```
import time #used for the sleep function
from pypot.dynamixel import autodetect_robot  #used to get the robot object
import pypot.dynamixel  #used to get the motors,legs etc.
import math #quite obvious
import json #to use a json file
from contextlib import closing  #to close properly the robot at  the end
import pypot.robot

asterix = None
legs = []
xCorrection = [80,0,0,80,0,0]
yCorrection = [0,0,0,0,0,0]



def leg_ik(x3,y3,z3,alpha = 20.69, beta = 5.06,l1=51,l2=63.7,l3=93):
    d13 = math.sqrt(x3*x3 + y3*y3) - l1
    d = math.sqrt(d13*d13 + z3*z3)
    tmp = (l2**2 + d**2 - l3**2)/(2*l2*d)
    a1 = z3 / d13
    a2 = (l2**2 + l3**2 - d**2)/(2*l2*l3)

    angles = (None,None,None)
    theta1 = math.degrees(math.atan2(y3,x3))    # OK

    theta2 = math.degrees(math.atan(a1) + math.acos(tmp))
    theta3 = 180 - math.degrees(math.acos(a2))
# Corrections to the angles theta2 and theta3
    theta2 = -(theta2 + alpha)
    theta3 = -(theta3 - 90 + alpha + beta)
    angles = (theta1,theta2,theta3)
    return angles

def get_legs(obj):
    return [obj.leg1,obj.leg2,obj.leg3,obj.leg4,obj.leg5,obj.leg6]

def get_xCorrection(leg):
  i = int(leg[0].id*0.1)
  return xCorrection[i-1]

def get_yCorrection(leg):
  i = int(leg[0].id*0.1)
  return yCorrection[i-1]

"""---------------------- Rotation Functions ---------------------------------"""
""" Written by Thor the 24/03/15 """
""" Tested by Corentin the 24/03/15 """


""" ---------------------Mathematics to correct the rotation ---------------"""
```

```
""" Written by Thor the 26/04/15"""
# This needs to be done so that we can define a common circle of rotation
# for all the legs. To communicate this common information to all the legs
# we need to express the radius of this common circle of rotation as a function of theta and the legs
""" I added the attributes needed to obtain this information to the json file
Currently these attributes are set to zero"""
def R_leg(theta,leg,R):
  xCorrection = get_xCorrection(leg)
  yCorrection = get_yCorrection(leg)
  cos = math.cos(math.radians(theta))
  sin = math.sin(math.radians(theta))
  tmp = (xCorrection**2)*((cos**2)-1)
  tmp += (yCorrection**2)*((sin**2)-1)
  tmp += xCorrection*yCorrection*math.sin(math.radians(2*theta))
  tmp += R**2
  return (-xCorrection*cos - yCorrection*sin + math.sqrt(tmp))


# This function takes care of 1 leg at a time
# This moves the leg given polar coordinates. Important because we when we need to do a rotation the legs should not move
# outside the circle of rotation. We want a perfect rotation!
# TEST : Working perfectly
def move_leg(theta,z,leg,R = 150):

  i=0
  # Tupl is a vector that carries the angles that represent the final position of the tip of the leg
  # The angles are calculated from the arguments of the function using inverse kinematics
  # R is the radius of the circle of rotation. Theta is given in degrees.
  # Lets transform our polar coordinates onto the Cartesian plane
  # print R_leg(theta,leg,R), " - ", leg[0].id
  x = R_leg(theta,leg,R)*math.cos(math.radians(theta))+get_xCorrection(leg)
  y = R_leg(theta,leg,R)*math.sin(math.radians(theta))+get_yCorrection(leg)
  motor_angles = leg_ik(x,y,z)
  for m in leg:
    m.goal_position = motor_angles[i]
    i+=1
  return (x,y,z)

# This should just give us our initial spider position
# We also use this function when rotating to refix the legs' frames of reference
"""SOLVED?"""
