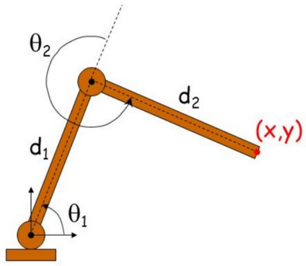
2 dimensional 2 joint inverse kinematics

problem: f(x,y,d1,d2) = Θ_2, then derive Θ_1
servo1: origin=(0,0), length=d_1, angle=Θ_1, servo 2: length=d_2 angle Θ_2

Θ_2 = acos((x^2+y^2-d_1^2-d_2^2)/(2*d_1*d_2))   solution set +/- Θ_2 +2kπ
Θ_1 = atan(y/x)-asin(d_2*sinΘ_2/sqrt(x^2+y^2))   solution set Θ_1 +kπ*

Implementing computation:
exclude divison by zero (when x=0, atan of infinity is pi/2)
use the correct solution (x>0 use second solution in the set ie. Θ_1+π



$$\theta_2 = \cos^{-1}\left(\frac{x^2 + y^2 - d_1^2 - d_2^2}{2d_1d_2}\right)$$

$$\theta_1 = \frac{-x(d_2\sin\theta_2) + y(d_1 + d_2\cos\theta_2)}{y(d_2\sin\theta_2) + x(d_1 + d_2\cos\theta_2)}$$

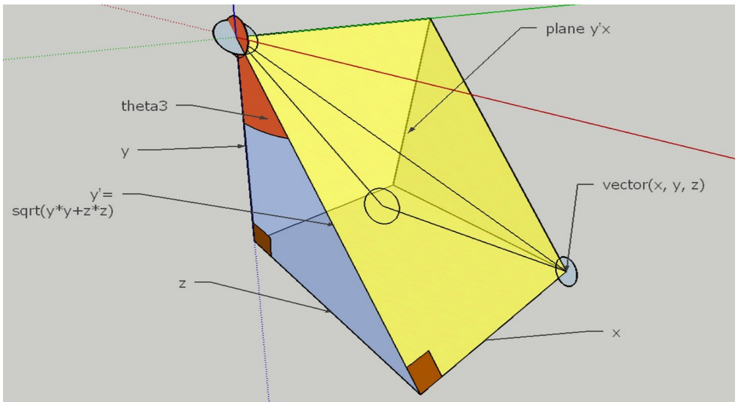
```
from math import *
def findangles(x=0.0, y=-0.59, d1=0.3, d2=0.3):
    a1=a2=a3=0.0
    test=(x*x+y*y-d1*d1-d2*d2)/(2*d1*d2)
    if test <-1 or test >1: solutions=0
    else:
        a2=acos(test)
        if x==0: temp=pi/2
        else: temp=atan(y/x)
        a1=temp-asin(d2*sin(a2)/sqrt(x*x+y*y))
        if  a2==0: solutions=1
        else: solutions=2;a3=temp-asin(d2*sin(-a2)/sqrt(x*x+y*y))
if x>0: a1=a1+3.14# i.e. use the second item in the solution set
    return(str(round(a1-0.43,3))+","+str(round(-a2,3)))
```

```
a=[]
a.append([0.0, -0.35, 0.3, 0.3]);a.append([-0.1, -0.35, 0.3, 0.3])
a.append([-0.2, -0.35, 0.3, 0.3]);a.append([-0.3, -0.35, 0.3, 0.3])
a.append([0.1, -0.35, 0.3, 0.3]);a.append([0.2, -0.35, 0.3, 0.3])
a.append([0.0, -0.4, 0.3, 0.3]);a.append([0.0, -0.5, 0.3, 0.3])
a.append([0.0, -0.59, 0.3, 0.3])
c=0
# generate motion script
print "#WEBOTS_MOTION,V1.0,motor1,motor4"
for b in a:
    print "00:"+"%02d"%(c*3)+":000,x="+str(b[0])+" y="+str(b[1])
+","+findangles(b[0],b[1],b[2],b[3])
    c=c+1
```

2.5 joint 3 dimensional inverse kinematics

problem: f(x,y,z,d1,d2) = Θ_2, then derive Θ_1 , Θ_3
servo0:origin(0,0),length=0, angle=Θ_3, rotates around x axis
servo1:origin=(0,0), length=d_1, angle=Θ_1,, rotates in y'x plane
servo 2: length=d_2 angle Θ_2, ,, rotates in y'x plane

