

Kurzanleitung `rstk`

`rstk` (Robot Systems Tool Kit) ist eine kleine Funktionsbibliothek zur dreidimensionalen Visualisierung von Objekten mit MATLAB. Die Bibliothek baut im auf der Matlab-Funktion `patch` auf, mit der sich beliebige dreidimensionale Geometrien in MATLAB darstellen lassen. Die Darstellung greift dabei über OpenGL auf die Grafikhardware zu, sodass auch komplexere Geometrien ohne signifikanten Performance-Verlust handhabbar sein sollten. Dieses Dokument soll die wesentlichen Konzepte der Bibliothek erläutern.

1 Installation

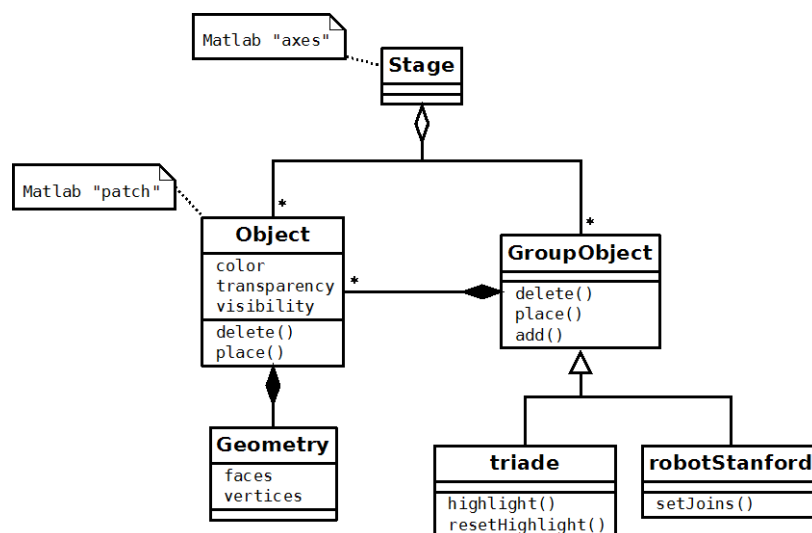
Entpacken Sie den Inhalt des heruntergeladenen Archivs in ein Verzeichnis Ihrer Wahl! Fügen Sie dieses Verzeichnis zum MATLAB-Suchpfad hinzu (File → Set Path... → Add Folder..., Speichern nicht vergessen)!

Geben Sie im MATLAB-Command-Window `rstk_demo` ein, um die Installation zu testen!

Kompatibilität: Die Bibliothek wurde mit Matlab 7.5.0 (R2007.b) entwickelt und getestet.

2 Aufbau der Bibliothek

Das folgende Diagramm beschreibt in UML-ähnlicher Notation die wesentlichen Konzepte der Bibliothek.



Als Bühne (Stage) für 3D-Objekte dient ein MATLAB-Koordinatensystem (`axes`). Mit der Funktion `createStage` kann ein entsprechend parametrisiertes Koordinatensystem erzeugt werden. 3D-Objekte werden mit der Funktion `createObject` erzeugt. Dabei ist ein Geometrie-Objekt anzugeben, welches die

Flächen (Polygone) bestimmt, aus denen ein Objekt besteht. Die Bibliothek beinhaltet eine Reihe von Funktionen (Präfix `geo...`) zum Erzeugen grundlegender Geometrien. Diese sind in Abschnitt 4 näher beschrieben. Bei Bedarf können 3D-Objekte mit der Funktion `createObjectGroup` zu Gruppen zusammengefasst werden. Die nachfolgenden Abschnitte beschreiben die einzelnen Elemente im Detail.

Hinweis zur Objektorientierten Programmierung (OOP): Die Bibliothek ist zwar objektorientiert konzipiert, verwendet aber aus Gründen der Einfachheit KEINE objektorientierten Sprach-Features von MATLAB. „Objekte“ werden durch MATLAB-Strukturen (`struct`) mit entsprechenden Datenfeldern nachgebildet. Objektmethoden werden mit Hilfe von Function-Handles (`@Funktionsname`) ebenfalls in den Strukturen hinterlegt, sodass ein Aufruf über die gewohnten Syntax `Objekt.Methode(...)` möglich ist.

3 Transformationen

Zur Beschreibung affiner Abbildungen (Translationen, Rotationen, Skalierung, etc.) werden innerhalb der Bibliothek 4x4-Matrizen verwendet (homogene Transformationen).

