SG-Library: Entwicklung einer konstruktiven MATLAB-Toolbox zur räumlichen Modellierung von Körpern, Gelenken und Getrieben

*Tim C. Lueth**

*Technische Universität München, Lehrstuhl Mikrotechnik und Medizingerätetechnik (MIMED), Boltzmannstr. 15, Geb. 1, tim.lueth@tum.de

Abstract (deutsch und englisch)

Der Beitrag beschreibt eine neuartige MATLAB-Toolbox "SG-Library" mit der sich innerhalb von MATLAB die Oberflächenmodelle von geometrischen Körpern über Kommandos oder im Rahmen von automatischen Konstruktionsprogrammen erstellen und für die additive Fertigung (3D-Druck) in STL-Files speichern lassen. Die Toolbox wird an der TU München kontinuierlich weiterentwickelt und soll zum vollautomatischen Entwurf von Mechanismen – analog zu den Hardwarebeschreibungssprachen im Rechnerentwurf – ausgebaut werden. Aus der Beschreibung einer Effektorbewegung sowie möglichen Gestellpunkten soll letztendlich automatisch ein Mechanismus berechnet und vollständig gedruckt werden können. Der vorliegende Artikel beschränkt sich auf die Anwendung konstruktiver Funktionen der Toolbox.

This paper describes a novel MATLAB toolbox "SG-Library" for creating surface models (STL) of geometrical bodies or mechanisms. This is possible by using command lines or by developing automatic design functions for machine elements. The surface models can be written in STL-Files and printed in 3D (additive manufacturing). The Toolbox is continually developed at the TU München. It will be expanded – comparable to the hardware description languages in Computer Engineering - for the fully automatic design of mechanisms. By describing a movement task and possible frame points, a mechanism can be automatically calculated and 3D-printed completely. Nevertheless, this article is limited to the constructive functions of the Toolbox.

DOI: 10.14459/2015md1276136

1 Motivation

Für die Synthese und Analyse von allgemeinen Mechanismen bietet sich das Mathematikprogramm MATLAB der Firma MathWorks an. Mit diesem lassen sich mathematische Problemstellungen sehr effizient numerisch aber auch symbolisch lösen. Es erlaubt darüber hinaus die dreidimensionale Darstellung von Oberflächenmodellen oder Vollkörpern am Bildschirm.

In einem früheren Artikel [1] haben wir beschrieben wie man mit wenigen MATLAB-Funktionen die Glieder und Verbindungselemente für planare Mechanismen einfach modellieren und für die Fertigung mit einem 3D-Drucker im STL-Format speichern kann. Zum Getriebetechnikkolloquium 2013 in Ilmenau hatten wir auch eine größere Zahl von gedruckten Viergelenken mitgebracht und verteilt.

In der Folgezeit wurde deutlich, dass der CAD-Entwurf von Volumenkörpern mit Programmen wie CATIA, ProEngineer oder SolidWorks in erster Linie Modellierungsfunktionen bereitstellt, die zerspanende Fertigungsverfahren nachbilden. Dabei werden von einem größeren Volumen wie einem "Werkstück" durch boolesche Operationen andere Volumen wie beispielsweise "Bohrungen" durch boolesche Volumensubtraktionen "geschnitten" oder "abgezogen". Additive Fertigungsverfahren erlauben jedoch die Fertigung allgemeinerer Geometrien. Aus diesem Grund besitzt die aus [1] entstandene MATLAB-Toolbox andere, deutlich generellere Modellierungsfunktionen.

Es zeigte sich jedoch auch, dass kaum ein klassisch geschulter CAD-Nutzer bereit war, sein bisheriges CAD-Modellierungsvorgehen zu verändern und so wurde es notwendig, eine umfangreichere MATLAB-Toolbox **VLFL-Library,** auch **SG-Library** genannt, zu entwickeln, mit der von Konstruierenden bei Bedarf auch analog zu klassischen CAD-Programmen modelliert werden kann.

Dennoch bleibt die Hauptaufgabe unserer Toolbox die Synthese und Analyse von Kinematiken und Mechanismen. Wichtig ist auch, dass alle Elemente als Vollkörper in MATLAB direkt modellierbar sind und kein Import von Geometrien über STL oder IGES/STEP notwendig ist.

2 Struktur der SG/VLFL-Library Toolbox

Der Beitrag umfasst die Vorstellung der folgenden Bereiche:

- Datenstrukturen,
- Generierung von Punktlisten (2D) und Vertexlisten (3D),
- Export und Import von STL, Texte und räumliche Transformationen,
- Geschlossene Polygone und boolesche Operationen in 2D,
- Extrudieren von Polygonen nach boolesche Operationen in 3D,
- Räumliches Anordnen, Vereinigen und Prüfen mehrerer Körper,
- Relative Anordnung und Ausrichtung von Körpern,
- Rotation von Polygonen nach boolesche Operationen in 3D,
- Slicen, Schneiden und Zerlegen von Körpern,
- Boolesche Operatoren von Oberflächenmodellen,
- Anordnen und Verpacken von Körpern für die Fertigung,
- Ergänzung von Koordinatensystemen,
- Realisierung von Kinematikmodellen und
- Kollisionsanalyse und Flächenanalyse.

3 Datenstrukturen der SG/VLFL-Library Toolbox

Damit der MATLAB-Nutzer volle Kontrolle über Geometrien und Kinematiken behält, ist es notwendig, die dazugehörigen Modelle möglichst einfach zu halten und zumindest kurz zu erläutern.

Punkte (Points) in 2D und in 3D (Vertices):

- PL = Point-List [n x 2] als xy-Koordinaten,
- CPL = Closed-Polygon-List [n x 2] als xy-Koordinaten,
- VL = Vertex-List [n x 3] als xyz-Koordinaten.

Beispiele:

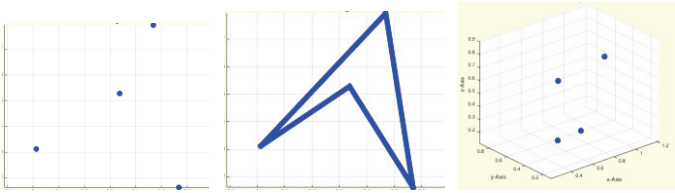

Abb. 1: a) PL=rand(4,2); PLplot(PL, 'b*'),
b) CPL=rand(4,2); CPLplot(PL, 'b*-',10),
c) VL=rand(4,3); VLplot(VL, 'b*',10)

Begrenzungen (Edges/Kanten) in 2D und in 3D (Facets/Dreiecke)

- EL = Edge-List [n x 2] als Indizes auf Punkt- oder Vertexlisten,
- FL = Facet-List [n x 3] als Indizes auf Punkt- oder Vertexlisten,
- TL = Tetrahedron-List [n x 4] als Indizes auf Vertexlisten.

