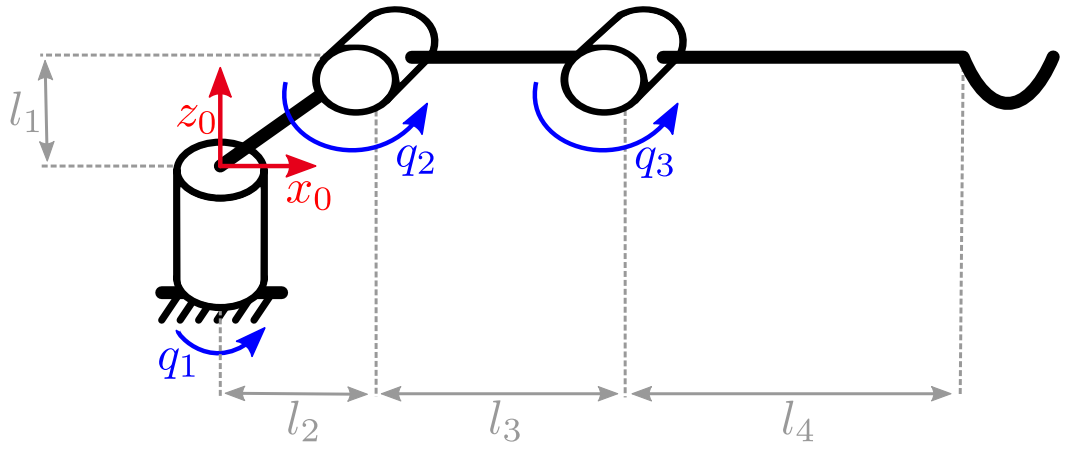
zu erreichen. Stelle dazu das Inverskinematikmodell auf und löse es zunächst symbolisch nach q_1 und q_3 auf. Setze erst dann konkrete Zahlenwerte für ${}^0\boldsymbol{r}_3$ und $l_i, i = 1,2,3,4$, ein und runde dein Ergebnis auf fünf Dezimalstellen.

e) Überlege und beschreibe anhand der Robotergeometrie (Abbildung 2), wie du die Gelenke einstellen müsstest, um die
Endeffektorposition

$$ {}^0\boldsymbol{r}_3 = \begin{pmatrix} 0\mathrm{m} \\ 1\mathrm{m} \\ 0\mathrm{m} \end{pmatrix} $$

zu erreichen. Warum kannst diese Parameter beim echten Roboter nicht einstellen?

Vorlage für Aufgabe 2 a)



Programmieraufgabe P1 Simulationsstudie 3D-Drucker (5 Punkte)

Im Rahmen einer Simulationsstudie soll das Verhalten eines 3D-Druckers untersucht werden. Mit dem in Abbildung 5 gezeigten Drucker sollen Buchstaben für einen Schriftzug gedruckt werden. Dafür soll der Drucker beliebige Buchstaben übergeben bekommen und ausgehend von einer Startposition diese drucken sowie im Anschluss wieder zurück zur Startposition fahren. Zur Realisierung sollen, wie nachstehend beschrieben, verschiedene Funktionen in PYTHON implementiert werden.

Bitte beachte die Hinweise zur Programmieraufgabe. Für diese Programmieraufgabe brauchst du nur die Datei *tasks/task1.py* bearbeiten.

