Abgabe 3

Grundlagen der Digitalisierung
Fachgebiet Datenverarbeitung in der Konstruktion
Prof. Dr.-Ing. R. Anderl

TECHNISCHE UNIVERSITÄT DARMSTADT

Inhaltsverzeichnis

1 Einleitung

Herzlich Willkommen bei der dritten und letzten Abgabe des Softwareprojektes vom DiK!

In dieser Abgabe müssen Sie:

- eine algebraische Lösung für die Bewegung des Roboters,
- eine Benutzeroberfläche für die Ansteuerung des Roboters, sowie alle dazu benötigten Funktionen erstellen
- und eine Bedienungsanleitung für das Bedienen Ihres Programms.

In den folgenden Kapiteln finden Sie ausführliche Hilfestellungen zu den Teilaufgaben dieser Abgabe sowie in Kapitel 6 FAQ eine Sammlung beantworteter Fragen zum letzten Abschnitt des Softwareprojektes.

Lesen Sie die folgenden Kapitel aufmerksam durch und konsultieren Sie vor den Sprechstunden vor allem die hier gesammelten FAQs. Dort werden im Vorfeld viele Ihrer Fragen beantwortet werden können!

Viel Erfolg bei der Bearbeitung dieser Abgabe!

2 Abgabe 3

2.1 Aufgabenstellung

Die 3. Abgabe startet am 02.12.2021 und endet am 21.01.2022 um 18:00 Uhr!!!

In der ersten Abgabe durften Sie innerhalb des PAP/NSD einen Block verwenden, mit dem der Roboter an eine bestimmte Position (X/Y/Z) fährt und direkt den *wrist*-Winkel einstellt. In der Realität gibt es diesen Block jedoch nicht. Der Roboter versteht nur Winkelangaben für seine verschiedenen Gelenke. In dieser Abgabe sollen Sie sich Gedanken dazu machen, wie dieser Teil des Programms realisiert werden kann, also wie aus Positionsdaten (X/Y/Z) Winkelangaben für die verschiedenen Gelenke des Roboters errechnet werden können und wie Sie das *wrist*-Gelenk im gewünschten Winkel ansetzen.

Diese Abgabe ist der letzte Teil Ihres Softwareprojektes. Wenn Sie diesen (durchaus komplexen) Teil des Softwareprojektes geschafft haben, können Sie stolz auf sich sein. Strengen Sie sich also an, damit Ihr Projekt gelingt.

2.1.1 Algebraische Lösung

Überlegen Sie sich, wie Sie aus Positionsdaten Winkelangaben erzeugen können. Schreiben Sie sich dazu mathematische Formeln heraus und überlegen Sie, wie Sie diese anwenden können. Verfeinern Sie Ihre Rechnungen, bis Sie für jedes Gelenk des Roboters eine Formel haben, die nur noch von den Positionsdaten abhängig ist. Verwenden Sie einen über den Übergabeparameter *hand2groundAng* festgelegten Winkel.

Ihre algebraische Lösung sollte eine Funktion mit folgenden In- und Outputs darstellen:

```
[baseRot, shoulder, elbow, wrist] = f(x, y, z, hand2groundAng)
```

Dies schränkt die Freiheitsgrade in Ihrer Rechnung ein und verringert die Möglichkeit, mehrere Lösungen zu erhalten. Weiterhin hilft es Ihnen beim späteren Programm, den Roboter einfach in die richtigen Positionen zum Scannen, Würfel greifen und Würfel bewegen zu bringen.

Kommentieren Sie ihre Rechnungen und benennen Sie komplexere Formeln, die Sie hierbei verwenden.

Tipp: Zur Berechnung der Winkelangaben können Sie den Cosinussatz verwenden.

Geben Sie für diesen Aufgabenteil Ihre vollständige Rechnung inkl. Skizze als PDF Datei ab.

2.1.2 Programmcode

Programmieren Sie den Algorithmus, der ein festgelegtes Muster erzeugt, die Benutzeroberfläche beinhaltet und die Bewegung des Roboters die Sie in der algebraischen Lösung errechnet haben umsetzt. Teilen Sie die anstehenden Aufgaben dabei sinnvoll auf Ihre Teammitglieder auf!

Jede Funktion Ihres Programmcodes sollte mindestens ein bis zwei Sätze Beschreibung enthalten, die Sie über

```
help funktionsname
```

aufrufen können. Diese Beschreibung enthält auch eine Erklärung für potentielle In- und Outputs. Grundlegend sollte Ihr Programmcode gut kommentiert sein. Die Kommentare helfen Ihren Kommilitonen beim Verständnis. Gemeinhin gilt der Anhaltspunkt, dass auf zwei Zeilen (schwierigen) Code mindestens eine Zeile Kommentar folgen sollte.

Es ist Ihnen überlassen, ob Sie alles in einem Skript (einer m.Datei) programmieren oder mehrere Skripte erstellen. Der Übersicht halber empfehlen wir Ihnen mehrere Skripte für Ihren Programmcode zu erstellen.

Folgende Funktionen muss Ihr Code ausführen können:

- Der Roboter muss ein aktuelles Muster scannen können
    - Dafür muss Ihr Programm die Würfelfarben, die oben liegen, zurückgeben können
- Der Roboter muss anschließend die Würfel so sortieren können, dass ein beliebig gewünschtes Muster erhalten werden kann
- Wenn ein Würfel bereits die gewünschte Farbe hat, soll der Roboter diesen nicht mehr greifen, sondern an seinem Platz liegen lassen
- Würfel müssen nicht getauscht werden können!
- Wenn die Farbe eines Würfels geändert werden muss, soll der Roboter jeden Würfel greifen können und ihn zum Drehen an den Extraplatz unterhalb des 3x3 Feldes legen
    - Auf dem Extraplatz soll der Roboter den Würfel drehen können
    - Es müssen so viele Drehvorgänge durchgeführt werden, bis die gewünschte Farbe oben liegt. Anschließend soll der Roboter den Würfel wieder zurück an seinen ursprünglichen Platz legen
    - Hinweis: Nach einem Drehvorgang kann es sein, dass der Roboter nicht weiß, welche Farbe oben liegt. Es kann daher sein, dass der Roboter den Würfel nach einem Drehvorgang auf dem Extraplatz nochmal scannen muss, um zu wissen, welche Farbe oben liegt

Geben Sie für diesen Aufgabenteil Ihren vollständigen Programmcode in einer oder mehreren m.Dateien ab.

