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
Steve Nguyen

1er 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 :

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

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

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

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

The positive root of this equation is :

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

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

1.2 Program

```
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

legs = []
xCorrection = [-10,-20,-20,10,-20,-20]
yCorrection = [0,-15,15,0,15,-15]


"""
  Indirect kinematic function.
  Parameters :
    - (x3,y3,z3) : The coordonnates where we want to put the leg
    - alpha : the correction for the second motor.
    - beta : the correction for the third motor.
    - l2 , l3 : the length of the different part of the leg
  Return a tuple with three values, corresponding to the angles of each motor of the leg.
"""
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 = (0,0,0)
    theta1 = angles[0]
    theta2 = angles[1]
    theta3 = angles[2]

    try:
        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)
    except ValueError:
        print "The leg of the robot cannot go that far!!"

    return angles


def get_legs(obj):
```

```
    """
  Return a list with all the legs of the robot passed in parameter, i.e a leg is three motors.
      The motorgroups is actually done manually.
  """
    return [obj.leg1,obj.leg2,obj.leg3,obj.leg4,obj.leg5,obj.leg6]


def get_xCorrection(leg):
  """
    Return the correction (in mm) for the x axis of a specified leg
    Parameters :
      - leg : The leg which is going to be moved
  """
  i = int(leg[0].id*0.1)
  return xCorrection[i-1]


def get_yCorrection(leg):
  """
  Return the correction (in mm) for the y axis of a specified leg
  Parameters :
    - leg : The leg which is going to be moved
  """
  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_center):
  """



  """
  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(2*math.radians(theta))
```

```
  tmp += R_center**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_center = 100):
  """
    Do a rotation on one leg.
    Parameters :
      - theta : the angle we want the leg to do
      - z : the height of the tip of the leg.
      - leg : the leg we want to move
      - R_center : The radius of the circle which the center is equal to center of the entire
        robot.
    Return a tuple with the coordonnates of the leg.
  """
  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_center)*math.cos(math.radians(theta))
  y = R_leg(theta,leg,R_center)*math.sin(math.radians(theta))
  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(theta,z):
  """
    Put the robot in an initial position.
    Parameters :
      - theta : The initial angle of the leg from on its own axis.
      - z : The height of the leg.
  """
  # 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]))
```

```
    initial_position.append(move_leg(-abs(theta),z,legs[1]))
    initial_position.append(move_leg(abs(theta),z,legs[2]))
    initial_position.append(move_leg(0,z,legs[3]))
    initial_position.append(move_leg(-abs(theta),z,legs[4]))
    initial_position.append(move_leg(abs(theta),z,legs[5]))

    time.sleep(0.1)

    return initial_position

"""
TODO: make sure that this works. If it works than we can easily do experiments to find the highest value on alpha
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
See the draft implementation for arbitrary_rotation above.
"""
# theta is the value we need to set the initial position
# alpha determines the amount of rotation (made by each call to the function) from this initial position
# alpha is physically limited because of the legs. We should define this limit as max_angle - see above.
# 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)
def rotation_angle(alpha,theta,z):
    """
     Do a rotation on all the legs of the robot.
     Parameters :
       - alpha :
       - theta : The angle we want the legs to do.
       - z : the height of the leg.
    """
    #clockwise 2 and 5 are limited
    breaklength = 0.1
    # Position 1: The 'spider' position. This position has a low center of gravity.
    # Here we define the initial position. i.e. the spider position
    # It is important to observe the x and y values of each leg in its own frame of reference

    # Position 2: Put legs 1, 3, 5 in the air and rotate at the same time
    move_leg(-abs(theta)+alpha,z+20,legs[1])
    move_leg(alpha,z+20,legs[3])
    move_leg(abs(theta)+alpha,z+20,legs[5])
    time.sleep(breaklength)

    # Position 3: Put legs 1,3 and 5 down
    move_leg(-abs(theta)+alpha,z,legs[1])
    move_leg(alpha,z,legs[3])
    move_leg(abs(theta)+alpha,z,legs[5])
    time.sleep(breaklength)

    # Position 4: Rotate legs 0, 2, 4
```

```
    move_leg(alpha,z+20,legs[0])
    move_leg(abs(theta)+alpha,z+20,legs[2])
    move_leg(-abs(theta)+alpha,z+20,legs[4])
    time.sleep(breaklength)

    # Position 5: Put legs 0, 2 and 4 down.
    move_leg(alpha,z,legs[0])
    move_leg(abs(theta)+alpha,z,legs[2])
    move_leg(-abs(theta)+alpha,z,legs[4])
    time.sleep(breaklength)

# max_angle = 20 is just a guess.
# 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
"""SOLVED?"""
#TEST : If we put negative value fot the beta angle, this is just not working.
# TEST : If the value of max_angle is not 20, the rotation does not work proprely
# theta and z are simply values that determine the initial position
# Other parameters are to define the rotation
def arbitrranany_rotation(beta, max_angle = 10, theta = 45, z = -60):
    """
        Do a fully operational rotation on the robot (the return to inital position is in this function)
        Parameters :
            - beta : The angle we want the center of the robot to do.
            - max_angle : The maximum angle the robot can do in one loop.
            - theta : The rotation of each leg.
            - z : the height of the robot.
    """
# Here we do euclidean division. We determine how often max_angle divides beta and the remainder of this division.
# This gives us the number of rotations we need to make by a predefined max_angle
# The remainder gives us the amount we need to rotate by to be able to finish the full rotation by an angle of beta
# i.e. beta = q*max_angle + r
    beta = 2*beta
    initial_pos(theta,z)
    if beta < 0:
        max_angle = -max_angle

    q = beta//max_angle
    r = beta%max_angle
    print q
    print r
    # rotate by max_angle q times
    for i in range(1,q):
        rotation_angle(max_angle, theta, z)
        initial_pos(theta,z)
    # finally rotate by r
    rotation_angle(r, theta, z)
    initial_pos(theta,z)
```

```python

""" ________________________________________________________________________________ """
```