Θ_3= atan(z/y) solution set Θ_3+ kπ/2
use 2d plane y'*x and from 2d equations above derive Θ_2 and Θ_1
y'=sqrt(y^2 +z^2) now substituting y' for y
Θ_2 = acos((x^2+y^2+z^2-d_1^2-d_2^2)/(2*d_1*d_2)) solution set +/- Θ_2 +2kπ
Θ_1 = atan(y'/x)-asin(d_2*sinΘ_2/sqrt(x^2+y^2+z^2)) solution set Θ_1 +kπ*



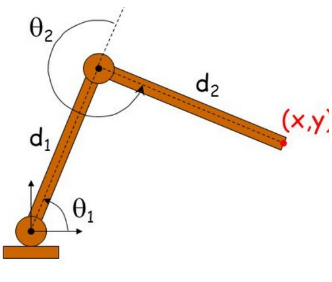

```
from math import *
def findangles(x=0.0, y=-0.4, z=-0.1, d1=0.3, d2=0.3):
    ydsq=y*y+z*z;yd=sqrt(ydsq)
    if y<0: yd=-yd
    a1=a2=a3=0.0
    test=(x*x+ydsq-d1*d1-d2*d2)/(2*d1*d2)
    if test <-1 or test >1: solutions=0
    else:
        a2=acos(test)
        if x==0: temp=pi/2
        else: temp=atan(yd/x)
        a1=temp-asin(d2*sin(a2)/sqrt(x*x+ydsq))
        if x>0: a1=a1+3.14# i.e. use the second item in the solution set
    a3=atan(z/y)
    return(str(round(a1-0.43,3))+","+str(round(a3,3))+","+str(round(-a2,3)))
```

```
a=[]
a.append([0.0, -0.35, 0.0]);a.append([-0.1, -0.35, 0])
a.append([-0.2, -0.35, 0.0]);a.append([-0.3, -0.35, 0.0])
a.append([0.1, -0.35, 0.0]);a.append([0.2, -0.35, 0.0])
a.append([0.0, -0.4, 0.0]);a.append([0.0, -0.5, 0.0])
a.append([0.0, -0.59, 0.0]);a.append([0.0, -0.35, 0.0])
a.append([0.0, -0.35, 0.1]);a.append([0.0, -0.35, 0.2])
a.append([0.0, -0.35, 0.3]);a.append([0.0, -0.35, -0.1])
a.append([0.0, -0.35, -0.2]);a.append([0.0, -0.35, -0.3])
a.append([0.0, -0.35, -0.2]);a.append([0.0, -0.35, 0.0])
c=0 #generate motion script
print "#WEBOTS_MOTION,V1.0,motor1,motor9,motor4"
for b in a:
    print "00:"+"%02d"%(c*3)+":000,x="+str(b[0])+" y="+str(b[1])+"
z="+str(b[2])+","+findangles(b[0],b[1],b[2])
    c=c+1
```

Forward Kinematics Now calculate which solutions hold true for our model, using forward kinematics.

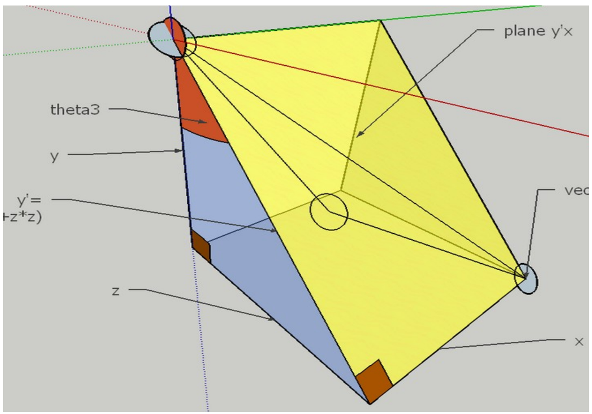
2 dimensions, 2 joints:

x = d_1*cos Θ_1 + d_2*cos Θ_1+Θ_2
y = d_1*sin Θ_1 + d_2*sin Θ_1+Θ_2



3 dimensions, 2.5 joints:

x = d_1*cos Θ_1 + d_2*cos Θ_1+Θ_2,
y' = d_1*sin Θ_1 + d_2*sin Θ_1+Θ_2
y = y' cos Θ_3,
z = -y' sin Θ_3



```
#improvements in inv kinematics.
#findangles function now casts x,y,z as floats, if x is positive the second solution is used for a1
#findangles function - a3 is measured from negative y so y needs to be inverted

from math import *
def findangles(x,y,z, d1=30, d2=60):
   x,y,z=float(x),float(y),float(z)
   ydsq=y*y+z*z;yd=sqrt(ydsq)
   if y<0: yd=-yd
   a1=a2=a3=0.0; test=(x*x+ydsq-d1*d1-d2*d2)/(2*d1*d2)
   if test <-1 or test >1: retun(0,0,0)
   else:
      a2=acos(test)
      if x==0: temp=pi/2
      else: temp=atan(yd/x)
      a1=temp-asin(d2*sin(a2)/sqrt(x*x+ydsq))
   a3=atan(z/-y) # invert the y axis due to location of a3
   if x<0: a1=a1+pi #use a different solution if x positive
   return(a1,a3,a2)

def forward(a1,a2,a3, d1=30, d2=60):
   x=(d1*cos(a1))+(d2*cos(a1+a2))
   yd=(d1*sin(a1)+d2*sin(a1+a2))
   y=yd*cos(a3)
   z=-yd*sin(a3)
   return(round(x,1),round(y,1),round(z,1))

def test(a,b,c):
   print("original coords: x="+str(a)+" y="+str(b)+" z="+str(c))
   a=findangles(a,b,c);print("angle(degs) a1="+str(a[0]*30.96)+" a2="+str(a[2]*30.96)+"
a3="+str(a[1]*30.96))
   b=forward(a[0],a[2],a[1]);print("derived  coords: x="+str(b[0])+" y="+str(b[1])+"
z="+str(b[2]))
   print"============================================================="

#now test in all 4 quadrants
test(10,-75,0);test(20,-52,10)
test(-20,-62,-10);test(10,52,0)
test(-10,62,0);test(20,-52,10)
 test(-20,-62,-10)
```

```
OUTPUT
original coords: x=10 y=-75 z=0
angle(degs) a1=-70.5817163203 a2=37.8821618443 a3=0.0
derived  coords: x=10.0 y=-75.0 z=0.0
=========================================================
original coords: x=20 y=-52 z=10
angle(degs) a1=-81.4716011396 a2=60.0334282584 a3=5.88203739396
derived  coords: x=20.0 y=-52.0 z=10.0
=========================================================
original coords: x=-20 y=-62 z=-10
angle(degs) a1=100.992205796 a2=49.9738745039 a3=-4.95091029298
derived  coords: x=-20.0 y=-62.0 z=-10.0
=========================================================
original coords: x=10 y=52 z=0
angle(degs) a1=-4.86842567404 a2=63.8192826663 a3=-0.0
derived  coords: x=10.0 y=52.0 z=0.0
=========================================================
original coords: x=-10 y=62 z=0
angle(degs) a1=15.3675237948 a2=53.4326705861 a3=-0.0
derived  coords: x=-10.0 y=62.0 z=0.0
=========================================================
original coords: x=20 y=-52 z=10
angle(degs) a1=-81.4716011396 a2=60.0334282584 a3=5.88203739396
derived  coords: x=20.0 y=-52.0 z=10.0
=========================================================
original coords: x=-20 y=-62 z=-10
angle(degs) a1=100.992205796 a2=49.9738745039 a3=-4.95091029298
derived  coords: x=-20.0 y=-62.0 z=-10.0
=========================================================
>>>
```

So final algorithm

```
def findangles(x, y, z, d1=30, d2=60):
    x, y, z = float(x), float(y), float(z)
    ydsq = y * y + z * z;
    test = (x*x + ydsq - d1*d1 - d2*d2) / (2*d1*d2)
    if test < -1 or test > 1 or x*x+y*y+z+z<(d1-d2)*(d1-d2):
        return(0, 0, 0, 0)
    else:
        a2 = acos(test)
        if x == 0: temp = pi / 2
        else: temp = atan(-sqrt(ydsq) / x)#yd = -sqrt(ydsq) i.e.
assume yd is negative for now
        a1 = temp - asin(d2 * sin(a2) / sqrt(x * x + ydsq))
        a3 = atan(z / -y)  # invert  y axis due to loc of a3
    if x <= 0: a1 = a1 - pi  #use different solution if x +'ve
    if y>=0: #deal with yd as positive
        if z<=0: a3= a3 - pi
        else: a3 = a3 + pi
    return (a1, a3, a2)
```

```
The checksums conclude:
When X is negative use second solution for θ1 (i.e θ1+pi)
When Y is positive: If  Z>0  use θ3+pi, if Z<=0 use θ3-pi
Oher wise you can use first solution for θ1 , θ2 and θ3 pi
```

