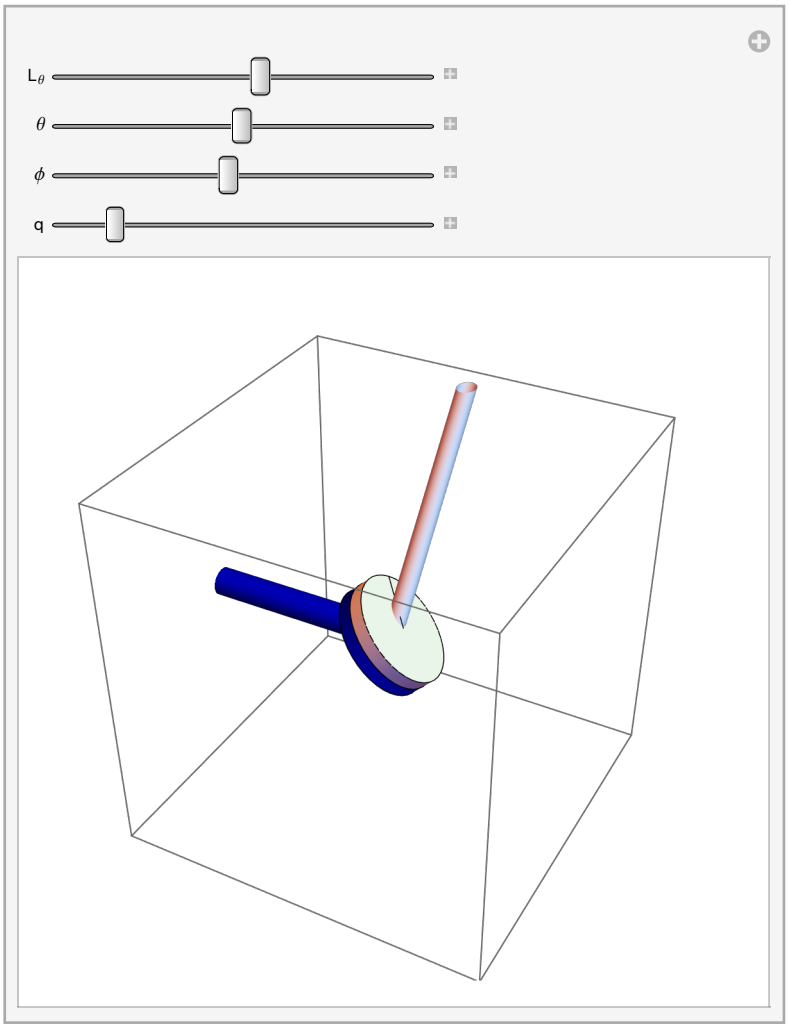
- Click Enable Dynamics in the top bar if it appears (or enable dynamic updating manually under Evaluation). Click Evaluation → Evaluate Initialization Cells to load the notebook's content into the kernel.
- In the following we demonstrate what angles are necessary to parameterize a fully general robotic joint.
  - $\theta$ describes the degree the "joint plane" is rotated back from the perpendicular towards the incoming arm segment.
  - $\phi$ describes the rotation *around* the incoming joint which we bend the rotation plane back along.
  - L$\theta$ describes the angle through which the robot arm is bent down vertically towards the joint plane.
  - q is a dynamic parameter describing the current rotation of the joint.
  - (In later code, we have a parameter $\psi$ which describes the azimuthal angle around the joint that the outgoing arm segment is bent down towards, but we ignore that here as it's nearly redundant with q.)
  - Note that we can achieve an "elbow joint" by taking L$\theta = \pi/2$ and $\theta = \pi/2$, and a "shoulder joint" by taking L$\theta = \pi/2$ and $\theta = 0$.

In[289]:=
```
Manipulate[
 Module[{t = .15},
  Graphics3D[{
    {Blue, Tube[{{-2, 0, 0}, {0, 0, 0}}, t / 2]},
    GeometricTransformation[{
      {Blue, Cylinder[{{0, 0, 0}, {-t / 2, 0, 0}}, 0.3]},
      {EdgeForm[Dashed], Cylinder[{{0, 0, 0}, {t / 2, 0, 0}}, 0.3]},
      {Line[{{t / 2, 0, 0}, {t / 2, 0.3, 0}}]},
      {Tube[{{0, 0, 0}, {2 Cos[Lθ], 2 Sin[Lθ], 0}}, 0.05]}
     },
     EulerMatrix[{ϕ, θ, -ϕ}, {1, 2, 1}].RotationMatrix[q, {1, 0, 0}]]
   },
   AxesOrigin → {0, 0, 0}, PlotRange → {{-1, -1, -1}, {1, 1, 1}}]],
 {{Lθ, 0.5, Subscript["L", "θ"]}, 0, Pi / 2},
 {{θ, 0.5}, 0, Pi / 2}, {{ϕ, Pi}, 0, 2 Pi}, {q, 0, 2 Pi}]
```

Out[289]=




- In general, we have several coordinate systems at play, named in bold.