Beispiele:

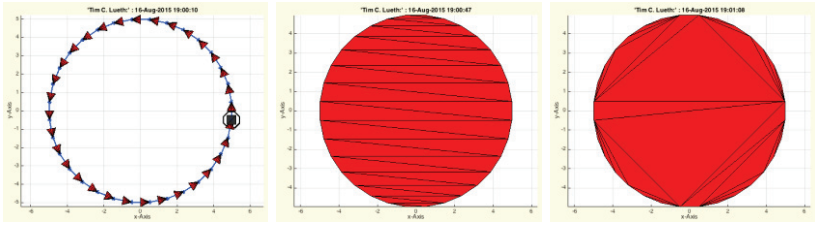

Abb. 2: a) PLELofCPL(PLcircle(5)),
b) PLFLofCPLpoly(PLcircle(5)),
c) PLFLofCPLdelaunay(PLcircle(5))

4 Punkte (2D), Vertices (3D), Flächen und Körper

Grundlegendes: Grundsätzlich werden mit *Punktlisten PL* immer xy-Eckpunktkoordinaten (2D) und mit *Vertexlisten VL* immer xyz-Eckpunktkoordinaten (3D) beschrieben. *Kantenlisten EL* besten aus Paaren der Indizes von Eckpunkten und beziehen sich daher immer auf PL oder VL. *Flächenlisten FL* bestehen aus Tripeln der Indizes von Dreieckseckpunkten und beziehen sich daher immer auf PL oder VL. *Oberflächenmodelle SG* sind "MATLAB-Structs". Sie bestehen immer mindestens aus einer Vertexliste SG.VL und einer Flächenliste SG.FL.

Modellierungsmöglichkeiten mit Punktlisten PL in 2D:

Punktliste:	PL=[x1 y1; x2 y2; …], PL=[0 0; 10 0; 10 10];
Vertexliste aus Punktliste:	VL=**VLaddz**(PL) oder VL=**VLaddz**(PL, z)
Punkteingabe per Maus:	VL=**VLui** oder VL=**VLui**(Punktezahl)
Kreispunkte:	PL=**PLcircle**(Radius,[Eckenzahl n])
Kreissegmentpunkte:	PL=**PLcircseg** (Radius, [n, Start-, Endwinkel])
Sternpunkte:	PL=**PLstar**(Außenradius, Eckenzahl)
Quadratpunkte:	PL=**PLsquare**(x, y)
Spiralpunkte:	PL=**PLhelix**(Radius) oder VL=**PLhelix**(R, h)
Evolventenpunkte:	PL=**PLevolvente**(Radius, Winkel, Eckenzahl)
Zahnradpunkte:	PL=**PLgearDIN**(Modul, Zähnezahl)
Bezier aus 4 Punkten:	VL=**VLBezier4P**(p1,p2,p3,p4)
Winkel zu Punktnachbarn:	wL=**PLangle**(PL), wL=**VLnorm**(VL)
Punktabstände,-richtung:	dL=**PLnorm**(PL), dL=**VLnorm**(VL)
Drehung/Verschiebung:	PL=**PLtrans**(PL,[rot(phiz),[dx; dy]])
Konvexe Hülle:	PL=**PLconvexhull**(PL)
Runden von Ecken:	PL=**PLradialEdges**(PL, Radius)
Zeichnen von Punkten:	**PLplot**(PL,[Farbe, Dicke]), PLplot(PL, 'r*--',2)
Beschriften von Punkten:	**textVL**(PL,[Farbe, Dicke, Punktindizes])

Modellierungsmöglichkeiten mit geschlossenen Polygonlisten CPL in 2D:

Schließen von Punktlisten:	CPL=**CPLofPL**(PL)
Kopieren von Konturen:	CPL=**CPLcopypattern**(CPL,[nx ny],[dx,dy])
Entfernen von Punkten:	CPL=**CPLremstraight**(PL)
Beispielkonturen:	CPL=**CPLsample(**Beispielnummer**)**
Boolesche Operationen:	CPL=**CPLbool**(f, CPLA, CPLB), f='&', '|', '-', 'xor'
Umlaufabhängige Kontur:	CPL=**CPLrecontour**(CPL)
Zeichnen von Konturen:	**CPLplot**(PL,[Farbe, Dicke])

Flächenerzeugung in 2D:
Delaunay-Dreiecksflächen: [PL,FL]=**PLFLofCPLdelaunay**(CPL)
Slicing-Dreiecksflächen: [PL,FL]=**PLFLofCPLpoly**(CPL)

Zeichnen der Flächen: **VLFLplot**(PL,FL,[Farbe, Dicke])

Modellierungsmöglichkeiten mit Punktlisten VL in 3D:
Vertexlisten: VL=[x1 y1; x2 y2; …]
Vertexliste aus Punktliste: VL=**VLaddz**(PL) oder VL=**VLaddz**(PLZ)
Vertex Eingabe per Maus: VL=**VLui** oder VL=**VLui**(Punktezahl)
Alle geplotteten Vertices: VL=**VLofgca**
Vertices eines Fotos**:** VL=**VLofimage(**Image, Pixelsize**)**

Bezierkurve mit 4 Punkten: VL=**VLBezier4**(p1,p2,p3,p4,[Stützpunktzahl])
Bezierkurve aus Punktliste: VL=**VLBezierC**(VL, Stützpunktzahl)
Radienkurve aus 3 Vert.: VL=**VLRadius3**(p1,p2,p3,[Stützpunktzahl])
Radienkurve mit 4 Punkten: VL=**VLRadius4**(p1,p2,p3,p4,[Stützpunktzahl])
Radienkurve mit Vertexliste: VL=**VLRadiusC**(VL, Stützpunktzahl)

