ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ v Praze
Ú12110 Ústav přístrojové a řídící techniky

Semestrální práce

Python pro vědecké výpočty

Generátor pulzů pro simulaci vstupů kontroly trakce vozu Formule student

Vypracoval:
Pavel Balcar
2014

Úvod do problematiky

V této práci se budu zabývat vývojem zařízení, které bude simulovat vstupní signály ze snímačů otáček kol do řídící jednotky motoru studentské formule. Zařízení bude použito pro testování funkčnosti systému kontroly trakce, kterou je schopna tato jednotka vykonávat. Toto testování bude probíhat na motorové brzdě, proto je třeba simulovat tyto vstupy. Snímání otáček kol na studentské formuli je realizováno pomocí hallových sond. Jejich výstup je tedy obdélníkový pulz s frekvencí závislou na konkrétních otáčkách a střídou cca. 50%. Střída je závislá na podobě tzv. trigger disku (kotouč s otvory), ale pro potřeby testování na motorové brzdě to není podstatný parametr a budu pracovat práve s 50%. Zařízení by mělo být snadno ovladatelné z uživatelského rozhraní na osobním počítači.

Požadavky

Je potřeba simulovat dva výstupní signály s různou frekvencí pro přední a zadní nápravu. Frekvence signálů musí být dopočítána ze zadaných veličin, které uživatel zadá v uživatelském rozhraní. Zadané veličiny jsou rychlost otáčení kol přední a zadní nápravy v km/h, průměr kol přední a zadní nápravy a počet děr na trigger disku přední a zadní nápravy.

Realizace

Výstupní signály generuje zařízení LabJack U3-LV. Uživatelské rozhraní bylo vytvořeno v prostředí wxGlade. Celý program byl pak psán v jazyce Python 2.7.4.

Začal jsem tvorbou samotného uživatelského rozhraní tzv. GUI v prostředí wxGlade. Zde jsem si zvolil, jak velké okno aplikace chci použít a to pak rozdělil na jednotlivé Sizery, do kterých jsem umísťoval jednotlivé prvky aplikace (tlačítka, labely, text commandy, triggery atp.) Program wxGlade poté výsledné GUI, vytvořené graficky, převede do Py-kódu.

Obr. 1 – GUI vytvořené v programu wxGlade, červené písmo jsou popisky.

Vygenerovaný Py-kód jsem otevřel v programu Idlex vkládal jednotlivým položkám GUI funkce. Kód vygenerovaný wxGlade nebudu popisovat. Popíšu jen změny, které jsem v něm provedl.

Vložení grafu do sizeru cystom widget

V začátku kódu musí být importovány funkce z knihovny matplotlib a to takto:

import matplotlib

matplotlib.use('WXAgg')

from matplotlib.backends.backend_wxagg import FigureCanvasWxAgg as FigureCanvas

from matplotlib.backends.backend_wx import NavigationToolbar2Wx

from matplotlib.figure import Figure

a dále v „class Pulse_generator(wx.Frame):“ musí být vloženo toto:

self.figure = Figure()

self.axes = self.figure.add_subplot(111)

self.Graph = FigureCanvas(self, -1, self.figure)

Čtení hodnot zadávaných do GUI a aktualizace zobrazovaných hodnot

Tyto části programu obsahují všechny event handlery, tím je zaručeno, že když změním jakoukoli hodnotu, přepíší se labely, které ukazují aktuální hodnoty (rychlosti, frekvence, slip, graf atd.)

```
##### Cteni dat z GUI, obsah je stejny pro vsechny event handlery, proto se obsah labelu meni
##### s jakoukoli zmenou v GUI
def spdfr(self, event):  # wxGlade: Pulse generator.<event_handler>
    frspeed=self.frspdtrig.GetValue()/10.   #cteni posuvniku rychlosti prednich kol
    sfrspeed=str(frspeed)                   #prevod na string
    self.frspdvalue.SetLabel(sfrspeed)      #nastaveni labelu na aktualni nastavenou rychlost
    rrspeed=self.rrspdtrig.GetValue()/10.   #posuvnik rychlosti zadnich kol
    srrspeed=str(rrspeed)
    self.rrspdvalue.SetLabel(srrspeed)
    slip=((rrspeed-frspeed)/frspeed)*100.   #vypocet slip ratia
    slip="%.3f" %                           #zkraceni cisla na 3 desetinne mista
    sslip=str(slip)                         #prevod na sting
    self.SlipRatioValue.SetLabel(sslip)     #nastaveni labelu slip

    #### Vypocet hodnot frekvence pulzu zadnich kol
    trigfr=self.frTriggerdiscHoles.GetValue()   #cteni hodnoty der predniho trigger disku
    FrTyre=self.FrDiameter.GetValue()           #prumer predniho kola
    trigrr=self.rrTriggerdischoles.GetValue()   #cteni hodnoty der zadniho trigger disku
    RrTyre=self.RrDiameter.GetValue()           #prumer zadniho kola
    if not RrTyre or not trigrr:            #pokud je jeden z parametru nula
        freqrr=0                            #zapise se do lablu frekvence=0
        omegarr=freqrr                      #tim odpada problem s delenim nulou
        freqrr="%.2f" % freqrr
        freqrr=str(freqrr)
        self.freqr.SetLabel(freqrr)
    else:                                   #zde se pocita frekvence ze zvolenych hodnot
        RrTyre=int(RrTyre)
        trigrr=int(trigrr)
        freqrr=(rrspeed*trigrr)/(3.6*pi*(RrTyre/1000.))
        omegarr=freqrr
        freqrr="%.2f" % freqrr
        freqrr=str(freqrr)
        self.freqr.SetLabel(freqrr)
```

