Substitutions and Replacements in Mechanism Prototyping

Cases studies with LinkageDesigner

Dr. Gábor Erdős

Computer and Automatisation Research Institute, Hungarian Academy of Sciences, H−1518 Budapest Kende u 13−17. Hungary
gabor.erdos@sztaki.hu

In parameterized mechanism design, there are two contradicting requirements; i.) keep the governing equation as general as possible (everything in symbolic form), but ii.) be able to quickly look at (or simulate) the mechanism and see how the it moves at every stage of the design process (which requires numeric representation) . Handling symbolic and numerical representation together is the base paradigm of symbolic manipulator programs like *Mathematica.* The solutions developed for this paradigm can be used for enhancing the flexibility of the mechanism prototyping application package written in *Mathematica.*
In this talk I will show some examples of mechanism modelling in *LinkageDesigner* application package, which utilizes this bundled symbolic and numeric representation. The bridge between the two representation is the substitution, that enables a one−way route going from symbolic to numeric representation. Two mechanism a historical parabola drawing mechanism and the spirograph will be considered as examples . Beside the powerful substitution, the replacement also widely utilized in *LinkageDesigner*. This simple tool can be handy in solving problems like inverse kinematic problem. Two examples, the inverse kinematic problem of a 6R robot and a 3DOF parallel mechanism will be considered here.

■ Introduction

Parameterized mechanism modelling, was always appealing to designers, because it can greatly simplify the design process. Mechanism design is an iterative process, since the specifications are continuously changing and the mechanism should adopt these changes. Parametrized mechanism models can support the evolution of the design, if the changes in the specification can be followed with small adaptation of the parameters. The theory of computer−aided mechanism modelling distinguishes two main mechanism modelling techniques the i.) augmented and the ii.) embedded methods. The first approach models the configuration of the mechanism by a vector of Cartesian coordinates, that describes the locations and orientations of the links relative to the reference frame. In the embedded approach generalized −or joint− coordinates are used to specify the postures of the links. Both modelling technique can handle parameterized mechanisms to some extent, however the selected modelling technique naturally influences the data model of the mechanism, the type of solver one has to employ and also type of problems, that can be effectively addressed.

In the augmented technique the topology of the mechanism is flattened, because the position and orientation of every links are maintained relative to the Global Reference Frame. The topology of every mechanism can be modelled with a simple tree having the ground link in the root and all the moving link connected to the root. Every link of the mechanism is defined with 6 coordinates (in case of planar mechanism with 3 coordinates). The physical constraints of the mechanism (joints) are modelled with a set of constraint equation. The different type of constraints are modelled with specific number of constraint equations (e.g hinge impose 5 constraint equation) see [1]. Redundant constraint equations are created if the mechanism contains loops. For example a loop closing rotational joint might impose [5,4,...0] independent constraint equations depending on the actual configuration of the mechanism. Because of the topology of the links are not taken into consideration, usually not every constraint equations are independent. This implies, that the mechanism can not be solved with an ordinary root finder algorithm (like FindRoot), but it requires a special solver. This special solver is a inherently numerical solver(see [2]) , therefore a parametrized mechanism – where the constraint equations are generated with the parameters– should be converted to numerical ones before one posture can be calculated.

The big disadvantage of the augmented method, that it requires a special solver even in case of kinematic modelling. However if one could eliminate the redundant constraint equations, before solving them, a simple solver (like FindRoot) could be employed too. One natural solution would be to let the user resolve the problem. This way the independent constraint equations can be further processed, even solved in closed form if there is a solution, therefore it is better suited for parametrized mechanism modelling.

LinkageDesigner is using the embedded notation.In this method the topology matches the kinematic graph of the mechanism. The kinematic graph is an undirected graph where the nodes represent the links and the edges of the graph stands for the joints between the links. To describe the configuration of the mechanism the relative transformation between the connected links are attached to the edge of the kinematic graph. The independent variables of these transformation are the *driving variables* of the mechanism. If the kinematic graph has a loop, the loop closing kinematic pair is treated differently. Instead of the relative transformation, the set of independent constraint equations is attached to the edge of the graph. Because the embedded notation works always with minimal number of constraint equations and does not requires special solver, it can support effectively the design process even in case of complicated multi–looped mechanism.

The parameterized mechanism allows the designer to handle a family of mechanism together. By substituting the parameters with numerical values, a specific instance of the family is obtained. In the following sections a couple of examples modelled in LinkageDesigner application package will be presented, where one can see that building a bridge between the generic and specific representation of a family of mechanism can be implement with such a simple approach like substitution and replacements.

Initialization

■ Parabola drawing mechanism

Even the ancient greek mathematicians and engineers invented different kind of curve drawing mechanism, that helped them to solve different design problems. All of these linkages incorporate one special feature of the curves, that became the working principle of the mechanism. The following parabola drawing mechanism, described in [3] is most probably one of the latest invention in this field, since after the appearance of the personal computers, these mechanism were not used anymore. The mechanism based on the following corollary.

Corollary 1

Given *g* line perpendicular to the axis (*t*) of the parabola (see on Figure 1). If the distance of *g* from the nose point of the parabola is $OB_0 = 2p$, than for an arbitrary P point of the parabola, the angle of
$POB\angle = 90°$, where *B* is the intersection of *e* and *g* lines. *e* line is paralell with *t* and *p* is the distance of the focus and the directrix of the parabola.