2.1.3 GUI

Für die 3. Abgabe und die Bedienung des Roboters, müssen Sie unter anderem eine Benutzeroberfläche erstellen (GUI = Graphical User Interface). In der nachfolgenden Abbildung ist ein Beispiel für eine solche Benutzeroberfläche dargestellt.

Abbildung 1

Natürlich haben Sie beim Designen freie Hand und Ihre Benutzeroberfläche kann auch ganz anders aussehen.

Folgende Funktionen müssen allerdings enthalten sein:

- Ein Knopf zum Scannen des aktuellen Würfelmusters
- Eine Darstellung des gescannten Musters, das bspw. nach dem Scannen und nach dem Sortieren aktualisiert wird
- Die Möglichkeit ein neues, beliebiges Farbmuster einzustellen
    - Tipp: Es ist Ihnen überlassen wie ein Nutzer das Farbmuster einstellt. Es könnte beispielsweise durch Klicken auf die Würfel in der GUI oder durch ein Dropdown-Menü realisiert werden.
- Eine Darstellung des Wunschmusters
- Ein Knopf zum Sortieren des Würfelmusters (Wenn das Wunschmuster in der GUI eingestellt wurde, muss per Knopfdruck der Roboter damit beginnen die Würfel so zu sortieren, dass am Ende die Würfel das gewünschte Muster abbilden

2.1.4 Dokumentation

Damit ein beliebiger Nutzer später Ihr Programm ausführen kann, ist es wichtig, dass Sie es dokumentieren. Dokumentieren Sie also die Bedienung Ihres Programmes. Schreiben Sie dazu für den späteren Benutzer eine Bedienungsanleitung. Aus der Bedienungsanleitung sollte hervorgehen welches Programm das Startprogramm ist (wie startet man Ihr Programm) und wie mit der Benutzeroberfläche interagiert werden kann. **Maximaler** Umfang der Bedienungsanleitung sind zwei doppelseitig beschriebene Seiten.

Geben Sie für diesen Aufgabenteil Ihre vollständige Bedienungsanleitung als PDF Datei ab.

2.2 Hilfe

Don't Panic!

Bei der Erstellung dieses Tutorials haben wir uns bemüht, Ihnen möglichst präzise Hilfen anzubieten, ohne Sie dabei in Ihrer Freiheit zu sehr einzuschränken. Trotzdem konnten wir wahrscheinlich an dieser Stelle nicht alle Ihre Fragen beantworten. Wenn Sie also im weiteren Verlauf des Softwareprojektes nicht weiterkommen, dann zögern Sie nicht, Ihren Tutoren oder dem Orga-Team Fragen zu stellen! In Kapitel 6 FAQ finden sie auch eine Sammlung wichtiger Anforderungen an das Projekt, viele Ihrer Fragen können dort möglicherweise geklärt werden.

Die Tutoren sind in erster Linie für die Studierenden in der entsprechenden Übungsgruppe zuständig und können Ihre Fragen nur dann beantworten, wenn sonst gerade keine anderen Fragen gestellt werden. Erfahrungsgemäß ist in späteren Blöcken mehr Zeit für einzelne Fragen. Wenn nicht genügend Zeit bleibt, können Sie auch in den Sprechstunden (s.u.) vorbeischauen.

2.3 Sprechstunden

Für das Softwareprojekt werden gesonderte Sprechstunden angeboten. Dort können Sie mit dem Orga-Team und zur Verfügung stehenden Tutorinnen und Tutoren Ihre Probleme klären.

Die Sprechstunden finden diesmal ohne Anmeldung in moodle zu folgenden Terminen von je 11:40 Uhr bis 13:30 Uhr auf Discord statt:

- 14.12.2021
- 11.01.2022
- 18.01.2022

Um den größten Nutzen aus der Sprechstunde zu ziehen, bereiten Sie sich bitte gut vor und stellen Sie konkrete Fragen zu Problemen, die Sie aktuell bei der Umsetzung des Softwareprojektes haben.

Werfen Sie also vor der Sprechstunde einen Blick in das Kapitel 6 FAQ, möglicherweise werden manche Ihrer Fragen dort geklärt.

Hinweis: In der emb Woche und in der Weihnachtspause finden keine Sprechstunden statt.

2.4 Abgabeformat

Das Abgeben Ihrer Lösungen erfolgt online über moodle. Für die 3. Abgabe müssen Sie **3 Teilaufgaben** in einer **Zip-Datei** abgeben:

- **Algebraische Lösung (PDF)**
- **Bedienungsanleitung (PDF)**
- **Programmcode (eine oder mehrere m.Dateien)**

Andere Dateiformate werden nicht akzeptiert. Für den Fall, dass Sie nicht wissen wie Sie eine Zip-Datei erstellen, haben wir eine Erklärung auf moodle hochgeladen.

Da es sich bei der 3. Abgabe um eine Gruppenabgabe handelt, muss jedes Teammitglied auf moodle die Abgabe bestätigen, damit Sie Ihre Aufgabe abgeben können! Hierbei sei auch noch angemerkt, dass nachdem die letzte Person aus der Gruppe die Abgabe für die Gruppenlösung bestätigt hat, eine Änderung der Abgabe **NICHT** mehr möglich ist. Es müssen **alle** Gruppenmitglieder auf moodle die Abgabe bestätigen, damit die Abgabe erfolgreich ist.

3 Erste Schritte mit der Entwicklungsumgebung

In diesem Abschnitt möchten wir Ihnen Vorgehensweisen anbieten, durch die eine erfolgreiche Bearbeitung des Softwareprojektes erleichtert wird. Schauen Sie sich dieses Kapitel im Besonderen an und versuchen Sie, die vorgestellten Schritte umzusetzen, bevor Sie andere Teile des Softwareprojektes angehen.

