Programming the mobile robot Khepera

The aim of this exercise is to become familiar with programming of the mobile robot in Python language.

1 Commands

Khepera's cooperation module is written using object-oriented programming, therefore every instruction should start with suitable instance (ex. k.stop()).
`about()` – informs about interface version,
`stop()` – stop the robot,
`reset()` – load default values of parameters,
`read_version()` – readout of bios version and khepera's module,
`set_speed(left,right)` – set the speed of wheels,
`[left,right]=read_speed()` – read the speed of wheels,
`set_poscounter(left,right)` – set the values of counters in wheels,
`set_pos(left,right)` – set the position which the robot should achieve,
`complete()` – wait until the robot achieves position,
`[left,right]=read_pos()` – read counters value of the wheels position,
`conf_speed_profile(vl,al,vr,ar)` – set maximum speed of wheels `(vl, vr)` and maximum acceleration `(al, ar)`,
`read_status()` – read motion controller state,
`change_led(number, action)` – change LED diods state, `numer`=0 side, `numer`=1 frontal, `action`=0 turn off, `action`=1 turn on, `action`=2 change state,
`read_adc(channel)` – read analog–to–digital converter value,
`read_proximity()` – proximity sensor readings,
`read_proximity_avg()` – average proximity sensor readings,
`read_light()` – light sensor readings,
`read_light_avg(n)` – average light sensor readings,
`sleep(t)` – time delay,
`demo(n)` – call the demo procedure, `n`=0 obstacles avoidance, `n`=1 light following.

Measurement readings are represented as eight elements table (first element has index 0) `[left_90, left_45, left_10, right_10, right_45,`

right_90, right_back, left_back].
On the top of the robot there is a monochromatic 8–bits 64 pixels camera module. Commands which cooperate with the camera:
turret_read_image() – read complete image (64 pixels),
turret_read_light_intensity() – read brightness of environment,
turret_read_image_lowers() – read complete image in 4-bits resolution mode,
turret_read_subimg8(start) – read next 8 pixels starting from start,
turret_read_subimg16(start) – read next 16 pixels starting from start,
turret_read_subscanned2() – read every second pixel of complete image (32 elements table),
turret_read_subscanned4() – read every fourth pixel of complete image (16 elements table).

To start program, please write command

```
python nameofprogram
```

in the console.

2 Program example

```
#!/usr/bin/python
# -*- coding: latin2 -*-

from khepera import *

# user function
def fun(k):
   k.set_speed(4,4)
   k.sleep(1.5)
   k.stop()
   print "End"

# run the function
go(fun)
```

3 Laboratory

3.1 Preliminaries

- to be familiar with this instruction,
- to be familiar with khepera's parameters and construction.

3.2 Safety rules

Khepera is very a fragile device. During the exercise it is necessary to keep the robot inside the work–place. Velocities of wheels should be bounded by safe values. In case of non–standard behaviour the robot should be picked up by a corpus to prevent wheels block. Pulling the connective cable is forbidden.

During program execution there is the possibility of stopping the robot using **k.stop()**. Using `Ctrl-C` should also stop the program (on condition that the program is started as it was shown in section 1). The robot stops when the program ends. In dangerous situations the robot can be turned off by disconnect the AC adapter.

4 Task to do

- Draw the rectangle using speed control (set_speed, sleep).
- Draw the circle.
- Draw the rectangle using counters (set_poscounter, set_pos).
- Draw the eight shape.
- Write the program that stops the robot in front of the obstacle.
- Write the program that allows the robot to avoid the obstacles. Please, look at the difference between sensor readings and average sensor readings.
- Write the program that allows the robot to go in the direction of light.
- Complete above–mentioned programs with the procedure that untwist the cable in the case of `Ctrl-C` breaking.

`Ctrl-C` exception handling

```
def fun(k):     # user function
        ...
def safefun(k): # master function
        try:
              fun(k)
        except KeyboardInterrupt:
              print 'Ctrl-C'
go(safefun)     # run
```

A Python's syntax

Python is widely spread object–oriented programing language. Due to its clear syntax, the programs are easy to understand. The build-in variable types are very varied, from numerical (floating-point, complex, integer with infinite precision) through text string, dictionaries, to lists and others.

One of the variable type in Python is the list. List is written as a several elements separated by a comma and taken into the square bracket. It is not necessary that the elements inside the list are of the same type, e.g. `a=['spam', 'eggs', 100, 1234]`.

The index operator can be used to call the particular element of the list, e.g. `a[2]`. The first element of the list has the 0 index. The negative indexes can be used to take the corresponding list element counting from the end. The individual list elements could be changed in that way: `a[2]=a[2]+3`.

Python's program example

```
# Fibonnaci series
a, b = 0, 1
while b < 10:
   print b
   a, b = b, a+b
```

In the first line there is the multiplied assign: variables `a` and `b` simultaneously take the values 0 and 1. In every assign command, the right-hand expressions are firstly evaluated and then assigned to the left-hand expressions.

The loop `while` is performed as long as the relation (`b < 10`) is true. Every non–zero value of the variables is treated as a logical true, similar to every non–empty lists or strings. The standard relation operators are the same as in C language: `<` (smaller than), `>` (grater than), `==` (equal to), `<=` (smaller than or equal to), `>=` (grater than or equal to) and `!=` (not equal to). There are also two logic expressions: `True` and `False`. To combine the logic expression into more complex one the `and`, `or` and `not` operators should be used.

To group the expressions use the indentation. Every line in the same group must be indented with the same number of chars. The end of the group is marked with the empty line.

A.1 Python's control structures

The conditional structure in Python is similar to other programming language

```
if x < 0:
   print 'less than zero'
elif x == 0:
   print 'zero'
else:
   print 'greater than zero'
```

The loop `for` in Python is a little different than in other language like C or Pascal. Here, the list is iterating through the values of the inputted sequence of the list. If there is the need to iterate through the standard sequence the `range()` should be used, e.g.

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

The expression `range(5,10)` defines a series starting from 5, and `range(0,10,3)` defines the series with the step equal 3.

Expression `break` prevents to continue the current loop (`for` or `while`) and the `continue` starts the present loop from the beginning.

A.2 Build-in functions

`len(a)` – returns the length (number of elements) of `a` ,
`max(a)` – returns the maximum value of the elements of `a`,
`min(a)` – returns the minimum value of the elements of `a`,
`print...` – printing the variables value, the comma at the end of the line prevents the line breaking.

For the list `l` there are few function:
`l.index(w)` – returns the index of the first element containing the `w` value,
`w in l` – testing the list `l` for the presence of the `w` value.