Tutorial 11: Attaching Coordinates Frames to Create Kinematik Models

2015-06-08: Tim C. Lueth, MIMED - Technische Universität München, Germany (URL: http://www.mimed.de) - Last Change: 2017-07-07

Contents

Complete List of all Tutorials with Publishable MATLAB Files of this Solid-Geoemtries Toolbox

The following topics are covered an explained in the specific tutorials:

- Tutorial 01: First Steps Using the VLFL-Toolbox for Solid Object Design
- Tutorial 02: Using the VLFL-Toolbox for STL-File Export and Import
- Tutorial 03: Closed 2D Contours and Boolean Operations in 2D
- Tutorial 04: 2½D Design Using Boolean Operators on Closed Polygon Lists (CPL)
- Tutorial 05: Creation, Relative Positioning and Merging of Solid Geometries (SG)
- Tutorial 06: Relative Positioning and Alignment of Solid Geometries (SG)
- Tutorial 07: Rotation of Closed Polygon Lists for Solid Geometry Design
- Tutorial 08: Slicing, Closing, Cutting and Separation of Solid Geometries
- Tutorial 09: Boolean Operations with Solid Geometries
- Tutorial 10: Packaging of Sets of Solid Geometries (SG)
- Tutorial 11: Attaching Coordinates Frames to Create Kinematik Models
- Tutorial 12: Define Robot Kinematics and Detect Collisions
- Tutorial 13: Mounting Faces and Conversion of Blocks into Leightweight-structures
- Tutorial 14: Manipulation Functions for Closed Polygons and Laser Cutting (SVG)
- Tutorial 15: Create a Solid by 2 Closed Polygons
- Tutorial 16: Create Tube-Style Solids by Succeeding Polygons
- Tutorial 17: Filling and Bending of Polygons and Solids
- Tutorial 18: Analyzing and modifying STL files from CSG modeler (Catia)
- Tutorial 19: Creating drawing templates and dimensioning from polygon lines
- Tutorial 20: Programmatically Interface to SimMechanics Multi-Body Toolbox
- Tutorial 21: Programmatically Convert Joints into Drives (SimMechanics)
- Tutorial 22: Adding Simulink Signals to Record Frame Movements
- Tutorial 23: Automatic Creation of a Missing Link and 3D Print of a Complete Model
- Tutorial 24: Automatic Creation of a Joint Limitations
- Tutorial 25: Automatic Creation of Video Titels, Endtitels and Textpages
- Tutorial 26: Create Mechanisms using Universal Planar Links
- Tutorial 27: Fourbar-Linkage: 2 Pose Syntheses and Linkage Export for 3D Printing
- Tutorial 28: Fourbar-Linkage: 3 Pose Syntheses and Linkage Export for 3D Printing
- Tutorial 29: Create a multi body simulation using several mass points
- Tutorial 30: Creating graphical drawings using point, lines, surfaces, frames etc.
- Tutorial 31: Importing 3D Medical DICOM Image Data and converting into 3D Solids
- Tutorial 32: Exchanging Data with a FileMaker Database
- Tutorial 33: Using a Round-Robin realtime multi-tasking system
- Tutorial 34: 2D Projection Images and Camera Coordinate System Reconstruction
- Tutorial 35: Collection of Ideas for Tutorials
- Tutorial 36: Creating a Patient-Individual Arm-Skin Protector-Shell

Motivation for this tutorial: (Originally SolidGeometry 2.3 required)

function VLFL_EXP11 % MUST RUN AS A SCRIPT BECAUSE OF loadweb+

```
close all;
```

2. Loading the 5 components of a 4DoF robot solid model

Before explaining how to create the parts of a robot kinematik we just load such components in. The command line load AIM_robot

```
loadweb ('AIM_SGrobot.mat');
% SG0=SGfixerrors(SG0,1e-3); SGchecker(SG0);
```

