



Collège **Sciences et technologies**

Introduction à la robotique par la pratique

DOCUMENTATION DE PROJET

Étudiants :

Corentin Charles

Clément Renazeau

Þorsteinn Hjörtur Jónsson

Enseigneurs :

Rémi Fabre

31 mars 2015

1 Introduction

This is the introduction.

2 Rotation

2.1 Description

To make Asterix do a rotation we need to define a circle of rotation, with regards to the center of the robot, on which the legs will move on. As we should always use inverse kinematics to decide the motor angles we will need to express the the circle of rotation in each legs' coordinate frame. To do this we'll need to translate the coordinates of the center system by the vector (x_{corr}, y_{corr}) , i.e. a point (x_c, y_c) in the center system has the coordinates $(x_l + x_{corr}, y_l + y_{corr})$ with respect to the coordinate system of the leg.

The following table shows measurements of this vector.

Let R_c be the radius of the circle of rotation. We can define the circle of rotation by $(R_c \cos(\theta_c), R_c \sin(\theta_c))$ for $\theta_c \in [0, 2\pi]$.

Let R_l be the distance from the tip of a leg to the origin of the coordinate system which it defines and let $\theta_l \in [0, 2\pi]$ such that $(R_l \cos(\theta_l), R_l \sin(\theta_l))$ describes the tip of the leg.

Assume that the tip of a leg is on the circle of rotation. Then,

$$(R_c \cos(\theta_c), R_c \sin(\theta_c)) = (R_l \cos(\theta_l) + x_{corr}, R_l \sin(\theta_l) + y_{corr}).$$

Therefore,

2.2 Program

```

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
9 asterix = None
10 legs = []
11 xCorrection = [0,0,0,0,0,0]
12 yCorrection = [0,0,0,0,0,0]
13
14
15
16 def leg_ik(x3,y3,z3,alpha = 20.69, beta = 5.06,l1=51,l2=63.7,l3=93):
17     d13 = math.sqrt(x3*x3 + y3*y3) - l1
18     d = math.sqrt(d13*d13 + z3*z3)
19     tmp = (l2**2 + d**2 - l3**2)/(2*l2*d)
20     a1 = z3 / d13
21     a2 = (l2**2 + l3**2 - d**2)/(2*l2*l3)
22
23     angles = (0,0,0)
24     theta1 = angles[0]
```

```

25     theta2 = angles[1]
26     theta3 = angles[2]
27
28     try:
29         theta1 = math.degrees(math.atan2(y3,x3))    # OK
30         theta2 = math.degrees(math.atan(a1) + math.acos(tmp))
31         theta3 = 180 - math.degrees(math.acos(a2))
32         # Corrections to the angles theta2 and theta3
33         theta2 = -(theta2 + alpha)
34         theta3 = -(theta3 - 90 + alpha + beta)
35         angles = (theta1,theta2,theta3)
36     except ValueError:
37         print "The legs of the robot cannot go that far!!"
38
39     return angles
40
41 def get_legs(obj):
42     return [obj.leg1, obj.leg2, obj.leg3, obj.leg4, obj.leg5, obj.leg6]
43
44 def get_xCorrection(leg):
45     i = int(leg[0].id*0.1)
46     return xCorrection[i-1]
47
48 def get_yCorrection(leg):
49     i = int(leg[0].id*0.1)
50     return yCorrection[i-1]
51
52 """----- Rotation Functions -----"""
53 """ Written by Thor the 24/03/15 """
54 """ Tested by Corentin the 24/03/15 """
55
56
57 """ -----Mathematics to correct the rotation -----"""
58 """ Written by Thor the 26/04/15"""
59 # This needs to be done so that we can define a common circle of rotation
60 # for all the legs. To communicate this common information to all the legs
61 # we need to express the radius of this common circle of rotation as a function of theta and
62 # the legs
63 """ I added the attributes needed to obtain this information to the json file
64 Currently these attributes are set to zero"""
65 def R_leg(theta, leg, R_center):
66     xCorrection = get_xCorrection(leg)
67     yCorrection = get_yCorrection(leg)
68     cos = math.cos(math.radians(theta))
69     sin = math.sin(math.radians(theta))
70     tmp = (xCorrection**2)*((cos**2)-1)
71     tmp += (yCorrection**2)*((sin**2)-1)
72     tmp += xCorrection*yCorrection*math.sin(math.radians(2*theta))
73     tmp += R_center**2
74     return (-xCorrection*cos - yCorrection*sin + math.sqrt(tmp))

