



Collège **Sciences et technologies**

Introduction à la robotique par la pratique

DOCUMENTATION DE PROJET

Étudiants :

Corentin Charles

Clément Renazeau

Þorsteinn Hjörtur Jónsson

Enseignants :

Rémi Fabre

Steve Nguyen

1^{er} avril 2015

1 Rotation

1.1 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,

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

So we have two equations :

$$R_c \cos(\theta_c) = R_l \cos(\theta_l) + x_{corr} \quad (1)$$

$$R_c \sin(\theta_c) = R_l \sin(\theta_l) + y_{corr} \quad (2)$$

By putting the both equations to the power of two and adding them with each other we obtain the following second degree polynomial equation :

$$R_l^2 + 2R_l (\cos(\theta_l) + x_{corr} + \sin(\theta_l) + y_{corr}) - R_c^2 + x_{corr}^2 + y_{corr}^2 = 0$$

The positive root of this equation is :

$$R_l = -x_{corr} \cos(\theta_l) - y_{corr} \sin(\theta_l) + \sqrt{x_{corr}^2 (\cos^2(\theta_l) - 1) + y_{corr}^2 (\sin^2(\theta_l) - 1) + x_{corr} y_{corr} \sin(2\theta_l)}$$

So this should give us the radius of each leg as a function of the correction and the angle given.

1.2 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 legs = []
10 xCorrection = [-10,-20,-20,10,-20,-20]
11 yCorrection = [0,-15,15,0,15,-15]
12
13
14 """
15 Indirect kinematic function.
16 Parameters :
17     - (x3,y3,z3) : The coordonnates where we want to put the leg
18     - alpha : the correction for the second motor.
19     - beta : the correction for the third motor.
20     - l2, l3 : the length of the different part of the leg
21 Return a tuple with three values, corresponding to the angles of each motor of the leg.
22 """
23 def leg_ik(x3,y3,z3,alpha = 20.69, beta = 5.06,l1=51,l2=63.7,l3=93):
24     d13 = math.sqrt(x3*x3 + y3*y3) - l1
25     d = math.sqrt(d13*d13 + z3*z3)
26     tmp = (l2**2 + d**2 - l3**2)/(2*l2*d)
27     a1 = z3 / d13
28     a2 = (l2**2 + l3**2 - d**2)/(2*l2*l3)
29
30     angles = (0,0,0)
31     theta1 = angles[0]
32     theta2 = angles[1]
33     theta3 = angles[2]
34
35     try:
36         theta1 = math.degrees(math.atan2(y3,x3)) # OK
37         theta2 = math.degrees(math.atan(a1) + math.acos(tmp))
38         theta3 = 180 - math.degrees(math.acos(a2))
39         # Corrections to the angles theta2 and theta3
40         theta2 = -(theta2 + alpha)
41         theta3 = -(theta3 - 90 + alpha + beta)
42         angles = (theta1,theta2,theta3)
43     except ValueError:
44         print "The leg of the robot cannot go that far!!"
45
46     return angles
47
48
49 def get_legs(obj):

```

```

50     """
51     Return a list with all the legs of the robot passed in parameter, i.e a leg is three motors.
52     The motorgroups is actually done manually.
53     """
54     return [obj.leg1 ,obj.leg2 ,obj.leg3 ,obj.leg4 ,obj.leg5 ,obj.leg6 ]
55
56 def get_xCorrection(leg):
57     """
58     Return the correction (in mm) for the x axis of a specified leg
59     Parameters :
60     - leg : The leg which is going to be moved
61     """
62     i = int(leg[0].id*0.1)
63     return xCorrection[i-1]
64
65
66 def get_yCorrection(leg):
67     """
68     Return the correction (in mm) for the y axis of a specified leg
69     Parameters :
70     - leg : The leg which is going to be moved
71     """
72     i = int(leg[0].id*0.1)
73     return yCorrection[i-1]
74
75 """----- Rotation Functions -----"""
76 """ Written by Thor the 24/03/15 """
77 """ Tested by Corentin the 24/03/15 """
78
79
80 """ -----Mathematics to correct the rotation -----"""
81 """ Written by Thor the 26/04/15"""
82 # This needs to be done so that we can define a common circle of rotation
83 # for all the legs. To communicate this common information to all the legs
84 # we need to express the radius of this common circle of rotation as a function of theta and
85     the legs
86 """ I added the attributes needed to obtain this information to the json file
87 Currently these attributes are set to zero"""
88 def R_leg(theta,leg,R_center):
89     """
90
91     """
92     xCorrection = get_xCorrection(leg)
93     yCorrection = get_yCorrection(leg)
94     cos = math.cos(math.radians(theta))
95     sin = math.sin(math.radians(theta))
96     tmp = (xCorrection**2)*((cos**2)-1)
97     tmp += (yCorrection**2)*((sin**2)-1)
98     tmp += xCorrection*yCorrection*math.sin(2*math.radians(theta))

```