#TEST : We SHOULD NOT put negative value in this function (otehrwise the legs (except legs 1-4) will 'meet each other')
def initial_pos(asterix,theta,z):
  # Experiments have shown that using the values 100 and 30 for changing x and y respectively is working okay
  initial_position = []
  initial_position.append(move_leg(0,z,legs[0]))
```

```
97    initial_position.append(move_leg(-abs(theta),z,legs[1]))
98    initial_position.append(move_leg(abs(theta),z,legs[2]))
99    initial_position.append(move_leg(0,z,legs[3]))
100   initial_position.append(move_leg(-abs(theta),z,legs[4]))
101   initial_position.append(move_leg(abs(theta),z,legs[5]))
102
103   time.sleep(0.1)
104
105   return initial_position
106
107 """
108 TODO: make sure that this works. If it works than we can easily do experiments to find the
      highest value on alpha
109 If we know the highest value of alpha we can determine the number of turns needed to do an
      arbitrary amount of rotation by using Euclidean division
110 See the draft implementation for arbitrary_rotation above.
111 """
112 # theta is the value we need to set the initial position
113 # alpha determines the amount of rotation (made by each call to the function) from this initial
       position
114 # alpha is physically limited because of the legs. We should define this limit as max_angle -
      see above.
115 # TEST : A value of 45 will make the legs (2-3 and 4-5) touch for a little while (actually
      until the next leg move)
116 def rotation_angle(asterix,alpha,theta,z):
117   #clockwise 2 and 5 are limited
118   breaklength = 0.1
119   # Position 1: The 'spider' position. This position has a low center of gravity.
120   # Here we define the initial position. i.e. the spider position
121   # It is important to observe the x and y values of each leg in its own frame of reference
122
123   # Position 2: Put legs 1, 3, 5 in the air and rotate at the same time
124   move_leg(-abs(theta)+alpha,z+20,legs[1])
125   move_leg(alpha,z+20,legs[3])
126   move_leg(abs(theta)+alpha,z+20,legs[5])
127   time.sleep(breaklength)
128
129   # Position 3: Put legs 1,3 and 5 down
130   move_leg(-abs(theta)+alpha,z,legs[1])
131   move_leg(alpha,z,legs[3])
132   move_leg(abs(theta)+alpha,z,legs[5])
133   time.sleep(breaklength)
134
135   # Position 4: Rotate legs 0, 2, 4
136   move_leg(alpha,z+20,legs[0])
137   move_leg(abs(theta)+alpha,z+20,legs[2])
138   move_leg(-abs(theta)+alpha,z+20,legs[4])
139   time.sleep(breaklength)
140
141   # Position 5: Put legs 0, 2 and 4 down.
142   move_leg(alpha,z,legs[0])
```

```
143   move_leg(abs(theta)+alpha,z,legs[2])
144   move_leg(-abs(theta)+alpha,z,legs[4])
145   time.sleep(breaklength)
146
147 # max_angle = 20 is just a guess.
148 # TEST : Working not too bad. beta = 180 are doing a rotation of 90deg. It seems that we have
      to multiply the wanted value by 2 to have a proper rotation
149 """SOLVED?"""
150 #TEST : If we put negative value fot the beta angle, this is just not working.
151 # TEST : If the value of max_angle is not 20, the rotation does not work proprely
152 # theta and z are simply values that determine the initial position
153 # Other parameters are to define the rotation
154 def arbitrary_rotation(asterix,beta, max_angle = 20, theta = 45, z = -60):
155 # Here we do euclidean division. We determine how often max_angle divides beta and the
      remainder of this division.
156 # This gives us the number of rotations we need to make by a predefined max_angle
157 # The remainder gives us the amount we need to rotate by to be able to finish the full rotation
      by an angle of beta
158 # i.e. beta = q*max_angle + r
159   initial_pos(asterix,theta,z)
160   if beta < 0:
161     max_angle = -max_angle
162
163   q = beta//max_angle
164   r = beta%max_angle
165   print q
166   print r
167   # rotate by max_angle q times
168   for i in range(1,q):
169     rotation_angle(asterix,max_angle, theta, z)
170     initial_pos(asterix,theta,z)
171   # finally rotate by r
172   rotation_angle(asterix,r, theta, z)
173   initial_pos(asterix,theta,z)
174
175
176 """ ------------------------------------------------------------------------------ """
```

rotation.py

```
1  import time #used for the sleep function
2  from pypot.dynamixel import autodetect_robot  #used to get the robot object
3  import pypot.dynamixel  #used to get the motors,legs etc.