```

```

75
76 # This function takes care of 1 leg at a time
77 # This moves the leg given polar coordinates. Important because we when we need to do a
    rotation the legs should not move
78 # outside the circle of rotation. We want a perfect rotation!
79 # TEST : Working perfectly
80 def move_leg(theta,z,leg,R_center = 100):
81
82     i=0
83     # Tupl is a vector that carries the angles that represent the final position of the tip of
        the leg
84     # The angles are calculated from the arguments of the function using inverse kinematics
85     # R is the radius of the circle of rotation. Theta is given in degrees.
86     # Lets transform our polar coordinates onto the Cartesian plane
87     # print R_leg(theta,leg,R), " - ", leg[0].id
88     x = R_leg(theta,leg,R_center)*math.cos(math.radians(theta))
89     y = R_leg(theta,leg,R_center)*math.sin(math.radians(theta))
90     motor_angles = leg_ik(x,y,z)
91     for m in leg:
92         m.goal_position = motor_angles[i]
93         i+=1
94     return (x,y,z)
95
96 # This should just give us our initial spider position
97 # We also use this function when rotating to refix the legs' frames of reference
98 """SOLVED?"""
99 #TEST : We SHOULD NOT put negative value in this function (otehrwise the legs (except legs 1-4)
    will 'meet each other')
100 def initial_pos(asterix,theta,z):
101     # Experiments have shown that using the values 100 and 30 for changing x and y respectively
        is working okay
102     initial_position = []
103     initial_position.append(move_leg(0,z,legs[0]))
104     initial_position.append(move_leg(-abs(theta),z,legs[1]))
105     initial_position.append(move_leg(abs(theta),z,legs[2]))
106     initial_position.append(move_leg(0,z,legs[3]))
107     initial_position.append(move_leg(-abs(theta),z,legs[4]))
108     initial_position.append(move_leg(abs(theta),z,legs[5]))
109
110     time.sleep(0.1)
111
112     return initial_position
113
114 """
115 TODO: make sure that this works. If it works than we can easily do experiments to find the
    highest value on alpha
116 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
117 See the draft implementation for arbitrary_rotation above.
118 """
119 # theta is the value we need to set the initial position

```

```

120 # alpha determines the amount of rotation (made by each call to the function) from this initial
    position
121 # alpha is physically limited because of the legs. We should define this limit as max_angle -
    see above.
122 # 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)
123 def rotation_angle(asterix, alpha, theta, z):
124     #clockwise 2 and 5 are limited
125     breaklength = 0.1
126     # Position 1: The 'spider' position. This position has a low center of gravity.
127     # Here we define the initial position. i.e. the spider position
128     # It is important to observe the x and y values of each leg in its own frame of reference
129
130     # Position 2: Put legs 1, 3, 5 in the air and rotate at the same time
131     move_leg(-abs(theta)+alpha, z+20, legs[1])
132     move_leg(alpha, z+20, legs[3])
133     move_leg(abs(theta)+alpha, z+20, legs[5])
134     time.sleep(breaklength)
135
136     # Position 3: Put legs 1,3 and 5 down
137     move_leg(-abs(theta)+alpha, z, legs[1])
138     move_leg(alpha, z, legs[3])
139     move_leg(abs(theta)+alpha, z, legs[5])
140     time.sleep(breaklength)
141
142     # Position 4: Rotate legs 0, 2, 4
143     move_leg(alpha, z+20, legs[0])
144     move_leg(abs(theta)+alpha, z+20, legs[2])
145     move_leg(-abs(theta)+alpha, z+20, legs[4])
146     time.sleep(breaklength)
147
148     # Position 5: Put legs 0, 2 and 4 down.
149     move_leg(alpha, z, legs[0])
150     move_leg(abs(theta)+alpha, z, legs[2])
151     move_leg(-abs(theta)+alpha, z, legs[4])
152     time.sleep(breaklength)
153
154 # max_angle = 20 is just a guess.
155 # 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
156 """SOLVED?"""
157 #TEST : If we put negative value for the beta angle, this is just not working.
158 # TEST : If the value of max_angle is not 20, the rotation does not work properly
159 # theta and z are simply values that determine the initial position
160 # Other parameters are to define the rotation
161 def arbitrary_rotation(asterix, beta, max_angle = 20, theta = 45, z = -60):
162     # Here we do euclidean division. We determine how often max_angle divides beta and the
        remainder of this division.
163     # This gives us the number of rotations we need to make by a predefined max_angle
164     # The remainder gives us the amount we need to rotate by to be able to finish the full rotation
        by an angle of beta