- **in-joint:** we assume that each rotating joint comes with its own coordinate system in which the axis of rotation is along the z axis and the "initial" or "default" state is by assumption along the x axis. This is the coordinate system of the blue "base" in the above image, on top of which we imagine the joint rotating. There is ambiguity in where we place the z = 0 plane, though.
- **out-joint:** The next coordinate system is the current coordinate system of the rotated part of the joint; so, it's the same as the in-joint coordinate system, but rotated around the z-axis by some angle. These coincide when the joint is at its default position.
- We could specify the outgoing link in this system by two angles and a length, but let's just use a vector. (Note: the above animation essentially chooses a particular "default position", and therefore only includes one angle.) We assume this vector indicates the position of the end of the outgoing link when the joint is in its default position. Note, again, there is ambiguity in determining exactly where the link should end. This is analogous to the z = 0 ambiguity. How do we fix this?
- Note that while z = 0 in the next joint is ambiguous, what is not ambiguous is the line determined by the axis of rotation of the next joint. We could potentially consider the space of such lines generated by joint movement, but here we'll assume there's a privileged point of connection given.
- **link:** We then need an induced coordinate system on the end of the outgoing link. We construct this by considering the rotation in the plane spanned by the z-axis and the link vector which transforms the z axis into the link vector (up to positive multiple). (When there is no such plane (the z-axis and the link vector are collinear) this is 1 if their dot product is positive, and we rotate around the x axis otherwise. If the link vector is 0, we take the identity matrix.) This is a (nontrivially translating) affine transformation from the out-joint system.
- (We could have split this up into an in-link and out-link system, but that seemed redundant, and did not expose the structure of the robot.)
- So, let's construct the transformations in general.
- Function that expands matrices by a dimension to allow affine transformations:

In[221]:=
```
ClearAll[Af, vector, nvector]
```

In[222]:=
```
vector = {Except[_List], Except[_List], Except[_List]};
nvector = {_?NumericQ, _?NumericQ, _?NumericQ};
```

In[224]:=
```
Af[v : vector] := Append[v, 1]
```

In[225]:=
```
Af[M_?MatrixQ, v : vector : Null] := If[MatchQ[v, Null],
  ReplacePart[ArrayPad[M, {{0, 1}, {0, 1}}], {4, 4} → 1],
  ReplacePart[ArrayPad[M, {{0, 1}, {0, 1}}],
   MapIndexed[Append[#2, 4] → #1 &, Append[v, 1]]]]
```

In[226]:=
```
AfRotationInverse[AfM_?MatrixQ] :=
 Af[#, -#.(AfM[[1 ;; 3, 4]])] &@Transpose[AfM[[1 ;; 3, 1 ;; 3]]]
```

- The inverse operator works as expected:

In[227]:=
```
Block[{R, a, b, c, v}, R = Af[EulerMatrix[{a, b, c}], {v[1], v[2], v[3]}];
  Simplify[R.AfRotationInverse[R]]] // MatrixForm
```

Out[227]//MatrixForm=

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- **in-joint ← out-joint**

In[228]:=
```
ClearAll[oi, io]
```

In[229]:=
```
Block[{q}, io[q_] = Af@RotationMatrix[q, {0, 0, 1}];]
```

In[230]:=
```
Block[{q}, oi[q_] = AfRotationInverse[io[q]];]
```

- **out-joint ← link**
- Mathematica actually has built-in functionality to get the rotation matrix that rotates one vector into the direction of another. Though, for the record, we could build this up by taking ArcCos[L[[3]]/Norm[L]], and rotating by that much around L cross {0,0,1}. However, this only works if L is nonzero, so we need to add some defaults.