4  import math #quite obvious
5  import json #to use a json file
6  from contextlib import closing  #to close properly the robot at  the end
7  import pypot.robot
8  import rotation
9
10 import Tkinter as tk # to get the a graphic interface for the control function
11
12
```

```
13 asterix = None
14 legs = []
15 initial = []
16
17 def leg_ik(x3,y3,z3,alpha = 20.69, beta = 5.06,l1=51,l2=63.7,l3=93):
18     d13 = math.sqrt(x3*x3 + y3*y3) - l1
19     d = math.sqrt(d13*d13 + z3*z3)
20     tmp = (l2**2 + d**2 - l3**2)/(2*l2*d)
21     a1 = z3 / d13
22     a2 = (l2**2 + l3**2 - d**2)/(2*l2*l3)
23
24     angles = (0,0,0)
25     theta1 = angles[0]
26     theta2 = angles[1]
27     theta3 = angles[2]
28
29     try:
30         theta1 = math.degrees(math.atan2(y3,x3))    # OK
31         theta2 = math.degrees(math.atan(a1) + math.acos(tmp))
32         theta3 = 180 - math.degrees(math.acos(a2))
33         # Corrections to the angles theta2 and theta3
34         theta2 = -(theta2 + alpha)
35         theta3 = -(theta3 - 90 + alpha + beta)
36         angles = (theta1,theta2,theta3)
37     except ValueError:
38         print "The legs of the robot cannot go that far!!"
39
40     return angles
41
42 """
43 Get the legs of the given robot object (from the json file).
44 """
45 def get_legs(obj):
46   return [obj.leg1,obj.leg2,obj.leg3,obj.leg4,obj.leg5,obj.leg6]
47
48 """
49   Makes one leg move.
50   parameters:
51     L -- The length between the start point and the end point (in a right line)
52     leg -- The leg we want to move
53     initial -- a tuple with three values wich correspond to the intial coordonnate of the end
         of the leg
54 """
55 def move_leg(L,z,leg):
56   num = int(leg[0].id*0.1)-1
57   theta = math.atan(initial[num][1]/initial[num][0])
58   hypo = math.sqrt(initial[num][0]**2 + initial[num][1]**2)
59   x = math.cos(theta)*(hypo+L)
60   y = math.sin(theta)*(hypo+L)
61   z = z
62   angles = leg_ik(x,y,z)
```

```
63    i=0
64    for motors in leg:
65      motors.goal_position = angles[i]
66      i+=1
67  """
68  Make the robot move along his two separate legs
69  """
70  def move_center_forward(L,z):
71    break_length = 1
72    theta = 20  #more than 20 would make the legs touch for a sec (because of the speed)
73    order = [1,5,2,4]
74    if L<0:
75      order = [4,2,5,1]
76
77    move_leg(L,z+40,legs[0])
78    move_leg(-L,z+40,legs[3])
79    time.sleep(break_length)
80
81    move_leg(L,z,legs[0])
82    move_leg(-L,z,legs[3])
83    time.sleep(break_length)
84    for i in order:
85      if i==order[0] or i==order[2]:
86        rotation.move_leg(-theta,z+40,legs[i])
87      else:
88        rotation.move_leg(theta,z+40,legs[i])
89        time.sleep(break_length)
90    for i in order:
91      if i==order[0] or i==order[2]:
92        rotation.move_leg(-theta,z,legs[i])
93      else:
94        rotation.move_leg(theta,z,legs[i])
95        time.sleep(break_length)
96    time.sleep(break_length)
97
98  """