Entsprechende Matrizen können mit Hilfe der folgenden Funktionen erzeugt werden:

- `T_unity()` → Einheitstransformation (= `eye(4)`)
- `T_shift(...)` → Verschiebung entlang der Koordinatenachsen
- `T_rot(...)` → Rotation (verschiedene Parametrierungsmöglichkeiten)
- `T_scale(...)` → Skalierung (Größenänderung)

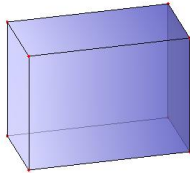
Über die möglichen Funktionsparameter geben die Hilfetexte (`help T_...`) Auskunft. Die Hintereinanderausführung (Kombination) von Transformationen kann wie gewohnt durch Matrizenmultiplikationen erreicht werden.

Beispiele:

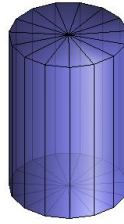
```
T1 = T_shift([1 0 2]); % Verschiebung: 1 entlang X, 2 entlang Z
T2 = T_rot('X', pi / 2); % Drehung 90° um X-Achse
T3 = T_rot('xyz', pi/2, pi/4, pi/4); % Drehung um konsekutive Achsen
T4 = T_rot('XYZ', pi/2, pi/4, pi/4); % Drehung um RAUMFESTE Achsen
T5 = T_scale([0 0 -1]); % Spiegelung an XY-Ebene
```

4 Geometrien

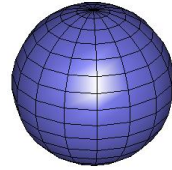
Jedes 3D-Objekt benötigt eine Geometrie zur Definition seiner Polygonflächen. Eine Geometrie ist eine MATLAB-Struktur mit den zwei Einträgen `v` (Eckpunkte, „vertices“) und `f` (Flächen, „faces“). Bedeutung und Aufbau dieser Einträge entsprechen den Eigenschaften 'Vertices' und 'Faces' eines MATLAB-patch-Objektes. Für weitere Details sei daher auf die MATLAB-Dokumentation der `patch`-Funktion verwiesen! Die Bibliothek bietet eine Reihe von Hilfsfunktionen (Präfix `geo...`) zur Erzeugung grundlegender Geometrien. Die wichtigsten Funktionen sind in der nachfolgenden Übersicht zusammengestellt.



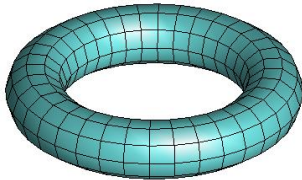
geoBox



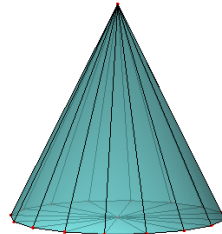
geoCylinder



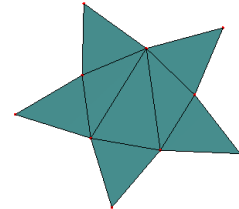
geoEllipsoid



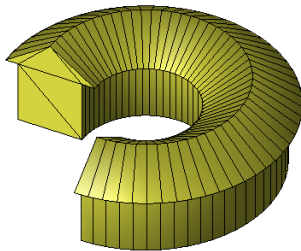
geoTorus



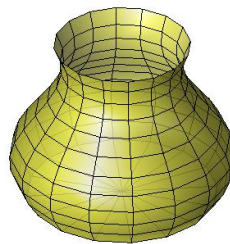
geoCone



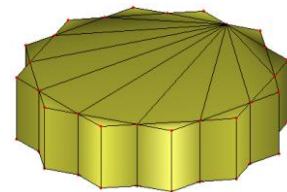
geoShape



geoRotateContour

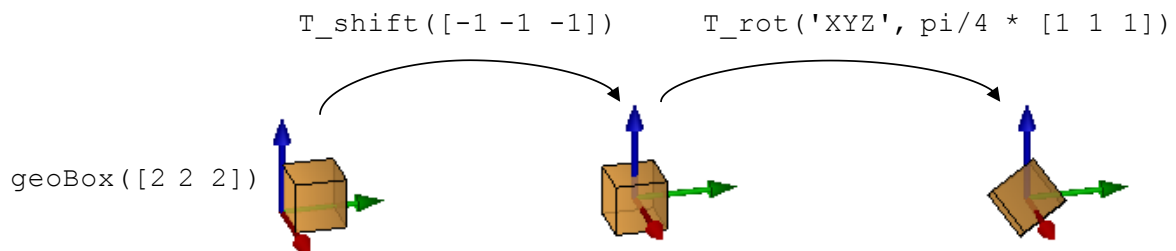


geoRotateCurve

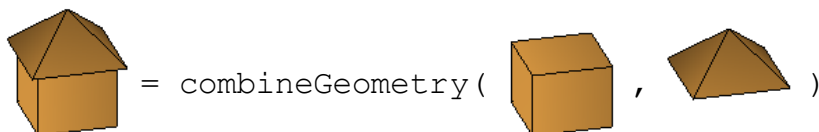


geoExtrude

Mit Hilfe der Funktion `transform` können Geometrien einer Koordinatentransformation unterzogen und somit beliebig im Raum platziert werden:




Komplexere Geometrien lassen sich mit der Funktion `combineGeometry` aus mehreren Elementargeometrien zusammensetzen:



5 Objekte

Ein Objekt wird mit der Funktion `createObject` erstellt. Deren erster und einziger nichtoptionaler Parameter ist eine Geometrie-Struktur (siehe Abschnitt 4). Alle weiteren Parameter werden an das intern erzeugte MATLAB-patch-Objekt weitergereicht. Auf diese Weise kann z. B. bereits beim Erstellen die Farbe eines Objektes festgelegt werden. Beispiel:

```
box = createObject(geoBox([1 1 1]), 'FaceColor', [0 1 0]); % green cube
```



```
intern: patch('Vertices', geo.f, 'Faces', geo.v, ..., 'FaceColor', [0 1 0]);
```

Alle Objekte besitzen folgende Methoden:

```
object.setFaceColor,  
object.setVertexColor,  
object.setWireframeColor
```

- ➔ Setzt die Darstellungsfarbe für Flächen, Eckpunkten bzw. Kanten eines Objektes

```
object.setShowFaces,  
object.setShowVertices,  
object.setShowWireframe
```

- ➔ Schaltet die Darstellung von Flächen, Eckpunkten bzw. Kanten ein- oder aus.

```
object.setTransparency
```

- ➔ Setzt den Transparenzfaktor der Darstellung. Gültiger Wertebereich: 0 (vollständig transparent) ... 1 (nicht transparent)

```
object.show(),  
object.hide()
```

- ➔ Zeigt ein Objekt an bzw. blendet es aus (verändert `Visibility`-Eigenschaft des patch-Objektes)

```
object.delete()
```

- ➔ Löscht das Objekt und alle damit verbundenen Ressourcen

```
object.setGeometry()
```

- ➔ Ersetzt die aktuellen Geometriedaten des Objektes

```
object.addGeometry()
```

- ➔ Erweitert die Geometrie eines Objektes mit Hilfe von `combineGeometry`

```
object.place()
```

- ➔ Setzt die Transformationsmatrix für die Platzierung des Objektes im Raum. Die Transformationsmatrix verändert nicht die Objektgeometrie. Sie wird vor der Übergabe der Geometriedaten an das MATLAB-patch-Objekt angewendet.

```
object.transform()
```

- ➔ Wendet eine Transformationsmatrix auf die Objektgeometrie selbst an. Die Transformationsmatrix des Objektes (vgl. `object.place()`) wird hierdurch nicht beeinflusst!

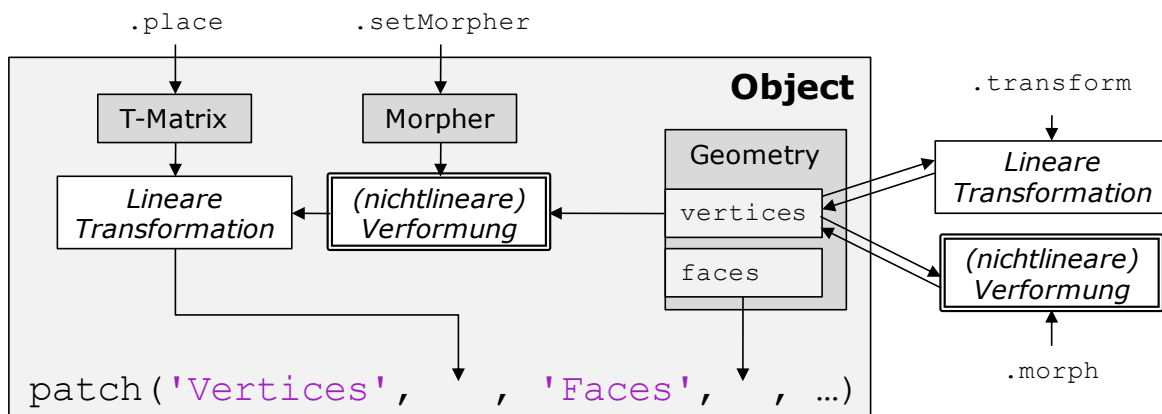
```
object.setMorpher()
```

- ➔ Stellt eine Objektverformung (siehe Abschnitt 7) ein. Analog zur Transformationsmatrix (`object.place`) verändert die Verformung nicht die im Objekt gespeicherte Geometrie, sondern wird vor der Übergabe an die interne `patch`-Funktion auf die Geometriedaten angewendet.

```
object.morph()
```

- ➔ Verformt analog zu `object.transform` die im Objekt gespeicherte Geometrie. Eine zusätzlich eingestellte Objektverformung (`object.setMorpher`) bleibt hierdurch unverändert.

