Exercise 11: Attaching Coordinates Frames to Create Kinematik Models

2015-06-08: Tim C. Lueth, MIMED - Technische Universität München, Germany 2015-08-18: Tim C. Lueth, changed for better kinematic model structure (URL: http://www.mimed.de)

Contents

- 1. Summary of the already explained aspects of the VLFL-Toolbox
- 2. Loading the 5 components of a 4DoF robot solid model
- 3. The concept of attaching coordinate frames as 4x4 honogenous transformation matrix
- 4. Attaching manually coordinate frames as 4x4 honogenous transformation matrix
- 5. Creating kinematic models consisting of named solids
- 6. Automatic creation of a chain
- Final remarks on toolbox version and execution date

1. Summary of the already explained aspects of the VLFL-Toolbox

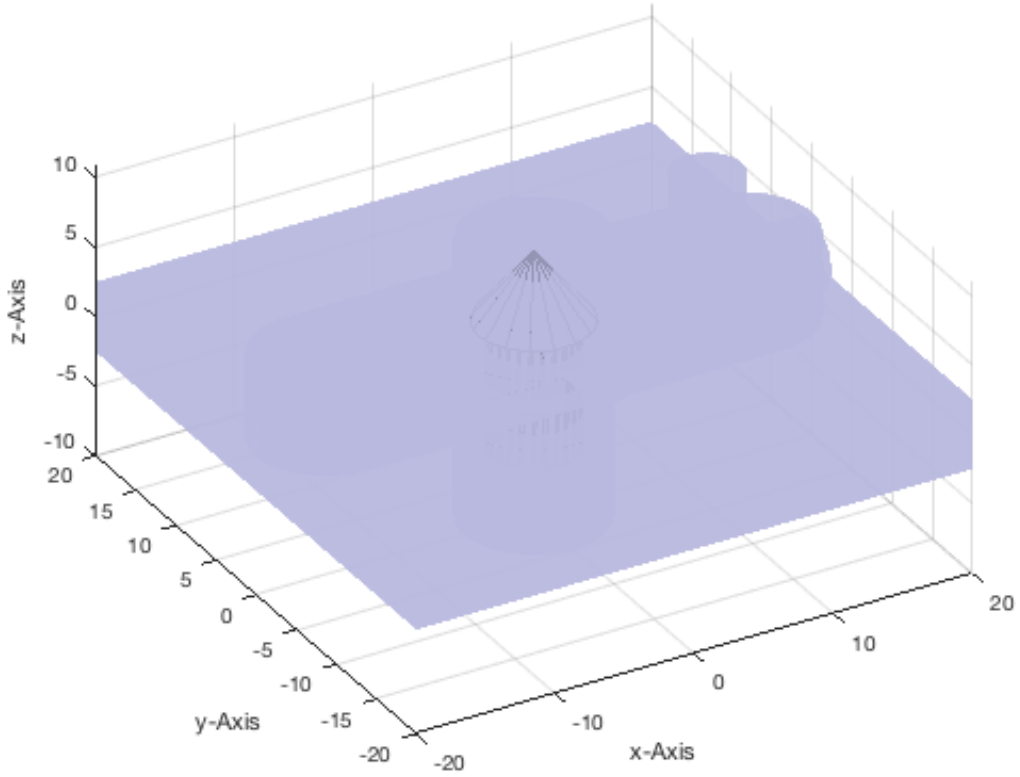
In examples 1-10 we've learned about:

- *Point lists* (PL), *Vertex lists* (VL), *Facet lists* (VL), *Edge lists* (EL)
- Functions to import, check, and export of *STL-File* data
- Functions for generating text strings as solid volumes
- *Closed polygon lists* (CPL) for boolesche operations of contours
- 2.5D extrusion of closed polyogon list (CPL) to *Solid Geometries* (SG)
- Spatial transformation and merging of solid geometry sets
- Relative positioning of solids geometries and boolean operations on solids
- Packaging and boxing of several parts into a complete assemble kit

