Arm Planning

Andrew Lewis

Spring 2012

DH Parameters

In[1]:=
```
Params = {{1, 0, 0, θ1}, {1, 0, 0, θ2}, {.5, 0, 0, θ3}}; (* ai, di, αi, θi *)
```

This function takes a table of DH paramaters and outputs a list of transformation matrices.

In[2]:=
```
transMatrices[p_] := Module[{k, A, n, a, d, α, θ},
  n = Dimensions[p][[1]];
  A = Table[0, {k, 1, n}];
  For[k = 1, k ≤ n, k++,
   a = p[[k, 1]];
   d = p[[k, 2]];
   α = p[[k, 3]];
   θ = p[[k, 4]];
   A[[k]] = {{Cos[θ], -Sin[θ] Cos[α], Sin[θ] Sin[α], a Cos[θ]},
     {Sin[θ], Cos[θ] Cos[α], -Cos[θ] Sin[α], a Sin[θ]},
     {0, Sin[α], Cos[α], d}, {0, 0, 0, 1}};
  ];
  Return[A]
 ]
```

In[3]:=
```
A = transMatrices[Params]
```

Out[3]=
```
{{{Cos[θ1], -Sin[θ1], 0, Cos[θ1]}, {Sin[θ1], Cos[θ1], 0, Sin[θ1]}, {0, 0, 1, 0},
  {0, 0, 0, 1}}, {{Cos[θ2], -Sin[θ2], 0, Cos[θ2]}, {Sin[θ2], Cos[θ2], 0, Sin[θ2]},
  {0, 0, 1, 0}, {0, 0, 0, 1}}, {{Cos[θ3], -Sin[θ3], 0, 0.5 Cos[θ3]},
  {Sin[θ3], Cos[θ3], 0, 0.5 Sin[θ3]}, {0, 0, 1, 0}, {0, 0, 0, 1}}}
```

```
A[[1]] // MatrixForm
```

$$\begin{pmatrix} \text{Cos}[\theta 1] & -\text{Sin}[\theta 1] & 0 & \text{Cos}[\theta 1] \\ \text{Sin}[\theta 1] & \text{Cos}[\theta 1] & 0 & \text{Sin}[\theta 1] \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

```
Length[A]
```

3

Forward Kinematics Pose

This function takes in a list of θ and d states and transformation matrices and outputs the list of joint position coordinates.

In[4]:=
```
θs = {θ1, θ2, θ3, θ4, θ5, θ6, θ7, θ8};
ds = {d1, d2, d3, d4, d5, d6, d7, d8};
```

In[6]:=
```
pose[θ_, d_, A_] := Module[{n, p, k, rules, T},
  n = Length[A];
  rules = MapThread[Rule, {Take[θs, n], Flatten[{θ}]}];
  rules = Flatten[{rules, MapThread[Rule, {Take[ds, n], Flatten[{d}]}]}];
  p = Table[0, {k, 1, n + 1}];
  p[[1]] = {{0}, {0}, {0}, {1}};
  T = A[[1]];
  For[k = 2, k ≤ n, k++,
   p[[k]] = T.p[[1]];
   T = T.A[[k]];
  ];
  p[[n + 1]] = T.p[[1]];
  p = p /. rules;
  Return[p]
 ]
```

$$\theta = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix};$$

$$d = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix};$$

```
p = pose[θ, d, A]
```

```
{{{0}, {0}, {0}, {1}}, {{1}, {0}, {0}, {1}},
 {{2}, {0}, {0}, {1}}, {{2.5}, {0.}, {0.}, {1.}}}
```

```
p[[2]] // MatrixForm
```

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

Visual Simulation

In[7]:=
```
to3D[p_] := Module[{k, n, out},
  n = Length[p];
  out = Table[0, {k, 1, n}];
  For[k = 1, k ≤ n, k++,
   out[[k]] = Take[Flatten[{p[[k]]}], 3];
  ];
  Return[out]
 ]
```

In[8]:=
```
to2D[p_] := Module[{k, n, out},
  n = Length[p];
  out = Table[0, {k, 1, n}];
  For[k = 1, k ≤ n, k++,
   out[[k]] = Take[Flatten[p[[k]]], 2];
  ];
  Return[out]
 ]
```

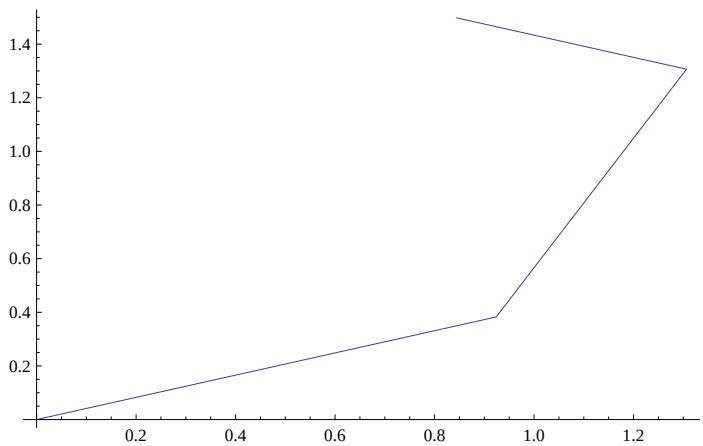
```
pdraw = N[to2D[p]]
```

```
{{0., 0.}, {1., 0.}, {2., 0.}, {2.5, 0.}}
```