Winkel zwischen Vertices: wL=**VLangle**(VL)
Vertexabstände,-richtung: dL=**VLnorm**(VL)
Drehung/Verschiebung: VL=**VLtrans**(VL,[rot(wx,wy,wz),[dx;dy;dz]])
Spiegeln an X-Achsen: VL=**VLswapX**, **VLswapXY**, **VLswapXZ**,
Spiegeln an Y-Achsen: VL=**VLswapY**, **VLswapYX**, **VLswapYZ**,
Spiegeln an Z-Achsen: VL=**VLswapZ**, **VLswapZX**, **VLswapZY**,

Zeichnen von Vertices: **VLplot**(VL,[Farbe, Dicke]), **VLplot**(VL, 'r*',2)
Beschriften von Vertices: **textVL**(VL,[Farbe, Dicke, Punktindizes])

Modellierungsmöglichkeiten von Oberflächenmodellen in 3D:
Extrudieren von Punkten: [VL,FL]=**VLFLofPLz**(PL, Höhe)
Extrudieren einer Kontur: [VL,FL]=**VLFLofCPLz**(CPL, Höhe)

Extrudieren einer Kontur: SG=**SGofCPLz**(CPL, Höhe)
Rotieren einer Kontur: SG=**SGofCPLrot**(CPL, Winkel)
Begrenzt durch Konturen: SG=**SGofVLtrans**(VL, ,[rot(wx,wy,wz),[dx;dy;dz]])

Weitere Hinweise zu MATLAB:
Zusätzlich gibt es auch Tetraederlisten TL. Diese bestehen aus Quadrupeln der Indices von Tetraedereckpunkten und beziehen sich daher immer auf VL (3D). Matlab unterstützt dieses Tetraeder, räumliche Oberflächen und Ebene Flächen direkt in der Basisversion, jedoch wechselten von 2012a zum 2015a mehrfach die Feldnamen. Daher wird in diesem Text immer von PL/VL bzw. EL/FL/TL gesprochen.

Beispiele:

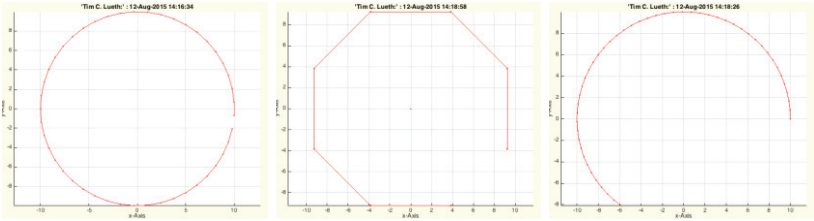

Abb. 3: PLcircle(10), b) PLcircle(10,8), c) PLcircseg(10,50,0,pi*1.3)

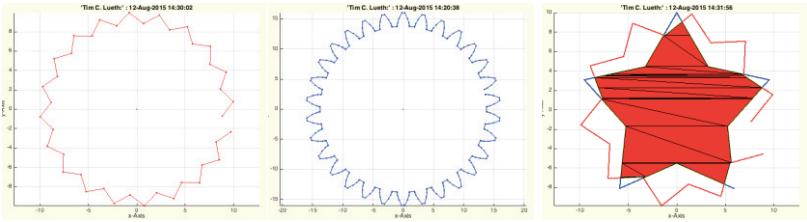

Abb. 4: a) PLstar(10,40), b) PLgearDIN(1,30),
c) CPLbool('&',PLstar(10,10),PLstar(10,20))

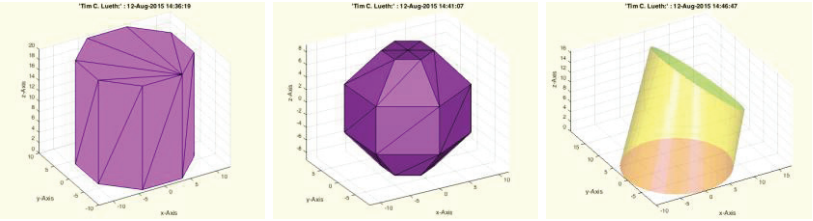

Abb. 5: a) SGofCPLz(PLcircle(10,8),20), b) SGofCPLrot(PLcircle(10,8),6),
c) SGofVLtrans(PLcircle(10),[rotdeg(20,0,0),[10; 10; 10]])

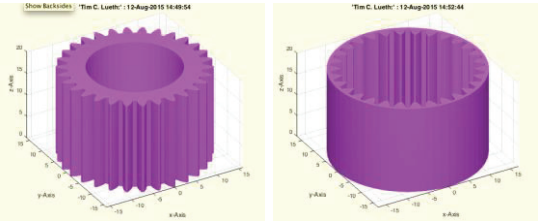

Abb. 6: a) SGofCPLz(CPLbool('-',PLgearDIN(1,30),PLcircle(10)),20),
b) SGofCPLz(CPLbool('-',PLcircle(17),PLgearDIN(-1,30)),20)

5 Export-Import von STL-Files und Textgenerierung

Grundlegendes:
Das Exportieren von STL-Dateien wurde bereits in [1] beschrieben. Die STL-Files können bei www.shapeways.com additiv gefertigt oder auch in CAD-Programme importiert werden.

Lesen und Schreiben von STL-Dateien:
Einlesen einer STL-Datei: [SG.VL,SG.FL]=**VLFLreadSTL**(Filename)
Auswahl einer STL-Datei: [SG.VL,SG.FL]=**VLFLui**([Vergrößerungsfaktor])
Schreiben der STL-Datei: **VLFLwriteSTL**(VL, FL, Filename)
Schreiben der STL-Datei: **SGwriteSTL**(SG, Filename)

Erzeugen von Buchstaben-Geometrien aus Zeichenfolgen:
Textgeometrien: [VL,FL]=**VLFLtextimage**(Text, Pixelsize, Dicke)

Mathematische Hilfsfunktionen:
Rotationsmatrix, Radiant R=**rot**(wz) oder R=**rot**(wx, wy, wz)
Rotationsmatrix, Grad R=**rotdeg**(wz) oder R=**rotdeg**(wx, wy, wz)

Beispiele:

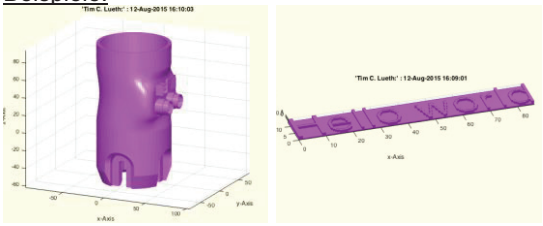

Abb. 7: a) VLFLui, Einlesen einer CAD-Datei für eine JACO-Roboterbasis, b) VLFLtextimage('Hello world')

Weitere Hinweise zu MATLAB: Die Umwandlung von Text oder Latex-Formeln in Oberflächenmodelle ist noch nicht geschwindigkeitsoptimiert und kann einige Sekunden benötigen.