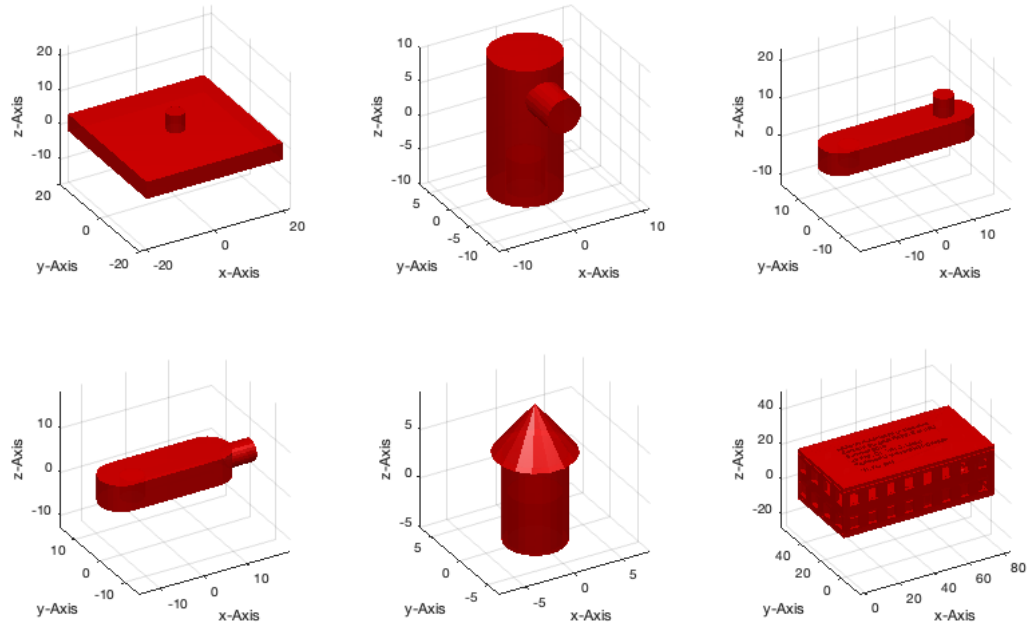
```
% SG1=SGfixerrors(SG1,1e-3); SGchecker(SG1);
% SG2=SGfixerrors(SG2,1e-3); SGchecker(SG2);
% SG3=SGfixerrors(SG3,1e-3); SGchecker(SG3);
% SG4=SGfixerrors(SG4,1e-3); SGchecker(SG4);
% save ('AIM_SGrobot','SG0','SG1','SG2','SG3','SG4','SGrobot');
```

```
loadweb: Access path to changed from "www.mimed.mw.tum.de" to "www.mw.tum.de/mimed/" in 2020 Aug.
loadweb: Access path to changed from "www.mw.tum.de/mimed/" to "www.mec.ed.tum.de/mimed/" in 2021 Nov.
Downloading "https://www.mec.ed.tum.de/fileadmin/w00cbp/mimed/Matlab_Toolboxes/AIM_SGrobot.mat" into: /Volumes/LUETH-WIN/WIN AIM Matlab Libraries/Solic
```

- returns a solid geometry SG0 which is a base plate with a rotatorial joint
- returns a solid geometry SG1 which is a link with a rotatorial joint
- returns a solid geometry SG2 which is a link with a rotatorial joint
- returns a solid geometry SG3 which is a link with a rotatorial joint
- returns a solid geometry SG4 which is a hand with a pointing tip
- returns a solid geometry SGrobot which can be written as STL-File an printed using a 3D printer.

```
SGfigure;

subplot(2,3,1); SGplot(SG0); view (-30,30); VLFLplotlight(1,0.9);
subplot(2,3,2); SGplot(SG1); view (-30,30); VLFLplotlight(1,0.9);
subplot(2,3,3); SGplot(SG2); view (-30,30); VLFLplotlight(1,0.9);
subplot(2,3,4); SGplot(SG3); view (-30,30); VLFLplotlight(1,0.9);
subplot(2,3,5); SGplot(SG4); view (-30,30); VLFLplotlight(1,0.9);
subplot(2,3,6); SGplot(SGrobot); view (-30,30); VLFLplotlight(1,0.9);
SGwriteSTL (SGrobot,'4-DOF Robot Set');
```

```
publishSGPDF:<a href = "matlab: openbydoubleclick ('/Users/timlueth/Desktop')">/Users/timlueth/Desktop/</a><a href = "matlab: openbydoubleclick ('/User
```