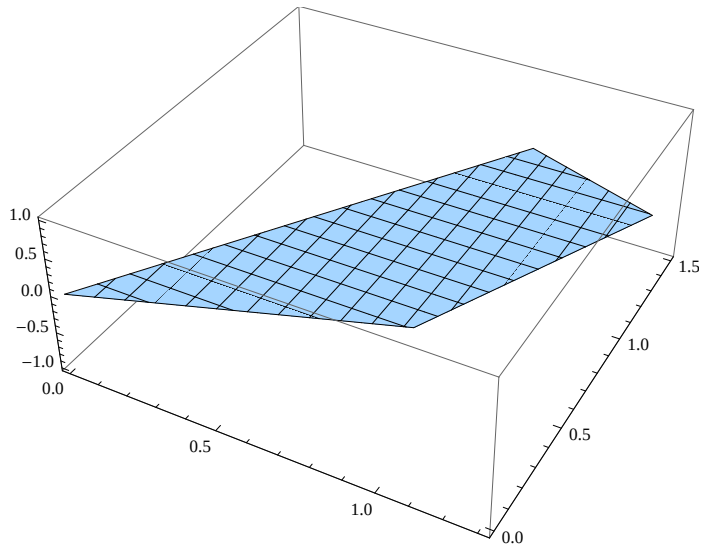
$$\theta = \begin{pmatrix} \pi/8 \\ \pi/4 \\ \pi/2 \end{pmatrix};$$

$$d = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix};$$

```
p = pose[θ, d, A];
pdraw = N[to2D[p]];
g11 = ListPlot[pdraw, Joined → True]
```

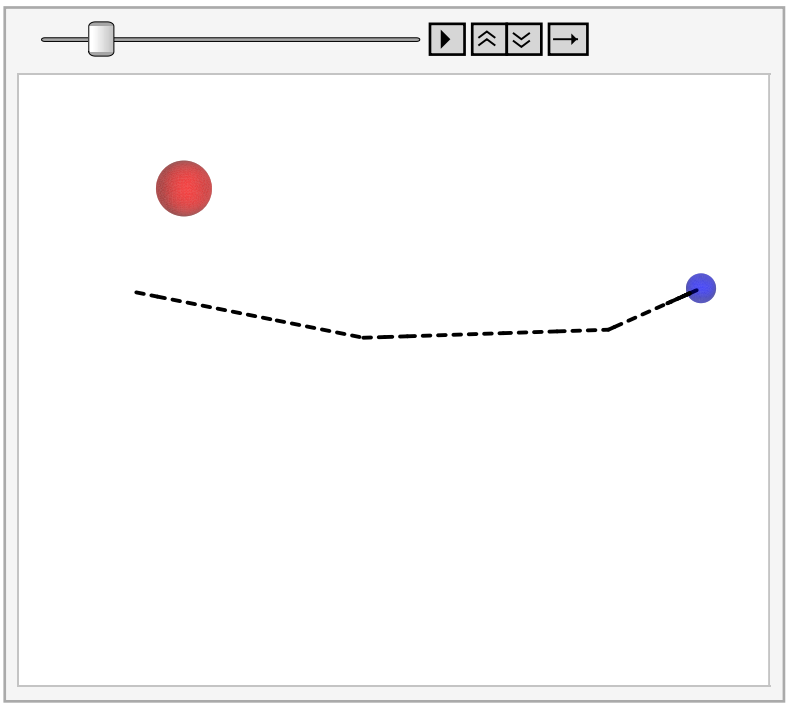


```
θ = {π / 8, π / 4, π / 2};

d = {0, 0, 0};

p = pose[θ, d, A];
pdraw = N[to3D[p]];
ListPlot3D[pdraw]
```




```
x0 = {-.25, .75, 0};

θlist = Table[{π / 6, π / 3, π / 2} * .1 k, {k, 1, 25, .5}];

plist = pose[#, d, A] & /@ θlist;
plist = to3D[#] & /@ plist;
glist =
  Graphics3D[{Black, Thick, Dashed, Line[#], Blue, Opacity[.5], Sphere[#[[4]], .0625],
      Red, Sphere[Flatten[x0], .125]}, Boxed → False] & /@ plist ;
```

```
ListAnimate[glist , AnimationRunning → False]
```

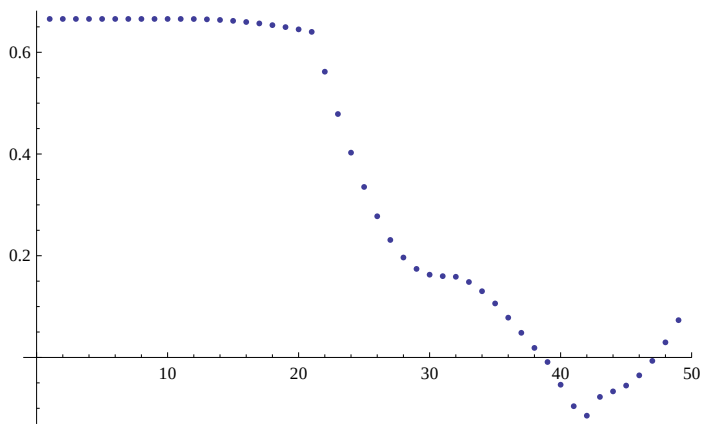


Collision Detection

- **Minimum Distance**

  This function gives the minimum distance between a sequential list of 3d points and a point in 3d space.

In[9]:=
```
minDistance[p_, x0_] := Module[{n, d, d2, c, a, b, minD, i, e},
  n = Length[p];
  minD = ∞;
  For[i = 1, i < n, i++,
   a = p[[i]]; b = p[[i + 1]];
   c = (x0 - a).(b - a) / Norm[(b - a)];
   (*Print["--------------------"];
   Print["i -> ", i, "  a -> ", a, "  b -> ", b, "  x0 -> ", x0];
   Print["c -> ", c];*)
   If[c > 0 && c < Norm[b - a], (*normal intesects line segment*)
     e = Norm[x0 - a];
     d = Sqrt[e^2 - c^2];
     , (*else d = dist to endpts*)
     d = Norm[a - x0];
     d2 = Norm[b - x0];
     If[d2 < d, d = d2];
    ]
    (*Print["d ->", d];*)
    If[ d < minD, minD = d];
  ];
  Return[minD]
 ]
ListPlot[(minDistance[#, Flatten[x0]] & /@ plist) - .125]
```



■ Collision Detection

This function returns true or false for a given arm orientation and list of objects and their radii.