In[231]:=
```
ClearAll[Lo, oL, LinkRotate]
```

In[232]:=
```
LinkRotate[L : nvector] := Which[
  L[[1]] == 0 && L[[2]] == 0 && L[[3]] ≥ 0,
  IdentityMatrix[3],
  L[[1]] == 0 && L[[2]] == 0 && L[[3]] < 0,
  RotationMatrix[Pi, {1, 0, 0}],
  True,
  RotationMatrix[{{0, 0, 1}, L}]]
```

In[233]:=
```
LinkRotate[L : vector] :=
 RotationMatrix[{{0, 0, 1}, L}] /; ! MatchQ[L, nvector]
```

In[234]:=
```
oL[L : vector] := Af[LinkRotate[L], L]
```

In[235]:=
```
Lo[L : vector] := AfRotationInverse[oL[L]]
```

- The origin in the link system should correspond to the original link vector in the outjoint system (which is what L should be):

In[236]:=
```
(oL[{l1, l2, l3}].{0, 0, 0, 1})[[1 ;; 3]]
```

Out[236]=
```
{l1, l2, l3}
```

- Note that taking the last column is equivalent to acting on the origin:

In[237]:=
```
oL[{l1, l2, l3}][[1 ;; 3, -1]]
```

Out[237]=
```
{l1, l2, l3}
```

- Likewise, the origin in the outjoint system should be in the –z direction with length Norm[L]. (To get Mathematica to confirm this to us this algebraically, we need to make some assumptions.)

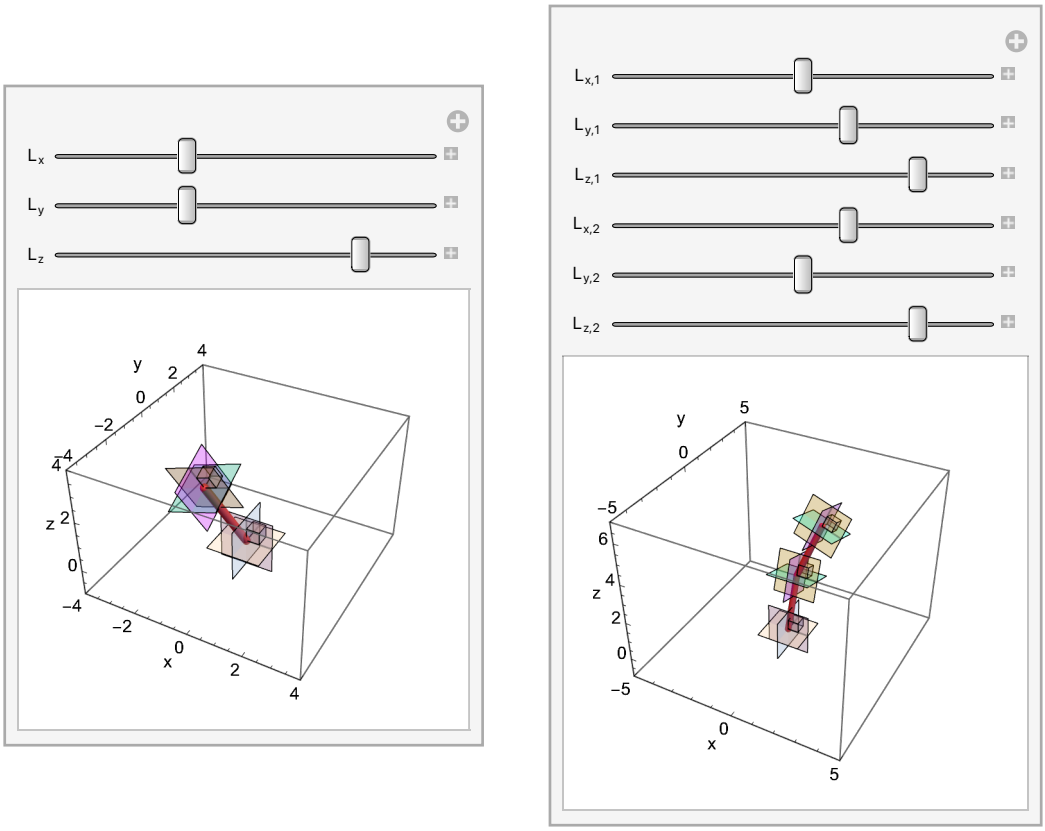
In[238]:=
```
FullSimplify[Lo[{l1, l2, l3}].{0, 0, 0, 1},
  Assumptions → {l1 > 0 && l2 > 0 && l3 > 0}][[1 ;; 3]]
```

Out[238]=
$$\left\{0,\ 0,\ -\sqrt{l1^2 + l2^2 + l3^2}\right\}$$

- To illustrate this (and to debug), I set up the following Manipulates to show how the induced system at the end of the link relates to the one at the start of the link (colorless).