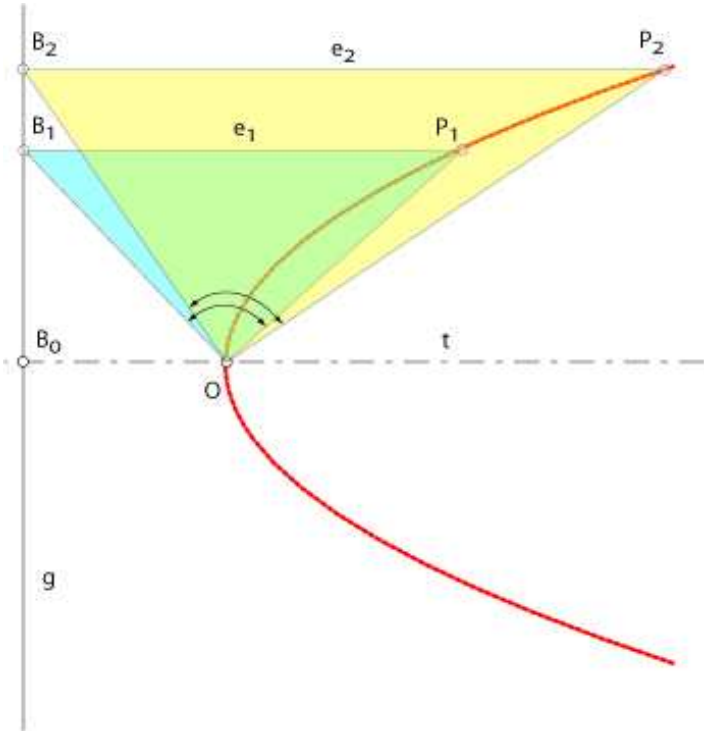


Figure 0 Parabola construction

This corollary implies the working principle of the parabola drawing mechanism, namely if a strait angled triangle is rotated in the nosepoint of the parabola, the third side of the triangle touches the points of the parabola. The mechanism naturally should be designed in such a way, that the BP side of the triangle (see in Figure 1) should move up and down along the *g* line and also its size should allow shrink and grown. The designed mechanism shown on Figure 2.

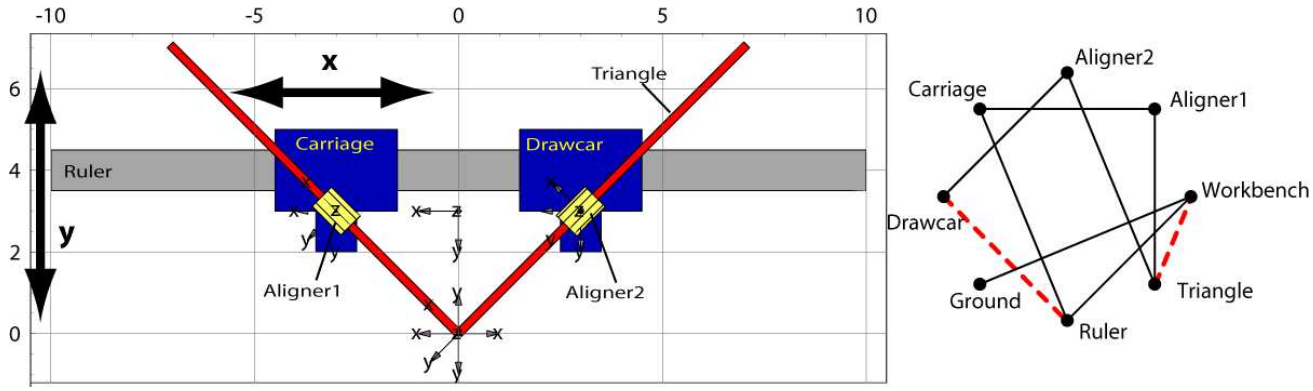


Figure 0 Parabola drawing mechanism

The mechanism definition is not presented here, only the pre–defined linkage is used. As it was already mentioned in the introduction, LinkageDesigner employs the embedded mechanism modelling method. This implies, that constraint equations are generated only for loop closing kinematic pairs. To investigate the equations load the mechanism and the LinkageData of the parabola drawing mechanism.

Load the LinkageData of the parabola drawing mechanism

```
In[10]:= Get["parabola.ld"]

Out[10]= -LinkageData,8-
```

The defined mechanism contains two loops, displayed with red dotted line in Figure 2. The loop closing constraint equations constraints joint variables, that were independent before.The independent joint variables of the mechanism stored in the $DrivingVariables record of the LinkageData, while the implicitly defined (or constrained) joint variables are stored in the $DerivedParametersB record. Both records stores the variable names and its actual substitution value. $DerivedParametersB record also stores the constraint equation to be solved.

List the generated constraint equations of the mechanism

```
In[11]:= TableForm[Rationalize[
           Simplify[parabola[["$DerivedParametersB"]][[All, {2, 3}]]]]]

Out[11]//TableForm=
q1 → -0.785398      x + q2 Cos[q1] == 0
q2 → -4.24264       q2 Sin[q1] == y
q4 → 4.24264        q4 Cos[q1] == q2 Sin[q1]
q5 → 0.785398       Cos[q1] Cos[q5] == 1 + Sin[q1] Sin[q5]
```

List the driving variables of the mechanism

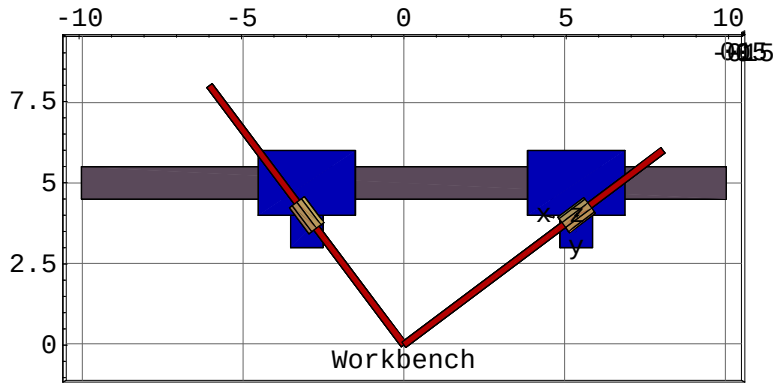
```
In[12]:= parabola[["$DrivingVariables"]] // Rationalize

Out[12]= {y → 3, x → 3}
```

The parametric representation of the constraint equation makes it simple to calculate the new posture of the mechanism. If a new substitution values of the independent parameters (in this case *x* and *y* parameters) is assigned, they are substitued into the constraint equations, which become definite and they are solved for the dependent variables. This iterative substitution–solving process is implemented in the AnimateLinkage function, that generates the animation of the mechanism if its independent variables are interpolated between the specified limit values.