In[10]:=
```
collisionFree[θ_, A_, {obj_, rad_}] := Module[{j, pList, r, out, n, rads, objs},
  out = True;
  pList = to3D[pose[θ[[1]], θ[[2]], A]];
  objs = Partition[Flatten[{obj}], 3];
  rads = Flatten[{rad}];
  n = Length[rads];
  For[j = 1, j ≤ n, j++,
   r = minDistance[pList, objs[[j]]];
   If[r ≤ rads[[j]], out = False];
  ];
  Return[out]
 ]
```

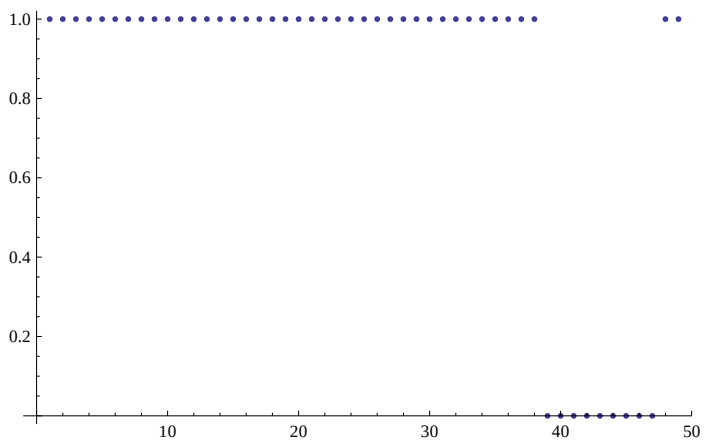
```
cList = collisionFree[{θlist[[41]], d}, A, {x0, .125}]
```

```
False
```

```
cList = collisionFree[{#, d}, A, {x0, .125}] & /@ θlist
```

```
{True, True, True, True, True, True, True, True, True, True, True, True,
 True, True, True, True, True, True, True, True, True, True, True, True,
 True, True, True, True, True, True, True, True, True, True, True, True,
 False, False, False, False, False, False, False, False, False, True, True}
```

```
ListPlot[cList]
```



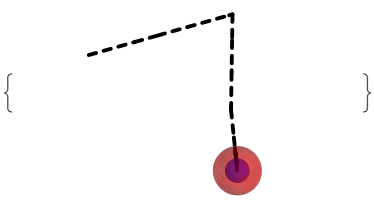
Inverse Kinematics

This function provides the joint angles for a 3 rotational dof serial arm for a given 3d point and a specified length between the first and third joint.

In[11]:=
```
inversePose[x0_, params_, l4_] := Module[{l, x, y, e, r, α, a, β, b, d, γ, θ},
  l = params[[All, 1]];
  x = x0[[1]]; y = x0[[2]];
  e = ArcTan[x, y];

  r = Sqrt[x^2 + y^2];
  a = ArcCos[(l4^2 + r^2 - l[[3]]^2)/(2 l4 r)];
  β = ArcCos[(l[[1]]^2 + l4^2 - l[[2]]^2)/(2 l[[1]] l4)];
  b = ArcCos[(l[[3]]^2 + l4^2 - r^2)/(2 l[[3]] l4)];
  d = ArcCos[(l[[2]]^2 + l4^2 - l[[1]]^2)/(2 l[[2]] l4)];
  γ = ArcCos[(l[[1]]^2 + l[[2]]^2 - l4^2)/(2 l[[1]] l[[2]])];
  θ = {(e + a + β), {-(π - γ)}, -(π - (b + d))};
  Return[θ]
 ]

xd = {{1.25}, {-.5}, {0}};
θ = inversePose[xd, Params, 1]
θ * 180 / π
p = to3D[pose[θ, d, A]]
Graphics3D[{Black, Thick, Dashed, Line[#], Blue, Opacity[.5],
    Sphere[#[[4]], .0625], Red, Sphere[Flatten[xd], .125]}, Boxed → False] &/@ {p}
```

$$\left\{\{0.978798\}, \left\{-\frac{2\pi}{3}\right\}, \{0.0738076\}\right\}$$

```
{{56.081}, {-120}, {4.22887}}

{{0, 0, 0}, {0.55802, 0.829827, 0}, {0.997662, -0.0683458, 0}, {1.25, -0.5, 0.}}
```



```
Element[Flatten[θ], Reals]

True
```

RRT Planning

Connect 1

In[12]:=
```
connect[{θ1_, d1_}, {θ2_, d2_}, A_, n_, {obj_, rad_}] :=
 Module[{cnct, θ, d, configs, cout, diffθ, diffd, i, r, pList, j},
  (*determine discrete configurations between two points*)
  diffθ = (θ2 - θ1) / n;
  diffd = (d2 - d1) / n;
  configs = Table[{θ1 + i diffθ, d1 + i diffd}, {i, 0, n}];
  cout = {};
  For[i = 1, i ≤ Length[configs], i++,
   If[collisionFree[configs[[i]], A, {obj, rad}],
     cout = Append[cout, configs[[i]];,
     (*else*)
     Return[{cout, False}]];
  ];
  cnct = True;
  Return[{cout, cnct}]
 ]
```

In[76]:=
$$\theta a = \begin{pmatrix} 3\pi/8 \\ \pi/4 \\ \pi/2 \end{pmatrix};$$

$$d0 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix};\quad da = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix};$$

$$\theta b = \left\{\{1.0471975511965976`\}, \left\{-\frac{2\pi}{3}\right\}, \{-0.5235987755982987`\}\right\};$$

$$db = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix};$$

```
n = 5;
diffθ = (θb - θa) / n;
diffd = (db - da) / n;
configs = Table[{θa + i diffθ, da + i diffd}, {i, 0, n}]
```