6 Geschlossene Polygone und boolesche Operationen

Grundlegendes: Geschlossene Polygonzüge (CPL) können in MATLAB mit einer Koordinatenzeile [CPL1;NaN NaN; CPL2…] beliebig aneinander gehängt werden, um beispielweis eingeschlossene Polygonzüge zu beschrei-

ben. Eine andere Darstellung beruht auf geschlossenen Kantenlisten EL, die sich auf Punktlisten PL oder Vertexlisten VL beziehen.

Funktionen für geschlossene Polygone sowie Punkt- und Kantenlisten:
Boolesche Operationen: CPL=**CPLbool**(f,CPLA,CPLB), f='&', '|', '-', 'xor'
Umlaufabhängige Addition CPL=**CPLrecontour**(CPL)
Punkte/Kanten von CPL: [PL,EL]=**PLELofCPL**(CPL)
Zeichnen der Kantenliste: **PLELplot**(PL,EL,[Farbe, Dicke])

Beispiele:

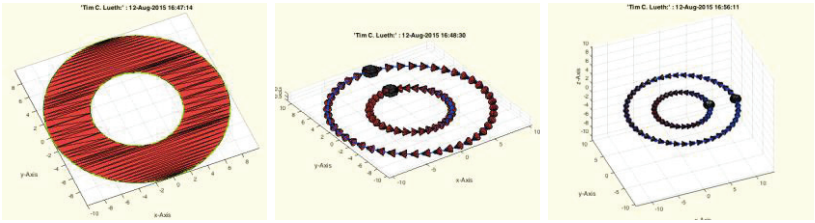

Abb. 8: a) CPL=CPLbool('-',PLcircle(10),PLcircle(5)), b) PLELofCPL(CPL), c) PLELofCPL([PLcircle(10);NaN NaN; PLcircle(5)])

7 Extrudieren von geschlossenen Polygonen in 3D

Grundlegendes: Beim Extrudieren von geschlossenen Polygonen deren Kanten sich kreuzen, entstehen getrennte Konturen, die gemeinsame Punkte besitzen. Diese gemeinsamen Punkte sind in der CPL-Darstellung als getrennte Punkte erkennbar, da sie in den zwei Polygonen, durch NaN-Einträge getrennt, auftauchen. Beim Extrudieren von zwei Polygonen mit zwei gemeinsamen Punkten entstehen automatisch zwei getrennte Modelle, die jedoch zwei senkrechte Kanten gemeinsam haben. Um diese Situation zu vermeiden, sollten geschlossene Polygone sich nie durchdringen oder berühren.

Funktionen für geschlossene Polygone:
Umlaufabhängige Addition: CPL=**CPLunite**(CPL)

Beispiele:

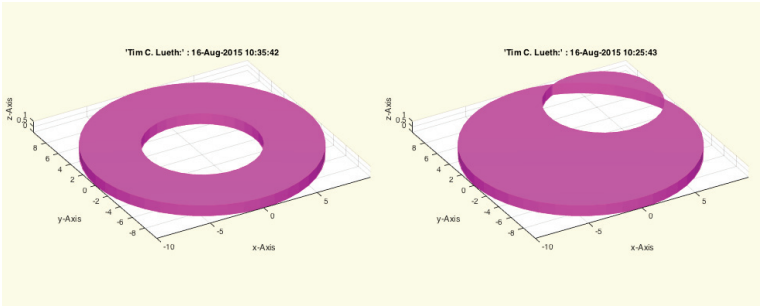

Abb. 9: a) SGofCPLz([PLcircle(10);NaN NaN;PLtrans(PLcircle(5),[0 0])],1),
b) SGofCPLz([PLcircle(10);NaN NaN;PLtrans(PLcircle(5),[5 5])],1)

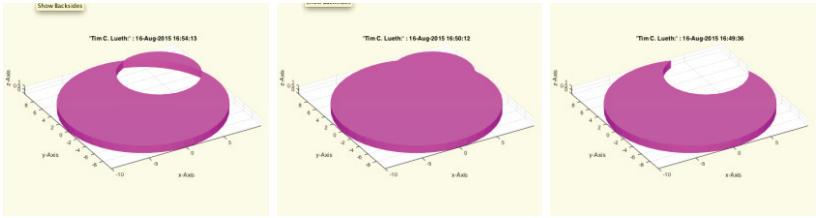

Abb. 10: PLA= PLcircle(10); PLB=PLtrans(PLcircle(5),[5 5]);
a) SGofCPLz([PLA; NaN NaN; PLB],1),
b) SGofCPLz(CPLunite([PLA; NaN NaN; PLB]),1),
c) SGofCPLz(CPLunite([PLA; NaN NaN; flipud(PLB)]),1)

Weitere Hinweise zu MATLAB: Es ist nur eine Frage der Zeit bis eine Analyse der Umlaufkanten und eine richtungsabhängige Addition und Subtraktion in der Reihenfolge der Polygone in MATLAB integriert wird. Die Grundlage ist in CPLbool bereits vorhanden. Dennoch bleibt dieses Problem erhalten, da auch bei Buchstaben oder Bildern gemeinsame Kanten aus Pixeln entstehen können.

8 Räumliches Anordnen und Vereinigen von Körpern

Grundlegendes: Bei der räumlichen Anordnung werden nur die Koordinaten der Vertices eines Körpers transformiert. Die Flächenindizes bleiben erhalten. Bei der Vereinigung von Körpern werden einfach die Vertexlisten und Flächenlisten hintereinander gehängt. Sich durchdringende Körper können anschließend wieder separiert werden.