3. The concept of attaching coordinate frames as 4x4 honogenous transformation matrix

If we analyze the structure of one of the components of the robot we see that we have now more than just the surface of the geometry.

```
SG0

% We see that beside vertices and facets (VL, FL) we have a color and a
% alpha value for transparency when plotting.
```

```
SG0 =
  struct with fields:

      VL: [79×3 double]
      FL: [154×3 double]
   alpha: 0.8000
   color: 'm'
   Tname: {'A'  'B'}
```

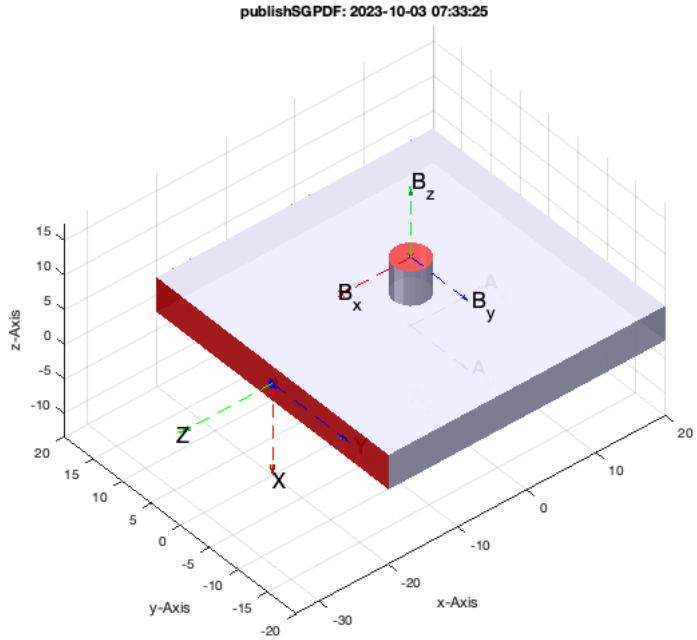
```
        T: {[4×4 double]  [4×4 double]}
     TFiL: {[2×1 double]  [21×1 double]}
```

- *Tname* is a cell list contain the ascii-string of the names of the coordinate frames
- *T* is the 4x4 homogenous transformation matrix related to the indexed name
- *TFiL* is an optional index of the facets that belong to the surface that defines the coordinate system To the homogenous transformation matrix out of the struct, the most convinient way is to use the function:

- **SGT** returns for a given solid and a given frame name the 4x4 matrix
- **SGT** draws the part and the frames and the defining facets if there is no output parameter

```
A=SGTui(SG0,'A')
B=SGTui(SG0,'B')

SGTplot(SG0);
view(-40,40);
```

```
A =
  struct with fields:

       VL: [79×3 double]
       FL: [154×3 double]
    alpha: 0.8000
    color: 'm'
    Tname: {'A'  'B'}
        T: {[4×4 double]  [4×4 double]}
     TFiL: {[2×1 double]  [21×1 double]}
B =
  struct with fields:

       VL: [79×3 double]
       FL: [154×3 double]
    alpha: 0.8000
    color: 'm'
    Tname: {'A'  'B'}
        T: {[4×4 double]  [4×4 double]}
     TFiL: {[2×1 double]  [2×1 double]}
```



