CYBER-PHYSICAL SYSTEMS AND ROBOTICS

Lab 1. Perception and locomotion

# Preparation

1. The pass keyword is used when there is a need to write a statement but as of now, nothing has been implemented yet. Nothing will happen when executing this line of code. The idea is to replace it afterward.
2. The Idle class allows to execute the loop in main.py with a fixed time step. This way one can control the update rate of the measurements as well as the commands given to the robot. This limits the data transfer to an acceptable value which is a trade-off between resource efficiency and precision of the robot control.
3. It is impossible to instantiate the Robot class. Indeed, it is an abstract class, implemented thanks to the ABC class which is part of the abc package. An abstract class must be complemented to be instantiated. The Robot class has abstract methods, like move and sense, which must be implemented by the class RobotP3DX, inheriting from Robot. Then, it is possible to create an instance of RobotP3DX.
4. In the \_\_init\_\_ method of class RobotP3DX, there is a call to the \_\_init\_\_ method of the mother class Robot, which initialize attributes defined in Robot. RobotP3DX can access them as it inherits from this class.

# Code

## Wandering around

First, we modify the \_init\_motor method, with calls to the V-REP API to recover the handles (reference code to an object) of the two motors given their names. Finally, we return them in the form of a dictionary with keys “right” and “left”. The names could be retrieved in the simulator V-REP.

rc, handle\_right = vrep.simxGetObjectHandle(self.\_client\_id, "Pioneer\_p3dx\_rightMotor",  
 vrep . simx\_opmode\_blocking)  
rc, handle\_left = vrep.simxGetObjectHandle(self.\_client\_id, "Pioneer\_p3dx\_leftMotor", vrep.simx\_opmode\_blocking)  
  
motors = {"left": handle\_left, "right": handle\_right}

Code 1: Recovering the motors handles

## Taking a glance at the environment

Then, we had to do the same with the sensors. We could use the same API function but as there are 16 sensors, we used a loop to scan them all. Names were generated using a simple concatenation and handles were appended to a list previously created.

Sensors are also read a first time to ensure there work correctly. In a first approach, the return code rc is read and a very simple verification is made, to ensure there is no error.

handle\_sensor = []  
for i in range (1, 17):  
 rc, handle = vrep.simxGetObjectHandle(self.\_client\_id, "Pioneer\_p3dx\_ultrasonicSensor" + str(i),  
 vrep.simx\_opmode\_blocking)  
 handle\_sensor.append(handle)  
 if rc != vrep.simx\_return\_ok:  
 print("error")  
 vrep.simxReadProximitySensor(self.\_client\_id, handle, vrep.simx\_opmode\_streaming)  
  
return handle\_sensor

Code 2: Recovering the sensors handles and storing them in a list

To read the sensors, the method sense was implemented, using the API function vrep.simxReadProximitySensor with a loop. The distance are computed thanks to the norm function of the numpy package.

distance = []  
for i in range(0, 16):  
 rc, is\_valid, detected\_point, \_, \_ = vrep.simxReadProximitySensor(self.\_client\_id, self.\_sensors[i],  
 vrep.simx\_opmode\_buffer)  
  
 if is\_valid:  
 distance.append(np.linalg.norm(detected\_point))  
  
 else:  
 distance.append(1)  
  
# print(distance)  
return distance

Code 3: Reading the sensors

## Exploring the wild

The first strategy used was to implement a simple 2-state machine: either the robot is moving on a straight line of the maize or it is turning, making a curve. Tests were performed to distinguish these cases depending on the measurements of the sensors.

if measurements[4-1] > 0.4 and measurements[5-1] > 0.4 and measurements[3-1] > 0.3 and measurements[6-1] > 0.3:

Code 4: Test to determine the state of the robot

1

8

16

9

## Road to the final project