3.1 In moodle zur Verfügung gestellte Matlab-Dateien

Für die Bearbeitung des 3. PA haben wir Ihnen in moodle bereits einige Matlab-Dateien zur Verfügung gestellt. Die Dateien finden Sie in moodle unter dem Abschnitt „3.Abgabe“ unter dem Namen „Matlab Entwicklungsumgebung“. Es handelt sich hierbei um eine Zip-Datei. Laden Sie die Datei herunter und speichern Sie sie in einem beliebigem Ordner auf Ihrem Computer. Nun müssen Sie die Datei entpacken (Rechtsklick → 7-Zip → Extract here). Die Datei enthält die Matlab Dateien:

- Cube (.m und .p Datei), *diese Datei stellt Ihnen bereits vordefinierte Funktionen für das Arbeiten mit den* Farbwürfeln zur Verfügung
- Robot (.m und .p Datei), *diese Datei stellt Ihnen bereits vordefinierte Funktionen für das Bewegen des Roboterarms zur Verfügung*
- textColorOut (bereits aus dem 1.PA bekannt)
- VirtualRobot (.m und .p. Datei), *diese Datei stellt Ihnen eine virtuelle Entwicklungsumgebung des Roboters zur Verfügung und alle Funktionen, die für das Arbeiten mit der virtuellen Entwicklungsumgebung benötigt werden*

Kopieren Sie nun alle Dateien in Ihren gewünschten Matlab Ordner (der Ordner, auf den Sie mit Matlab zugreifen). Sowohl die .p Dateien als auch die .m Dateien!! Wie im 1. PA müssen Sie mit den .p Dateien **NICHTS** machen, sondern Sie arbeiten nur mit den .m Dateien.

In den Dateien, die wir Ihnen zur Verfügung stellen, müssen Sie nichts ändern! Der Inhalt der Dateien dient lediglich dazu, dass Sie nicht alle benötigten Funktion selbst programmieren müssen, sondern wir geben Ihnen die wichtigsten Funktionen an die Hand. Wie in der Vorlesung und Hörsaalübung zu Objektorientierung gelernt, befinden sich in den einzelnen .m Dateien bereits definierte Funktionen (Methoden). Diese Funktionen, müssen Sie also nicht mehr selbst erstellen, sondern Sie können Sie einfach in Ihrem Skript oder Command Window aufrufen und durch das Aufrufen dann auch anwenden.

Wenn Sie sich unsicher sind, welche Funktionen wir Ihnen bereits zur Verfügung stellen und welche Inputs und Outputs diese haben, öffnen Sie einfach die entsprechende .m Datei. Zu Beginn jeder der 4 .m Dateien ist eine Erklärung über alle Funktionen geben und wenn Sie im Code dann zu der entsprechenden Funktion scrollen, können Sie es dort die Funktionsweise im Detail nachlesen.

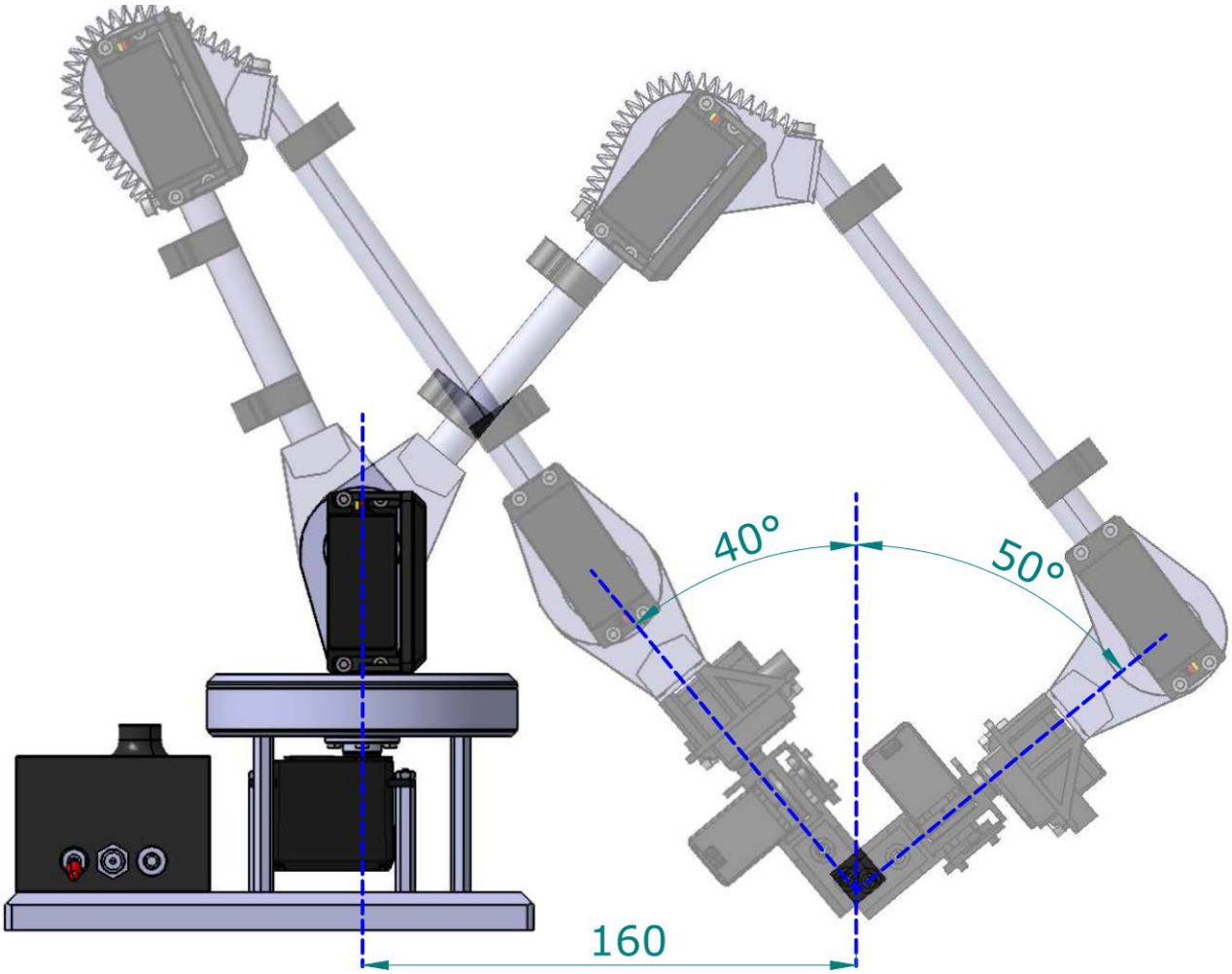
Zusätzlich zur Zip-Datei haben wir Ihnen auf moodle noch die Sensorinterpretation aus Abgabe 1 als .m-Datei hochgeladen. Diese müssen Sie nicht verwenden, aber, wenn Sie mit Ihrer Lösung nicht zufrieden sind, können Sie die von uns bereitgestellte Sensorinterpretation verwenden (siehe hierzu Kapitel 3.2 Auslesen des Sensors).