The mechanism draw a parabola if the *x* driving variable (that is corresponding to the translational position of the *Carriage*) is set to a fixed value and y driving variable

interpolated in an interval. *y* driving variable corresponds to the translational position of the *Ruler* (see on Figure 2).

```
In[13]:= AnimateLinkage[parabola, {{y -> 4.0, x → 3.0}, {y → 0}},
  Resolution → 10, MaxIterations -> 500, AccuracyGoal -> 8,
  LinkMarkers → {"Drawcar"}, MarkerSize -> 1,
  TracePoints → {{"Drawcar", {0, 0, 1}}}, Axes → True,
  FaceGrids → {{0, 0, -1}}, ViewPoint → {0, 0 the , 10}];
```



■ Spirograph

The spirograph is a very simple mechanism consisting of two rigid body, that are connected with a rolling constraint. Rolling constraint is a higher order constraint, because the general constraint definition requires geometric data of the kinematic pair(rolling curve definitions), unlike the lower order constraints (like hinge or translational joints) , where a marker is fully defines the joint. However if the rolling curves are circle, the higher order joint could be easily substituted with two rotational joint, because the rolling constraints between circle are very simple.

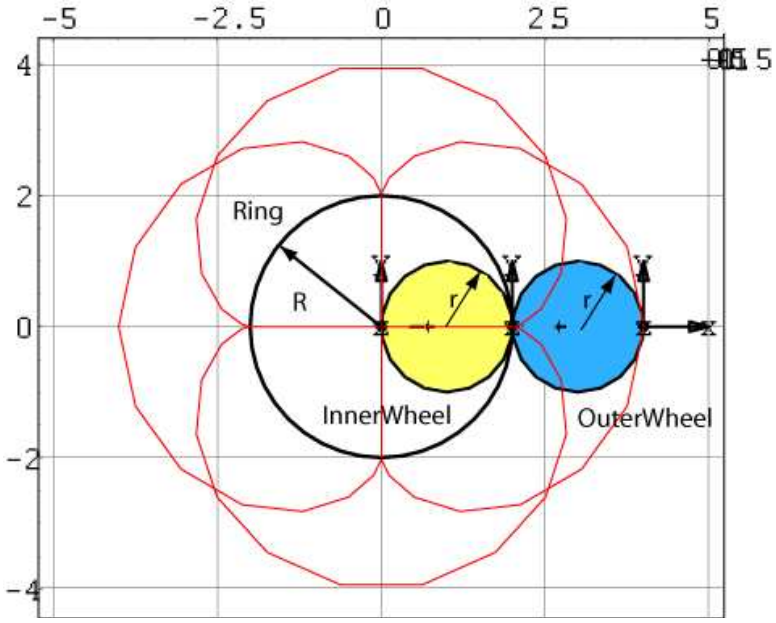


Figure 0 Spirograph mechanism

In this example a spirograph having an inner and outer shunweel is about to built. Both wheels have the same radius and rolling on the same ring. The radius of the ring is denoted with R, while the radius of the wheels are denoted with r. To substitute the rolling constraint a virtual body is also introduced (called *Arm*), that is responsible to rotate the center of the wheels on a circle. This way the rolling constraint is substituted with two rotational joints , the first rotates *Arm* around the origin of the *Ring* , while the second rotates the wheels attached to the arms at R−r and R+r distance from the origin. The joint variables representing this two rotational joints are not independent, because the rolling constraint makes them dependent on each other. This dependency is incorporated in the mechanism during the mechanism definition. Because of the definition of this mechanism is very short I will included it below.

Create the LinkageData of the mechanism with two geometrical parameters

```
In[101]:= spiro = CreateLinkage["spiro",
            WorkbenchName -> "Ring", SimpleParameters -> {R → 10, r → 5}];
```

Define the rotational joint between Ring and the virtual link called Arm

```
In[102]:= DefineKinematicPairTo[spiro, "Rotational",
            {q}, {"Ring", MakeHomogenousMatrix[{0, 0, 0}]},
            {"Arm", MakeHomogenousMatrix[{0, 0, 0}]}];
```

Define the rotational joint between Arm and the OuterWheel

```
In[103]:= DefineKinematicPairTo[spiro, "Rotational",
            {θout}, {"Arm", MakeHomogenousMatrix[{R + r, 0, 0}]},
            {"OuterWheel", MakeHomogenousMatrix[{0, 0, 0}]}];
```

Define the rotational joint between Arm and the InnerWheel

```
In[104]:= DefineKinematicPairTo[spiro, "Rotational",
            {θin}, {"Arm", MakeHomogenousMatrix[{R - r, 0, 0}]},
            {"InnerWheel", MakeHomogenousMatrix[{0, 0, 0}]}];
```

The mechanism is so far 3 DOF mechanism having 3 independent joint variables.The rolling constraint can be incorporated in such a way that constraint *θout* and *θin* rotational joint variables to be dependend on the radius of the wheel and the ring and the joint variables of the hinge joint between *Ring* and *Arm* (*q*). LinkageDesigner provides a function called ReplaceDrivingVariables, that allows such a transactions.

`ReplaceDrivingVariables[linkage,new,old,opts]` moves the *old* driving variables into *$DerivedParametersA* record and adds the *new* driving variables to the *$DrivingVariables* record of linkage.

Introduce the rolling constraint of the mechanism

```
In[105]:= spiro = ReplaceDrivingVariables[spiro,
            {θ → 0}, {q → θ, θout → R / r * θ, θin → -R / r * θ}]
```

```
ReplaceDrivingVariables::dofchg :
 Warning! The number of driving variables are changed from 3
   to 1! This might cause error in the D.O.F. calculations!
```

```
Out[105]= -LinkageData, 7-
```

Attach geometry to the Ring, InnerWheel and OuterWheel links

```
In[117]:= spiro[["$LinkGeometry", "Ring"]] = Graphics3D[Cylinder[R, 1]];
          spiro[["$LinkGeometry", "OuterWheel"]] =
            Graphics3D[{SurfaceColor[Blue], LinkShape[0, r, r, 0.1]}];
          spiro[["$LinkGeometry", "InnerWheel"]] =
            Graphics3D[{SurfaceColor[Yellow], LinkShape[0, r, r, 0.1]}];
```