rotation.py

2 Walk

2.1 Program

```
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
import rotation

import Tkinter as tk # to get the a graphic interface for the control function


legs = []
initial = []


"""
  Indirect kinematic function.
  Parameters :
    - (x3,y3,z3) : The coordonnates where we want to put the leg
    - alpha : the correction for the second motor.
    - beta : the correction for the third motor.
    - l2 , l3 : the length of the different part of the leg
  Return a tuple with three values, corresponding to the angles of each motor of the leg.
"""
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 = (0,0,0)
    theta1 = angles[0]
    theta2 = angles[1]
    theta3 = angles[2]

    try:
        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)
    except ValueError:
        print "The leg of the robot cannot go that far!!"
```

```python

    return angles

"""
  Return a list with all the legs of the robot passed in parameter, i.e a leg is three motors. The motorgroups is actually done manually.
"""
def get_legs(obj):
  return [obj.leg1,obj.leg2,obj.leg3,obj.leg4,obj.leg5,obj.leg6]

"""
  Makes one leg move.
  parameters:
    - L : The length between the start point and the end point of each leg (in a straight line).
    - z : THe height of the center of the robot.
    - leg : The leg we want to move
"""
def move_leg(L,z,leg):
  num = int(leg[0].id*0.1)-1
  theta = math.atan(initial[num][1]/initial[num][0])
  hypo = math.sqrt(initial[num][0]**2 + initial[num][1]**2)
  x = math.cos(theta)*(hypo+L)
  y = math.sin(theta)*(hypo+L)
  z = z
  angles = leg_ik(x,y,z)
  i=0
  for motors in leg:
    motors.goal_position = angles[i]
    i+=1

""""
  Make the robot move along his two separate legs
  Parameters:
    - L : The length between the start point and the end point of each leg (in a straight line).
    - z : THe height of the center of the robot.
"""
def move_center_forward(L,z,max_length=50):
  print "move_center_forward({0},{1},...)".format(L,z)

  break_length = 2
  theta = 20  #more than 20 would make the legs touch for a sec (because of the speed)
  order = [1,5,2,4]

  if L>max_length:
    move_center_forward(L-max_length,z)
    L = L -max_length

  initial = rotation.initial_pos(30,z)
  if L<0:
```

```
    order = [4,2,5,1]
  move_leg(L,z+40,legs[0])
  print "move_leg({0},{1},...)".format(L,z)
  move_leg(-L,z+40,legs[3])
  print "here 3"
  time.sleep(break_length)

  move_leg(L,z,legs[0])
  move_leg(-L,z,legs[3])
  time.sleep(break_length)

  rotation.move_leg(-theta,z+40,legs[order[0]])
  rotation.move_leg(theta,z+40,legs[order[1]])
  time.sleep(break_length)
  rotation.move_leg(-theta,z,legs[order[0]])
  rotation.move_leg(theta,z,legs[order[1]])
  time.sleep(break_length)

  rotation.move_leg(-theta,z+40,legs[order[2]])
  rotation.move_leg(theta,z+40,legs[order[3]])
  time.sleep(break_length)
  rotation.move_leg(-theta,z,legs[order[2]])
  rotation.move_leg(theta,z,legs[order[3]])
  time.sleep(break_length)

"""
THEORICAL WORK: The order of the leg or the direction could be wrong...TO TEST
Make the robot move along its two legged side.
"""
  theta = 20

  if L>max_length:
    move_center_aside(L-max_length,z)
    L = L -max_length

  if L<0:
    theta = -theta

  initial = rotation.initial_pos(30,z)
  time.sleep(break_length)
  print "putting legs 2-3 in the air"
  print "move_leg({0},{1},...)".format(L,z+40)
  move_leg(L,z+40,legs[1])
  move_leg(L,z+40,legs[2])
  time.sleep(break_length)
  move_leg(L,z,legs[1])
  move_leg(L,z,legs[2])
  time.sleep(break_length)

  print "putting legs 5-6 on the ground"
  move_leg(-L,z+40,legs[4])
```

```
  move_leg(-L,z+40,legs[5])
  time.sleep(break_length)
  move_leg(-L,z,legs[4])
  move_leg(-L,z,legs[5])
  time.sleep(break_length)

  print "rotating the legs 1-4 and putting them in the air"
  rotation.move_leg(theta,z+40,legs[0])
  rotation.move_leg(-theta,z+40,legs[3])
  time.sleep(break_length)
  print "rotating the legs 1-4 and putting them on the ground"
  rotation.move_leg(theta,z,legs[0])
  rotation.move_leg(-theta,z,legs[3])
  time.sleep(break_length)
#move the center with leg 2, 4, 6 in the air


"""
  Move all the leg in their own coordonnate system.
  Parameters:
    - L : The length between the start point and the end point of each leg (in a straight line)
      .
    - z : THe height of the center of the robot.
"""
def moving_all_legs(L,z):
  move_leg(L,z,legs[0])
  move_leg(L,z,legs[1])
  move_leg(L,z,legs[2])
  move_leg(-L,z,legs[3])
  move_leg(L,z,legs[4])
  move_leg(L,z,legs[5])
```