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

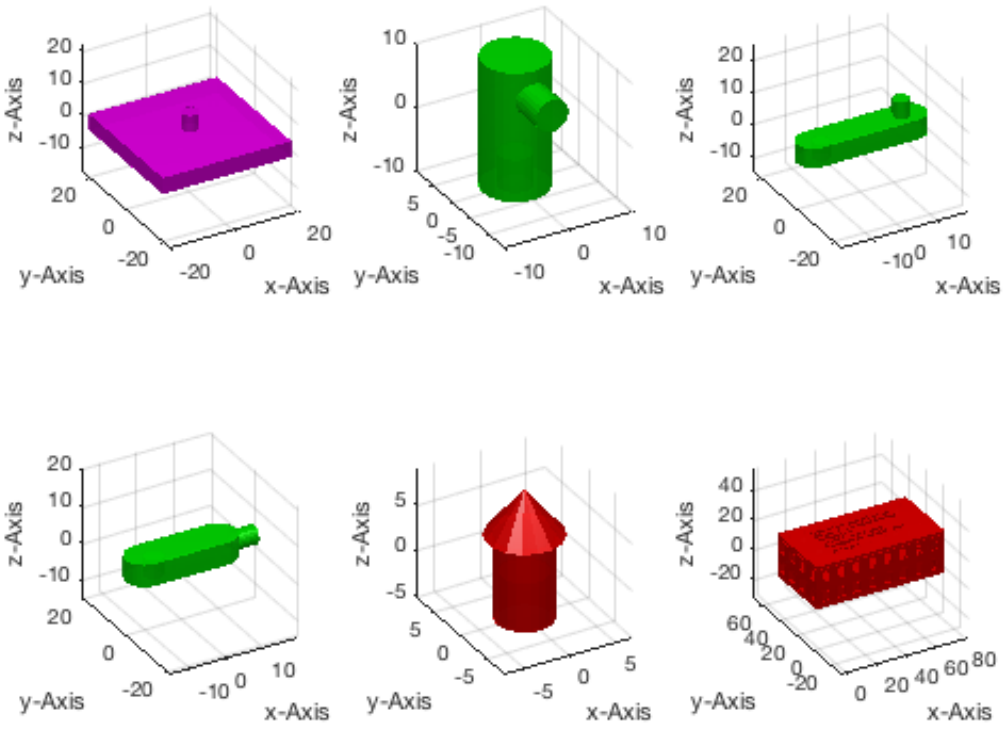
```
load AIM_SGrobot
SG0=SGfixerrors(SG0,1e-3); SGchecker(SG0);
SG1=SGfixerrors(SG1,1e-3); SGchecker(SG1);
SG2=SGfixerrors(SG2,1e-3); SGchecker(SG2);
SG3=SGfixerrors(SG3,1e-3); SGchecker(SG3);
SG4=SGfixerrors(SG4,1e-3); SGchecker(SG4);
save ('AIM_SGrobot','SG0','SG1','SG2','SG3','SG4','SGrobot');
```



- 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.

```
VLFLfigure;
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');
```

```
1000..2000..3000..4000..5000..6000..7000..8000..9000..10000..11000..12000..13000..14000..15
000..16000..17000..18000..19000..20000..21000..22000..23000..
```



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=SGT(SG0,'A')
B=SGT(SG0,'B')

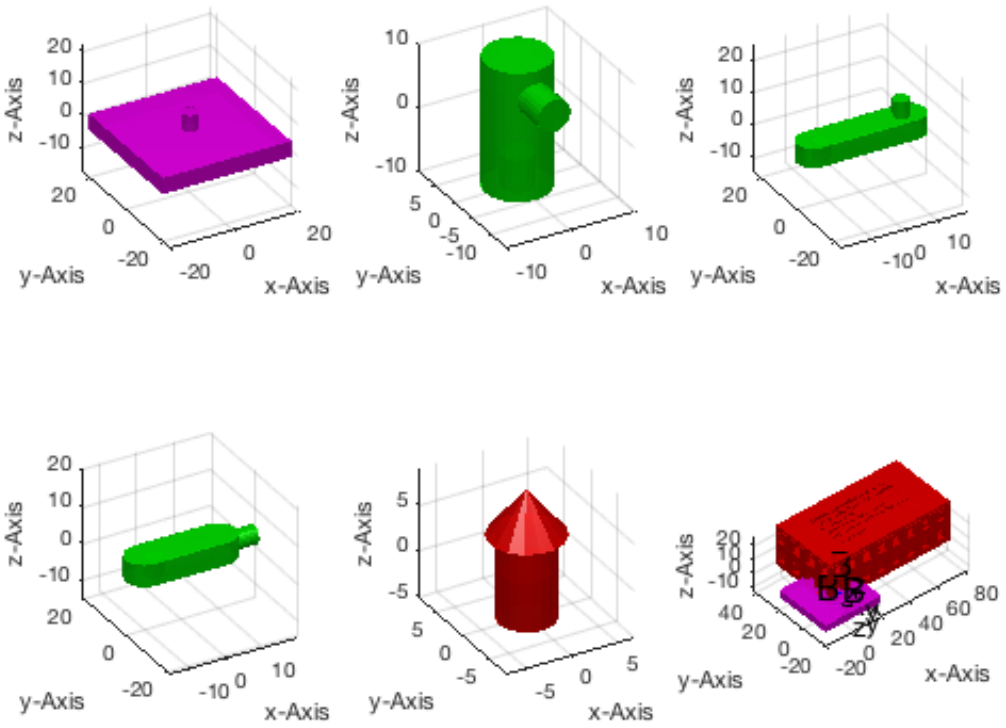
SGT(SG0);
view(-40,40);
```

```
A =

    1.0000         0         0         0
         0   -1.0000         0         0
    0.0000         0   -1.0000   -2.5000
         0         0         0    1.0000