```

99     tmp += R_center**2
100     return (-xCorrection*cos - yCorrection*sin + math.sqrt(tmp))
101
102
103 # This function takes care of 1 leg at a time
104 # This moves the leg given polar coordinates. Important because we when we need to do a
    rotation the legs should not move
105 # outside the circle of rotation. We want a perfect rotation!
106 # TEST : Working perfectly
107 def move_leg(theta,z,leg,R_center = 100):
108     """
109     Do a rotation on one leg.
110     Parameters :
111         - theta : the angle we want the leg to do
112         - z : the height of the tip of the leg.
113         - leg : the leg we want to move
114         - R_center : The radius of the circle which the center is equal to center of the entire
            robot.
115     Return a tuple with the coordonnates of the leg.
116     """
117     i=0
118     # Tupl is a vector that carries the angles that represent the final position of the tip of
        the leg
119     # The angles are calculated from the arguments of the function using inverse kinematics
120     # R is the radius of the circle of rotation. Theta is given in degrees.
121     # Lets transform our polar coordinates onto the Cartesian plane
122     # print R_leg(theta,leg,R), " - ", leg[0].id
123     x = R_leg(theta,leg,R_center)*math.cos(math.radians(theta))
124     y = R_leg(theta,leg,R_center)*math.sin(math.radians(theta))
125     motor_angles = leg_ik(x,y,z)
126     for m in leg:
127         m.goal_position = motor_angles[i]
128         i+=1
129     return (x,y,z)
130
131 # This should just give us our initial spider position
132 # We also use this function when rotating to refix the legs' frames of reference
133 """SOLVED?"""
134 #TEST : We SHOULD NOT put negative value in this function (otehrwise the legs (except legs 1-4)
    will 'meet each other')
135 def initial_pos(theta,z):
136     """
137     Put the robot in an initial position.
138     Parameters :
139         - theta : The initial angle of the leg from on its own axis.
140         - z : The height of the leg.
141     """
142     # Experiments have shown that using the values 100 and 30 for changing x and y respectively
        is working okay
143     initial_position = []
144     initial_position.append(move_leg(0,z,legs[0]))

```

```

145     initial_position.append(move_leg(-abs(theta),z,legs[1]))
146     initial_position.append(move_leg(abs(theta),z,legs[2]))
147     initial_position.append(move_leg(0,z,legs[3]))
148     initial_position.append(move_leg(-abs(theta),z,legs[4]))
149     initial_position.append(move_leg(abs(theta),z,legs[5]))
150
151     time.sleep(0.1)
152
153     return initial_position
154
155 """
156 TODO: make sure that this works. If it works than we can easily do experiments to find the
157       highest value on alpha
158 If we know the highest value of alpha we can determine the number of turns needed to do an
159       arbitrary amount of rotation by using Euclidean division
160 See the draft implementation for arbitrary_rotation above.
161 """
162 # theta is the value we need to set the initial position
163 # alpha determines the amount of rotation (made by each call to the function) from this initial
164     position
165 # alpha is physically limited because of the legs. We should define this limit as max_angle -
166     see above.
167 # TEST : A value of 45 will make the legs (2-3 and 4-5) touch for a little while (actually
168     until the next leg move)
169 def rotation_angle(alpha,theta,z):
170     """
171     Do a rotation on all the legs of the robot.
172     Parameters :
173         - alpha :
174         - theta : The angle we want the legs to do.
175         - z : the height of the leg.
176     """
177     #clockwise 2 and 5 are limited
178     breaklength = 0.1
179     # Position 1: The 'spider' position. This position has a low center of gravity.
180     # Here we define the initial position. i.e. the spider position
181     # It is important to observe the x and y values of each leg in its own frame of reference
182
183     # Position 2: Put legs 1, 3, 5 in the air and rotate at the same time
184     move_leg(-abs(theta)+alpha,z+20,legs[1])
185     move_leg(alpha,z+20,legs[3])
186     move_leg(abs(theta)+alpha,z+20,legs[5])
187     time.sleep(breaklength)
188
189     # Position 3: Put legs 1,3 and 5 down
190     move_leg(-abs(theta)+alpha,z,legs[1])
191     move_leg(alpha,z,legs[3])
192     move_leg(abs(theta)+alpha,z,legs[5])
193     time.sleep(breaklength)
194
195     # Position 4: Rotate legs 0, 2, 4

```