walk.py

3 Main

3.1 Program

```
import walk as walk
import rotation as rotation

import itertools
import time
import numpy
from pypot.dynamixel import autodetect_robot
import pypot.dynamixel
import math
import json
import time
from contextlib import closing
from Tkinter import *

import pypot.robot

asterix = None
legs = []



def get_object():
  """
    Return a robot object created from a json file. initialize the legs variable in the three files.
  """
  asterix = pypot.robot.from_json('my_robot.json')
  legs = get_legs(asterix)
  rotation.legs = get_legs(asterix)
  walk.legs = get_legs(asterix)

  return asterix


def detection():
  """
    Do the detection of the robot with all its motors. It puts the configuration into a json file named 'my_robot.json'
  """
  my_robot = autodetect_robot() #detect al the legs of the robot. Might take a while to operate.

  #write the configuration found into a json file. We shouldn't use the complete detection whith this json file.
  config = my_robot.to_config()
  with open('my_robot.json', 'wb') as f:
      json.dump(config, f)
```

```

  with closing(pypot.robot.from_json('my_robot.json')) as my_robot:
      # do stuff without having to make sure not to forget to close my_robot!
      pass

<<<<<<< HEAD
"""
  Initialize the robot. Firstly get the robot object, and then put the angles of the motor at 0
      deg.
  Return the robot object.
"""
=======

>>>>>>> f49be2317d5b8553dc3cac64b48885436bbd60b1
def initialize():
  """
    Initialize the robot. Firstly get the robot object, and then put the angles of the motor at
        0 deg.
    Return the robot object.
  """
  asterix = get_object()
  # print asterix
  # Note that all these calls will return immediately,
  # and the orders will not be directly sent
  # (they will be sent during the next write loop iteration).
  for m in asterix.motors:
      m.compliant = False    # <=> enable_torque.
     # m.goal_position = 0
  time.sleep(0.1)
  return asterix
"""
  if asterix['motorgroups'] == None:
    asterix['motorgroups'] = {
    'leg1': ["motor_11","motor_12","motor_13"],
    'leg2': ["motor_21","motor_22","motor_23"],
    'leg3': ["motor_31","motor_32","motor_33"],
    'leg4': ["motor_41","motor_42","motor_43"],
    'leg5': ["motor_51","motor_52","motor_53"],
    'leg6': ["motor_61","motor_62","motor_63"]
    }
"""


def get_legs(obj):
  """
    Return a list with all the legs of the robot passed in parameter, i.e a leg is three motors
        . The motorgroups is actually done manually.
  """
  return [obj.leg1, obj.leg2, obj.leg3, obj.leg4, obj.leg5, obj.leg6]