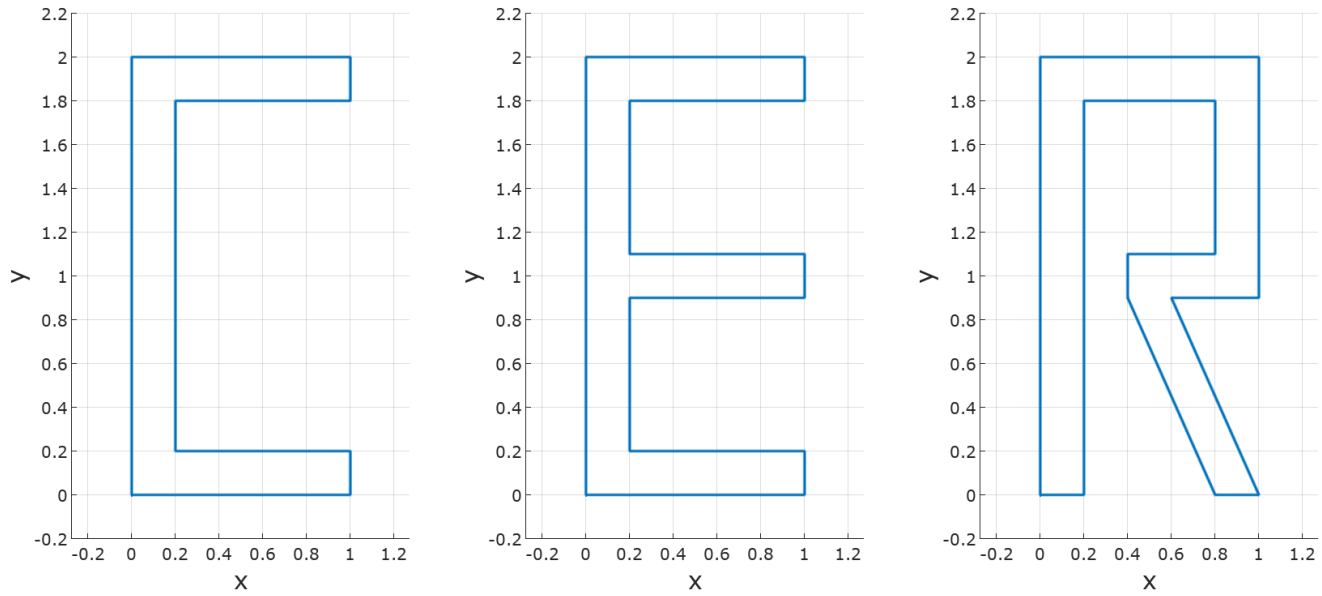


Abbildung 3: Vorlage Buchstaben

1. Implementiere eine Funktion, welche die Endeffektorpositionen der zu druckenden Buchstaben bereitstellt. Erstelle dazu eine Funktion

   `path, extrusion = dictionary(letter)`,

   welche zu dem mit dem Parameter `letter` übergebenen Buchstaben eine Matrix, bestehend aus den anzufahrenen Wegpunkten, in der Variable `path` und einen Spaltenvektor `extrusion` mit der Information zur Filamentextrusion für jeden Wegpunkt zurückliefert. *(1 Punkt)*

   - Jede Zeile der Matrix `path` soll die 3D-Koordinaten der Wegpunkte des jeweiligen angefragten Buchstaben enthalten.
   - Jeder Eintrag des Spaltenvektors `extrusion` soll für jeden Wegpunkt in `path` einen boolschen Wert enthalten, der angibt ob nach Erreichen des korrespondierenden Wegpunkts das Druckmaterial extrudiert werden soll (Wert 1) oder nicht (Wert 0).
   - Definiere die Wegpunkte entsprechend der in Abbildung 3 gezeigten Buchstaben.
   - Die Buchstaben sollen eine Einheit breit, zwei Einheiten hoch und 0.2 Einheiten dick sein. Dabei soll sich jeweils der Punkt „links-unten“ der einzelnen Buchstaben in der Koordinate $[0, 0, 0]$ befinden.
   - Die Start- und Zielkoordinate der Druckpfade mit Filamentextrusion sollen identisch sein und in der Koordinate $[0, 0, 0]$ liegen, sodass der Buchstabe geschlossen und in einem Zug ohne Absetzen gedruckt werden kann. Der Druckpfad soll in positive y-Richtung starten.

- Vor und nach dem Drucken jedes einzelnen Buchstabens soll sich der Druckkopf eine Einheit oberhalb der Start- und Zielkoordinate $[0, 0, 0]$ befinden
- Die Funktion `dictionary` muss lediglich die Buchstaben 'C', 'E' und 'R' unterstützen.
- Beispiel Buchstabe 'I':
  `path = [[0, 0, 1], [0, 0, 0], [0, 2, 0], [0.2, 2, 0], [0.2, 0, 0], [0, 0, 0], [0, 0, 1]]`
  und
  `extrusion = [0, 1, 1, 1, 1, 0, 0]`.