Die folgende Abbildung verdeutlicht die Wirkungen der vier unterschiedlichen Transformations- und Morphing-Funktionen:



6 Gruppenobjekte

Die Funktion `createObjectGroup` erstellt ein Gruppenobjekt, welches mehrere separate Objekte zusammenfasst. Dies ist z. B. immer dann sinnvoll, wenn mehrere Objekte einen zusammenhängenden (Starr-)Körper formen. Die zusammenzufassenden Objekte sind der Funktion `createObjectGroup` als Argumente zu übergeben. Die Gruppe lässt sich nachträglich mit der `add`-Methode um zusätzliche Objekte erweitern.

Darüberhinaus bieten Gruppenobjekte fast alle Methoden, die auch für Einzelobjekte zur Verfügung stehen. Sie rufen jeweils die gleichnamigen Methoden aller in der Gruppe enthaltenen Einzelobjekte auf.

```

group.setFaceColor()
group.setVertexColor()
group.setWireframeColor()
group.hide()
group.show()

```

```

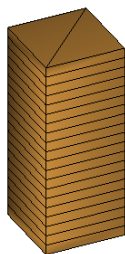
group.setShowFaces()
group.setShowVertices()
group.setShowWireframe()
group.setTransparency()
group.delete()
group.place()

```

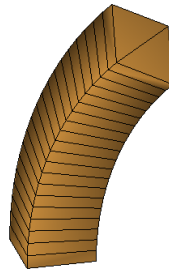
Einige spezielle Gruppenobjekte werden in Abschnitt 8 beschrieben.

7 Verformungen (morphing)

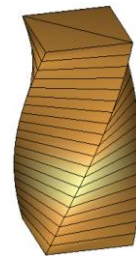
Zusätzlich zu affinen Abbildungen (Transformationsmatrizen) unterstützt die Bibliothek auch nichtlineare Verformungen (z. B. verdrehen, verbiegen...). Alle Objekte besitzen hierfür die Methoden `setMorpher`, mit der sich ein sog. „Morpher“ einstellen lässt. Morpher werden von Bibliotheksfunktionen mit dem Präfix `morph...` erstellt. Die Bibliothek unterstützt momentan die beiden in der folgenden Abbildung dargestellten Morpher. Eigene Morpher können bei Bedarf nach diesen Vorlagen implementiert werden.



Normal



morphBend

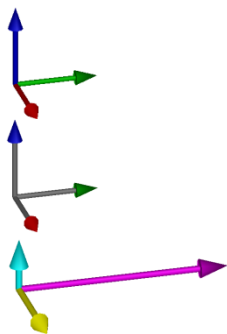


morphTwist

8 Spezielle Funktionen

8.1 Triade

Die Funktion `triade` erzeugt eine Objektgruppe aus drei orthogonal angeordneten Pfeilen, welche die Lage eines Koordinatensystems im Raum visualisiert. Größe und Färbung der Pfeile können über die Parameter der Funktion beliebig vorgegeben werden (siehe `help triade`). Standardmäßig wird die **X-Achse in rot**, die **Y-Achse in Grün** und die **Z-Achse in Blau** dargestellt. Die folgende Abbildung zeigt diese und einige weitere Parametrierungsbeispiele.

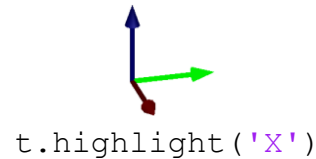
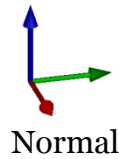


```
triade();
```

```
triade(T_unity(), 0.5 * [1 1 1]);
```