3.2 Erste Schritte

Wir möchten Ihnen hier einen Vorschlag machen, wie Sie sich auf einfachem Weg der Programmierung des Roboters nähern können. Sie müssen diesen Weg nicht gehen, sondern können auch Ihren eigenen wählen, denn gerade in der Programmierung gibt es viele Möglichkeiten, ein und dasselbe Ziel zu erreichen *(Für die Matlab Befehle der folgenden Schritte siehe Kapitel 3.2)*.

Unsere 10 Schritte zum Erfolg

1. Erstellen Sie ein *VirtualRobot* Objekt und machen Sie sich mit dem Koordinatensystem vertraut.
2. Lassen Sie den Roboter ein Gelenk bewegen; bewegen Sie danach andere Gelenke.
3. Bewegen Sie mehrere Gelenke gleichzeitig.
4. Probieren Sie verschiedene Gelenkpositionen aus. Testen Sie auch, was passiert, wenn Sie ein Gelenk außerhalb seiner Reichweite bewegen wollen.
5. Versuchen Sie, den Roboter so zu bewegen, dass die *hand* vertikal oder horizontal über dem Boden steht.
6. Schließen und öffnen Sie die Hand.
7. Schreiben Sie ein erstes Skript, in dem der Roboter erstellt wird und verschiedene Bewegungen ausführt.
8. Versuchen Sie, eine Ausgabe vom Farbsensor zu erhalten. Er wird normalerweise ‚k', also schwarz, ausgeben.
9. Erstellen Sie eine erste GUI, mit der ein *Roboter*-Objekt erstellt wird.
10. Sie haben nun den Einstieg in das Softwareprojekt geschafft. Als nächste Schritte sollten Sie sich mit der Geometrie des Roboters auseinandersetzen und die verschiedenen Abgaben bearbeiten.



3.3 vRob nutzen

Bevor Sie nun mit der Bearbeitung starten, vergewissern Sie sich nochmal, dass sich alle von uns zur Verfügung gestellten Matlab Dateien in Ihrem aktuellen Ordern in Matlab befinden, damit Matlab auf die Dateien zugreifen kann. Sonst funktionieren die folgenden Befehle nicht. Nachfolgend stellen wir Ihnen einige wichtige Funktionen zum Arbeiten mit dem Roboter vor. Natürlich gibt es noch weitere Funktionen, die Sie verwenden müssen, die jedoch nicht alle in diesem Tutorial einzeln erklärt werden müssen.

Roboter starten:

Um die Entwicklungsumgebung aufzurufen, greifen Sie auf die Klasse *VirtualRobot* zu. Auf objektorientierte Programmierung wird in Kapitel 5 OOP in Matlab näher eingegangen oder Sie schauen sich nochmal die Folien der 5. Hörsaalübung auf moodle an. Hier für die 3. Abgabe erstellen Sie ein Objekt mit beliebigem Namen (wir verwenden als Benennung gerne *vRob*), was von der Klasse *VirtualRobot* instanziiert wird:

```
vRob = VirtualRobot;
```

Mit diesem Aufruf erhalten Sie ein neues Matlab-Fenster namens „Figure 1: Virtueller Roboter“. Anhand dieser Darstellung können Sie den aktuellen Zustand des Roboters beobachten.