Funktionen für das Anordnen von Punkten im Raum:

Räumliche Verschiebung: VL=**VLtrans**(VL, [rot(wx,wy,wz),[dx;dy;dz]])
Verschiebung in Ursprung: VL=**VLtrans**(VL,0)
Verschiebung 1. Oktant: VL=**VLtrans**(VL,1)
Verschiebung mit Vektor: VL=**VLtrans**(VL, [dx dy dz])
Rotation mit 3x3 Matrix: VL=**VLtrans**(VL, R), **VLtrans** (VL, rot(wx,wy,wz))
Transformation mit 4x4: VL=**VLtrans**(VL,T)

Funktionen für das Anordnen von Körpern im Raum:

Räumliche Verschiebung: SG=**SGtrans**(SG, [**rot**(wx,wy,wz),[dx;dy;dz]])
Verschiebung in Ursprung: SG=**SGtrans**(SG, 0)
Verschiebung 1. Oktant: SG=**SGtrans**(SG, 1)
Verschiebung mit Vektor: SG=**SGtrans**(SG, [dx dy dz])
Rotation mit 3x3 Matrix: SG=**SGtrans**(SG,R),**SGtrans**(VL, **rot**(wx,wy,wz))
Transformation mit 4x4: SG=**SGtrans**(SG, T)

Funktionen für das Verbinden von Punkten und Körpern:

Verbinden von VL,FL: [VL,FL]=**VLFLcat2**(VL1,FL1,VL2,FL2)
Verbinden von SG: SG=**SGcat2**(SG1,SG2)
Separieren von VL,FL: OL=**VLFLseparate**(VL,FL)
Separieren von SG: [SG,i]=**SGseparate**(SG)

Beispiele:

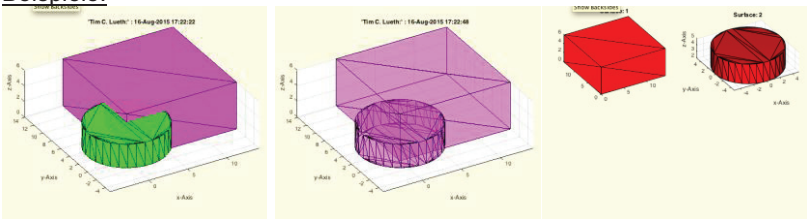

Abb. 11: a) Körper A und B, b) SGcat2(A,B), c) SGseparate(SGcat2(A,B))

9 Relative Anordnung und Ausrichtung von Körpern

Grundlegendes: Bei der relativen Anordnung und Ausrichtung werden die umschließenden Bounding-Boxen als Anhaltspunkt verwendet.

Funktionen für das relative Anordnen

Darunter (z-Richtung):	SG=**SGunder**(SG, SGB, [Lücke])
Darauf (z-Richtung):	SG=**SGontop**(SG, SGB, [Lücke])
Links (x-Richtung):	SG=**SGleft**(SG, SGB, [Lücke])
Rechts (x-Richtung):	SG=**SGright**(SG, SGB, [Lücke])
Davor (y-Richtung):	SG=**SGinfront**(SG, SGB, [Lücke])
Dahinter (y-Richtung):	SG=**SGbehind**(SG, SGB, [Lücke])
Zentriert (x, y, z):	SG=**SGincenter**(SG, SGB, [Lücke])

Funktionen für das relative Ausrichten

Unten bündig (z):	SG=**SGalignbottom**(SG, SGB, [Lücke])
Oben bündig (z):	SG=**SGaligntop**(SG, SGB, [Lücke])
Links bündig(x):	SG=**SGalignleft**(SG, SGB, [Lücke])
Rechts bündig (x):	SG=**SGalignright**(SG, SGB, [Lücke])
Vorne bündig (y):	SG=**SGalignfront**(SG, SGB, [Lücke])
Hinten bündig (z):	SG=**SGalignback**(SG, SGB, [Lücke])

Beispiele:

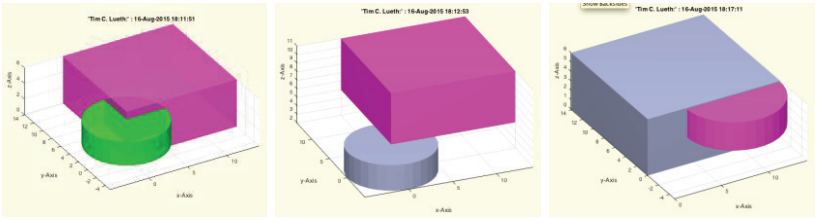

Abb. 12: a) Körper A und B, b) SGontop(A,B), c) SGaligntop(SGalignright(B,A),A,+0.1)

10 Rotation von Polygonen in 3D

Grundlegendes: Neben dem Extrudieren von Flächen zu Körpern ist die Nutzung von Rotationskörpern wichtig. Eine weitere Modellierungsweise ist das räumliche Anschmiegen an Kugeloberflächen.

Funktionen für das Slicen, Schneiden und Zerlegen von Körpern:

Rotationskörper:	SG=**SGofCPLrot**(CPL)
Kugelschmiegekörper:	SG=**SGofCPLsphere**(CPL, Radius, Dicke)

Beispiele:

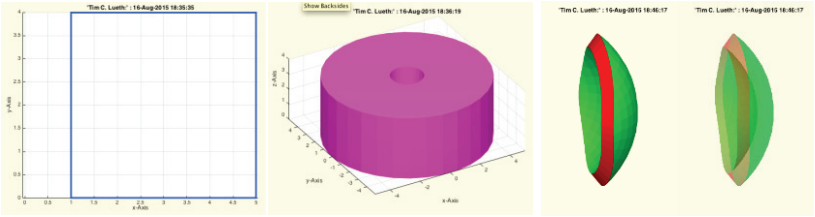

Abb. 13: a) PL=[1 0; 5 0; 5 4; 1 4], b) SGofCPLrot(PL), c) SGofCPLsphere(PLcircle(9),10,2)

11 Slicen, Schneiden und Zerlegen von Körpern

Grundlegendes: Das Slicen ist eine Funktion, die für die additive Fertigung eine besondere Bedeutung hat. Dabei wird für eine definierte z-Koordinate geprüft, welche Dreiecksflächen eines Körpers geschnitten werden, und die Kreuzungspunkte sowie die Kanten zwischen den Kreuzungspunkten berechnet. Mit dieser Methode lassen sich auch Körper in zwei Teile auftrennen. Liegt die Schnittebene jedoch parallel zu einer Fläche des Körpers, kann die eindeutige "Rekonstruktion" nicht garantiert werden. Diese Tatsache verhindert die eindeutige Auftrennung eines Körpers an schnittebenenparallelen Oberflächen.