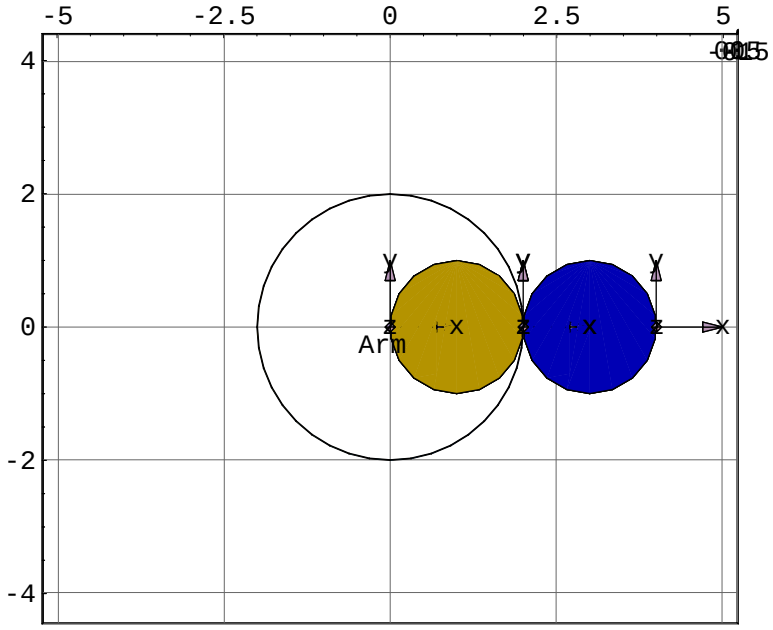
The spirograph mechanism is fully defined having 1 DOF , which is represented in the θ driving variable. θout and θin variables became explicitly derived parameters as a result of the ReplaceDrivingVariables function. The spirograph mechanism describes a family of similar mechanism differing only in the substitution value of the geometric parameters. To select one mechanism from the family set a new numerical value to the parameters, that will be substituted in numerical calculations.

Set the radius of the Ring and the wheels to {R –> 2, r –> 1}

In[113]:=
```
SetSimpleParametersTo[spiro, {R -> 2, r -> 1},
  MaxIterations -> 150, AccuracyGoal -> 8];
```

Animate the linkage

In[120]:=
```
AnimateLinkage[spiro,
  {{θ -> 0}, {θ -> 2 π}}, Resolution → 30, LinkMarkers ->
   {{"InnerWheel", MakeHomogenousMatrix[{r, 0, 0}],
     MakeHomogenousMatrix[{-r, 0, 0}]},
    {"OuterWheel", MakeHomogenousMatrix[{-r, 0, 0}],
     MakeHomogenousMatrix[{r, 0, 0}]}},
  MarkerSize → 1,
  TracePoints → {{"InnerWheel", {r, 0, 1}, {-r, 0, 1}},
    {"OuterWheel", {-r, 0, 1}, {r, 0, 1}}},
  TraceStyle → {Thickness[0.01], Red}, ViewPoint → {0, 0, 50},
  Axes → True, FaceGrids → {{0, 0, -1}}];
```



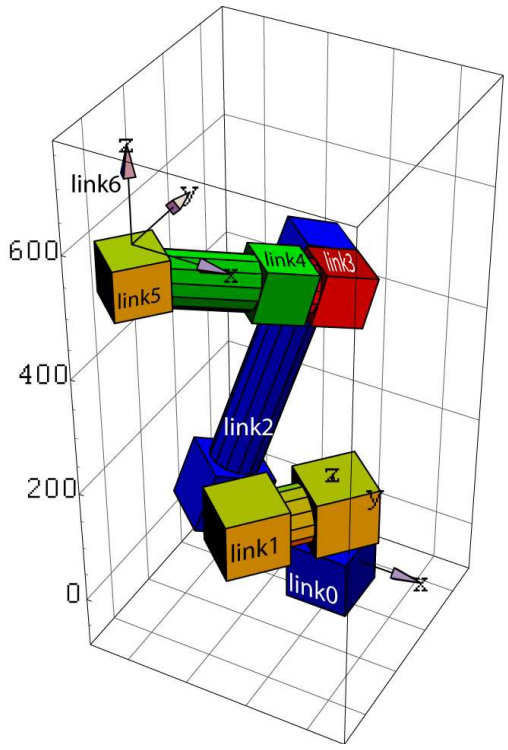
■ Inverse kinematics with replacements

The solution of the inverse kinematics problem (IKP) is one of the most challenging problem in manipulator design. The problem is formulated as follows: Given the desired position and orientation of the tool relative to the reference coordinate frame, calculate the set of joint angles, that moves the tool into this posture. There are numerous solution technique has been developed ranging from the numerical solution to the closed form solutions. For a summary of the existing techniques you can consult any standard textbook on robotics like ([4], [5]). In this section a shortcut solution will be presented, that enables the designer to quickly solve the inverse kinematic equation. This solution based on simple replacements of the driving variables and utilize the same ReplaceDrivingVariables function, that was employed in the previous section.

□ Serial manipulator

The 6R manipulator shown on Figure 4 is generated with the presented DH parameters. Since the manipulator is open chain mechanism, therefore the embedded method does not generate constraint equations. The manipulator is 6DOF mechanism, of which DOF represented by the $\theta 1$,$\theta 2$... driving variables. In case of the Inverse problem one would like to "drive" the mechanism by commanding the tool w.r.t Cartesian reference frame. This gives the idea, that if the position and orientation of the tool are parameterized, these parameters could become the new driving values of the mechanism and the old one should be constrained with equations containing the new driving variables. This way setting the substitution value of the new driving values, than solving the constraint equations of the inverse kinematic problem would result in the substitutional values of the joint variables, which in turn substituted into the homogenous transformation and the new posture of the mechanism is calculated.

To follow this process the mechanism definition is not presented her, the pre−defined mechanism will be loaded and only the replacement procedure is discussed in details below.



	a	d	A	Θ
1	a1	d1	90 °	Θ1
2	a2	d2	180 °	Θ2
3	0	d3	90 °	Θ3
4	0	d4	90 °	Θ4
5	0	0	90 °	Θ5
6	0	d6	0	Θ6

Figure 0 6R manipulator

Load the LinkageData of the 6R manipulator

```
In[14]:= Get["manipulator.ld"];
```

In order to solve the Inverse kinematic problem, first the equation should be defined. In this example the Local Reference Frame of link6 (see on Figure 4) is considered as the tool frame. This selection is arbitrary, but the method would work for any other tool frame too. The new driving variables are the position vector of the Origin and the Euler angles of the orientation of this frame, denoted by $\{x,y,z,\theta,\phi,\psi\}$. To generate the constraint equation the Homogenous Transformation matrix of *link6* is calculated *w.r.t* World Reference Frame (Ground).