Speeding up caculations
fill a 3d matrix with remdembered calculations (calculate once, read many)

```
#scipy is limited in not allowing tuples in its sparse data
structures
#to overcome this I use a pointer to a list of tuples
#the 0th value in the list will represent Not calculated

from scipy import sparse
m = sparse.dok_matrix((100, 2000), dtype='int16')
list=["Not calculated"]
def add_element((x, y, z), (a,b,c)):
    m[x, y + z * 100] = len(list)
    list.append((a,b,c))
def get_element(x, y, z):
    return list[m[x, y + z * 100]]
add_element([3, 2, 4], [0.15,1.3,-1.57])
add_element([20, 15, 7], [1.2,-3.23,-1.2])
add_element([0, 0, 1], [-1.20,3.22,-1.34])
print get_element(0, 0, 1)[0]
print get_element(3, 2, 4)
print get_element(20, 15, 7)
print get_element(0, 0, 0)
print "  This is m sparse:";print m
m=m.tocsr()
sparse.csr_matrix.sort_indices(m)
print "  This is m sparse sorted:";
print m
sparse.save_npz("test.npz", m.tocsr(), compressed=True)
```

```
OUTPUT
-1.2
(0.15, 1.3, -1.57)
(1.2, -3.23, -1.2)
Not calculated
  This is m sparse:
  (3, 402)      1
  (20, 715)     2
  (0, 100)      3
  This is m sparse sorted:
  (0, 100)      3
  (3, 402)      1
  (20, 715)     2
```

Walking gaits in quadrupeds

walking gaits in quadrupeds are cyclical limb movements that can be described by temporal displacement loops which vary from limb to limb.

Gait	Phases (radians)	limbs in same phase
Prance	0	4
Trot	0,pi (diagonal)	2
Pace	0,pi (ipsilateral)	2
Gallop	0,?2/pi (front, back)	2
Walk	0,pi/2,pi,3*pi/2	1

Displacement, time loops

The shape of the displacement time loop and its time period is likely to be the same for all limbs and these 2 variables will fully describe the gait.

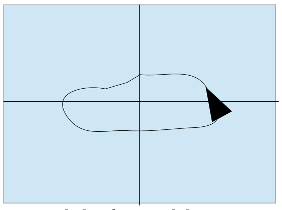
f(t)=(x,y); where constants are origin, x displacement, y displacement skew in all 4 quadrants (x,y for skew point)

maximum x forward = maximum backward displacement

gait height and gait vertical/horizontal displacement are customisable

ideal gait would: maximise x displacement, minimise y displacement

Initial programmatic solution assumed that velocity was approximately the same through-out the cycle only allowing certain aspects such as heights of the curve to be changed– this limits the possible gaits and dynamics so a more complex definition of the displacement-time loop is needed.



Solution

Have a set number of points in time per cycle i.e. 12 and each of them has a 3 dimensional dipslacement this allows for forward, sideways and rotational movements and also for much better dynamic movements where most displaement takes place in a single phase such as jumping, hopping, proncing etc..

this is still not covering all possible gaits as its possible to use limbs differently ie front feet for balnce and back for propulsion or ?limping gaits .

Rotation of quadrupeds

length and width of body are necessary to calculate rotations

Details of model dimensions (for/from Webots)

Legs	**Body dimensions**
lower leg 116mm upper leg 70mm	width 70mm length 180mm (90+45*2)

Actual dimensions

Body: 90mmx70mm

hips: height 45mm, radius 40mm

arm1 shape: height 70mmxradius 20 mm

translation: x-35mm, y 22.5mm

leg group: height 116mmxradius 20 mm (take 5mm off for foot ->> 111

translation: y: 90mm

foot: sphere radius: 10mm

transalation (from middle: y 55.5mm)

joint anchors

hip1:0,0,0

hip2:y=-0.0225 (for opposite front/back legs reverse signs)

knee: 0.035

centre of robot is at O, body w*l

abs servo front left (l/2,-w/2)

rel foot front left (x,z)

abs foot front left => (x+l/2,z-w/2)

abs foot front right => (x+l/2, z+w/2)

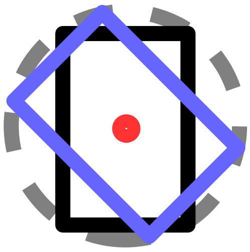
abs foot back left => (x-l/2,z-w/2)

abs foot back right => (x-l/2, z+w/2)

now rotate by angle ϴ about O:

x' = x cosϴ - y sinϴ

y' = y cosϴ + x sinϴ



```
def rotate45(x,z, a, w=70, l=180):
   z, z,a =float(x), float(z), float(a);ca=cos(a);sa=sin(a)
   nx=x+l/2;nz1=z-w/2;nz2=-z+w/2#get absolute coords about origine
   x1=nx*ca+nz1*sa;z1=nz1*ca-nx*sa #x1-l/2 and z1+w2 gives the relative coordinates
   x2=nx*ca+nz2*sa;z2=nz2*ca-nx*sa #x2-l/2 and z2-w/2 gives the relative coordinates
   return( x2-l/2, z2-w/2,x1-l/2,z1+w/2,-x1+l/2,-z1-w/2, -x2+l/2, -z2+w/2)#new leg positions for FR, FL, BR, BL
```