Funktionen für das Slicen, Schneiden und Zerlegen von Körpern:
Beispielkörper für Tests: SG=**SGsample**([Körper-Nummer])
Slicen des Körpers: TR=**SGslicer**(SG, z-Koordinate)
Kanten des Schnittfläche: [VL,EL]=**VLELofTR**(TR)
Dreiecke der Schnittfläche:[VL,FL]=**VLFLofTR**(TR)
Schnittpunkte im Körper: SG=**SGcut2**(SG, z-Koordinate)
Schneiden eines Körpers: [SGA,SGB]=**SGcut**(SG, z-Koordinate)
Ausschneiden aus Körper: SG=**SGcut**(SG, [z1 z2])

Beispiele:

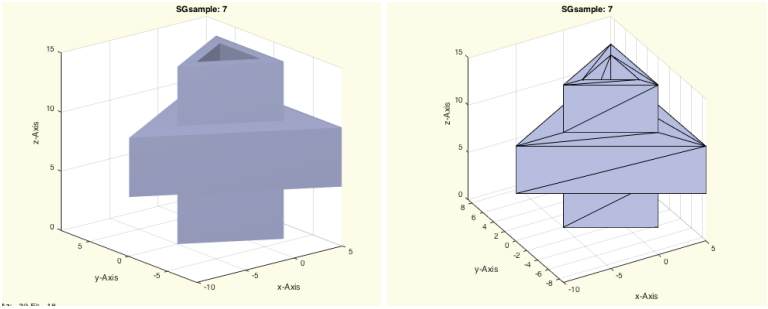

Abb. 14: SGsample(7): Ein Testkörper mit durchgehender Öffnung

Bei diesem Körper wird deutlich, dass sich die Schnittebene an der ersten Stufe für z=10 nicht eindeutig berechnen lässt, da sie für die Betrachtung des oberen oder unteren Körper unterschiedlich ausfallen würde.

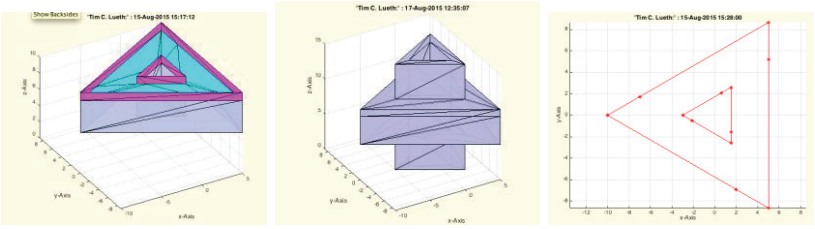

Abb. 15: a) SGslicer(SGsample(7),9), b) SGcut2(SGsample(7),9), c) VLELofTR(SGslicer(SGsample(7),9))

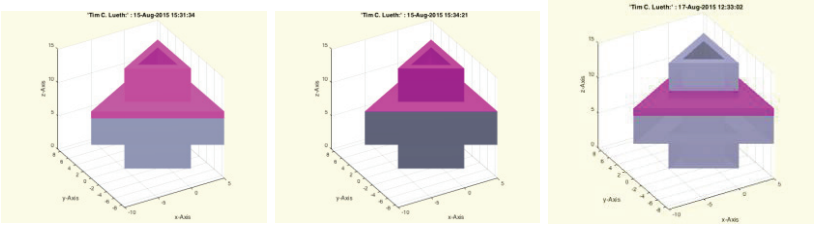

Abb. 16: a) SGcut(SGsample(7),9), b) SGcut(SGsample(7),9.99), c) SGcut(SGsample(7),[9 11])

12 Boolesche Operatoren von Oberflächenmodellen

Grundlegendes: Boolesche Operationen von Oberflächenmodellen haben den Nachteil, dass sich an den Schnittflächen die Zahl der Punkte und die Zahl der Dreiecke versechsfachen kann. Darüber hinaus entstehen kleine degenerierte Dreiecke. In fast allen Fällen kann man jedoch durch den geschickten Entwurf von Baugruppen derartige boolesche Operationen umgehen. Die Bauteile benötigen weniger Rechenzeit bei additiver Konstruktion.

Boolesche Operationen mit Oberflächenmodellen:
Verschmelzen der Körper: SG=**SGbool('+',SGA,SGB)**
Schnittmenge der Körper: SG=**SGbool('x',SGA,SGB)**
Körper A ohne Körper B: SG=**SGbool('A',SGA,SGB)**
Körper B ohne Körper A: SG=**SGbool('B',SGA,SGB)**

Beispiele:

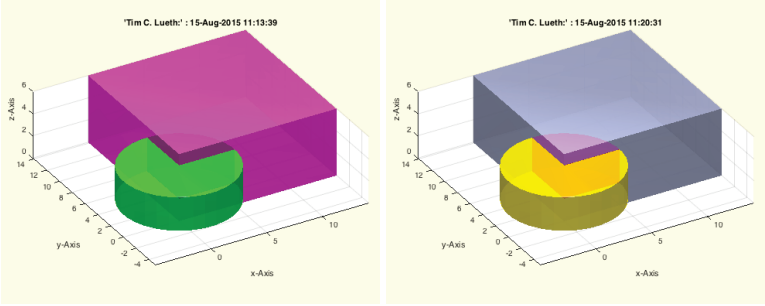

Abb. 17: Zwei Körper A und B erzeugen Schnittpunkte und Schnittflächen

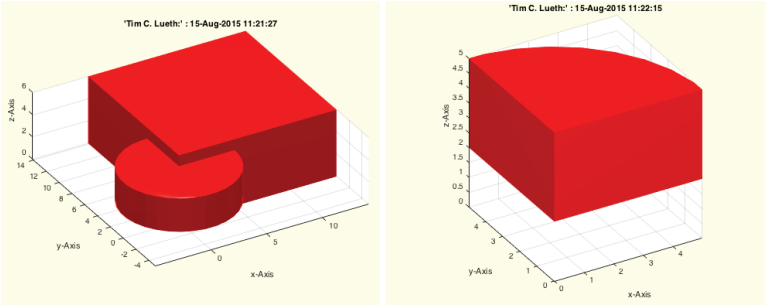

Abb. 18: a) SGbool('+',SGA,SGB), b) SGbool('x',SGA,SGB)