Retrieve the transformation matrix of $LLRF_6$

```
In[15]:= mx = GetLLRFMatrix[manipulator, "link6",
            ReferenceFrame -> "Ground", SubstituteParameters → False]
```

Extract the position vector of the origin from mx and make it equal to {x,z,y} vector

```
In[16]:= eq1 = MapThread[Equal, {Drop[mx.{0, 0, 0, 1}, -1], {x, y, z}}]
```

```
Out[16]= {a1 Cos[θ1] + a2 Cos[θ1] Cos[θ2] + d2 Sin[θ1] - d3 Sin[θ1] +
     d4 (-Cos[θ1] Cos[θ3] Sin[θ2] + Cos[θ1] Cos[θ2] Sin[θ3]) + d6
      (Cos[θ5] (-Cos[θ1] Cos[θ3] Sin[θ2] + Cos[θ1] Cos[θ2] Sin[θ3]) -
        (Cos[θ4] (Cos[θ1] Cos[θ2] Cos[θ3] + Cos[θ1] Sin[θ2]
              Sin[θ3]) - Sin[θ1] Sin[θ4]) Sin[θ5]) == x,
   -d2 Cos[θ1] + d3 Cos[θ1] + a1 Sin[θ1] + a2 Cos[θ2] Sin[θ1] +
     d4 (-Cos[θ3] Sin[θ1] Sin[θ2] + Cos[θ2] Sin[θ1] Sin[θ3]) + d6
      (Cos[θ5] (-Cos[θ3] Sin[θ1] Sin[θ2] + Cos[θ2] Sin[θ1] Sin[θ3]) -
        (Cos[θ4] (Cos[θ2] Cos[θ3] Sin[θ1] + Sin[θ1] Sin[θ2]
              Sin[θ3]) + Cos[θ1] Sin[θ4]) Sin[θ5]) == y,
   d1 + a2 Sin[θ2] + d4 (Cos[θ2] Cos[θ3] + Sin[θ2] Sin[θ3]) +
     d6 (Cos[θ5] (Cos[θ2] Cos[θ3] + Sin[θ2] Sin[θ3]) -
        Cos[θ4] (Cos[θ3] Sin[θ2] - Cos[θ2] Sin[θ3]) Sin[θ5]) == z}
```

The orientation of a frame is coded in the 3x3 rotation matrix part of the homogenous matrix. The rotation matrix is an orthonormal 3x3 matrix that can be represented with 3 independent parameters. In the literature there are many rotation matrix representation like Euler angles , Euler parameters, Rodriguez parameters, Roll−Yaw−Pitch,.... . These are equivalent representation , therefore we could pick any of them to use as driving variables of the inverse kinematics problem. From technical point of view, the Rodriguez parameters are the easiest to calculate from a given rotation matrix. Therefore the generation of the constraint equation for the orientation of the tool is done in three steps:

1. Select the rotation matrix representation (e.g. Euler angles) and define the rotation matrix using the parameters of the selected representation. This matrix is called **rotmx1.**

2. Extract the rotation matrix from the tool matrix. This matrix is called **rotmx2**

3. Extract the Rodriguez parameters from **rotmx1** and **rotmx2** and make them equal.

Define the ExtractRodriguezParameters function

```
In[17]:= ExtractRodriguezParameters[Amx_?MatrixQ] :=
          Module[{Vmx, ret},
           Vmx = (Amx - Transpose[Amx]);
           ret = {Vmx[[3, 2]], Vmx[[1, 3]], Vmx[[2, 1]]};
           Return[ret]
          ]
```

Define the rotation matrix with the new driving variables $\{\phi, \theta, \psi\}$ based on the Euler angles parametrization

```
In[18]:= (rotmx1 = RotationMatrix[{0, 0, 1}, ϕ].
             RotationMatrix[{1, 0, 0}, ϑ].RotationMatrix[{0, 0, 1}, ψ])

Out[18]= {{Cos[ϕ] Cos[ψ] - Cos[ϑ] Sin[ϕ] Sin[ψ],
           -Cos[ϑ] Cos[ψ] Sin[ϕ] - Cos[ϕ] Sin[ψ], Sin[ϑ] Sin[ϕ]},
          {Cos[ψ] Sin[ϕ] + Cos[ϑ] Cos[ϕ] Sin[ψ],
           Cos[ϑ] Cos[ϕ] Cos[ψ] - Sin[ϕ] Sin[ψ], -Cos[ϕ] Sin[ϑ]},
          {Sin[ϑ] Sin[ψ], Cos[ψ] Sin[ϑ], Cos[ϑ]}}
```

Extract the rotation matrix from the $LLRF_6$ homogenous matrix

```
In[19]:= rotmx2 = ExtractRotationMatrix[mx]
```

Create the constraint equation for the orientation of the tool

```
In[20]:= eq2 = Thread[ExtractRodriguezParameters[rotmx1] ==
             ExtractRodriguezParameters[rotmx2]]
```

Replace the old driving variables of the manipulator with the parameter specializing the position and orientation of the tool

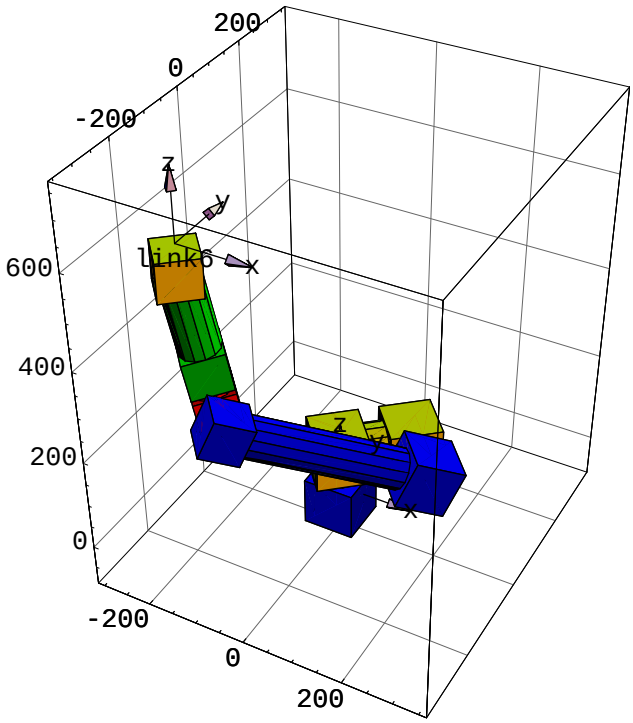
```
In[21]:= manipINV = ReplaceDrivingVariables[manipulator,
             {x → 135, y → -20, z → 800 , ϑ → 0, ϕ → 0, ψ → 0},
             {ϑ1 → 0, ϑ2 → -90 °, ϑ3 → -1.57, ϑ4 → 0.0, ϑ5 → 0.0, ϑ6 → 0.0},
             EquationList → Join[eq1, eq2]];

         SetDrivingVariablesTo[manipINV,
          {x -> 135, y -> 200, z -> 500, ϑ -> 0, ϕ -> 0, ψ -> 0},
          MaxIterations -> 150, AccuracyGoal -> 8]
```