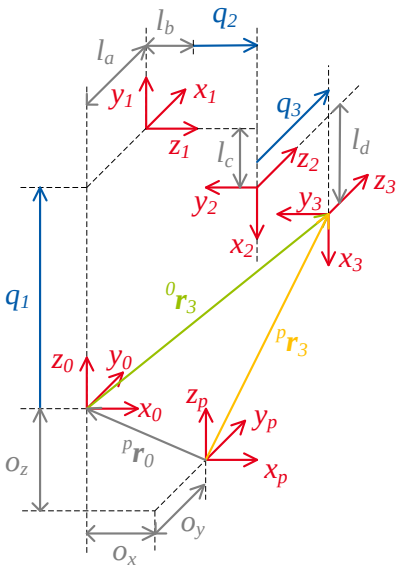


Abbildung 4: 3D-Drucker Kinematik

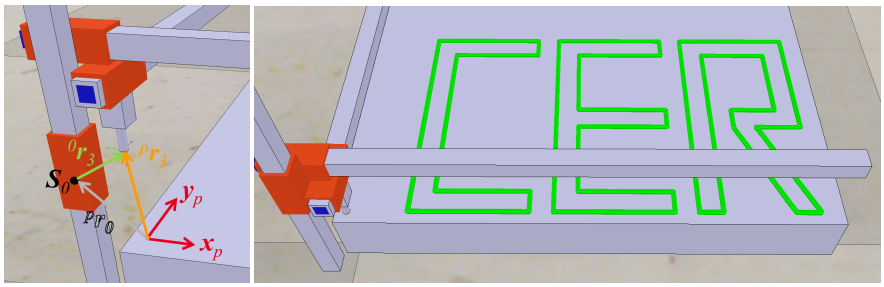


Abbildung 5: Visualisierung des Druckers in CoppeliaSim

Gelenk i	θ_i	d_i	a_i	α_i
1	$\frac{\pi}{2}$	q_1 (variabel)	l_a	$\frac{\pi}{2}$
2	$-\frac{\pi}{2}$	$q_2 + l_b$ (variabel)	l_c	$-\frac{\pi}{2}$
3	0	q_3 (variabel)	l_d	0

Tabelle 2: DH-Parameter des 3D-Druckers zur Bestimmung von ${}^0\boldsymbol{T}_3$

2. Als Teil der Druckvorbereitung werden in dem Skript `run_simulation.py` die Pfade der Buchstaben 'CER' ermittelt und für den Druck entsprechend angeordnet und skaliert. Daraus resultieren dann die für den gesamten Druck abzufahrenden Wegpunkte dargestellt im Koordinatensystem der Druckplatte S_p. Damit diese Vorgaben abgefahren werden können, müssen aus ihnen Gelenkvorgaben berechnet werden. Zudem muss die Druckkopfposition aus den aktuellen Gelenkstellungen berechnet werden können, um das Erreichen eines Wegpunktes zu überprüfen. Ermittle dafür die explizite Vorwärts- und Inverskinematik des Druckers. Erstelle eine Funktion

   `position = compute_forward_kinematic(q, kinematic_params, joint_limits)`,

   welche aus den übergebenen Gelenkstellungen `q` die Druckkopfposition ${}^p\boldsymbol{r}_3$ in 3D-Koordinaten erstellt. *(1.5 Punkte)*

   - Hierbei ist `q` ein $[3 \times 1]$-Spaltenvektor der Gelenkstellungen, zu denen die Druckkopfposition berechnet werden soll, `kinematic_params` ein $[7 \times 1]$-Spaltenvektor der Verschiebungen zwischen den einzelnen Koordinatensystemen der kinematischen Kette, `joint_limits` ein $[3 \times 2]$-Matrix der Gelenkpositionsbereiche und `position` ein $[3 \times 1]$-Spaltenvektor der Druckkopfposition, dargestellt im Koordinatensystem der Druckplatte S_p. Der Rückgabewert `position` soll ein leerer Vektor sein, wenn die übergebenen Gelenkstellungen die Gelenkpositionsbeschränkungen mit einer Toleranz von $\varepsilon =$ 1e-6 verletzen.
   - Bediene dich der DH-Parameter und Kinematikdarstellung aus Tabelle 2 und Abbildung 4 zur Berechnung der Transformation ${}^0\boldsymbol{T}_3$ und ${}^p\boldsymbol{T}_0$ und daraus der Vorwärtskinematik ${}^p\boldsymbol{T}_3$ mit dem Koordinatensystem S_p der Druckplatte und S_3 des Druckkopfes.