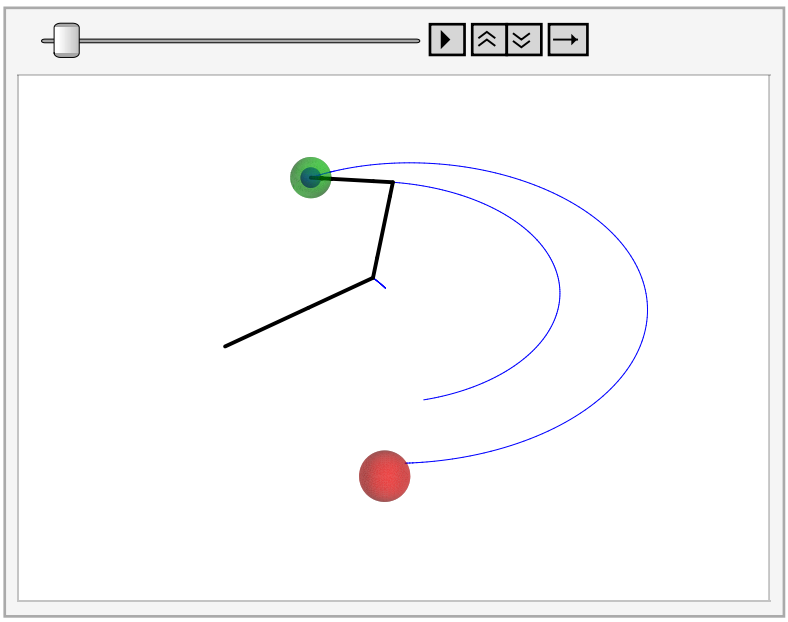
Out[83]=
$$\left\{\left\{\{1.1781\}, \left\{\frac{\pi}{4}\right\}, \{1.5708\}\right\}, \{\{0\}, \{0\}, \{0\}\}\right\},$$
$$\left\{\left\{\{1.15192\}, \left\{\frac{\pi}{15}\right\}, \{1.15192\}\right\}, \{\{0\}, \{0\}, \{0\}\}\right\},$$
$$\left\{\left\{\{1.12574\}, \left\{-\frac{7\pi}{60}\right\}, \{0.733038\}\right\}, \{\{0\}, \{0\}, \{0\}\}\right\},$$
$$\left\{\left\{\{1.09956\}, \left\{-\frac{3\pi}{10}\right\}, \{0.314159\}\right\}, \{\{0\}, \{0\}, \{0\}\}\right\},$$
$$\left\{\left\{\{1.07338\}, \left\{-\frac{29\pi}{60}\right\}, \{-0.10472\}\right\}, \{\{0\}, \{0\}, \{0\}\}\right\},$$
$$\left\{\left\{\{1.0472\}, \left\{-\frac{2\pi}{3}\right\}, \{-0.523599\}\right\}, \{\{0\}, \{0\}, \{0\}\}\right\}\right\}$$

```
configs[[1, 1]]
```

$$\left\{\{1.1781\}, \left\{\frac{\pi}{4}\right\}, \{1.5708\}\right\}$$

```
{configs, c} = connect[{θa, da}, {θb, db}, A, 150, {{xd}, {.125}}];
θc = configs[[All, 1]];
pList = pose[#, d, A] & /@ θc;
p = to3D[#] & /@ pList;
glist2 =
  Graphics3D[{Blue, Line[p[[All, 2]]], Blue, Line[p[[All, 3]]], Blue, Line[p[[All, 4]]],
      Black, Thick, Line[#], Blue, Opacity[.5], Sphere[#[[4]], .0625], Green,
      Sphere[p[[1, 4]], .125], Red, Sphere[Flatten[xd], .125]}, Boxed → False] & /@ p;
ListAnimate[glist2, AnimationRunning → False]
```

```
obstacle!
```



■ RRT Connect

In[14]:=

```
discθ1 = Table[i, {i, -3 π / 10, 13 π / 10, .1}];
discθ2 = Table[i, {i, -5 π / 8, 5 π / 8, .1}];
discθ3 = Table[i, {i, -12 π / 15, 12 π / 15, .1}];
nθ1 = Length[discθ1];
nθ2 = Length[discθ2];
nθ3 = Length[discθ3];
```