In[304]:=
```
coordsystem[o_] :=
  {Opacity[o], {Cyan, Polygon[{{-1, -1, 0}, {1, -1, 0}, {1, 1, 0}, {-1, 1, 0}}]},
   {Yellow, Polygon[{{-1, 0, -1}, {1, 0, -1}, {1, 0, 1}, {-1, 0, 1}}]},
   {Magenta, Polygon[{{0, -1, -1}, {0, 1, -1}, {0, 1, 1}, {0, -1, 1}}]},
   {Thickness[0.005], Line[{{1, 0, 0}, {0, 0, 0}}],
    Line[{{0, 1, 0}, {0, 0, 0}}],
    Line[{{0, 0, 1}, {0, 0, 0}}]},
   {White, Cube[{0.25, 0.25, 0.25}, 0.5]}};
coordsystem0[o_] := coordsystem[o] /. (x : (Cyan | Magenta | Yellow) :→ White);
AfGeometricTransformation[g_, AfM_] :=
 GeometricTransformation[g, {AfM[[1 ;; 3, 1 ;; 3]], AfM[[1 ;; 3, 4]]}];
iHue[i_, n_] := Hue[i / n, 0.7, 1]
LinkDirectives[i_, n_] := {iHue[i, n]}
linkthickness = 0.15;
LinkModel[v0_, v1_, n_ : 1, i_ : 0, thicknessf_ : (linkthickness &)] :=
 Append[LinkDirectives[i, n], Tube[{v0, v1}, thicknessf[i, n]]];
Row[{Manipulate[Graphics3D[{coordsystem0[0.3], LinkModel[{0, 0, 0}, {lx, ly, lz}],
     AfGeometricTransformation[coordsystem[0.3], oL[{lx, ly, lz}]]},
    PlotRange → {{-4, 4}, {-4, 4}, {-1, 4}}, Axes → True, AxesLabel → {"x", "y", "z"},
    ImageSize → 200], {{lx, -1, Subscript["L", "x"]}, -3, 3},
   {{ly, -1, Subscript["L", "y"]}, -3, 3}, {{lz, 2.5, Subscript["L", "z"]}, 0, 3}],
  "    ",
  Manipulate[Graphics3D[{coordsystem0[0.3], LinkModel[{0, 0, 0}, {lx1, ly1, lz1}],
     LinkModel[(oL[{lx1, ly1, lz1}].{0, 0, 0, 1})[[1 ;; 3]],
      (oL[{lx1, ly1, lz1}].oL[{lx2, ly2, lz2}])[[1 ;; 3, 4]]],
     AfGeometricTransformation[coordsystem[0.3], oL[{lx1, ly1, lz1}]],
     AfGeometricTransformation[coordsystem[0.3],
      oL[{lx1, ly1, lz1}].oL[{lx2, ly2, lz2}]]},
    PlotRange → {{-5, 5}, {-5, 5}, {-1, 7}}, Axes → True,
    AxesLabel → {"x", "y", "z"}, ImageSize → 200],
   {{lx1, 0, Subscript["L", "x,1"]}, -3, 3},
   {{ly1, 0.8, Subscript["L", "y,1"]}, -3, 3},
   {{lz1, 2.5, Subscript["L", "z,1"]}, 0, 3},
   {{lx2, 0.8, Subscript["L", "x,2"]}, -3, 3},
   {{ly2, 0, Subscript["L", "y,2"]}, -3, 3},
   {{lz2, 2.5, Subscript["L", "z,2"]}, 0, 3}]}]
```

Out[308]=




- **link ← injoint**
- We have some freedom here with how to represent the transformation, since it can be an arbitrary rotation (no translation). This is determined entirely by robot parameters, not by the state, and we could parameterize it any way we'd like. Just to make a choice, let's have two angles which determine the angle $\theta$ the joint base should be rotated away from the z axis, and the angle $\phi$ away from the x-axis that that rotation should proceed in the direction of, together with a parameter $\psi$ determining how much the default angle should be rotated.

In[244]:=
```
ClearAll[Li, iL]
```

In[245]:=
```
Block[{θ, ϕ, ψ}, Li[θ_, ϕ_, ψ_ : 0] =
    Af@(EulerMatrix[{ϕ, θ, -ϕ}, {3, 2, 3}].RotationMatrix[ψ, {0, 0, 1}]);]
```

- If you expand this cell, you can see what I used to coax RotationMatrix into giving me something I could compute with numerically before simply using EulerMatrix.