   Erstelle eine Funktion

   `q = compute_inverse_kinematic(position, kinematic_params, joint_limits)`,

welche aus der übergebenen Druckkopfposition `position` die dafür benötigten Gelenkpositionen `q` berechnet. *(1.5 Punkte)*

- Hierbei ist `position` ein $[3 \times 1]$-Spaltenvektor ${}^{p}\boldsymbol{r}_3$ der Druckkopfposition dargestellt im Koordinatensystem der Druckplatte S_p, `kinematic_params` bzw. `joint_limits` wie oben stehend beschrieben und `q` ein $[3 \times 1]$-Spaltenvektor der Gelenkstellungen zum Erreichen der angefragten Druckkopfposition. Der Rückgabewert `q` soll ein leerer Vektor sein, wenn die angefragte Druckkopfposition aufgrund der Gelenkpositionsbeschränkungen (mit einer Toleranz von ε = 1e-6) nicht erreichbar ist.

3. Über die realisierte Drucksimulation ist es nun möglich eine Kostenabschätzung basierend auf der Menge des verwendeten Druckmaterials durchzuführen. Zur Abschätzung der verwendeten Materialmenge kann der gefahrene Weg des Druckkopfes bei aktiver Extrusion über die Zeit des Drucks verwendet werden. Implementiere eine Funktion

   `consumption = estimate_filament_consumption(points, extrusions)`,

   welche am Ende jedes `letter` aufgerufen wird. Die übergebenen Druckkopfpositionen `points` und Extrusionszustande `extrusions`, enthalten alle Punkten am Ende ein Wegstrecke und die aktive Extrusion. Im `consumption`, die gesamt Materialverbrauch ist zurückliefert. *(1 Punkt)*

   - Hierbei ist `points` ein $[N \times 3]$ Spaltenvektor der N Wegstrecke Punkte Druckkopfpositionen, dargestellt im Koordinatensystem der Druckplatte, `extrusion` ein boolscher Wert, der angibt, ob zwischen Elemente in `points` das Druckmaterial extrudiert wird, und `consumption` ein Skalar mit der akkumulierten Wegstrecke bei aktiver Material Extrusion.

Hinweise zur Programmieraufgabe

- Für jedes Programmierenaufgabe, greife die git *repository*, der im Moodle verlinkt ist, und ziehe die letzte Updates mit: `git pull`.
- Auf der Startseite der Veranstaltung im Moodle ist eine kurze Einführung in Python und CoppeliaSim verlinkt, welche die wichtigsten Grundlagen zur Verwendung der Programme erläutert.
- Folge dich die Anleitung zum Installation und Ausführung von Programmen vom git *repository*.
- Für die Aufgabe brauchst du nur die Dateien im Ordner *tasks* auszufüllen.
- Zur Überprüfung deiner Lösung der einzelnen Teilaufgaben findest du im Ordner `tests` öffentliche Testfälle. Diese können durch den Aufruf der Befehl `make test1` (für task1) im Hauptordner ausgeführt werden. Zur Bewertung werden neben diesen Tests **weitere** Testfälle überprüft!
- Neben den mitgelieferten Testfällen kannst du die Simulation mit `make p1` (für task1) starten. Weitere Informationen findest du im git *repository*.
- Gib als Kommentar am Anfahng jedes Dateien, unter den Ordner `tasks`, deinen Name und deine Matrikelnummer an.
- **Es ist ausreichend, wenn EIN Gruppenmitglied die Lösung einsendet. Alle angegebenen Gruppenmitglieder einer Abgabe bekommen dann dieselbe Bewertung.**
- Für die Abgabe der Programmieraufgabe lade **nur die zu bearbeitenden Dateien** im Moodle hoch. Die unter Ordner `tasks` sich finden.

Hinweis zu wissenschaftlichem Arbeiten

Es ist nicht gestattet, Lösungen anderer Personen als die der Gruppenmitglieder als Lösung der Aufgabe abzugeben. Des Weiteren müssen alle zur Lösungsfindung verwendeten, darüber hinausgehenden, relevanten Quellen explizit angegeben werden. Dem widersprechendes Handeln ist Plagiarismus und ist ein ernster Verstoß gegen die Grundlagen des wissenschaftlichen Arbeitens, das ernsthafte Konsequenzen bis hin zur Exmatrikulation haben kann.