$$d0 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix};$$

```
rrtConnect[ps_, pg_, A_, params_, {obj_, radius_}] :=
 Module[{confGoal, confStart, sTree, gTree, curTree, otherTree, l4, l, f1, f2,
   f3, path, p1, pf, dir, k, min, minK, dist, newConfigs, cnctd, dummyTree,
   commonConf, cur, gFinal, sFinal, nextSConf, nextGConf, prevConf, rad},
  p1 = Partition[Flatten[{ps}], 1];
  rad = 1.15 radius;
```

```
pf = Flatten[{pg}];
l = params[[All, 1]];
l4 = .9 (l[[1]] + l[[2]]);
f1 = False; f2 = False; f3 = False;
(*Find start and ending configurationsusing successively smaller l4*)
While[f1 ≠ True,
 If[f2 == False, confStart = inversePose[p1, params, l4]];
 (*Check if config is real*)
 f2 = Element[confStart, Reals];
 (*Check if config isn't colliding*)
 If[ f2 == True,
  If[collisionFree[confStart, A, {obj, rad}], f2 = True, f2 = False]
 ];
 If[f3 == False, confGoal = inversePose[pg, params, l4]];
 (*Check if config is real*)
 f3 = Element[confGoal, Reals];
 (*Check if config isn't colliding*)
 If[ f3 == True,
  If[collisionFree[confGoal, A, {obj, rad}], f3 = True, f3 = False]
 ];
 (*decrease l4 by 90%*)
 l4 = l4 * .9;
 f1 = f2 && f3;
 If[l4 < (l[[1]] - l[[2]]) || l4 < 0, Return["No end point configurations"]];
]; (*loop*)
Print["start and end configs found"];
path = {confStart, confGoal};

(*initialize RRT storage*)
sTree = {{{confStart, d0}, {confStart, d0}}};
curTree = sTree;
gTree = {{{confGoal, d0}, {confGoal, d0}}};
otherTree = gTree;
dir = {confGoal, d0};
f1 = False;
cur = 1; (*1 means cur is start*)

(*RRT loop*)
cnctd = False;
While[! cnctd,
 (*find closest to dir in current*)
 min = ∞;
 minK = 1;
 For[k = 1, k ≤ Length[curTree], k++,
  dist = Norm[Flatten[dir] - Flatten[curTree[[k, 1]]]];
  If[dist < min,
   min = dist;
   minK = k;
  ];
 ];
 (*grow current tree in dir*)
```

```
 {newConfigs, cnctd} = connect[curTree[[minK, 1]], dir, A, 125, {obj, rad}];
 (*append new nodes to current Tree*)
 For[k = 2, k ≤ Length[newConfigs], k++,
  curTree = Append[curTree,
     {newConfigs[[k]], newConfigs[[k - 1]]}];
 ];
 (*find closest to tip of new branch in other*)
 dir = curTree[[-1, 1]];
 min = ∞;
 minK = 1;
 For[k = 1, k ≤ Length[otherTree], k++,
  dist = Norm[Flatten[dir] - Flatten[otherTree[[k, 1]]]];
  If[dist < min,
   min = dist;
   minK = k;
  ];
 ];
 (*grow other tree towards tip of new branch*)
 {newConfigs, cnctd} = connect[otherTree[[minK, 1]], dir, A, 125, {obj, rad}];
 (*append new nodes to other Tree*)
 For[k = 2, k ≤ Length[newConfigs], k++,
  otherTree = Append[otherTree,
     {newConfigs[[k]], newConfigs[[k - 1]]}];
 ];
 (*Break if two trees are connected*)
 If[cnctd,
  commonConf = newConfigs[[-1]];
  Break[]
 ];
 (*swap trees and pick new random dir*)
 dummyTree = curTree;
 curTree = otherTree;
 otherTree = dummyTree;
 cur *= -1; (*-1 → cur is goal tree, 1 → cur is sTree*)
 dir = {{discθ1[[RandomInteger[{1, nθ1}]]],
         discθ2[[RandomInteger[{1, nθ2}]]],
         discθ3[[RandomInteger[{1, nθ3}]]]}, {0, 0, 0}};

]; (*end of growth and connection loop*)
(*construct path*)
    (*determine current tree*)
If[cur == 1,
 sTree = curTree; gTree = otherTree;,
 (*else*)
 sTree = otherTree; gTree = curTree;
];
f1 = False;
f2 = False;
gFinal = commonConf;
sFinal = {};
nextSConf = commonConf;
```

```
  nextGConf = commonConf;
  Print["connected!"];
  k = 0;
  While[f1 ≠ True,
   prevConf = nextSConf;
   nextSConf = sTree[[Position[sTree, nextSConf][[1, 1]]][[2]];
   sFinal = Append[sFinal, nextSConf];
   If[Flatten[nextSConf] == Flatten[{confStart, d0}], Break[]];
   k++;
   If[k > 1500, Print["break1"]; Break[]];
  ];
  k = 0;
  While[f2 ≠ True,
   prevConf = nextGConf;
   nextGConf = gTree[[Position[gTree, nextGConf][[1, 1]]][[2]];
   gFinal = Append[gFinal, nextGConf];
   If[Flatten[nextGConf] == Flatten[{confGoal, d0}], Break[]];
   k++;
   If[k > 1500, Print["break2"]; Break[]];
  ];
  path = Flatten[{Reverse[sFinal], gFinal}];
  path = Partition[Partition[Partition[path, 1], 3], 2];
  Return[path]
 ]
```

In[59]:=
$$x0 = \begin{pmatrix} 0 \\ 1.5 \\ 0 \end{pmatrix}; \quad xa = \begin{pmatrix} -1 \\ 1.5 \\ 0 \end{pmatrix}; \quad xb = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}; \quad x1 = \begin{pmatrix} .15 \\ 1.75 \\ 0 \end{pmatrix};$$

```
rad = .125
```

Out[60]= 0.125

In[68]:=
```
time = AbsoluteTiming[path = rrtConnect[xa, xb, A, Params, {x1, rad}];][[1]]
```

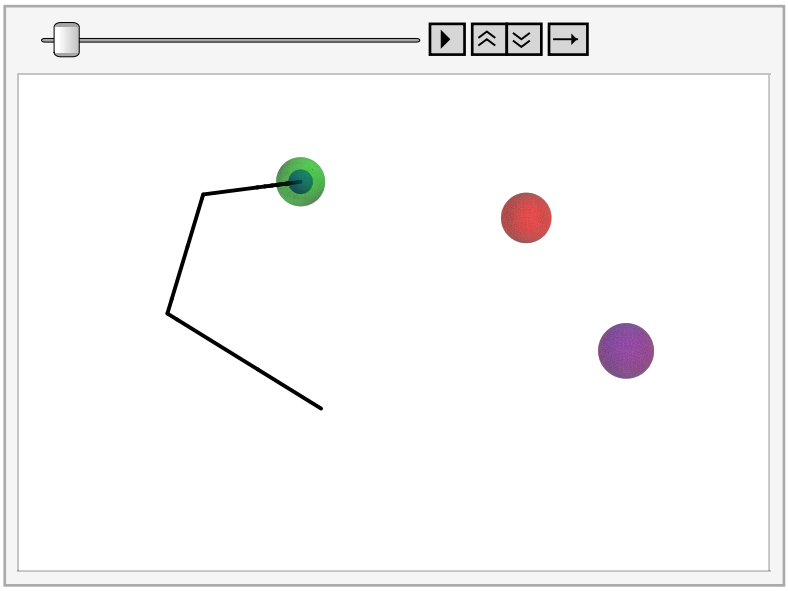
```
start and end configs found

connected!
```