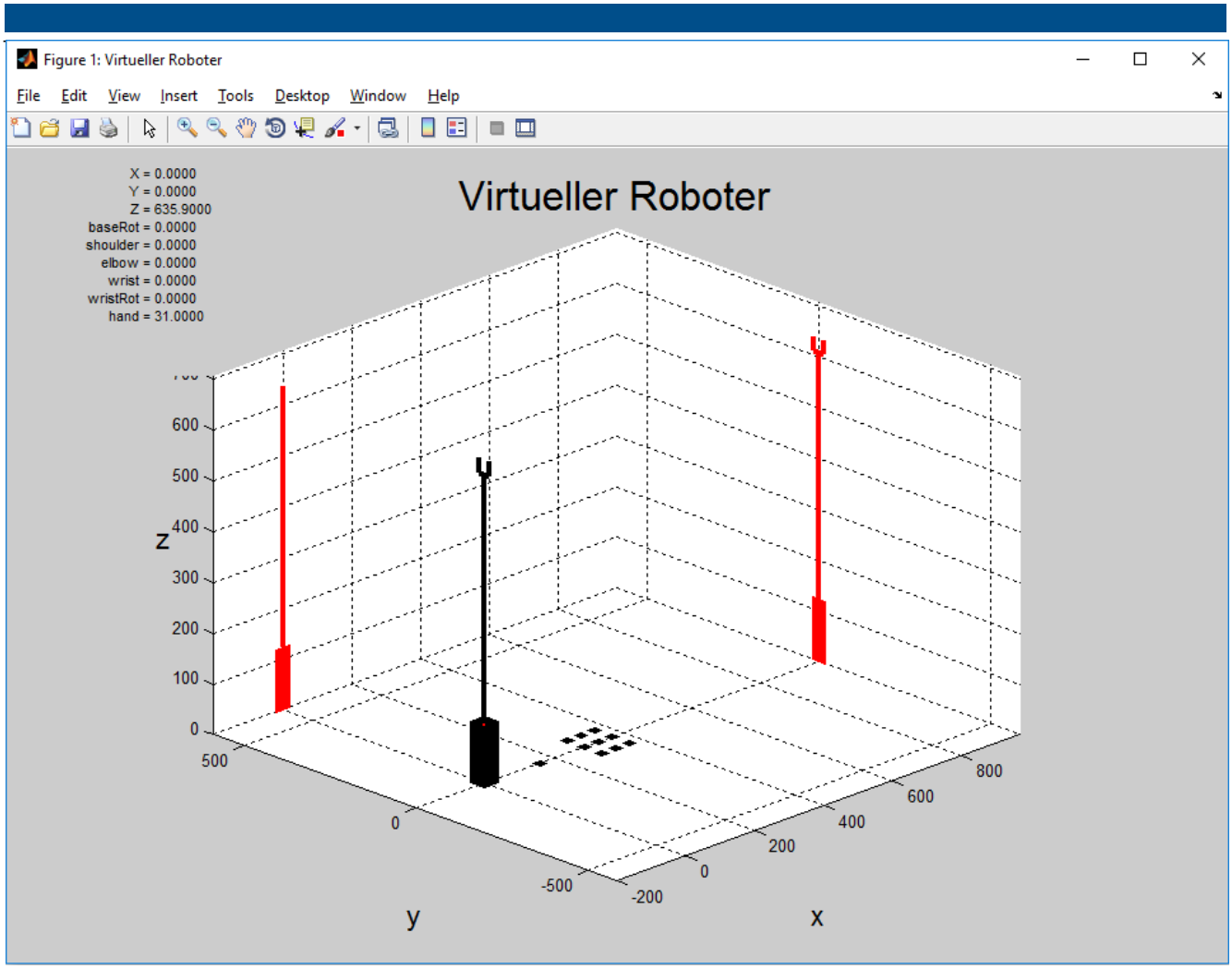


Abbildung 5-1: Die Entwicklungsumgebung vRob

In diesem Fenster erhalten Sie ein dreidimensionales Koordinatensystem mit dem Roboterarm als Ausgangspunkt. In Rot erhalten Sie Projektionen des Arms (quasi die Schatten, die der Roboterarm an die Wand wirft), um bei aus einer einzelnen Perspektive schwer zu unterscheidenden Positionen nicht den Überblick zu verlieren. Vor dem Roboter selbst befinden sich jeweils in schwarz markiert der einzelne Drehpunkt und das Matrixfeld; bisher ohne Würfel. Über den Aufruf des Roboters mit den entsprechenden Parametern erhalten Sie bereits zufällig gedrehte Würfel auf den dafür vorgesehenen Feldern.

```
vRob = VirtualRobot('random')
```

Schauen Sie sich hierfür auch die help-Funktion der **Methode** *VirtualRobot* an (wichtig: nicht der Klasse! Sollte Ihnen der Unterschied nicht bewusst sein, schauen Sie sich *OOP in Matlab* in Kapitel 5 an).

Generell erhalten Sie über die *help*-Funktion die Beschreibungen aller Methoden, die Sie im Klassendiagramm finden (siehe Kapitel 4.1). Beachten Sie aber, dass Sie **vor** Aufruf der *help*-Funktion die Klasse als Objekt instanziiert haben müssen (Hintergrund dazu finden Sie auch wieder in Kapitel 5).

Gelenke bewegen:

Um Ihnen den Einstieg zu erleichtern, erhalten Sie hier den Output des Hilfeaufrufs zu *VirtualRobot.moveAngles*:

```
moveAngles lets the robot move his joints to specified angles.
  Input:
  channels is a vector containing the joints to move, where
    1 is base rotation
    2 is the shoulder
    3 is the elbow
    4 is the wrist
    5 is the wrist rotation.
  destinations is a vector containing the angle destinations
    for the angles specified in channels.
  time is the movement time.
  call: rob.moveAngles(channels, destinations, time)
  example: rob.moveAngles([1,5,3],[45,70,30],3) will move the
  base rotation to 45 degrees, wrist to 70 degrees and elbow to
  30 degrees within 3 seconds.
```

D.h. der Befehl `moveAngles` ist wie folgt aufgebaut: `moveAngles(channels, destinations, time)`:

Die einzelnen Inputs bedeuten dabei das folgende:

- `channels`: bestimmt das Gelenk, welches bewegt werden soll
  - 1 base
  - 2 shoulder
  - 3 elbow
  - 4 wrist
  - 5 wrist rotation.
  - Wenn mehrere Gelenke gleichzeitig bewegt werden sollen, kann dies über einen Vektor eingegeben werden (z.B. `[1,5,3]`), wären dann die Gelenke base(1), wrist rotation(5) und elbow(3).
- `destinations`: Gibt die Gradzahl an, um die sich das Gelenk bewegen soll. Wenn sich das Gelenk in negative Koordinatenrichtung bewegen soll, muss ein minus vor die Gradzahl geschrieben werde
- `time`: Gibt die Zeit in Sekunden an, in der die Bewegung ausgeführt werden soll. Eine maximal schnelle Bewegung kann durch -1 eingestellt werden.

Die Informationen zu den einzelnen Gelenken finden Sie noch einmal im Tutorial zu Abgabe 1 veranschaulicht. Für den Einstieg können Sie also erst einmal den *elbow* auf seine 80°-Position bewegen:

```
vRob.moveAngles(3, 80, 3);
```

Für das Bewegen von mehreren Gelenken gleichzeitig, müssen Sie jeweils die Gelenknummern (channels) und Winkel als Vektor, sowie die Bewegungsdauer übergeben:

```
vRob.moveAngles([2 3 4], [30 110 -50], -1);
```

Hierbei kann der Parameter der Bewegungszeit auf -1 gesetzt werden, um eine schnellstmögliche Bewegung durchzuführen. Der Roboter orientiert sich hierbei an seinen intern abgespeicherten Limits.

Hand öffnen/schließen:

Für das Öffnen bzw. Schließen der Hand des Roboters gibt es die folgenden beiden Befehle:

```
vRob.openHand();
vRob.closeHand();
```

Wenn hierbei die Hand bereits geöffnet wird, wenn der Befehl zum Öffnen der Hand ausgeführt wird, gibt Matlab keine Fehlermeldung aus und die Hand wird auch nicht noch weiter geöffnet, sondern es passiert einfach gar nichts. Es gibt also nur die beiden Zustände Hand geöffnet und Hand geschlossen.

Warten auf das Ende der Bewegung:

Diese Funktion sorgt dafür, dass der Roboter seine Bewegung zu Ende ausführt, bevor ein neuer Befehl ausgeführt wird.

```
vRob.waitFor();
```

Auslesen des Sensors:

Wie Sie den Farbsensor auslesen können war bereits Bestandteil der 1. Abgabe. Der Vollständigkeit halber, nehmen wir den Befehl hier aber nochmal mit auf:

```
 colorVector = rob.getSensorColor();
```

Hierfür gelten allerdings bestimmte Voraussetzungen:

- Die *wrist* sollte sich horizontal auf einer Höhe von Z=65 befinden
- Der Sensor ist von den „Fingerspitzen“ 43,5mm radialer Abstand zum Roboter hin versetzt und sollte sich mittig über der Oberseite des Würfels befinden.

Falls Sie mit Ihrer Lösung für diese Aufgabe aus dem 1. PA nicht zufrieden sind, geben wir Ihnen eine m. Datei an die Hand (*Beispiel_Sensorinterpretation.m*), mit der Sie von hier an arbeiten können. Diese Datei ist nicht die einzige richtige Musterlösung, sondern es existieren viele Wege diese Funktion zu programmieren. Sie können aber gerne die Datei *Beispiel_Sensorinterpretation.m* für Ihren weiteren Code verwenden.