99  THEORICAL WORK: The order of the leg or the direction could be wrong...TO TEST
100 Make the robot move along its two legged side.
101 """
102 def move_center_aside(L,z):
103
104   break_length = 1
105   theta = 20
106   if L<0:
107     theta = -theta
108
109   initial = rotation.initial_pos(0,-60)
110   time.sleep(break_length)
111   move_leg(L,z+40,legs[1])
112   move_leg(L,z+40,legs[2])
113   time.sleep(break_length)
```

```
114   move_leg(L,z,legs[1])
115   move_leg(L,z,legs[2])
116
117
118   move_leg(-L,z+40,legs[4])
119   move_leg(-L,z+40,legs[5])
120   time.sleep(break_length)
121   move_leg(-L,z,legs[4])
122   move_leg(-L,z,legs[5])
123   time.sleep(break_length)
124
125   rotation.move_leg(theta,z+40,legs[0])
126   rotation.move_leg(theta,z+40,legs[3])
127   time.sleep(break_length)
128   rotation.move_leg(theta,z,legs[0])
129   rotation.move_leg(theta,z,legs[3])
130   time.sleep(break_length)
131
132 def moving_all_legs(L,z):
133   move_leg(L,z,legs[0])
134   move_leg(L,z,legs[1])
135   move_leg(L,z,legs[2])
136   move_leg(-L,z,legs[3])
137   move_leg(L,z,legs[4])
138   move_leg(L,z,legs[5])
```

walk.py

```
1  import walk as walk
2  import rotation as rotation
3
4  import itertools
5  import time
6  import numpy
7  from pypot.dynamixel import autodetect_robot
8  import pypot.dynamixel
9  import math
10 import json
11 import time
12 from contextlib import closing
13 import Tkinter as tk
14
15 import pypot.robot
16
17 asterix = None
18 legs = []
19
20
21 def get_object():
22   asterix = pypot.robot.from_json('my_robot.json')
23   legs = get_legs(asterix)
24   rotation.legs = get_legs(asterix)
```

```
25    walk.legs = get_legs(asterix)
26
27    return asterix
28
29  def detection():
30
31    my_robot = autodetect_robot() #detect al the legs of the robot. Might take a while to operate
          .
32
33    #write the configuration found into a json file. We shouldn't use the complete detection
          whith this json file.
34    config = my_robot.to_config()
35    with open('my_robot.json', 'wb') as f:
36        json.dump(config, f)
37
38    with closing(pypot.robot.from_json('my_robot.json')) as my_robot:
39        # do stuff without having to make sure not to forget to close my_robot!
40        pass
41
42  def initialize():
43
44    asterix = pypot.robot.from_json('my_robot.json')
45    # print asterix
46    # Note that all these calls will return immediately,
47    # and the orders will not be directly sent
48    # (they will be sent during the next write loop iteration).
49    for m in asterix.motors:
50        print m.present_position
51        m.compliant = False    # <=> enable_torque.
52        m.goal_position = 0
53      # with closing(pypot.robot.from_json('my_robot.json')) as my_robot:
54      #   # do stuff without having to make sure not to forget to close my_robot!
55      #   pass
56
57    time.sleep(2)
58    return asterix
59
60  def get_legs(obj):
61      return [obj.leg1,obj.leg2,obj.leg3,obj.leg4,obj.leg5,obj.leg6]
62
63  if __name__ == '__main__':
64
65    asterix = get_object()
66    initialize()
67    walk.initial = rotation.initial_pos(asterix,0,-60)
68    # time.sleep(2)
69    # walk.move_leg(30,0,rotation.legs[0])
70    # time.sleep(1)
71    # walk.move_leg(30,-60,rotation.legs[0])
72    # time.sleep(1)
73    while 1:
```

```
74      walk.initial = rotation.initial_pos(asterix,0,-60)
75      time.sleep(0.2)
76      walk.move_center_forward(30,-60)
77      walk.initial = rotation.initial_pos(asterix,0,-60)
78      time.sleep(0.2)
79
80      # print rotation.legs[0][0].id
81    # rotation.move_leg(0,-60,rotation.legs[0])
82    # rotation.arbitrary_rotation(asterix,720)
```

main.py