Out[68]= 23.1019709

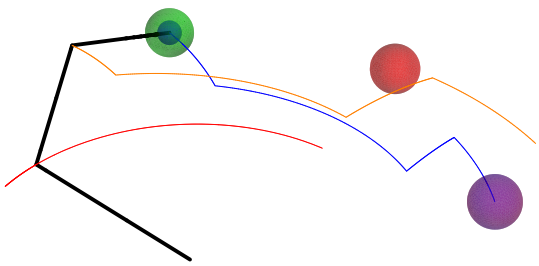
In[69]:=
```
Print["Length of path -> ", Length[path]];
Print["Time to calculate path -> ", time, " seconds"];
pList = pose[#[[1]], d0, A] &/@ path;
p = to3D[#] &/@ pList;
glist2 = Graphics3D[{
       Black, Thick, Line[#],
       Blue, Opacity[.5], Sphere[#[[4]], .0625],
       Green, Sphere[Flatten[xa], .125],
       Red, Sphere[Flatten[x1], rad],
       Purple, Sphere[Flatten[xb], .125]},
      Boxed → False] &/@ p;
ListAnimate[glist2, AnimationRunning → False]
Show[glist2[[1]],
 Graphics3D[{Blue, Line[p[[All, 4]]], Orange, Line[p[[All, 3]]], Red, Line[p[[All, 2]]] }]]
```

```
Length of path -> 410

Time to calculate path -> 23.1019709 seconds
```

Out[74]=



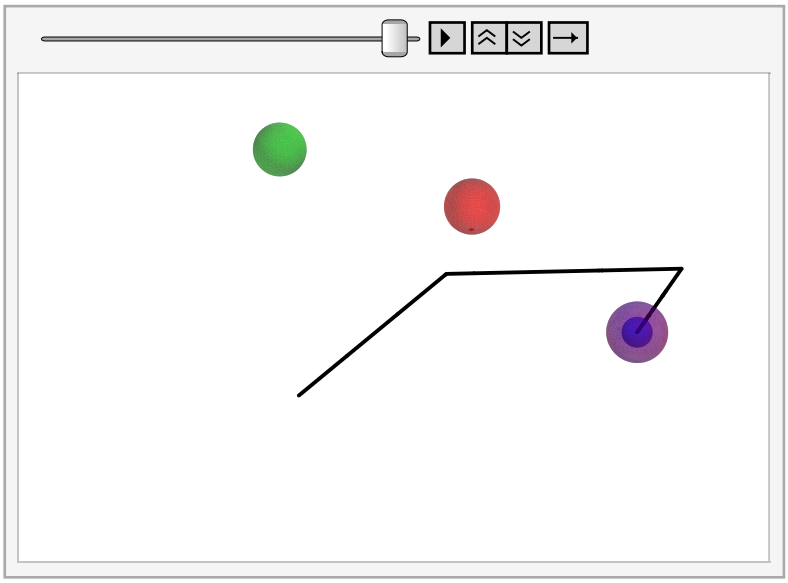
Out[75]=



Solutions

- **One Obstacle**
- **a**

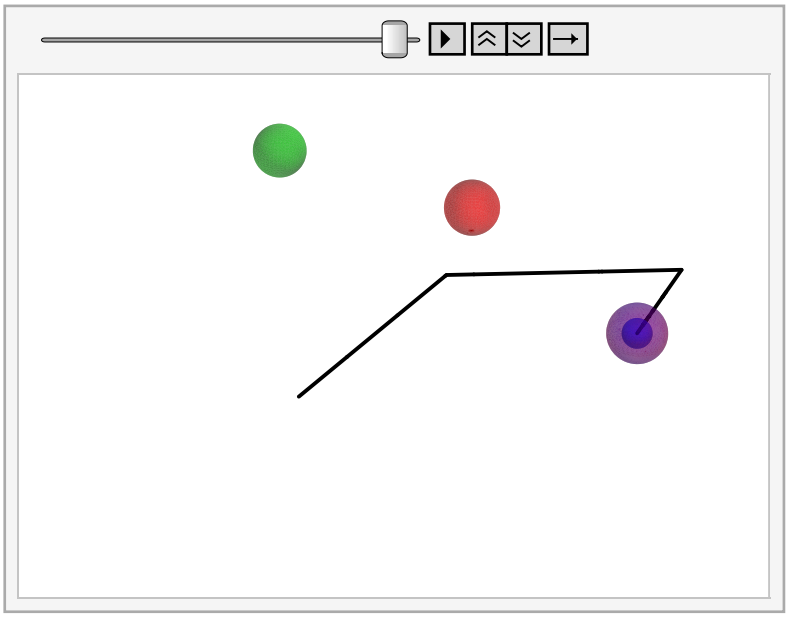
```
Length of path -> 995

Time to calculate path -> 25.4650000 seconds
```



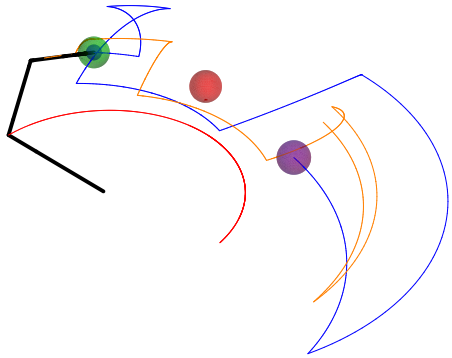
- **b**

```
Length of path -> 753

Time to calculate path -> 5.8440000 seconds
```