4. Attaching manually coordinate frames as 4x4 honogenous transformation matrix

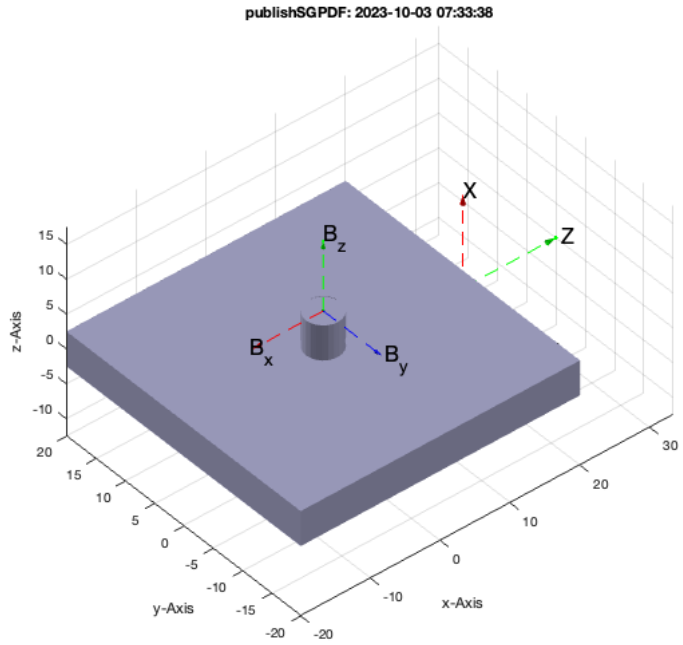
Since there is no requirement to use the facets TFiL, T matrices and their name can easily added by a programm during the design phase. Nevertheless, there is also a need to add frames interactively. For that purpose there are two other functions to add or to remove frames.

- **SGTremove** removes a named frame from the structure
- **SGTui** opens a figure and allows to generate a frame by touching a surface or point

To use SGTui you should a) first rotate the part on the screen until you see the surface where you like to set a frame, b) press enter and c) klick on the plane to set the frame. Now set two Frames 'C' and 'D'

```
A=SGTui(SG0,'C')
A=SGTui(A,'D')
view(-40,40);
```

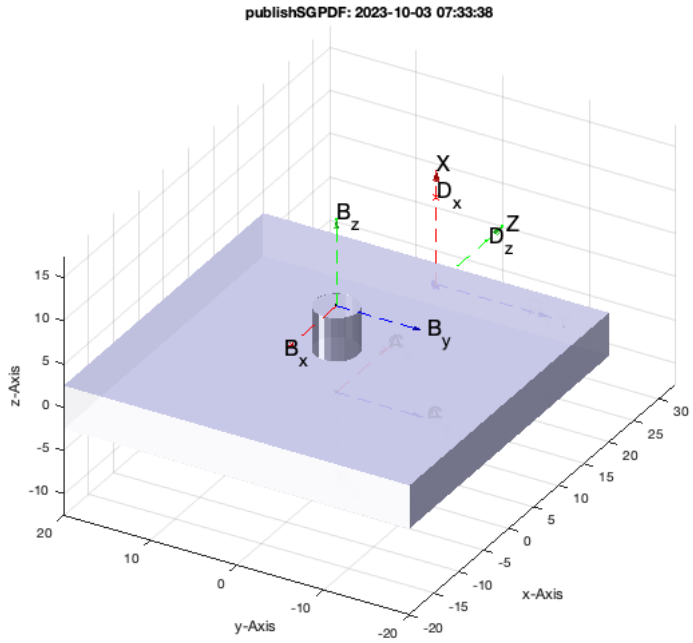
```
A =
  struct with fields:

      VL: [79×3 double]
      FL: [154×3 double]
   alpha: 0.8000
   color: 'm'
   Tname: {'A'  'B'  'C'}
       T: {[4×4 double]  [4×4 double]  [4×4 double]}
    TFiL: {[2×1 double]  [21×1 double]  [2×1 double]}
A =
  struct with fields:

      VL: [79×3 double]
      FL: [154×3 double]
   alpha: 0.8000
   color: 'm'
   Tname: {'A'  'B'  'C'  'D'}
       T: {[4×4 double]  [4×4 double]  [4×4 double]  [4×4 double]}
    TFiL: {[2×1 double]  [21×1 double]  [2×1 double]  [2×1 double]}
```