Generate a list of substitution of the manipulator's parameter, as the IKP parameters interpolated along a path

```
In[23]:= sub = GetLinkageRules[manipINV, {
            {ϕ -> 0°, θ → 0°, ψ -> 0, x → - 200, y -> -200, z -> 600},
            {ϕ -> 0°, θ → 90°,
             ψ -> 0, x → - 200, y -> -200, z -> 600},
            {ϕ -> 0°, θ → 90°, ψ -> 0, x -> -200,
             y -> 200, z -> 600},
            {ϕ -> 90°, θ → 90°, ψ -> 0, x -> -200,
             y -> 200, z -> 600},
            {ϕ -> 90°, θ → 90°, ψ -> 0, x → 200, y -> 200, z -> 600},
            {ϕ -> 90°, θ → 90°,
             ψ -> 90°, x → 200, y -> 200, z -> 600},
            {ϕ -> 90°, θ → 90°, ψ → 90°, x → 200,
             y -> -200, z -> 600},
            {ϕ -> 90°, θ → 90°, ψ → 90°, x → - 200,
             y -> -200, z -> 600}},
           Resolution → 50, MaxIterations -> 1500,
           SubstituteParameters -> True];
```

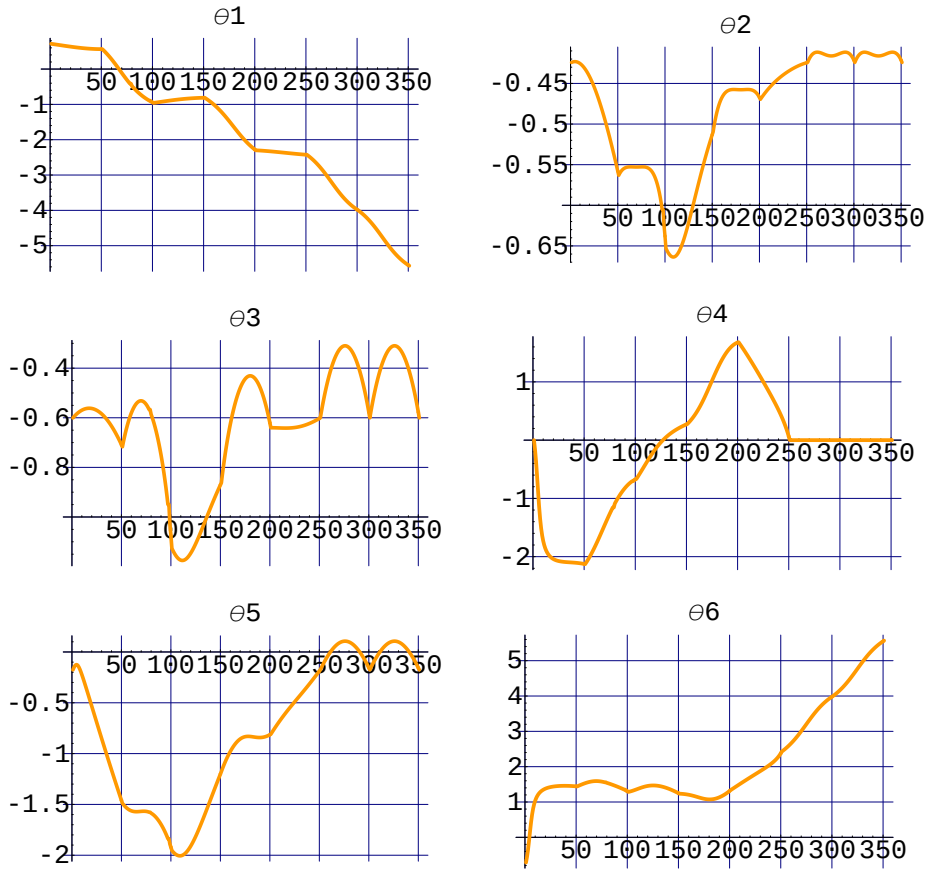
Plug in the calculated joint variables of the original(direct manipulator) to visually check the result of the calculation (to minimize the size of the notebook only every 10th interpolation point is used in the animation)

```
In[24]:= AnimateLinkage[manipulator, Partition[sub, 10][[All, 1]],
           Resolution → None, LinkMarkers → {"Ground", "link6"},
           MarkerSize → 150, TracePoints → {"link6"},
           FaceGrids → {{0, 1, 0}, {-1, 0, 0}, {0, 0, -1}}, Axes → True];
```



Plot the axis interpolation as the tool follows the prescribed path

```
In[25]:= Show[GraphicsArray[With[{
      ls = ListPlot[# /. sub,
              PlotJoined → True,
              GridLines → Automatic,
              PlotStyle → {Thickness[0.01], Hue[0.1]},
              PlotLabel → #,
              DisplayFunction → Identity] & /@
        {θ1, θ2, θ3, θ4, θ5, θ6}}, Partition[ls, 2]
  ]
 ]]
```




```
Out[25]= - GraphicsArray -
```

□ Parallel manipulator

Unlike the serial manipulators the parallel manipulators are containing loops in their kinematic graph, therefore during the modelling phase constraint equations are created. The process presented for the serial manipulator, to calculate quickly the IKP, could be applied in this case exactly in the same way. The following manipulator is patented by NASA (U.S.Patent No.5,816,105). The mechanism is a 3DOF and having 3 loops in the kinematic graph (sse on Figure 5).

