Programowanie robota mobilnego Khepera w języku Python

Paweł Ludwików

28 lutego 2011

1 Zasady bezpieczeństwa

Robot Khepera jest bardzo delikatną konstrukcją. Podczas ćwiczenia należy pilnować, aby nie wyjechał poza obszar stanowiska. Pod żadnym pozorem nie wolno dopuścić do spadnięcia robota ze stołu. Nie należy rozwijać prędkości przekraczających zdolność reakcji obsługujących. W przypadkach awaryjnych robota należy chwytać za korpus, nie blokując kół palcami. Nie należy ciągnąć za przewód łączący. Nie powinno się dopuścić do nadmiernego skręcenia przewodu.

Podczas wykonywania programu robota można zatrzymać wywołując komendę `k.stop()`. Przerwanie wykonywania programu poprzez Ctrl-C powinno również zatrzymać robota (pod warunkiem uruchamiania funkcji użytkownika identycznie jak na przykładzie podanym w punkcie 3). Zakończenie wykonywania programu powoduje zatrzymanie robota (warunek j.w.). W przypadkach awaryjnych robota można podnieść ręcznie lub ostatecznie wyłączyć przez wyjęcie zasilacza z listwy zasilającej.

2 Dostępne komendy

Moduł współpracy z robotem mobilnym Khepera został napisany w wersji obiektowej, więc wszystkie instrukcje powinny być poprzedzone odpowiednią instancją (np. `k.stop()`). Jeżeli w opisie nie zaznaczono inaczej wszystkie argumenty instrukcji i wartości zwracane są typu całkowitoliczbowego.

`about()` — informacje o wersji interfejsu obsługi,

`stop()` — zatrzymanie robota,

`reset()` — przywrócenie domyślnych wartości parametrów,

`read_version()` — odczyt wersji biosu i protokołu khepery,

`set_speed(vlewy, vprawy)` — ustawienie prędkości obrotu kół,

`[vl,vp]=read_speed()` — odczyt prędkości kół,

`set_poscounter(l,p)` — ustawienie wartości liczników obrotów kół,

`set_pos(poz_lew, poz_praw)` — ustawienie położenia do którego robot powinien dojść,

`complete()` — odczekanie, aż robot osiągnie zadane położenie,

`[l,p]=read_pos()` — odczyt wartości liczników obrotów kół,

`conf_speed_profile(v_l, a_l, v_p, a_p)` — ustawienie profilu sterownika: maksymalnej prędkości koła lewego i prawego (`v_l`, `v_p`) oraz przyspieszenia (`a_l`, `a_p`), wartości domyślne to (`20, 64, 20, 64`),

`read_status()` — odczyt stanu kontrolera ruchu,

`change_led(nr, dzialanie)` — zmiana stanu diód LED, `nr=0` – boczna, `nr=1` – przednia; `dzialanie=0` – wyłączenie, `dzialanie=1` – włączenie, `dzialanie=2` – zmiana stanu,

`read_adc(kanal)` — odczyt wartości przetwornika A/C

`read_proximity()` — odczyt czujników zbliżeniowych,

`read_proximity_avg(n)` — uśredniony odczyt czujników zbliżeniowych,

`read_light()` — odczyt czujników oświetlenia,

`read_light_avg(n)` — uśredniony odczyt czujników oświetlenia,

`sleep(t)` — opóźnienie `t` sekund (może być wartością zmiennoprzecinkową),

`demo(d)` — wywołanie procedury demonstracyjnej, `d=0` omijanie przeszkód, `d=1` podążanie do światła.

Stan czujników zwracany jest w 8 elementowej tablicy, indeksowanej od 0 zawierającej kolejno `[lewy_90, lewy_45, lewy_10, prawy_10, prawy_45, prawy_90, prawy_tył, lewy_tył]`.

Na szczycie robota zamontowany jest moduł monochromatycznej 8-bitowej 64 pikselowej kamery. Domyślnie zwracane są wartości z pełnego zakresu (8-bitów szarości). Zestaw komend współpracujących z kamerą:

`turret_read_image()` — odczytanie pełnego obrazu – 64 pikseli,

`turret_read_light_intensity()` — odczytanie jasności otoczenia,

`turret_read_image_lowres()` — odczytanie pełnego obrazu w trybie 4-bitowej rozdzielczości,

`turret_read_subimg8(poczatek)` — odczyt 8 kolejnych pikseli rozpoczynając od `poczatek`,

`turret_read_subimg16(poczatek)` — odczyt 16 kolejnych pikseli rozpoczynając od `poczatek`,

`turret_read_subscanned2()` — odczyt co drugiego piksela pełnego obrazu (tablica 32 wartości),

`turret_read_subscanned4()` — odczyt co czwartego piksela pełnego obrazu (tablica 16 wartości).