B =

   -1.0000         0         0    0.0120
         0   -1.0000         0         0
         0         0    1.0000    7.5000
         0         0         0    1.0000
```



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:

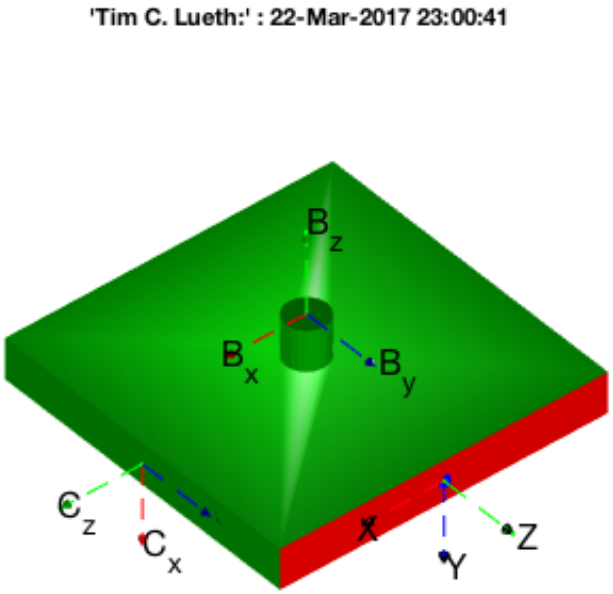
      VL: [79×3 double]
      FL: [154×3 double]
   alpha: 0.8000
   color: 'm'
   Tname: {'A'  'B'  'C'}
       T: {[4×4 double]  [4×4 double]  [4×4 double]}
```

```
      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:
```

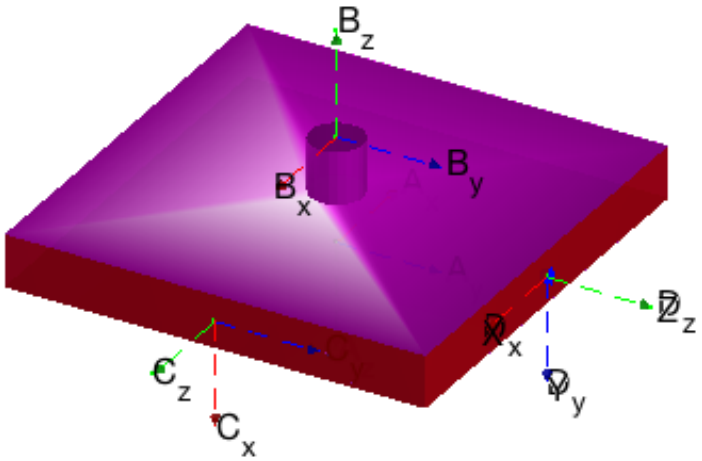
```
        VL: [79×3 double]
        FL: [154×3 double]
     alpha: 0.8000
     color: 'm'
     Tname: {'B'  'C'  'D'}
         T: {[4×4 double]  [4×4 double]  [4×4 double]}
      TFiL: {[21×1 double]  [2×1 double]  [2×1 double]}


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]}
```