```

```
#----------------------------------------------------------
#------------------ events function ----------------
#----------------------------------------------------------


def forward(event):
  """
    Call the move_center_forward function with some defined values.
    Parameters :
      -- event : an event that 'catch' what key the users is pressing.
  """
  print 'forward'
  z = -60
  L = 30
  theta = 0
  break_length = 0.2
  walk.initial = rotation.initial_pos(theta,z)
  time.sleep(break_length)
  walk.move_center_forward(L,z)
  walk.initial = rotation.initial_pos(theta,z)
  time.sleep(break_length)


def backward(event):
  """
    Call the move_center_forward function with some defined valuees (one is negative to go be
        able to go backward).
    Parameters :
      -- event : an event that 'catch' what key the users is pressing.
  """
  print 'backward'
  z = -60
  L = -30
  theta = 0
  break_length = 0.2
  walk.initial = rotation.initial_pos(theta,z)
  time.sleep(break_length)
  walk.move_center_forward(L,z)
  walk.initial = rotation.initial_pos(theta,z)
  time.sleep(break_length)


def left(event):
  """
    Call the move_center_aside function with some defined values.
    Parameters :
      -- event : an event that 'catch' what key the users is pressing.
  """
  z= -60
  L = 30
  theta = 0
```

```
    break_length = 0.2
    walk.move_center_aside(L,z)


def right(event):
    """
        Call the move_center_aside function with some defined values.
        Parameters :
          -- event : an event that 'catch' what key the users is pressing.
    """
    z= -60
    L = -30
    theta = 0
    break_length = 0.2
    walk.move_center_aside(L,z)


def rotation_left(event):
    """
        Call the arbitrary_rotation function with some defined values.
        Parameters :
          -- event : an event that 'catch' what key the users is pressing.
    """
    angle = 90
    rotation.arbitrary_rotation(angle*2)


def rotation_right(event):
    """
        Call the arbitrary_rotation function with some defined values (one is negative).
        Parameters :
          -- event : an event that 'catch' what key the users is pressing.
    """
    angle = -90
    rotation.arbitrary_rotation(angle*2)


def position_initial(event):
    """
        Call the initial_pos function with some defined values, with a height of 60 cm.
        Parameters :
          -- event : an event that 'catch' what key the users is pressing.
    """
    theta = 0
    z = -60
    rotation.initial_pos(theta,z)


def user_interaction():
    """
        Bind all the events to the minimal graphinc interface.
```

```
    """
    root = tk.Tk()
    root.bind("<Up>",forward)
    root.bind("<Down>",backward)
    root.bind("<Right>",right)
    root.bind("<Left>",left)
    root.bind("<i>",rotation_left)
    root.bind("<i>",rotation_right)
    root.bind("<Return>",position_initial)
    root.mainloop()


#--------------------------------------------------
#------------ Graphic interface ------------
#--------------------------------------------------

def create_interface():

    window = Tk()

    cadre = Frame(window, width=700, height=500, borderwidth=1)
    cadre.pack(fill=BOTH)

    #move forward
    forward = Frame(cadre, width=700, height=100, borderwidth=1)
    forward.pack(fill=Y)
    message = Label(forward, text="Move Forward")
    message.pack(side="left")

    var_texte = StringVar()
    ligne_texte = Entry(forward, textvariable=var_texte, width=5)
    ligne_texte.pack(side="left")

    message = Label(forward, text="mm")
    message.pack(side="left")

    bouton_forward = Button(forward, text="Move Forward", fg="red",command=window.quit)
    bouton_forward.pack(side="left")

    #move aside
    aside = Frame(cadre, width=700, height=100, borderwidth=1)
    aside.pack(fill=Y)
    message = Label(aside, text="Move Aside")
    message.pack(side="left")

    var_texte = StringVar()
    ligne_texte = Entry(aside, textvariable=var_texte, width=5)
    ligne_texte.pack(side="left")

    message = Label(aside, text="mm")
    message.pack(side="left")
```

```
    bouton_aside = Button(aside, text="Move Aside", fg="red",command=window.quit)
    bouton_aside.pack(side="left")

    #rotation
    rotation = Frame(cadre, width=700, height=100, borderwidth=1)
    rotation.pack(fill=Y)
    message = Label(rotation, text="Rotation")
    message.pack(side="left")

    var_texte = StringVar()
    ligne_texte = Entry(rotation, textvariable=var_texte, width=5)
    ligne_texte.pack(side="left")

    message = Label(rotation, text="degres")
    message.pack(side="left")

    bouton_aside = Button(rotation, text="Rotation", fg="red",command=window.quit)
    bouton_aside.pack(side="left")

    window.mainloop()
    window.destroy()


if __name__ == '__main__':

    # asterix = get_object()
    initialize()
    walk.initial = rotation.initial_pos(30,-60)
    time.sleep(0.1)
    create_interface()
    # user_interaction()
    while 1:
        walk.move_center_forward(20,-60)
        walk.initial = rotation.initial_pos(30,-60)
    # We really need to sleep before we die
    time.sleep(0.1)
```

main.py