From In[2]:=

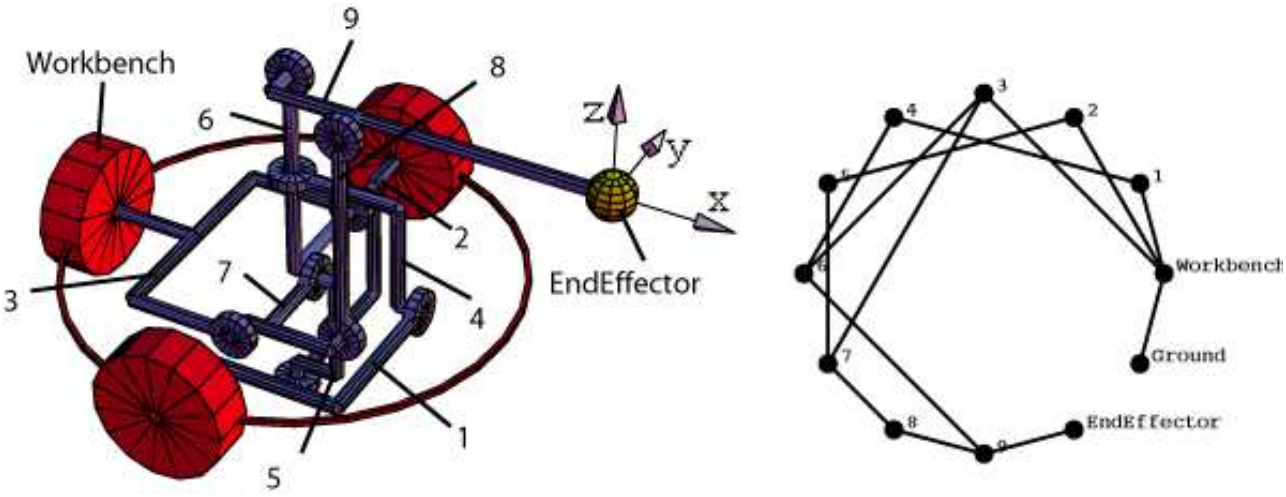


Figure 0 NASA Parallel manipulator

The mechanism definition is not presented her, the pre−defined mechanism will be loaded and only the replacement procedure is discussed in details below

Load the pre−defined mechanism

In[30]:=
```
Get["nasa.ld"];
```

List the defined constraint equations

In[46]:=
```
TableForm[Rationalize[
  Simplify[nasa[["$DerivedParametersB"]][[All, {2, 3}]]]]]
```

Out[46]//TableForm=

q1 → −1.57254	Cos[A] Cos[q1] == Cos[C] Sin[A] Sin[q1]
q3 → −701766.	Cos[q1] Cos[q3] Sin[A] + Cos[A] Cos[C] Cos[q3] Sin[q1] + Sin[C]
q2 → 12.5664	Cos[A] Cos[B] Cos[q4] == Sin[B] Sin[q4]
q4 → 1.56905	Cos[q2] (Cos[A] Cos[q4] Sin[B] + Cos[B] Sin[q4]) == 1 + Cos[q4] S
q5 → 669528.	p61 Cos[q6] + p72 Cos[q3 − q4 + q6] == p61 Cos[q3 − q4 + q5 + q6]
q6 → −398590.	p72 + p61 Sin[q6] + p72 Sin[q3 − q4 + q6] == p61 Sin[q3 − q4 + q5 + q6

List the simple parameters (they are the geometric parameters of the mechanism)

In[32]:=
```
nasa[["$SimpleParameters"]]
```

Out[32]=
```
{r → 10, p11 → 3, p12 → 6, p31 → 5, p32 → 5, p41 → 1, p42 → 5,
 p51 → 1, p52 → 1, p61 → 10, p71 → 1, p72 → 4, p73 → 1, p91 → 15}
```

The inverse kinematic problem in this case takes only the position of the end−effector as input, because the mechanism is 3DOF therefore can not specify the position and the orientation together. The output of the inverse kinematic problem is the values of the independent joint variables, which is the rotational joint defined on the 3 motor of the Workbench link (see on Figure 5). To define the inverse kinematic problem, the homogenous transformation matrix of the tool marker should be calculated.

Get the homogenous transformation matrix of the EndEffector

In[36]:=
```
mx = GetLLRFMatrix[nasa, "EndEffector",
  ReferenceFrame → "Ground", SubstituteParameters → False]
```

Unlike the serial manipulator, the parameters presented in the transformation matrix are not all independent, because some of them are already constrained by the loop closing constraint equations (see the listing $DerivedParametersB record above). Fortunately this doe not cause any problem, because the ReplaceDrivingVariables function appends the constraint equations of the IKP to the $DerivedParametersB record. In case of the independent parameters (stored in the $DrivingVariables and $SimpleParameters

records) are changing the solver lump together all constraint equations and solves them together. This way all constraints imposed either by loop closing or IKP are satisfied.

Calculate the constraint equation of the IKP

```
In[37]:= eq = Thread[ExtractTranslationVector[mx] == {X, Y, Z}]
```

$$\text{Out[37]=}\ \left\{-p61\,\text{Sin}[q3] + p91\,(\text{Cos}[q3]\,\text{Cos}[q6] - \text{Sin}[q3]\,\text{Sin}[q6]) == X,\ \frac{1}{2}\,(-p32 + p71 + p73)\,\text{Cos}[A] + p61\,\text{Cos}[q3]\,\text{Sin}[A] + p91\,(\text{Cos}[q6]\,\text{Sin}[A]\,\text{Sin}[q3] + \text{Cos}[q3]\,\text{Sin}[A]\,\text{Sin}[q6]) == Y,\ -p61\,\text{Cos}[A]\,\text{Cos}[q3] + \frac{1}{2}\,(-p32 + p71 + p73)\,\text{Sin}[A] + p91\,(-\text{Cos}[A]\,\text{Cos}[q6]\,\text{Sin}[q3] - \text{Cos}[A]\,\text{Cos}[q3]\,\text{Sin}[q6]) == Z\right\}$$

Replace the driving variables with the parameters of the IKP ({X,Y,Z})