```

191 move_leg(alpha , z+20, legs [0])
192 move_leg(abs(theta)+alpha , z+20, legs [2])
193 move_leg(-abs(theta)+alpha , z+20, legs [4])
194 time.sleep(breaklength)
195
196 # Position 5: Put legs 0, 2 and 4 down.
197 move_leg(alpha , z, legs [0])
198 move_leg(abs(theta)+alpha , z, legs [2])
199 move_leg(-abs(theta)+alpha , z, legs [4])
200 time.sleep(breaklength)
201
202 # max_angle = 20 is just a guess.
203 # 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
204 """SOLVED?"""
205 #TEST : If we put negative value for the beta angle, this is just not working.
206 # TEST : If the value of max_angle is not 20, the rotation does not work properly
207 # theta and z are simply values that determine the initial position
208 # Other parameters are to define the rotation
209 def arbitrranany_rotation(beta, max_angle = 10, theta = 45, z = -60):
210     """
211         Do a fully operational rotation on the robot (the return to initial position is in this
            function)
212         Parameters :
213             - beta : The angle we want the center of the robot to do.
214             - max_angle : The maximum angle the robot can do in one loop.
215             - theta : The rotation of each leg.
216             - z : the height of the robot.
217     """
218 # Here we do euclidean division. We determine how often max_angle divides beta and the
        remainder of this division.
219 # This gives us the number of rotations we need to make by a predefined max_angle
220 # The remainder gives us the amount we need to rotate by to be able to finish the full rotation
        by an angle of beta
221 # i.e. beta = q*max_angle + r
222 beta = 2*beta
223 initial_pos(theta , z)
224 if beta < 0:
225     max_angle = -max_angle
226
227 q = beta//max_angle
228 r = beta%max_angle
229 print q
230 print r
231 # rotate by max_angle q times
232 for i in range(1,q):
233     rotation_angle(max_angle, theta , z)
234     initial_pos(theta , z)
235 # finally rotate by r
236 rotation_angle(r, theta , z)
237 initial_pos(theta , z)

```

238

239

"""

rotation.py

"""

2 Walk

2.1 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 legs = []
14 initial = []
15
16
17 """
18 Indirect kinematic function.
19 Parameters :
20 - (x3,y3,z3) : The coordonnates where we want to put the leg
21 - alpha : the correction for the second motor.
22 - beta : the correction for the third motor.
23 - l2, l3 : the length of the different part of the leg
24 Return a tuple with three values, corresponding to the angles of each motor of the leg.
25 """
26 def leg_ik(x3,y3,z3,alpha = 20.69, beta = 5.06,l1=51,l2=63.7,l3=93):
27     d13 = math.sqrt(x3*x3 + y3*y3) - l1
28     d = math.sqrt(d13*d13 + z3*z3)
29     tmp = (l2**2 + d**2 - l3**2)/(2*l2*d)
30     a1 = z3 / d13
31     a2 = (l2**2 + l3**2 - d**2)/(2*l2*l3)
32
33     angles = (0,0,0)
34     theta1 = angles[0]
35     theta2 = angles[1]
36     theta3 = angles[2]
37
38     try:
39         theta1 = math.degrees(math.atan2(y3,x3)) # OK
40         theta2 = math.degrees(math.atan(a1) + math.acos(tmp))
41         theta3 = 180 - math.degrees(math.acos(a2))
42         # Corrections to the angles theta2 and theta3
43         theta2 = -(theta2 + alpha)
44         theta3 = -(theta3 - 90 + alpha + beta)
45         angles = (theta1,theta2,theta3)
46     except ValueError:
47         print "The leg of the robot cannot go that far!!"