Uruchomienie programu następuje po wpisaniu w konsoli (np. xterm) polecenia `python nazwaprogramu`

3 Przykład programu

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

from khepera import *

# funkcja uzytkownika
def fun(k):
        k.set_speed(4,4)
        k.sleep(1.5)
        k.stop()
        print "Koniec"

# uruchomienie funkcji
go(fun)
```

A Składnia języka Python

Python jest przejrzystym i wszechstronnym językiem programowania obiektowego. Używa eleganckiej składni, dzięki czemu programy są łatwe do zrozumienia. Zapewnia dużą różnorodność wbudowanych typów danych: liczbowych (zmiennoprzecinkowe, zespolone, liczby całkowite o nieskończonej precyzji), tekstowych, listy, słowniki, itp. Język zapewnia obsługę wyjątków pozwalając na efektywniej obsługiwać błędy.

Podstawowym złożonym typem danych w języku Python jest lista. Lista jest zapisywana jako wyszczególnienie wartości rozdzielone przecinkami ujęta w nawiasy kwadratowe. Elementy listy nie muszą byc tego samego typu. np. `a=['spam', 'eggs', 100, 1234]`.

Dostęp do elementu listy uzyskuje się stosując operator indeksowania, np. `a[2]`. Początkowy element posiada indeks 0. Użycie indeksów ujemnych powoduje zwrócenie odpowiedniej wartości elementu listy licząc od końca. Indywidualne elementy listy można dowolnie zmieniać, np: `a[2]=a[2]+3`.

Przykładowy program w języku Python (zapożyczony z wprowadzenia):

```
# liczby Fibonnaci'ego
a, b = 0, 1
while b < 10:
        print b
        a, b = b, a+b
```

Pierwsza linia zawiera przykład wielokrotnego przypisania: wartości `a` i `b` jednocześnie przyjmują wartości 0 i 1. Jak w każdym przypisaniu wartości prawostronne są obliczane najpierw, a potem podstawiane do lewej strony.

Pętla `while` wykonuje się tak długo jak długo warunek (tutaj `b < 10`) jest prawdziwy. W Pythonie, podobnie jak w C, każda niezerowa wartość całkowita ma wartość logicznej prawdy, każda niepusta lista lub ciąg znaków także. Standardowe operatory porównań są identyczne jak w C: `<` (mniejszy), `>` (większy), `==` (równy), `<=` (mniejszy lub równy), `>=` (większy lub równy) i `!=` (różny). W języku zdefiniowane są dwa wyrażenia logiczne: `True` oraz `False` posiadające logiczną wartość odpowiednio prawdy i fałszu. Łączenie warunków w złożone wyrażenia następuje przy pomocy opeatorów `and`, `or` i `not`.

Grupowanie wyrażeń odbywa się poprzez *wcięcia*. Każda linia tego samej grupy musi być wcięta o tą samą liczbę znaków. Koniec grupy oznacza się pustą linią. Zagnieżdżenie grup uzyskuje się przez zagnieżdżenie wcięć.

A.1 Kontrola przebiegu programu

Wyrażenie warunkowe uzyskuje się prawie identycznie jak w innych językach:

```
if x < 0:
        print 'ujemny'
elif x == 0:
        print 'zero'
else:
        print 'dodatni'
```

Pętla `for` różni się od odpowiedników w C i Pascalu. W Pythonie `for` iteruje po wartościach podanej sekwencji (listy lub ciągu) w kolejności ich pojawiania się.

Jeżeli zachodzi potrzeba iterowania po sekwencji liczb należy użyć funkcji `range()`, generującej listę zawierającą ciąg arytmetyczny.

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

Punkt końcowy nie jest częścią wygenerowanej listy, `range(10)` generuje listę 10 wartości, dopuszczalnych indeksów listy o długości 10. Można także określić początek: `range(5,10)` oraz także krok: `range(0, 10, 3)`

Wyrażenie `break`, podobnie jak w C, powoduje opuszczenie najbardziej zagnieżdżonej pętli `for` lub `while`. Wyrażenie `continue` powoduje wykonanie pętli od początku.

A.2 Funkcje wbudowane

`len(a)` — podaje długość (liczbę elementów) `a`
`max(a)` — podaje wartość maksymalnego elementu listy `a`
`min(a)` — podaje wartość minimalnego elementu listy `a`
`print ...` — drukowanie wartości, podanie przecinka na końcu nie powoduje przejścia do następnej linii

Dla listy `l` dostępne są następujące funkcje:
`l.index(w)` — zwraca indeks pierwszego elementu o wartości `w`
`w in l` — test, czy lista `l` zawiera element o wartości `w`

A.3 Obsługa wyjątku Ctrl-C

```
def fun(k):       # funkcja uzytkownika
        ...
def safefun(k):   # nadzorca
        try:
                fun(k)
        except KeyboardInterrupt:
                print 'Ctrl-C'

go(safefun)       # uruchomienie
```