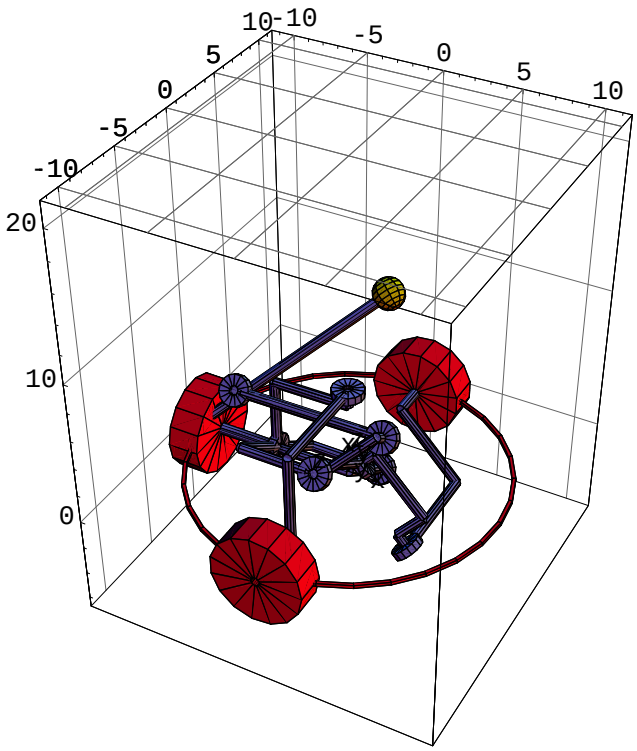
```
In[38]:= nasaINV =
           ReplaceDrivingVariables[linkage, {X → 10, Y → -1.5, Z → 10},
            {C → 0.0017, B → 0.0017, A → 0.0017}, EquationList → eq];
         SetDrivingVariablesTo[nasaINV, {X → 10, Y → -1.5, Z → 10},
           MaxIterations -> 150, AccuracyGoal -> 8];
```

Generate a list of substitution of the mechanism's parameter, as the IKP parameters interpolated along a path

```
In[40]:= sub = GetLinkageRules[nasaINV,
           {{X → 0, Y → 5, Z → 10}, {X → 10}, {Y → -5}, {X → 0}, {Y → 5},
            {Z → 20}, {X → 10}, {Y → -5}, {X → 0}, {Z → 10}, {X → 10, Y → 5}},
           Resolution → 20, MaxIterations → 1500,
           SubstituteParameters -> True];
```

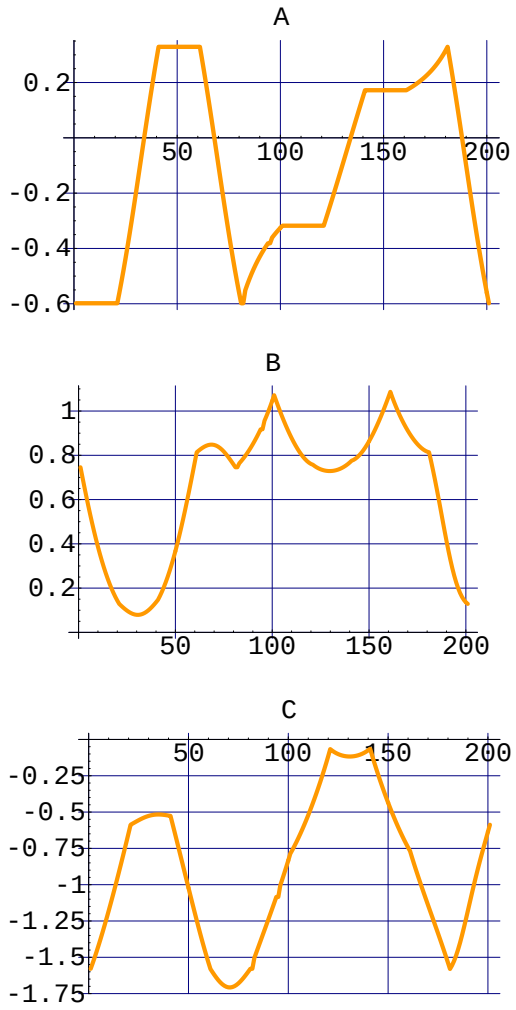
Plug in the calculated joint variables of the original(direct mechanism) to visually check the result of the calculation (to minimize the size of the notebook only every 10th interpolation point is used in the animation)

```
In[41]:= AnimateLinkage[linkage, Partition[sub, 10][[All, 1]],
           Resolution → None, LinkMarkers → {"1", "3"}, MarkerSize → 2,
           Boxed → True, FaceGrids → {{0, 0, 1}, {0, 1, 0}, {-1, 0, 0}},
           Axes → True, TracePoints → {{"EndEffector", {0, 0, 0}}}];
```



Plot the axis interpolation as the EndEffector follows the prescribed path

In[47]:=
```
Show[GraphicsArray[With[{
    ls = ListPlot[# /. sub,
            PlotJoined → True,
            GridLines → Automatic,
            PlotStyle → {Thickness[0.01], Hue[0.1]},
            PlotLabel → #,
            DisplayFunction → Identity] & /@ {A, B, C}},
  Partition[ls, 1]
 ]
]]
```



Out[47]= - GraphicsArray -

■ Conclusion

The embedded method working with a minimal set of constarint equations, that are automatically generated. It represents the mechanism in a graph, that enables mesaure the relative transformation of two arbitray points or frame of the mechanism. This two feature makes this mehod very attractive to use for mechanism prototyping, because any design equation can be easily generated in a parameterized form, that could be further processed in *Mathematica* to arrive the optimized substitution values of the parameters. Once the design is defined and optimized, the mathematical modell of the resulted mechanism is defined as a set of parameterized transformations and constraints, and a list of substitution values of the parameters.

■ Acknowledgement

This research was partially founded by the VRL−KCIP (FP6−507487−2) european project. This support is gratefully acknowledged.

■ References

[1] Haug,Edward J. *Computer Aided Kinematics and Dynamics of Mechanical Systems*, Allyn and Bacon, 1989.

[2] Edward J. Haug and R. A. Wehage. *Generalized coordinate partitioning fordimension reduction in analysis of constrained dynamic systems*. Journal of Mechanical Design, 104:247-255, 1982.

[3] Bela Szoke et al. *Levelvaltas egy parabolarajzolo elvet tartalmazo szerkesztesrol*. KGM Szerszamgepipari muvek kozlemenyek, XVII/2 p21−46, 1977.

[4] John J. Craig. *Introduction to robotics: mechanics and control*, Addison−Wesley, 2nd edition, 1989.

[5] R.P.Paul. *Robot Manipulators: Mathematics, Programming and Controll*, the MIT press, Cambridge, MA, 1981.