No we remove the first two frames 'A' and 'B'

```
A=SGTremove(A,'A')
A=SGTremove(A,'B')
SGT(A); view(-60,30);
```

```
A =
  struct with fields:

      VL: [79×3 double]
      FL: [154×3 double]
   alpha: 0.8000
   color: 'm'
   Tname: {'B'  'C'  'D'}
       T: {[4×4 double]  [4×4 double]  [4×4 double]}
    TFiL: {[21×1 double]  [2×1 double]  [2×1 double]}
A =
  struct with fields:

      VL: [79×3 double]
      FL: [154×3 double]
   alpha: 0.8000
   color: 'm'
   Tname: {'C'  'D'}
       T: {[4×4 double]  [4×4 double]}
    TFiL: {[2×1 double]  [2×1 double]}
```

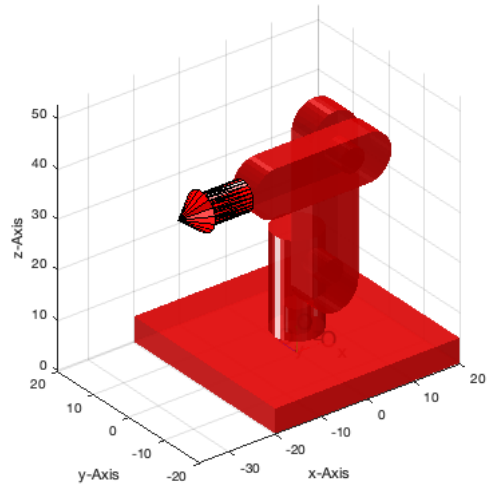


5. Creating kinematic models consisting of named solids

After being able to attach coodinate systems by frames to a solid, we can chain these solids by a string that describes which frames of the indiviudal objects are linked together. For this purpose we define a structure **KM (kinematik model)** that is a list of solids, followed by an ascii identifier and a transformation matrix for the origin of the solid. If the solids are chained, a function **KMchain** calculates those 3rd column transformation matrix to move and rotate the solid so that is fits to the given description of linked frames. **KMplot** shows the position of the individual solids in space.

```
% KM={SG0,'A',eye(4);SG1,'B',eye(4);SG2,'C',eye(4);SG3,'D',eye(4);SG4,'E',eye(4)}

KM.SG={SG0,SG1,SG2,SG3,SG4};
KM.Sname={'A','B','C','D','E'};
KM.BT={eye(4),eye(4),eye(4),eye(4),eye(4)};


KMchain(KM,'A.A-A.B-B.A-B.B-C.A-C.B-D.A-D.B-E.A-E.B-');
KM=KMchain(KM,'A.A-A.B-B.A-B.B-C.A-C.B-D.A-D.B-E.A-E.B-')
KMplot(KM);

% Now let us see how the 3rd column matrices describe the position of the
% solids in 3D space to create the robot model
KM.BT{:}
```