In[247]:=
```
Block[{θ, ϕ, ψ}, iL[θ_, ϕ_, ψ_ : 0] = AfRotationInverse[Li[θ, ϕ, ψ]];]
```

- Finally, we're ready to define forward kinematics!
- In general, the notation AB for matrices here means that AB.(x,1) takes x to be a point represented in the B coordinate system, and outputs its representation in the A coordinate system, hence the A ← B headings I've been using.
- Note that there's a special meaning for AB.(0,0,0,1): when B is a link coordinate system: it's the location of the end of the link in coordinate system A.

- There's also an especially easy way to compute it: just take the last column of the matrix AB. We'll also lop off the trailing 1.

In[248]:=
```
origin[M_?MatrixQ] := M[[1 ;; 3, 4]]
```

- Once we have the sequence of coordinate matrices, we can just take successive dot products. The typical Mathematica way to compute this would be something like

In[249]:=
```
ibase = IdentityMatrix[3];
```

In[250]:=
```
(* origin/@FoldList[Dot[#2,#1]&,Af[ibase],
   Table[oi[q[i]].iL[θ[i],ϕ[i],ψ[i]].Lo[L[i]],{i,n}]]; *)
```

- But we want something slightly different; we want a function whose inputs are the joint coordinates q.

In[251]:=
```
ClearAll[f]
```

In[252]:=
```
f[q : {___}] := origin[Dot[##, Af[ibase]] & @@
   Table[io[q[[i]]].oL[L[i]].Li[θ[i], ϕ[i], ψ[i]], {i, Length[q]}]]
```

- assuming everything is defined. **This is the forward kinematics function in the form asked for in the assignment.**
- However, to build our simulation, we're going to do something a bit different. We're going to set up our parameters first, and then manipulate the intermediate matrices. We're also going to keep around the matrices we produce.
- We'll use GeometricTransformation[g, {m, v}] to transform to these systems. So to convert:

In[253]:=
```
AfGeometricTransformation[g_, AfM_] :=
 GeometricTransformation[g, {AfM[[1 ;; 3, 1 ;; 3]], AfM[[1 ;; 3, 4]]}]
```

- Define the graphical models (I avoid pattern-matching the arguments here for speed):

In[254]:=
```
iHue[i_, n_] := Hue[i / n, 0.7, 1]
```

In[255]:=
```
LinkDirectives[i_, n_] := {iHue[i, n]}
```

In[256]:=
```
linkthickness = 0.15;
```

In[257]:=
```
LinkModel[v0_, v1_, n_ : 1, i_ : 0, thicknessf_ : (linkthickness &)] :=
 Append[LinkDirectives[i, n], Tube[{v0, v1}, thicknessf[i, n]]]
```

In[258]:=
```
injointthickness = 0.15;
injointradius = 0.3;
```

In[260]:=
```
InJointDirectives[i_, n_] := If[i == 0, {Black}, {iHue[i - 1, n]}]
```

In[261]:=
```
InJointModelRoot[n_ : 1, i_ : 1, thicknessf_ : (injointthickness &),
  radiusf_ : (injointradius &)] := Append[InJointDirectives[i, n],
  Cylinder[{{0, 0, 0}, {0, 0, -thicknessf[i, n]}}, radiusf[i, n]]]
```

In[262]:=
```
outjointthickness = 0.15;
outjointradius = 0.3;
```

In[264]:=
```
OutJointDirectives[i_, n_] := {iHue[i, n]}
```

In[265]:=
```
OutJointModelRoot[n_ : 1, i_ : 0, ψ_ : 0, thicknessf_ : (outjointthickness &),
  radiusf_ : (outjointradius &)] := Append[OutJointDirectives[i, n],
  {{EdgeForm[Thin], Cylinder[{{0, 0, 0}, {0, 0, thicknessf[i, n]}}, radiusf[i, n]]},
   {Black, Thick, Line[{
      {0, 0, thicknessf[i, n]},
      {radiusf[i, n] Cos[ψ], radiusf[i, n] Sin[ψ], thicknessf[i, n]}}]}
  }]
```

- In the following, click “Adjust Parameters” at the top to change, well, any of the parameters. Note: The following implementation relies on the model for injoint being rotationally symmetric, so that rotations are invisible. (Double click the cell border on the right to see the code.) If you can’t see all the parameter panels, try dragging the window wider; there should be 3 per row.