Jetzt haben Sie erst einmal alles Nötige, um in die Programmierung einzusteigen. Letztlich haben wir auch schon direkt den Anfang von *Erste Schritte in der Entwicklungsumgebung* abgedeckt; versuchen Sie sich also doch einfach mal an den nächsten Punkten der Liste und machen Sie sich an die Erstellung eines ersten Skripts für eine Bewegung des Roboters!

4 Objektbeschreibung

4.1 Klassendiagramm

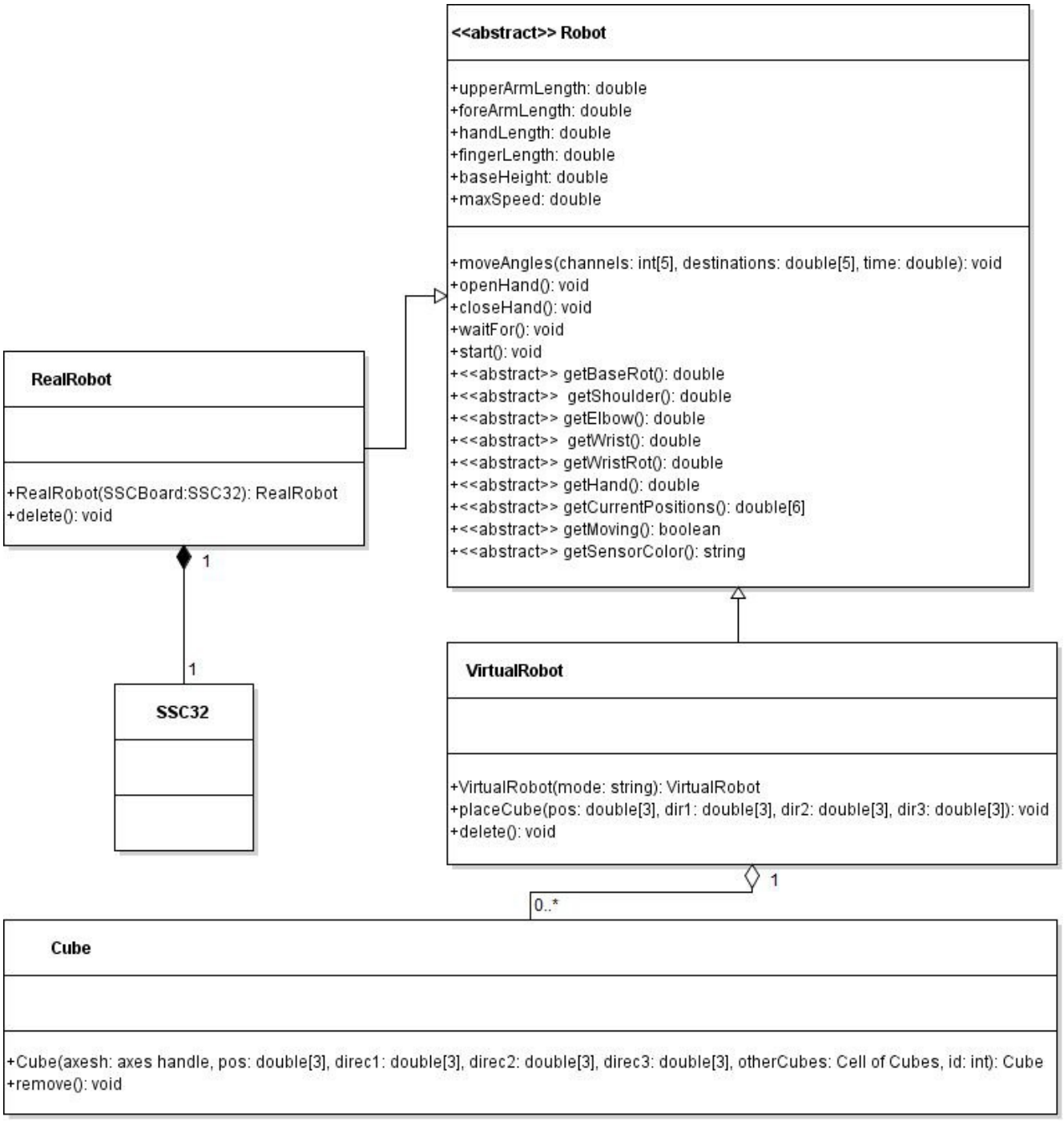


Abbildung 6-1: Klassendiagramm des Roboters

4.2 Informationen zu den einzelnen Methoden(/Funktionen)

Wie bereits in Kapitel 3.1 beschrieben, sind sämtliche im Objekt des Roboters verwendete Funktionen (in der objektorientierten Programmierung als „Methoden eines Objekts“ oder schlicht als „Methoden“ bezeichnet) für Sie zum Nachlesen dokumentiert. Alle Namen der Methoden sind Ihnen im obigen Diagramm gegeben, die Struktur eines Objekts lernen Sie in der Vorlesung kennen. Anstelle einer Erklärung der einzelnen Methoden hier im Tutorial haben Sie mit diesem Wissen die Möglichkeit, über die *help*-Funktion innerhalb von Matlab auf eben diese Zugriff zu erhalten.

Weiterhin finden Sie im Kapitel 5.1 *OOP in Matlab* tiefergehende Informationen zu Objekten und den Möglichkeiten, auf diese zuzugreifen.

5 Programmierhilfen

In diesem Softwareprojekt-Tutorial finden Sie weitere Programmierhilfen, die Ihnen eventuell im Verlauf des Softwareprojektes hilfreich erscheinen werden. Das Softwareprojekt in objektorientierter Programmierung abzuschließen ist optional, allerdings in der Industrie heutzutage weit verbreitet und in größeren Programmabläufen effizienter; wer also das Interesse hat, sich mit dem Thema zu beschäftigen, dem empfehlen wir es, auch wenn wir das objektorientierte Programmieren im Rahmen dieses Tutorials nicht tiefgründig behandeln. *(Es gibt weder Bonuspunkte, wenn Sie objektorientiert Programmieren noch Abzüge, wenn Sie nicht objektorientiert Programmieren)*.