Out[51]=



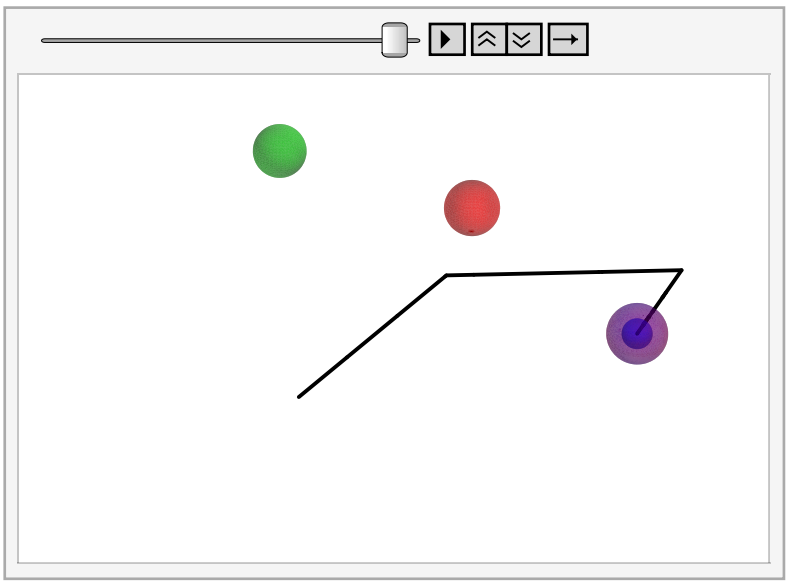
Out[52]=



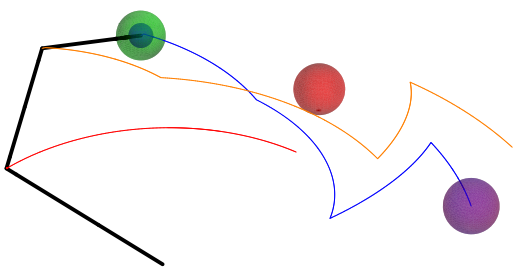
- **c**

```
Length of path -> 454

Time to calculate path -> 11.3730000 seconds
```

Out[59]=



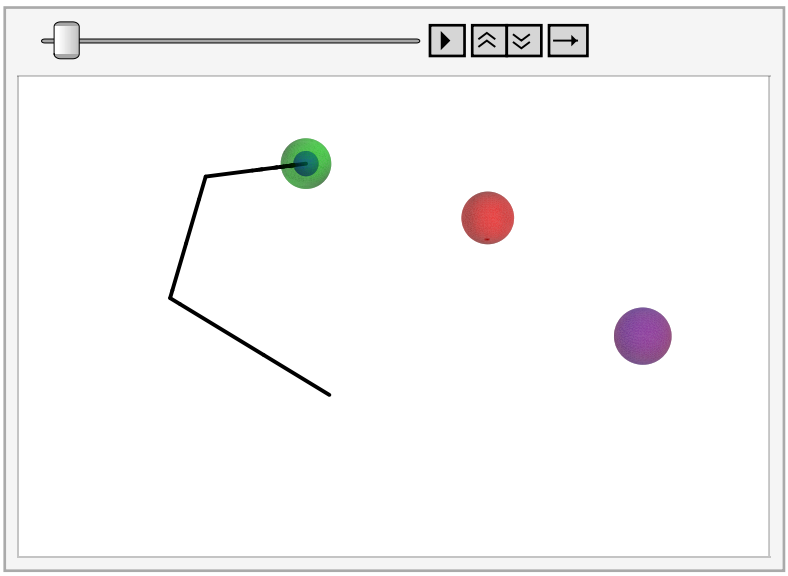
Out[60]=



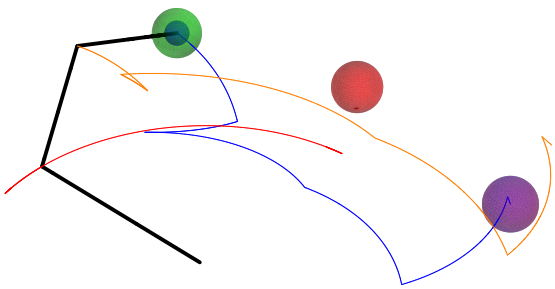
- **d**

```
Length of path -> 607

Time to calculate path -> 91.1690000 seconds
```

Out[67]=



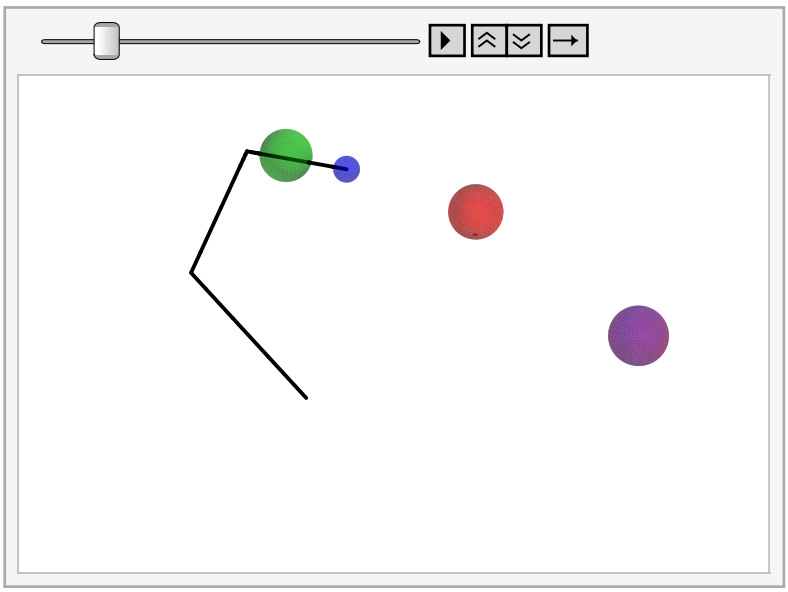
Out[68]=



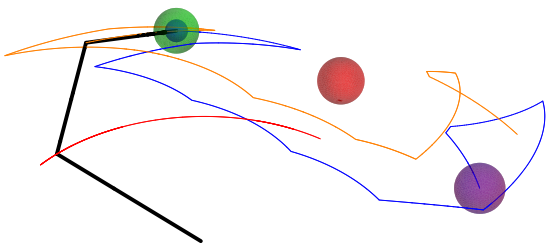
- **e**

```
Length of path -> 1005

Time to calculate path -> 80.8480000 seconds
```