In[309]:=
```
DynamicModule[{θ, ϕ, ψ, L, L0, Lθ, Lϕ, qrange,
  q, ibase, n = 7, nmax = 12, oLi, v, Fi, Fo, simulate = True},

 (* Init for θ[i], ϕ[i], L[i] *)
 Do[{θ[i], ϕ[i], ψ[i], L[i]} = {RandomReal[Pi / 2], RandomReal[2 Pi], 0,
    {RandomReal[{-1, 1}], RandomReal[{-1, 1}], RandomReal[{0.2, 1}]}}, {i, nmax}];
 Do[L0[i] = Norm[L[i]];
  Lθ[i] = ArcCos[L[i][[3]] / L0[i]];
  Lϕ[i] = If[L[i][[1]] == L[i][[2]] == 0, 0, ArcTan[L[i][[1]], L[i][[2]]]];,
  {i, nmax}];
 (* Init oLi matrix *)
 Do[oLi[i] = oL[L[i]].Li[θ[i], ϕ[i], ψ[i]];, {i, 1, nmax}];
 (* Init for base *)
 v[0] = {0, 0, 0};
 ibase = IdentityMatrix[3];
 Fi[0] = Af[ibase, v[0]];
 (* Init for qranges *)
 Do[qrange[i] = {-Pi, Pi}, {i, nmax}];
 (* Visible Dynamic *)
 Dynamic@If[simulate,
   (* Simulation *)
   Column[{Row[{Text["n = "],
       SetterBar[Dynamic[n], Range[nmax], Appearance → "Horizontal"]}],
     Button["Adjust parameters", simulate = False],
     Button["Randomize parameters", Do[{θ[i], ϕ[i], ψ[i], L[i]} =
        {RandomReal[Pi / 2], RandomReal[2 Pi], 0, {RandomReal[{-0.9, 0.9}],
          RandomReal[{-0.9, 0.9}], RandomReal[{0.2, 1}]}}, {i, n}];
      Do[L0[i] = Norm[L[i]];
       Lθ[i] = ArcCos[L[i][[3]] / L0[i]];
       Lϕ[i] = If[L[i][[1]] == L[i][[2]] == 0, 0, ArcTan[L[i][[1]], L[i][[2]]]];,
```

```
    {i, n}];
   (* Init oLi matrix *)
   Do[oLi[i] = oL[L[i]].Li[θ[i], ϕ[i], ψ[i]];, {i, 1, n}];],
  With[{qranges = Table[{{Symbol["q" <> ToString[i]], ψ[i], Subscript["q", i]},
      qrange[i][[1]], qrange[i][[2]]}, {i, 1, n}],
    qs = Table[Symbol["q" <> ToString[i]], {i, n}],
    workrange = (Total[Table[Norm[L[i]], {i, n}]] + linkthickness) *
      {{-1, 1}, {-1, 1}, {-1, 1}} * 0.7},
   Manipulate[
     q = qs;
     Fo[0] = Fi[0].io[q[[1]]];
     Do[Fo[i] = Fo[i - 1].oLi[i].io[q[[i + 1]]];
      v[i] = Fo[i][[1 ;; 3, 4]], {i, 1, n - 1}];
     Fo[n] = Fo[n - 1].oLi[n];
     v[n] = Fo[n][[1 ;; 3, 4]];
     (*Note: this is not a "true" value of Fo
       (or, it only is if we consider the final, invisible q to be 0 *)
     Graphics3D[
      Table[{LinkModel[v[i], v[i + 1], n, i], AfGeometricTransformation[
         {InJointModelRoot[n, i], OutJointModelRoot[n, i]}, Fo[i]]},
       {i, 0, n - 1}], PreserveImageOptions → True, RotationAction → Clip,
      PlotRange → workrange, ImageSize → 400],
     ##] & @@ qranges]
 }, Alignment → Center],
(* Parameter setting environment *)
Multicolumn[
 Prepend[Panel[Button["Simulate!", simulate = True, FrameMargins → 20]]] @