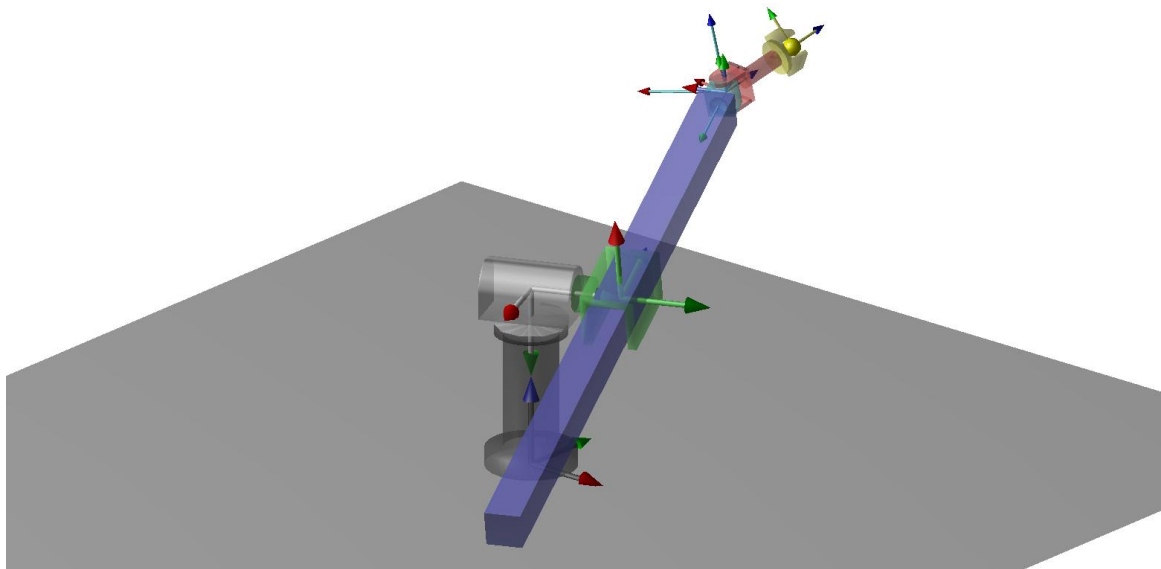
```
triade(T_unity(), ...
      repmat([1 1 0; 1 0 1; 0 1 1], 2, 1), ...
      [1 2 0.5]);
```

Die Objektmethode `highlight` ermöglicht es, Zur Verdeutlichung bestimmter Zusammenhänge eine oder mehrere Achsen der Triade farblich hervorzuheben. Mit `resetHighlight` wird der Normalzustand wieder hergestellt.



8.2 robotStanford

Diese Funktion erzeugt einen Stanford-Manipulator (6DOF-Manipulator). Über die Parameter lassen sich Gliedlängen und –Farben vorgeben.



```
robot = robotStanford(2, 1, 8, 1);
robot.colorLinks([.4 .4 .4], [1 1 1], [.5 1 .5], ...
                 [.5 .5 1], [.5 1 1], [1 .5 .5], [1 1 0.5]);
robot.setTransparency(0.5);
```

Folgende Objektmethoden stehen zur Verfügung:

`robot.colorLinks()`

- ➔ Setzt die Gliedfarben. Enthält die Parameterliste weniger Farben als Gelenke, werden die Farben wiederholt.

`robot.showOrigins()`

`robot.hideOrigins()`

- ➔ Zeigt die Gelenkkoordinatensysteme mittels Triaden an (siehe Bild oben) bzw. blendet sie aus.

`robot.setJoins()`

- ➔ Setzt alle 6 Gelenkwinkel. Die Winkel sind als sechs separate Argumente in *rad* anzugeben.

9 Animationen

Die Funktion `animate` bietet rudimentäre Unterstützung für Animationen. Die Funktionsweise soll anhand des folgenden Beispiels erklärt werden.

```
cube = createObject(geoBox([1 1 1]));  
animate(linspace(0, 2 * pi, 100), ...           % Zeitreihe  
        @(phi) cube.place(transRot('Z', phi)), ... % Animationsfunktion  
        0.1);                                   % Zeitintervall
```

Dieser Aufruf lässt einen Einheitswürfel einmal vollständig um die Z-Achse rotieren. Die Parameter haben dabei die folgende Bedeutung:

- Der erste Parameter gibt die Zeitreihe für die Animation vor. In diesem Fall sind es die Drehwinkel von 0 bis 2π um die Z-Achse in 100 Schritten.
- Der zweite Parameter ist das Handle einer Funktion, die in jedem Animationsschritt einmal ausgeführt wird, um die Animationsparameter einzustellen. Der übergebene Parameter ist das dem Animationsschritt zugeordnete Element des Zeitreihenvektors. Im Beispiel wird die Transformationsmatrix des Würfel-Objektes entsprechend dem für den jeweiligen Zeitschritt gewünschten Winkel gesetzt.
- Der dritte Parameter gibt die Schrittweite in Sekunden vor. Wird dieser Parameter ausgelassen, wird als Standardwert 0.1 s angenommen.