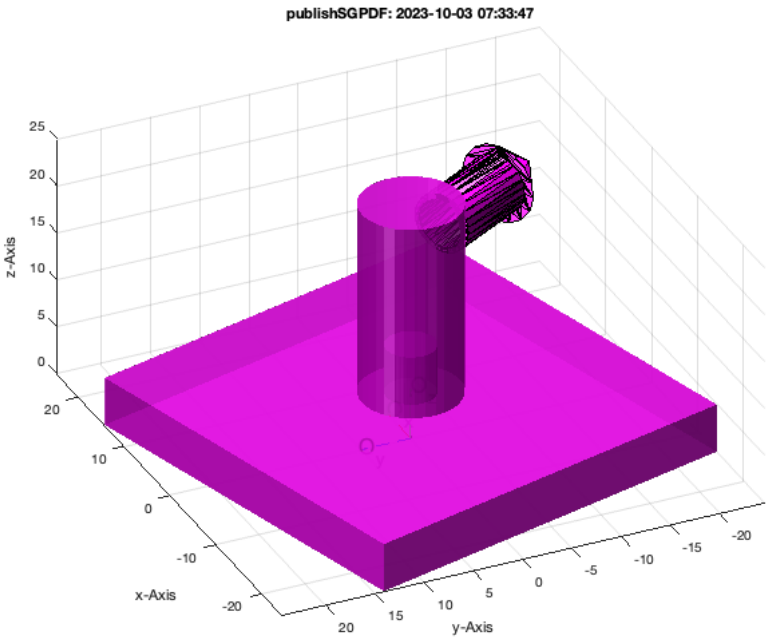
```
KM =
  struct with fields:

       SG: {1×5 cell}
    Sname: {'A'  'B'  'C'  'D'  'E'}
       BT: {1×5 cell}
ans =
   -0.0000    1.0000   -0.0000   -0.0000
   -1.0000   -0.0000         0         0
   -0.0000    0.0000    1.0000    2.5000
         0         0         0    1.0000
ans =
    1.0000    0.0000   -0.0000   -0.0120
   -0.0000    1.0000   -0.0000   -0.0120
         0    0.0000    1.0000   15.0000
         0         0         0    1.0000
ans =
   -0.0000   -1.0000    0.0000   -0.0000
    0.0000   -0.0000   -1.0000   -4.9880
    1.0000    0.0000   -0.0000   32.9590
         0         0         0    1.0000
ans =
   -1.0000    0.0000    0.0000   -7.4530
    0.0000   -0.0000   -1.0000  -10.9880
    0.0000   -1.0000   -0.0000   45.4350
         0         0         0    1.0000
ans =
    0.0000   -0.0000   -1.0000  -27.4290
   -0.0000   -1.0000    0.0000  -13.9760
   -1.0000   -0.0000    0.0000   45.4470
         0         0         0    1.0000
```



6. Automatic creation of a chain

```
KMofSGs({SG0,SG1,SG4})
```

```
KMofSGs: No collisions found for tolerance: 0.10
ans =
  struct with fields:

       SG: {[1×1 struct]  [1×1 struct]  [1×1 struct]}
    Sname: {3×1 cell}
       BT: {3×1 cell}
       KC: {'A.A-A.B-B.A-B.B-C.A-C.B-'}
```



Final remarks on toolbox version and execution date

```
VLFLlicense
```

```
This VLFL-Lib, Rel.  (2023-Oct-03), is for limited non commercial educational use only!
Licensee: Tim Lueth (Development Version)!
Please contact Tim Lueth, Professor at TU Munich, Germany!
WARNING: This VLFL-Lib (Rel. ) license will exceed at 06-Jul-2078 07:33:48!
Executed 03-Oct-2023 07:33:50 by 'timlueth' on a MACI64 using Mac OSX 13.6 | R2023a Update 5 | SG-Lib 5.4
```

```
======================================== Used Matlab products: ========================================
database_toolbox
distrib_computing_toolbox
fixed_point_toolbox
image_toolbox
map_toolbox
matlab
optimization_toolbox
pde_toolbox
phased_array_system_toolbox
signal_blocks
signal_toolbox
simmechanics
simscape
simulink
statistics_toolbox
=======================================================================================================
```

- *Tim Lueth, tested and compiled on OSX 10.7.5 with Matlab 2014b on 2015-06-08*
- *Christina Friedrich, executed and published on 64 Bit PC using Windows with Matlab 2015a on 2015-06-17*

Published with MATLAB® R2023a