  Append[Panel[Grid[Table[With[{i = i},
       {Row[{"Range of ", Subscript["q", i]}],
        IntervalSlider[Dynamic[qrange[i], (simulate = False;
            qrange[i] = N[#];
            q[i] = Clip[q[i], N[#]];) &], {-Pi, Pi}, MinIntervalSize → 0.1,
         Method → "Stop"]}], {i, n}]]] @ Table[With[{i = i, j = i + 1},
    Panel[Grid[{{
        Grid[{{Subscript["θ", i], Subscript["ϕ", i], Subscript["ψ", i]}, If[i == 0,
           Table[VerticalSlider[1, ImageSize → Tiny, Enabled → False], {3}],
           {VerticalSlider[Dynamic[θ[i], (θ[i] = N@#;
                oLi[i] = oL[L[i]].Li[θ[i], ϕ[i], ψ[i]]) &],
             {Pi / 2, 0}, ImageSize → Tiny],
            VerticalSlider[Dynamic[ϕ[i], (ϕ[i] = N@#;
                oLi[i] = oL[L[i]].Li[θ[i], ϕ[i], ψ[i]]) &],
             {0, 2 Pi}, ImageSize → Tiny],
            VerticalSlider[Dynamic[ψ[i], (ψ[i] = N@#;
                oLi[i] = oL[L[i]].Li[θ[i], ϕ[i], ψ[i]]) &],
```

```
                {0, 2 Pi}, ImageSize → Tiny]}]}],

            Grid[{{"L", Slider[Dynamic[L0[j], (L0[j] = #;
                    L[j] = # * Normalize[L[j]];
                    oLi[j] = oL[L[j]].Li[θ[j], ϕ[j], ψ[j]]) &],
                  {0.1, 1.5}, ImageSize → Tiny]},
                {Subscript["L", "θ"], Slider[Dynamic[Lθ[j], (Lθ[j] = #;
                    L[j] = EulerMatrix[{Lϕ[j], Lθ[j], 0}, {3, 2, 3}].{0, 0, L0[j]};
                    oLi[j] = oL[L[j]].Li[θ[j], ϕ[j], ψ[j]]) &],
                  {0, Pi / 2}, ImageSize → Tiny]},
                {Subscript["L", "ϕ"], Slider[Dynamic[Lϕ[j], (Lϕ[j] = #;
                    L[j] = EulerMatrix[{Lϕ[j], Lθ[j], 0}, {3, 2, 3}].{0, 0, L0[j]};
                    oLi[j] = oL[L[j]].Li[θ[j], ϕ[j], ψ[j]]) &],
                  {-Pi, Pi}, ImageSize → Tiny]}
              }]},
            {Dynamic[Graphics3D[{If[i == 0, {}, LinkModel[{0, 0, -1},
                  {0, 0, 0}, n, i - 1]], AfGeometricTransformation[
                {InJointModelRoot[n, i], OutJointModelRoot[n, i]},
                If[i == 0, IdentityMatrix[4], Li[θ[i], ϕ[i], ψ[i]]]]},
              PlotRange → (1.2 * {{-#, #}, {-#, #}, {-# - 0.1, #}} &[
                 Max[outjointradius, injointradius]])]],
             Dynamic[Graphics3D[
              {LinkModel[{0, 0, 0}, L[i + 1], n, i], OutJointModelRoot[n, i]},
              PlotRange → ({{-#, #}, {-#, #}, {0, #}} &[1.5])]]}
          }, Alignment → Center]]], {i, 0, n - 1}],
    3, Appearance → "Horizontal", Alignment → Center]
]]
```