```

```

48
49     return angles
50
51 """
52     Return a list with all the legs of the robot passed in parameter, i.e a leg is three motors.
53     The motorgroups is actually done manually.
54 """
55 def get_legs(obj):
56     return [obj.leg1 , obj.leg2 , obj.leg3 , obj.leg4 , obj.leg5 , obj.leg6 ]
57
58 """
59     Makes one leg move.
60     parameters:
61     - L : The length between the start point and the end point of each leg (in a straight line)
62     - z : The height of the center of the robot.
63     - leg : The leg we want to move
64 """
65 def move_leg(L,z,leg):
66     num = int(leg[0].id*0.1)-1
67     theta = math.atan(initial[num][1]/initial[num][0])
68     hypo = math.sqrt(initial[num][0]**2 + initial[num][1]**2)
69     x = math.cos(theta)*(hypo+L)
70     y = math.sin(theta)*(hypo+L)
71     z = z
72     angles = leg_ik(x,y,z)
73     i=0
74     for motors in leg:
75         motors.goal_position = angles[i]
76         i+=1
77
78 """
79     Make the robot move along his two separate legs
80     Parameters:
81     - L : The length between the start point and the end point of each leg (in a straight line)
82     - z : The height of the center of the robot.
83 """
84 def move_center_forward(L,z,max_length=50):
85     print "move_center_forward({0},{1},...)".format(L,z)
86
87     break_length = 2
88     theta = 20 #more than 20 would make the legs touch for a sec (because of the speed)
89     order = [1,5,2,4]
90
91     if L>max_length:
92         move_center_forward(L-max_length,z)
93         L = L -max_length
94
95     initial = rotation.initial_pos(30,z)
96     if L<0:

```

```

96     order = [4,2,5,1]
97     move_leg(L,z+40,legs[0])
98     print "move_leg({0},{1},...)".format(L,z)
99     move_leg(-L,z+40,legs[3])
100    print "here 3"
101    time.sleep(break_length)
102
103    move_leg(L,z,legs[0])
104    move_leg(-L,z,legs[3])
105    time.sleep(break_length)
106
107    rotation.move_leg(-theta,z+40,legs[order[0]])
108    rotation.move_leg(theta,z+40,legs[order[1]])
109    time.sleep(break_length)
110    rotation.move_leg(-theta,z,legs[order[0]])
111    rotation.move_leg(theta,z,legs[order[1]])
112    time.sleep(break_length)
113
114    rotation.move_leg(-theta,z+40,legs[order[2]])
115    rotation.move_leg(theta,z+40,legs[order[3]])
116    time.sleep(break_length)
117    rotation.move_leg(-theta,z,legs[order[2]])
118    rotation.move_leg(theta,z,legs[order[3]])
119    time.sleep(break_length)
120
121    """
122    THEORETICAL WORK: The order of the leg or the direction could be wrong...TO TEST
123    Make the robot move along its two legged side.
124    """
125    theta = 20
126
127    if L>max_length:
128        move_center_aside(L-max_length,z)
129        L = L -max_length
130
131    if L<0:
132        theta = -theta
133
134    initial = rotation.initial_pos(30,z)
135    time.sleep(break_length)
136    print "putting legs 2-3 in the air"
137    print "move_leg({0},{1},...)".format(L,z+40)
138    move_leg(L,z+40,legs[1])
139    move_leg(L,z+40,legs[2])
140    time.sleep(break_length)
141    move_leg(L,z,legs[1])
142    move_leg(L,z,legs[2])
143    time.sleep(break_length)
144
145    print "putting legs 5-6 on the ground"
146    move_leg(-L,z+40,legs[4])

```

```
147     move_leg(-L, z+40, legs [5])
148     time.sleep(break_length)
149     move_leg(-L, z, legs [4])
150     move_leg(-L, z, legs [5])
151     time.sleep(break_length)
152
153     print "rotating the legs 1-4 and putting them in the air"
154     rotation.move_leg(theta, z+40, legs [0])
155     rotation.move_leg(-theta, z+40, legs [3])
156     time.sleep(break_length)
157     print "rotating the legs 1-4 and putting them on the ground"
158     rotation.move_leg(theta, z, legs [0])
159     rotation.move_leg(-theta, z, legs [3])
160     time.sleep(break_length)
161 #move the center with leg 2, 4, 6 in the air
162
163
164 """
165     Move all the leg in their own coordonnate system.
166     Parameters:
167         - L : The length between the start point and the end point of each leg (in a straight line)
168         .
169         - z : THE height of the center of the robot.
170 """
171 def moving_all_legs(L, z):
172     move_leg(L, z, legs [0])
173     move_leg(L, z, legs [1])
174     move_leg(L, z, legs [2])
175     move_leg(-L, z, legs [3])
176     move_leg(L, z, legs [4])
177     move_leg(L, z, legs [5])
```