Out[75]=



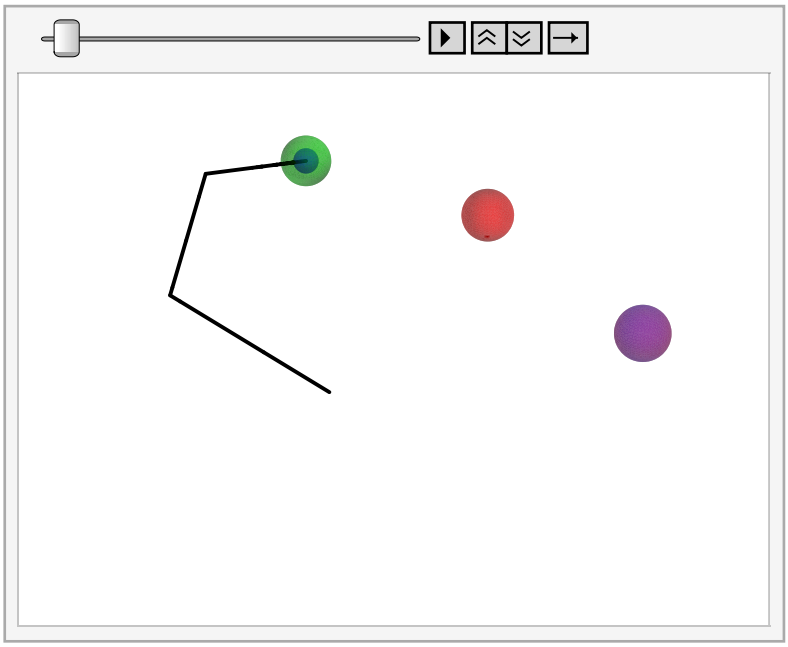
Out[76]=



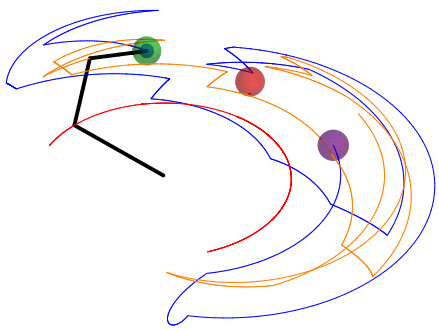
- **f**

```
Length of path -> 1596

Time to calculate path -> 137.1158000 seconds
```

Out[39]=



Out[40]=

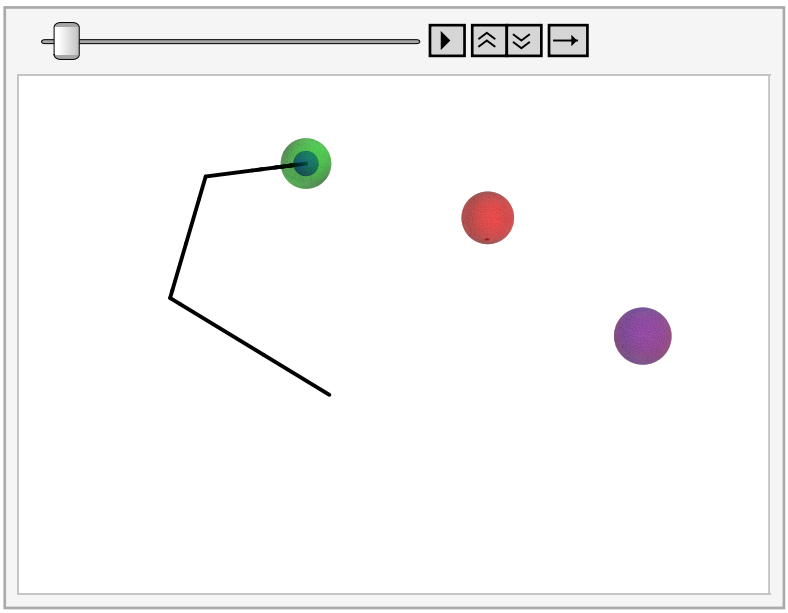


- **g**

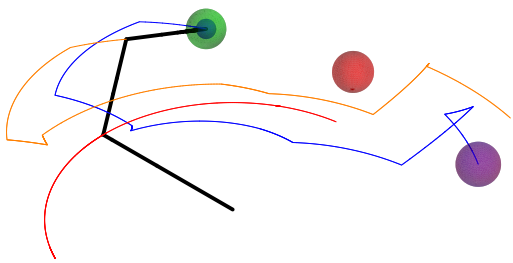
  Length of path -> 1045

  Time to calculate path -> 157.2480000 seconds

Out[51]=



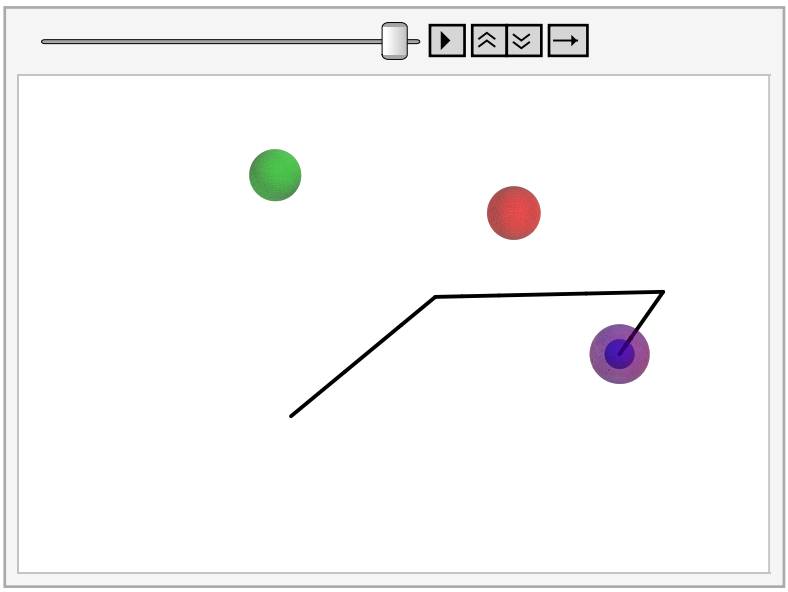
Out[52]=



- **h**

```
Length of path -> 410

Time to calculate path -> 23.1019709 seconds
```

Out[74]=



Out[75]=