Weiterhin gibt es natürlich allerlei weitere Snippets (kleine Code-Abschnitte) oder einzelne Befehle, die Ihnen in der Lösung der Aufgaben helfen. Lassen Sie sich aber nicht von anderen Teams beunruhigen, wenn diese alle Programmierhilfen verwendet haben. Das gibt keine Zusatzpunkte, sondern ist nur als Hilfestellung für Sie gedacht! Zusätzlich zu den hier gegebenen Informationen, können Sie natürlich auch jederzeit die MATLAB-Hilfe mit

```
help funktionsname
```

aufrufen oder Ihre_n Tutor_in befragen. Für den Aufruf der Hilfe zu Methoden gelten besondere Bedingungen, die Sie am Ende des folgenden Kapitels finden werden.

5.1 OOP in Matlab

Objektorientiertes Programmieren (kurz: OOP) ist eine Programmiertechnik, die Sie im GdD-Kurs bereits gelernt haben.

Für das Softwareprojekt ist es unabdingbar, dass Sie wissen, wie man in Matlab Methoden verwendet, die objektorientiert programmiert sind. Entsprechend beschränkt sich diese Programmierhilfe darauf, Ihnen einen Einblick in die Verwendung bereits programmierter Klassen und Methoden zu geben. Im Rahmen des Softwareprojektes benötigen Sie dieses Wissen, um beispielsweise auf Methoden des Roboters zugreifen zu können (s. Kapitel 0 und Kapitel 3.1).

Im Folgenden gehen wir beispielhaft auf die abstrakte Klasse Roboter des Softwareprojektes und die Verwendung ihrer Unterfunktion waitFor() ein.

Um auf abstrakte Klassen und ihre Funktionen zugreifen zu können, muss erst ein Objekt dieser Klasse instanziiert werden. In diesem Beispiel ist ein Erbe der Klasse Roboter VirtualRobot, das ist eine Klasse aus der Objekte abgeleitet werden können.

```
vRob = VirtualRobot; %instanziiert ein Objekt der Klasse VirtualRobot

vRob.moveAngles([2 3 4], [30 110 -50], 1);

vRob.waitFor() %vRob wartet bis die Bewegung von Befehl moveAngles() vollzogen
               ist
```

Denken Sie daran, dass sie auf die Dokumentation der von uns erstellten Klassen und Funktionen über den help Befehl in Matlab zugreifen können. Beachten Sie beim Aufruf der Hilfe-Funktion zu Methoden, dass Sie ebenso in diesem Fall die Klasse vorher instanziiert haben müssen, da die Methode für Matlab sonst nicht existiert. Auch müssen Sie unter Umständen Ihre Eingabe präzisieren; existieren beispielsweise eine Methode und ein Objekt mit dem gleichen Namen, wird Matlab Ihnen nur die Beschreibung des Objekts zurückgeben. Um spezifisch auf die Methode zuzugreifen, müssen Sie den Aufruf leicht anpassen:

```
help objektname.methodenname
```

Sollte eine solche Überschneidung nicht existieren, gibt Matlab auch bei einem unspezifischen Aufruf ohne Objektnamen direkt die Beschreibung der Methode aus.

5.2 GUI-Interaktion

Sie haben zwei Möglichkeiten eine GUI zu erstellen:

1. Durch Programmieren der einzelnen Elemente
2. Durch Nutzung des App Designers von Matlab

In diesem Kapitel gehen wir nur auf die erste Variante der GUI Umsetzung ein, für weitere Informationen für die GUI Umsetzung mittels App Designer verweisen wir auf den Theorie- und Praxisteil der Übung vom 16.12.2021 (bereits jetzt schon für Sie zum Nachlesen freigeschaltet).

Sie haben die Möglichkeit, über Eingaben auf der grafischen Benutzeroberfläche mit dem Benutzer zu interagieren. Maus- und/oder Tastatureingaben können dabei mit Funktionen, so genannten Callbacks, verknüpft werden. Je nach Typ des grafischen Elements stehen Ihnen verschiedene Callback-Einstiegspunkte zur Verfügung.

figure

Zunächst wird ein Fenster erstellt:

```
figureHandle = figure('Name','GUI','NumberTitle','off','MenuBar','none');
```

Erläuterungen zu den Attributen (Name, NumberTitle, MenuBar u.v.m.) finden Sie in der Matlab-Dokumentation unter 'Figure Properties'.

Um das Fenster in der Bildschirmmitte zu platzieren, können Sie beispielsweise folgende Zeilen verwenden:

```
screensize = get(0,'ScreenSize');
figureHandle.Position = [screensize(3)/2-400, screensize(4)/2-300, 800, 600];
```

uicontrol

Viele Elemente der Benutzeroberfläche in Matlab sind sogenannte uicontrol-Objekte: z.B. statische Textfelder, Eingabefelder, Schaltflächen und Pop-Up-Menüs. Eine Schaltfläche kann folgendermaßen erstellt werden:

```
buttonHandle = uicontrol(figureHandle,'Style','pushbutton','Position',
[80,80,100,40],'String','Drück mich!');
```

Schauen Sie sich hierzu auch das Matlab help-Kapitel *uicontrol* Properties an.

Damit durch Klicken auf den Knopf Ereignisse ausgeführt werden können, muss eine sogenannte Callback-Funktion definiert werden, die bei einer Interaktion aufgerufen wird:

```
buttonHandle.Callback = @clickEvent;
```

Durch das @-Zeichen werden der Callback-Funktion (hier clickEvent) automatisch zwei Argumente übergeben. Erstens die Quelle, also ein Handle zu dem Objekt das die Callback-Funktion aufgerufen hat (hier buttonHandle) und zweitens ein Event-Objekt, welches den Namen des Auslöse-Events enthält. Im Beispiel soll der Knopf die Hintergrundfarbe des Fensters ändern.

```
function clickEvent(src, eventdata)
    src.Parent.Color = 1 - src.Parent.Color;
    disp(eventdata.EventName);
end
```

Wird ein Übergabeargument nicht benötigt, kann es mit ~ unterdrückt werden:

```
function clickEvent(src,~)
```

Unter Umständen ist es zielführend, neben dem Source- und dem Event-Objekt noch andere Variablen oder Handles an eine Callback-Funktion zu übergeben. Dies kann folgendermaßen geschehen:

```
buttonHandle.Callback = {@clickEvent, variable1, variable2};
```

Der Kopf der Callback-Funktion enthält dann im Beispiel vier Übergabeargumente:

```
function clickEvent(src, ~, variable1, variable2)
```

properties

Jedes *uicontrol*-Objekt besitzt eine Reihe von Eigenschaften (sog. Attribute bzw. in Matlab properties; siehe *uicontrol* Properties in der Matlab-Hilfe). Die wichtigste Property ist 'Style', denn sie entscheidet über Aussehen und Funktion des Bedienelementes. Folgende Styles stehen zur Auswahl:

	Style	**Beschreibung**
Check Box	`checkbox`	Auswählbarer Status einer Option
Editable Text	`edit`	Durch den Benutzer editierbarer Text

Frame	`frame`	Vereinigt zusammengehörende Elemente in einer Box/einem Rahmen
Pop-up Menu	`popupmenu`	Ausklappbare Liste mit verschiedenen Optionen
List Box	`listbox`	Auswahlmenü mit verschiedenen Optionen, evtl. mit Scrollbar
Push Button	`pushbutton`	Klicken ruft ein Event hervor
Radio Button	`radiobutton`	Auswahl ändert den Status einer Option. I.A. sind Radio-Buttons in einer uibuttongroup* gruppiert.
Toggle Button	`togglebutton`	Es existiert nur der Zustand *ein* und *aus*
Slider	`slider`	Repräsentiert eine größere Anzahl von Optionen
Static Text	`text`	Zeigt unveränderbaren Text in einer Box

* Nur die in einer uibuttongroup gruppierten Radio-Buttons schließen sich gegenseitig aus. Zuerst muss die Buttongroup erstellt werden und dann deren Handle als Parent der Radio-Buttons angegeben werden.

Es gibt zahlreiche weitere Properties. An dieser Stelle sollen nur die wichtigsten erwähnt werden:

Property	**Beschreibung**	**Datentyp**
`Tag`	Der Tag ist der individuelle Name eines grafischen Objektes. Er ist ein nützliches Hilfsmittel, um ein gesuchtes Objekt zu finden, dessen Name bekannt ist.	String
`Parent`	Parent ist das Handle zum Eltern-Objekt. Wird nichts anderes angegeben, ist das aktuell aktive Fenster das Parent-Objekt.	figure/uipanel/uibuttongroup
`Position`	Position gibt den Abstand des GUI-Elements relativ zum Parent-Fenster an. Folgende Kodierung wird dabei benutzt: `[links unten Breite Höhe]`, wobei der innere Rand des Parents gemeint ist.	1x4 - Vektor
`Enable`	Der Wert von Enable entscheidet darüber, ob ein GUI - Element für den Benutzer benutzbar ist oder nicht.	String: 'on' oder 'off'
`Callback`	Callbacks sind interaktive Elemente, die auf Benutzereingaben reagieren. Meist wird entweder eine Funktion angesprochen oder eine Funktion definiert. Auf Callbacks wird in den nächsten Abschnitten genauer eingegangen.	Function-Handle: @funktionsname

`String`	String ist der dargestellte Name des GUI - Elementes und nicht zwangsläufig gleich dem `Tag`.	String

Properties können direkt beim Erstellen eines Objektes übergeben werden:

```
uiHandle = uicontrol('PropertyName1',PropertyValue1,'PropertyName2', Proper-
tyValue2);
```

Über das Handle können sie auch später geändert werden:

```
uiHandle.PropertyName = PropertyValue;
```

Oder ausgelesen werden:

```
PropertyValue = uiHandle.PropertyName;
```

5.3 Try & catch

Während des Programmierprozesses werden Sie Fehlermeldungen erhalten. Diese führen im Allgemeinen zu Programmabbrüchen. Um solche Abbrüche zumindest bei bekannten Fehlern zu vermeiden, existieren sogenannte try-catch-Blöcke. Produziert eine Anweisung im try-Block einen Fehler, so bricht das Programm nicht ab, sondern springt in den catch-Block und führt die darin enthaltenen Anweisungen aus. Wenn kein catch-Block vorhanden ist, springt das Programm zum Ende des Konstruktes. Sowohl im try- als auch im catch-Block können verschachtelte try-catch-Blöcke enthalten sein.

Syntax:

```
try
% versuche zu tun, was hier steht
catch
% wenn ein Fehler auftritt, führe aus, was hier steht
end
```

Beispiel

```
person = struct('Name', 'Max Mustermann', 'Matrikelnummer', 123456);
try
wohnort = person.Wohnort;
catch
disp('Das gesuchte Attribut scheint nicht zu existieren!');
end
```

6 FAQ

1. Muss der Roboter die Würfel tauschen können?
   → Nein, der Roboter muss nicht tauschen können. Es können auch alle Würfel gedreht werden.

2. An welchem Ort werden die Drehungen über die Kante und die Hochachse durchgeführt?
   → Beide Drehungen sollen nur am Drehplatz ([160, 0, 0]) durchgeführt werden. Beachten Sie, dass Sie den Würfel vor den Drehungen anheben, um eine Kollision mit dem Boden zu vermeiden.

3. Muss die Möglichkeit eines fehlenden Würfels oder die Erkennung von 'black' im Programm berücksichtigt werden?
   → Nein, bei korrekter Interpretation des Farbwertes können Sie davon ausgehen, dass immer eine Farbe erkannt wird. Sie müssen also Ihr Programm nicht darauf auslegen, dass 'black' erkannt wird.

4. Müssen Kollisionen mit anderen Würfeln beachtet werden?
   → Ja, der Arm und die Würfel, und die Würfel untereinander sollten nicht miteinander kollidieren.

5. Wo kann ich die dritte Abgabe abgeben?
   → Die Abgabe findet über moodle statt, dort müssen alle Dateien der Abgabe hochgeladen werden. Abgaben über E-Mail werden nur nach Absprache und nur bei technischen Problemen akzeptiert!

Allgemein hilfreiche Aussagen:

6. Wenn Sie vor Ende der Abgabe an einer Stelle in Ihrem Programm nicht weiterkommen, dann kommentieren Sie Ihr geplantes Vorgehen in Pseudocode. Das wird dann auch in der Bewertung berücksichtigt. Es wird zwar Abzüge geben, aber dann können wir wenigstens Teilpunkte vergeben

7. Sie müssen das Softwareprojekt NICHT objektorientiert lösen.

8. Wir empfehlen Ihnen Ihr Programm in mehreren Funktionen und Dateien zu schreiben. Dies erhöht die Lesbarkeit und Übersichtlichkeit des Programmes.

9. Bei Schwierigkeiten mit moodle prüfen Sie bitte zuerst, ob sich das Problem beheben lässt, indem Sie moodle über den Browser aufrufen.