walk.py

3 Main

3.1 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 from Tkinter import *
14
15 import pypot.robot
16
17 asterix = None
18 legs = []
19
20
21
22 def get_object():
23     """
24     Return a robot object created from a json file. initialize the legs variable in the three
25     files.
26     """
27     asterix = pypot.robot.from_json('my_robot.json')
28     legs = get_legs(asterix)
29     rotation.legs = get_legs(asterix)
30     walk.legs = get_legs(asterix)
31
32     return asterix
33
34 def detection():
35     """
36     Do the detection of the robot with all its motors. It puts the configuration into a json
37     file named 'my_robot.json'
38     """
39     my_robot = autodetect_robot() #detect al the legs of the robot. Might take a while to operate
40     .
41
42     #write the configuration found into a json file. We shouldn't use the complete detection
43     whith this json file.
44     config = my_robot.to_config()
45     with open('my_robot.json', 'wb') as f:
46         json.dump(config, f)
```

```

44
45 with closing(pypot.robot.from_json('my_robot.json')) as my_robot:
46     # do stuff without having to make sure not to forget to close my_robot!
47     pass
48
49 <<<<<<<< HEAD
50 """
51     Initialize the robot. Firstly get the robot object, and then put the angles of the motor at 0
52     deg.
53     Return the robot object.
54     """
55
56 >>>>>>> f49be2317d5b8553dc3cac64b48885436bbd60b1
57 def initialize():
58     """
59     Initialize the robot. Firstly get the robot object, and then put the angles of the motor at
60     0 deg.
61     Return the robot object.
62     """
63     asterix = get_object()
64     # print asterix
65     # Note that all these calls will return immediately,
66     # and the orders will not be directly sent
67     # (they will be sent during the next write loop iteration).
68     for m in asterix.motors:
69         m.compliant = False # <=> enable_torque.
70         # m.goal_position = 0
71     time.sleep(0.1)
72     return asterix
73 """
74 if asterix['motorgroups'] == None:
75     asterix['motorgroups'] = {
76         'leg1': ["motor_11", "motor_12", "motor_13"],
77         'leg2': ["motor_21", "motor_22", "motor_23"],
78         'leg3': ["motor_31", "motor_32", "motor_33"],
79         'leg4': ["motor_41", "motor_42", "motor_43"],
80         'leg5': ["motor_51", "motor_52", "motor_53"],
81         'leg6': ["motor_61", "motor_62", "motor_63"]
82     }
83 """
84
85 def get_legs(obj):
86     """
87     Return a list with all the legs of the robot passed in parameter, i.e a leg is three motors
88     . The motorgroups is actually done manually.
89     """
90     return [obj.leg1, obj.leg2, obj.leg3, obj.leg4, obj.leg5, obj.leg6]
91

```

```

92 #-----
93 #-----  events  function  -----
94 #-----
95
96
97 def forward(event):
98     """
99     Call the move_center_forward function with some defined values.
100     Parameters :
101     -- event : an event that 'catch' what key the users is pressing.
102     """
103     print 'forward'
104     z = -60
105     L = 30
106     theta = 0
107     break_length = 0.2
108     walk.initial = rotation.initial_pos(theta,z)
109     time.sleep(break_length)
110     walk.move_center_forward(L,z)
111     walk.initial = rotation.initial_pos(theta,z)
112     time.sleep(break_length)
113
114
115 def backward(event):
116     """
117     Call the move_center_forward function with some defined valuees (one is negative to go be
118     able to go backward).
119     Parameters :
120     -- event : an event that 'catch' what key the users is pressing.
121     """
122     print 'backward'
123     z = -60
124     L = -30
125     theta = 0
126     break_length = 0.2
127     walk.initial = rotation.initial_pos(theta,z)
128     time.sleep(break_length)
129     walk.move_center_forward(L,z)
130     walk.initial = rotation.initial_pos(theta,z)
131     time.sleep(break_length)
132
133 def left(event):
134     """
135     Call the move_center_aside function with some defined values.
136     Parameters :
137     -- event : an event that 'catch' what key the users is pressing.
138     """
139     z = -60
140     L = 30
141     theta = 0

```