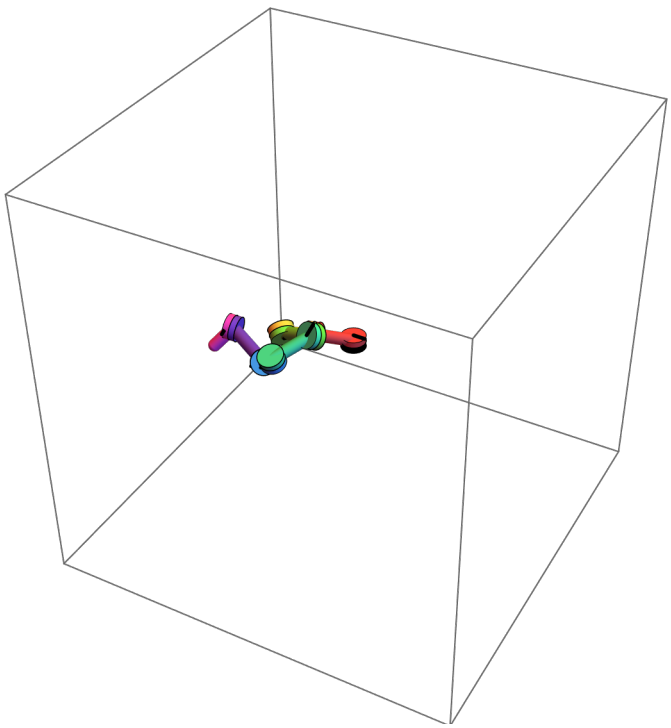
n = 1 2 3 4 5 6 7 8 9 10 11 12

Adjust parameters

Randomize parameters

$q_1$ $q_2$ $q_3$ $q_4$ $q_5$ $q_6$ $q_7$

Out[309]=



- The following generates some random starting angles, speeds, accelerations, and fluctuations, and exports a gif. It's behind an If branch to prevent it from evaluating if one evaluates the whole notebook; change False to True to create a gif.
  - gifn is a counter used to distinguish success exports, and is automatically incremented with each export. Its dynamically-updated value (which is used for the *next* export) is displayed below.

- Exports are named robot-arm-*gifn*.gif.

In[300]:=
```
If[MissingQ@PersistentSymbol["gifn"], PersistentSymbol["gifn"] = 5;];
gifn = PersistentSymbol["gifn"];
Dynamic[gifn]

Redo := (--PersistentSymbol["gifn"]; --gifn)
```

In[303]:=
```
If[True,
 Module[{θ, ϕ, ψ, L, L0, Lθ, Lϕ, q0, ibase, n = 4,
   oLi, v, Fi, Fo, workrange, ω, α, f, T = Identity, tmax = 7},
  Do[{θ[i], ϕ[i], ψ[i], L[i]} = {RandomReal[Pi / 2], RandomReal[2 Pi], 0,
     {RandomReal[{-1, 1}], RandomReal[{-1, 1}], RandomReal[{0.2, 1}]}}, {i, n}];
  Do[L0[i] = Norm[L[i]];
   Lθ[i] = ArcCos[L[i][[3]] / L0[i]];
   Lϕ[i] = If[L[i][[1]] == L[i][[2]] == 0, 0, ArcTan[L[i][[1]], L[i][[2]]]];,
   {i, n}];
  (* Init oLi matrix *)
  Do[oLi[i] = oL[L[i]].Li[θ[i], ϕ[i], ψ[i]];, {i, 1, n}];
  (* Init for base *)
  v[0] = {0, 0, 0};
  ibase = IdentityMatrix[3];
  Fi[0] = Af[ibase, v[0]];
  Do[ω[i] = RandomReal[{-1, 1}]; q0[i] = RandomReal[2 Pi];, {i, n}];
  If[True,
   Do[α[i] = RandomReal[{-1, 1}];, {i, n}];,
   α[_] := 0];
  If[True,
   Do[f[i] = With[{r = RandomReal[{-0.5, 0.5}]}, (Sin[r #] &)], {i, n}];,
   f[_] := (0 &)];
  workrange = (Total[Table[Norm[L[i]], {i, n}]] + linkthickness) *
    {{-1, 1}, {-1, 1}, {-1, 1}} * 0.7;
  If[True, (*Loop?*)
    T = t ↦ tmax Sin[Pi t / tmax] ^ 2] / 2;

  Export["robot-arm-" <> ToString[gifn++;
     PersistentSymbol["gifn"] ++] <> ".gif", Table[With[{t := T[t]},
     Fo[0] = Fi[0].io[q0[1] + t ω[1] + (1 / 2) t^2 α[1] + f[1][t]];
     Do[
      Fo[i] = Fo[i - 1].oLi[i].io[q0[i + 1] + t ω[i + 1] + (1 / 2) t^2 α[i + 1] + f[i + 1][t]];
      v[i] = Fo[i][[1 ;; 3, 4]], {i, 1, n - 1}];
     Fo[n] = Fo[n - 1].oLi[n]; v[n] = Fo[n][[1 ;; 3, 4]];
    Graphics3D[Table[{LinkModel[v[i], v[i + 1], n, i], AfGeometricTransformation[
        {InJointModelRoot[n, i], OutJointModelRoot[n, i]}, Fo[i]]}, {i, 0, n - 1}],
     PreserveImageOptions → True, RotationAction → Clip, PlotRange → workrange,
     ImageSize → 800, BoxStyle → Opacity[0]], {t, 0, tmax, 1 / 30}],
   RasterSize → 1200, "DisplayDurations" → 1 / 30]]]
```