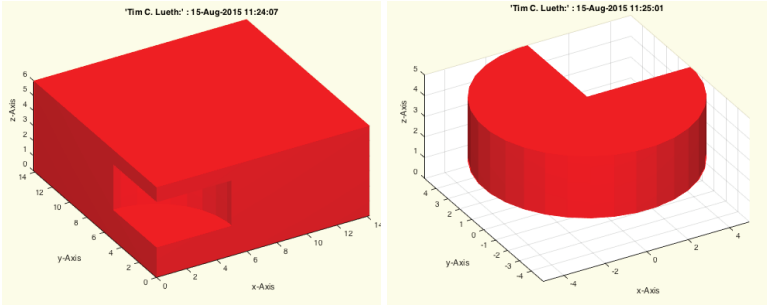

Abb. 19: a) SGbool('A',SGA,SGB), b) SGbool('B',SGA,SGB)

Im Falle eines additiven Entwurfs würde man den Körper "A ohne B" auch aus zwei Grundgeometrien (Quadrat und Quadrat ohne Kreissegment) extrudieren und den Körper dann aus drei Körpern geschichtet zusammensetzen. Die Zahl der Ecken und Flächen wäre geringer.

Der klassische CAD-Entwurf sieht jedoch eine boolesche Subtraktion vor.

Weitere Hinweise zu MATLAB: Aktuell dürfen bei booleschen Operationen keine gemeinsamen Grenzflächen der Körper vorhanden sein sondern die Körper müssen etwas überstehen. Bei der Operation werden die Punktkoordinaten auf ein Raster von 1e-5 (10nm) gerundet.

13 Anordnen und Verpacken für die Fertigung

Grundlegendes: Für die additive Fertigung sollten alle erforderlichen Bauteile in einem Druckprozess mit einer STL-Datei hergestellt werden können. Die Bauteile werden dazu mit einem kleinen Abstand nebeneinander und übereinander optimal raumausnutzend angeordnet. Eine Kiste mit einer Beschriftung erleichtert die Entnahme aus dem 3D-Drucker und dient später als Verpackung oder Aufbewahrungsort. Die Beschriftung kann eine Bauteilliste, die Anleitung zum Zusammenbau oder zur Einsortierung enthalten.

Definition von Körpern für Viergelenke :

Definition Gliedern:	SG=**fourBarLinkageKit**('Bar', Länge)
Definition Achsen:	SG=**fourBarLinkageKit**('Bolt')
Definition Wellen:	SG=**fourBarLinkageKit**('Shaft')
Definition Distanzstück:	SG=**fourBarLinkageKit**('Spacer')

Anordnen und Verpacken für die additive Fertigung:

Anordnen im Volumen: SG=**SGpackaging**({SG1,SG2,…},[x y z])
Sammeln im Container: SC=**SGCaddn** ([SC, SG, Kopienanzahl)
Anordnen des Containers: SG=**SGpackaging**(SC,[x y z])
Erzeugen der Verpackung: SG=**SGboxing**(SG)

Beispiele:

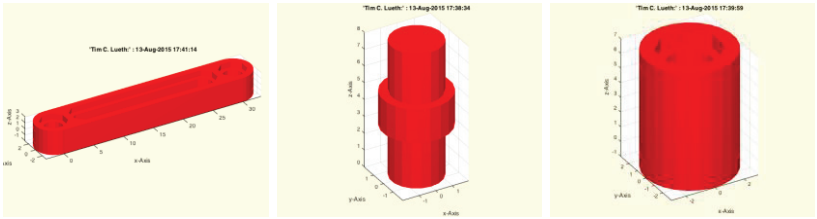

Abb. 20: a) fourBarLinkageKit('Bar',30), b) fourBarLinkageKit('Bolt'), c) fourBarLinkageKit('Spacer')

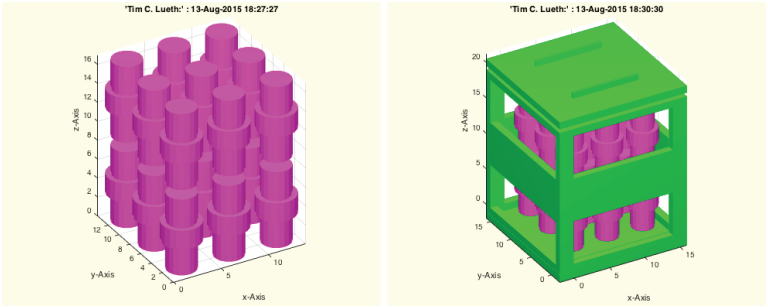

Abb. 21: a) SGpackaging([],SGBolt),[20 20 40]), b) SGboxing(SG)

14 Koordinatensysteme

Grundlegendes: Koordinatensysteme werden als *Transformationsmatrixliste T* von 4x4 Matrizen zusammen mit einer *Transformationsnamensliste Tname* und der Liste der *Gelenkgrenzflächen TFiL erzeugt,* die diese Transformationsmatrix definieren. Das *Oberflächenmodell SG* wird mit diesen Feldern erweitert.

Definition von Koordinatensystemen:
Manuelle Definition: SG.T{1}=eye(4); SG.Tname={'A'}

Matrix zu einem Namen: T=**SGT**(SG, Framenamen)
Interaktive Maus-Eingabe: SG=**SGTui**(SG,[Framenamen])
Entfernung von Frames: SG=**SGTremove**(SG,[Framenamen])

Beispiele:

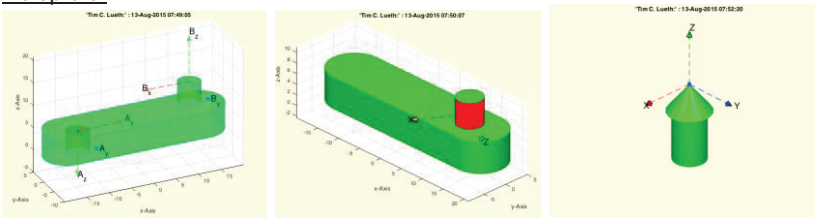

Abb. 22: a) Definition von Frames über planare Flächen, b) Definition von Frames über sphärische Flächen, c) Definition von Frames über Eckpunkte: SGTui(SG)

15 Kinematische Ketten

Grundlegendes: Um Körper als Gliederelemente zu einer kinematischen Kette zu verbinden, benötigt man eine *Liste der Körpergeometrien SG*, eine *Namensliste SN* der Körper sowie *Basistransformationsmatrizen BT* der Körper, mit der die Lage der Körper im Raum definiert werden. Sowohl die Vertexlisten als auch die Koordinatensysteme sind dann relativ zu der jeweiligen Transformationsmatrix BT definiert. Eine *kinematische Kette CS* wird durch mindestens eine Zeichenkette zur Verbindung von Frames zwischen zwei unterschiedlichen Körpern beschrieben. Bei der Definition muss daher für jeden Körper geprüft werden, ob er mindestens zwei Koordinatensysteme (SGT) besitzt. *Kinematikmodelle KM* sind "MATLAB-Structs".