```

```
165 # i.e. beta = q*max_angle + r
166     initial_pos(asterix,theta,z)
167     if beta < 0:
168         max_angle = -max_angle
169
170     q = beta//max_angle
171     r = beta%max_angle
172     print q
173     print r
174     # rotate by max_angle q times
175     for i in range(1,q):
176         rotation_angle(asterix,max_angle, theta, z)
177         initial_pos(asterix,theta,z)
178     # finally rotate by r
179     rotation_angle(asterix,r, theta, z)
180     initial_pos(asterix,theta,z)
181
182
183 """ _____ """
```

rotation.py

3 Walk

3.1 Description

3.2 Program

```

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
54               of the leg
55 """
56 def move_leg(L,z,leg):
57     num = int(leg[0].id*0.1)-1
58     theta = math.atan(initial[num][1]/initial[num][0])
59     hypo = math.sqrt(initial[num][0]**2 + initial[num][1]**2)
60     x = math.cos(theta)*(hypo+L)
61     y = math.sin(theta)*(hypo+L)
62     z = z
63     angles = leg_ik(x,y,z)
64     i=0
65     for motors in leg:
66         motors.goal_position = angles[i]
67         i+=1
68 """
69 Make the robot move along his two separate legs
70 """
71 def move_center_forward(L,z):
72     break_length = 1
73     theta = 20 #more than 20 would make the legs touch for a sec (because of the speed)
74     order = [1,5,2,4]
75     if L<0:
76         order = [4,2,5,1]
77
78     move_leg(L,z+40,legs[0])
79     move_leg(-L,z+40,legs[3])
80     time.sleep(break_length)
81
82     move_leg(L,z,legs[0])
83     move_leg(-L,z,legs[3])
84     time.sleep(break_length)
85     for i in order:
86         if i==order[0] or i==order[2]:
87             rotation.move_leg(-theta,z+40,legs[i])
88         else:
89             rotation.move_leg(theta,z+40,legs[i])
90             time.sleep(break_length)
91     for i in order:
92         if i==order[0] or i==order[2]:
93             rotation.move_leg(-theta,z,legs[i])
94         else:
95             rotation.move_leg(theta,z,legs[i])
96             time.sleep(break_length)

```

```

96     time.sleep(break_length)
97
98     """
99     THEORETICAL 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

4 Main

4.1 Description

4.2 Program

```
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
34     #write the configuration found into a json file. We shouldn't use the complete detection
35     #with this json file.
36     config = my_robot.to_config()
37     with open('my_robot.json', 'wb') as f:
38         json.dump(config, f)
39
40     with closing(pypot.robot.from_json('my_robot.json')) as my_robot:
41         # do stuff without having to make sure not to forget to close my_robot!
42         pass
43
44 def initialize():
```

```

44     asterix = get_object()
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         m.compliant = False    # <=> enable_torque.
51         m.goal_position = 0
52
53     time.sleep(2)
54     return asterix
55 """
56     if asterix['motorgroups'] == None:
57         asterix['motorgroups'] = {
58             'leg1': ["motor_11","motor_12","motor_13"],
59             'leg2': ["motor_21","motor_22","motor_23"],
60             'leg3': ["motor_31","motor_32","motor_33"],
61             'leg4': ["motor_41","motor_42","motor_43"],
62             'leg5': ["motor_51","motor_52","motor_53"],
63             'leg6': ["motor_61","motor_62","motor_63"]
64         }
65 """
66
67 def get_legs(obj):
68     return [obj.leg1 , obj.leg2 , obj.leg3 , obj.leg4 , obj.leg5 , obj.leg6 ]
69
70 if __name__ == '__main__':
71
72     # asterix = get_object()
73     initialize()
74     walk.initial = rotation.initial_pos(asterix,30,-60)
75     #rotation.arbitrary_rotation(asterix,360)
76     # time.sleep(2)
77     # walk.move_leg(30,0,rotation.legs[0])
78     # time.sleep(1)
79     # walk.move_leg(30,-60,rotation.legs[0])
80     # time.sleep(1)
81
82     # while 1:
83     #     move_center_aside(10,-60)
84
85     # while 1:
86     #     walk.initial = rotation.initial_pos(asterix,0,-60)
87     #     time.sleep(0.2)
88     #     walk.move_center_forward(30,-60)
89     #     walk.initial = rotation.initial_pos(asterix,0,-60)
90     #     time.sleep(0.2)
91
92     # print rotation.legs[0][0].id
93     rotation.move_leg(0,-60,rotation.legs[0])

```

main.py