```
142     break_length = 0.2
143     walk.move_center_aside(L,z)
144
145
146 def right(event):
147     """
148     Call the move_center_aside function with some defined values.
149     Parameters :
150     — event : an event that 'catch' what key the users is pressing.
151     """
152     z= -60
153     L = -30
154     theta = 0
155     break_length = 0.2
156     walk.move_center_aside(L,z)
157
158
159 def rotation_left(event):
160     """
161     Call the arbitrary_rotation function with some defined values.
162     Parameters :
163     — event : an event that 'catch' what key the users is pressing.
164     """
165     angle = 90
166     rotation.arbitrary_rotation(angle*2)
167
168
169 def rotation_right(event):
170     """
171     Call the arbitrary_rotation function with some defined values (one is negative).
172     Parameters :
173     — event : an event that 'catch' what key the users is pressing.
174     """
175     angle = -90
176     rotation.arbitrary_rotation(angle*2)
177
178
179 def position_initial(event):
180     """
181     Call the initial_pos function with some defined values , with a height of 60 cm.
182     Parameters :
183     — event : an event that 'catch' what key the users is pressing.
184     """
185     theta = 0
186     z = -60
187     rotation.initial_pos(theta,z)
188
189
190 def user_interaction():
191     """
192     Bind all the events to the minimal graphinc interface.
```



```

193     """
194     root = tk.Tk()
195     root.bind("<Up>", forward)
196     root.bind("<Down>", backward)
197     root.bind("<Right>", right)
198     root.bind("<Left>", left)
199     root.bind("<i>", rotation_left)
200     root.bind("<i>", rotation_right)
201     root.bind("<Return>", position_initial)
202     root.mainloop()
203
204
205 #-----
206 #----- Graphic interface -----
207 #-----
208
209 def create_interface():
210
211     window = Tk()
212
213     cadre = Frame(window, width=700, height=500, borderwidth=1)
214     cadre.pack(fill=BOTH)
215
216     #move forward
217     forward = Frame(cadre, width=700, height=100, borderwidth=1)
218     forward.pack(fill=Y)
219     message = Label(forward, text="Move Forward")
220     message.pack(side="left")
221
222     var_texte = StringVar()
223     ligne_texte = Entry(forward, textvariable=var_texte, width=5)
224     ligne_texte.pack(side="left")
225
226     message = Label(forward, text="mm")
227     message.pack(side="left")
228
229     bouton_forward = Button(forward, text="Move Forward", fg="red", command=window.quit)
230     bouton_forward.pack(side="left")
231
232     #move aside
233     aside = Frame(cadre, width=700, height=100, borderwidth=1)
234     aside.pack(fill=Y)
235     message = Label(aside, text="Move Aside")
236     message.pack(side="left")
237
238     var_texte = StringVar()
239     ligne_texte = Entry(aside, textvariable=var_texte, width=5)
240     ligne_texte.pack(side="left")
241
242     message = Label(aside, text="mm")
243     message.pack(side="left")

```

```
244 bouton_aside = Button(aside, text="Move Aside", fg="red", command=window.quit)
245 bouton_aside.pack(side="left")
246
247
248 #rotation
249 rotation = Frame(cadre, width=700, height=100, borderwidth=1)
250 rotation.pack(fill=Y)
251 message = Label(rotation, text="Rotation")
252 message.pack(side="left")
253
254 var_texte = StringVar()
255 ligne_texte = Entry(rotation, textvariable=var_texte, width=5)
256 ligne_texte.pack(side="left")
257
258 message = Label(rotation, text="degrés")
259 message.pack(side="left")
260
261 bouton_aside = Button(rotation, text="Rotation", fg="red", command=window.quit)
262 bouton_aside.pack(side="left")
263
264 window.mainloop()
265 window.destroy()
266
267
268 if __name__ == '__main__':
269
270     # asterix = get_object()
271     initialize()
272     walk.initial = rotation.initial_pos(30, -60)
273     time.sleep(0.1)
274     create_interface()
275     # user_interaction()
276     while 1:
277         walk.move_center_forward(20, -60)
278         walk.initial = rotation.initial_pos(30, -60)
279     # We really need to sleep before we die
280     time.sleep(0.1)
```

main.py