'Tim C. Lueth:' : 22-Mar-2017 23:00:41



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 =

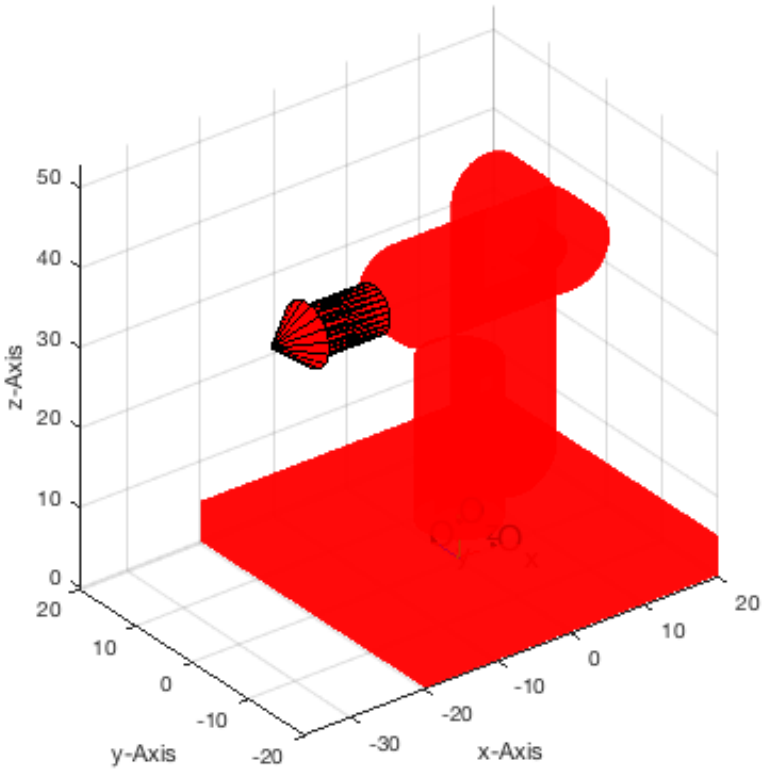
   -0.0000   -1.0000    0.0000   -0.0000
    0.0000   -0.0000   -1.0000   -4.9880
    1.0000    0.0000   -0.0000   32.9590
         0         0         0    1.0000


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

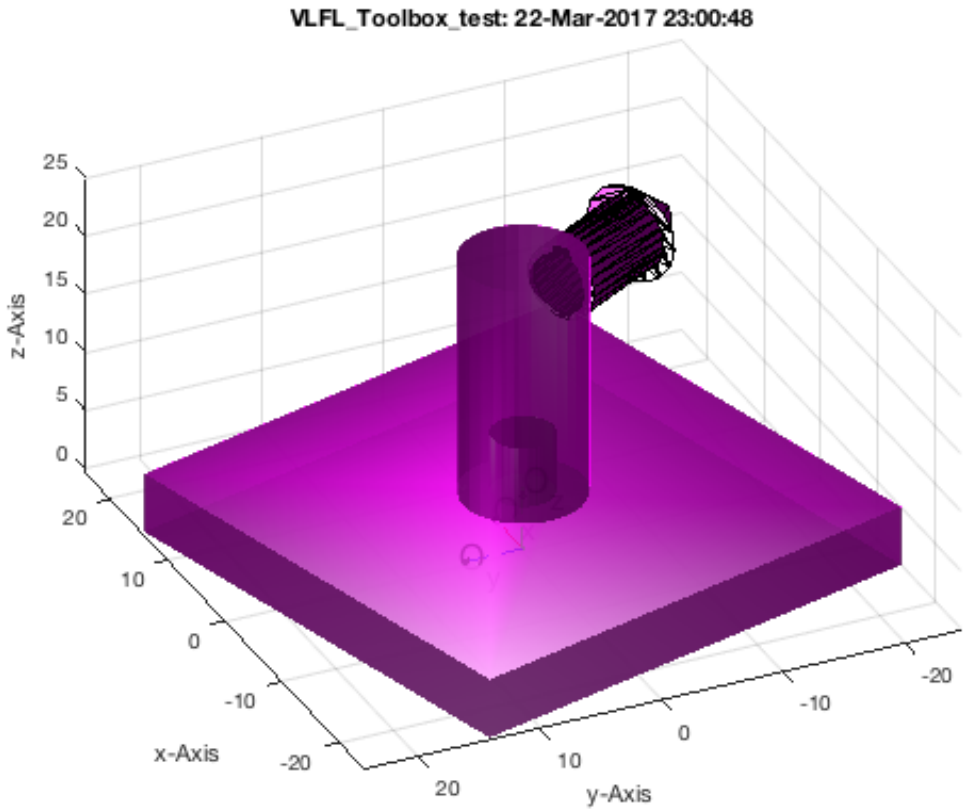
```
KMofSGs({SG0,SG1,SG4})
```

```
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. 3.6 (2017-Mar-23), is for limited non commercial educational use only!
Licensee: 3.6 Tim Lueth for academic and education 05-Mar-2017 !
Please contact Tim Lueth, Professor at TU Munich, Germany!
WARNING: This VLFL-Lib (Rel. 3.6) license will exceed at 18-Mar-2018!
Executed 22-Mar-2017 23:00:51 by 'lueth' using Matlab 9.1.0.441655 (R2016b) on a MACI64
======================================== Used Matlab products: ============================
==========
map_toolbox
matlab
simmechanics
simscape
simulink
================================================================================================
==========
```

- *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® R2016b