Definition von kinematischen Ketten:
Definition der Körper: KM=**KMofSGs**(SGs)

Beispiele:

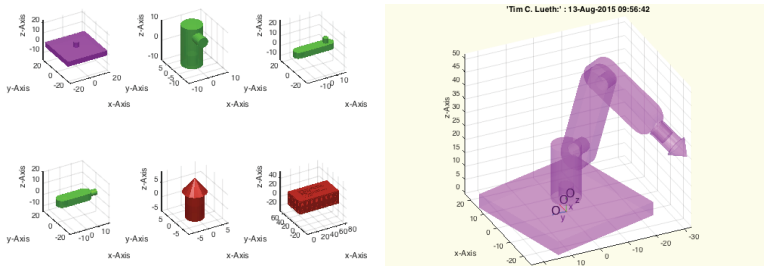

Abb. 23: a) Die Körper SG0, SG1, SG2, SG3, SG4 erzeugt mit AIM_Robot;
b) KMofSGs({SG0,SG1,SG2,SG3,SG4}),

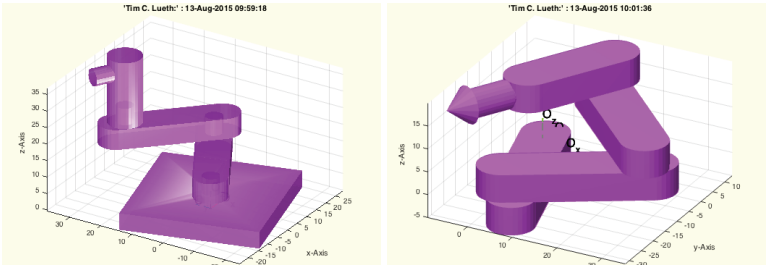

Abb. 24: a) KMofSGs({SG0,SG2,SG2,SG1}),
b) KMofSGs({SG2,SG2,SG2,SG3,SG4})

Weitere Hinweise zu MATLAB: Aktuell wird noch keine automatische Erkennung und Behandlung von geschlossenen kinematischen Ketten durchgeführt. Für das Viergelenk ist jedoch eine allgemeine Lösung implementiert.

16 Kollisionsanalyse oder Grenzflächenanalyse

Grundlegendes: Die Analyse von Kollisionen zweier Körper, beispielsweise der Glieder eines Mechanismus, geschieht grundsätzlich über die Prüfung, ob sich Dreiecksflächen der beiden Körper schneiden. Für Drehgelenke muss dabei berücksichtigt werden, dass durch die begrenzte Zahl der Eckpunkte eines Polygons, sich diese beispielsweise bei einem Drehgelenk unglücklich annähern und durchdringen können. In diesem Fall muss vor der Prüfung der Kollision einer der Körper um eine Toleranzschwelle von 0.1 mm geschrumpft werden. Die Kollisionsprüfung kann beschleunigt wer-

den, wenn man nicht a) alle Körper gegeneinander sondern nur b) die direkt verbundene Gelenkglieder oder nur c) die sich tatsächlich berührende Gelenkflächen auf Durchdringung prüft.

Kollisions- und Einschlussprüfung von Oberflächenmodellen in 3D:
Kollisionspunktberechnung: VL=**VLcrossingSG**(SGA,SGB)
Einschlussprüfung: ViL=**SGisInterior**(SG,VL)
Kollisionen in einer Kette: [KM,XVL]=**KMofSGs**(SGs,[Winkel],Toleranz)

Beispiele:

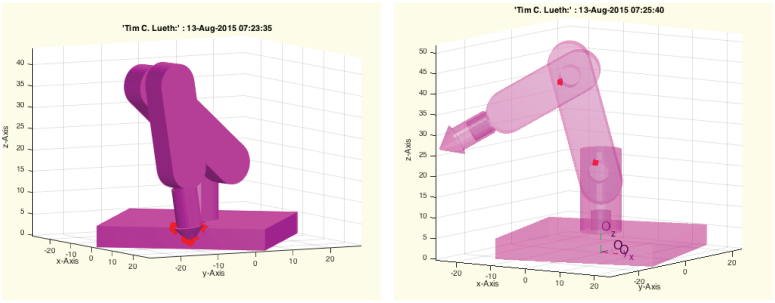

Abb. 25: Kollision (rot) des TCP mit der Grundplatte, b) Kollision an den Gelenkflächen bei einer reduzierten Toleranz von 50µm

Danksagung

Bedanken möchte ich mich bei Franz Irlinger für unsere Diskussionen, bei Mattias Träger und Laurens de Smedt für die intensiven Tests und Diskussionen zur Anwendbarkeit sowie bei Christina Friedrich für die Prüfung von Teilen der MATLAB-Bibliothek für das Betriebssystem Windows/PC. Yannick Krieger wie den Vorgenannten danke ich für Korrekturen. Die beschriebenen MATLAB-Funktionen wurden von Tim Lüth erstellt und für OSX/MAC getestet.

Erinnern wollen wir auch an die früheren Getriebelehrer des heutigen Lehrstuhls MIMED. Es waren die Professoren in Folge: F.A. Klingenfeld (1868-1880), W. Marx (1880-1886), Ludwig E. Burmester (1887-1912), S. Finsterwalder (1912-1931), G. Marx (-1940), H. Wögerbauer (1940-1945), Rudolf Beyer (1948-1960), R. Unterberger (1955-1977) und Joachim Heinzl (1978-2005).

Weiterer Dank gilt der Deutschen Forschungsgemeinschaft für die Bewilligung der Anlage (EOS) zum Selektiven Lasersintern.

17 Literatur

[1] Tim Lüth und Franz Irlinger: Berechnete Erzeugung von dreidimensionalen Oberflächen im STL-Format aus der Beschreibung planarer Mechanismen für die generative Fertigung durch selektives Lasersintern, 10. Getriebetechnikkolloquium, Ilmenau, Tagungsband, S .267-284