Obr. 2 – první část kódu pro čtení hodnot z text commandů a trigerrů, tento kód obsahuje každý event handler programu kromě handlerů pro obsluhu tlačítek

```
###### Zde je cast programu pro vykresleni grafu
self.axes.cla()
omega=0
if omegafr==0 and omegarr==0:       #podminky jsou zde pro urceni rozsahu x osy
    t = arange(0.0, 1, 0.0001)      #pokud jsou obe frekv. =0 osa x je <0,1>
    y = (signal.square(omegafr*t,duty=0.5)+1)/2
    x = (signal.square(omegarr*t,duty=0.5)+1)/2
elif omegafr<>0 and omegarr==0:     #zde je rozsah promenlivy podle ot prednich kol tak aby se graf vhodne vykresloval
    omega=omegafr
    t = arange(0.0, 30/omega, 0.0001)
    y = (signal.square(omegafr*t,duty=0.5)+1)/2
    x = (signal.square(omegarr*t,duty=0.5)+1)/2
elif omegafr==0 and omegarr<>0:     #zde je rozsah promenlivy podle otacek zadnich kol tak aby se graf vhodne vykresloval
    omega=omegarr
    t = arange(0.0, 30/omega, 0.0001)
    y = (signal.square(omegafr*t,duty=0.5)+1)/2
    x = (signal.square(omegarr*t,duty=0.5)+1)/2
elif omegafr<>0 and omegarr<>0:     # zde je rozsah promenlivy podle vetsi z frekvenci
    if omegarr>omegafr:
        omega=omegafr
    else:
        omega=omegarr
    t = arange(0.0, 30/omega, 0.0001)
    y = (signal.square(omegafr*t,duty=0.5)+1)/2
    x = (signal.square(omegarr*t,duty=0.5)+1)/2

#### samotne vykresleni grafu
self.axes.plot(t, y)
self.axes.plot(t, x)
self.axes.set_ylim(-0.5,1.5)
self.axes.set_title('Rear tyres - Green, Front tyres - Blue ')
self.axes.set_xlabel('t [sec]')
self.axes.set_ylabel('y [/]')
self.axes.grid()
self.Graph.draw()

event.Skip()
```

Obr. 3 – druhá část kódu pro čtení hodnot, tento kód obsluhuje aktualizaci grafu v custom widgetu

Threading

Generování pulzů je spuštěno tlačítkem start. Event handler pro tlačítko start obsahuje while cyklus. Protože je třeba generovat pulzy o dvou různých frekvencích, nestačí jeden while cyklus, ale musím použít dva. Funkce threading právě toto umožňuje. Při jejím použití se spustí více nezávislých vláken programu, které běží paralelně a tím pádem můžou bežet i dvě paralelní while smyčky.

```
#### Inicializace Threadu - jsou zde proto ze vystupni pulzy muzou mit ruznou frekvenci
#### proto potrebuji 2 paralelne bezici while cykly kazdy ve svem vlakne
class PulseThread(threading.Thread):
    def __init__(self,wait,pin):           #inicializace
        threading.Thread.__init__(self)     #wait je 1/2*frekvence
        self.wait=wait
        self.pin=pin                        #pin je zvoleny GPIO port LabJacku

    def run(self):                          #obsah threadu
        is_run = True
        while is_run:
            d.getFeedback(u3.BitStateWrite(IONumber = self.pin,State=1))
            time.sleep(self.wait)
            d.getFeedback(u3.BitStateWrite(IONumber = self.pin,State=0))
            time.sleep(self.wait)
            is_run=self.butstop
```

Obr. 4 – Inicializace a obsluha threadů

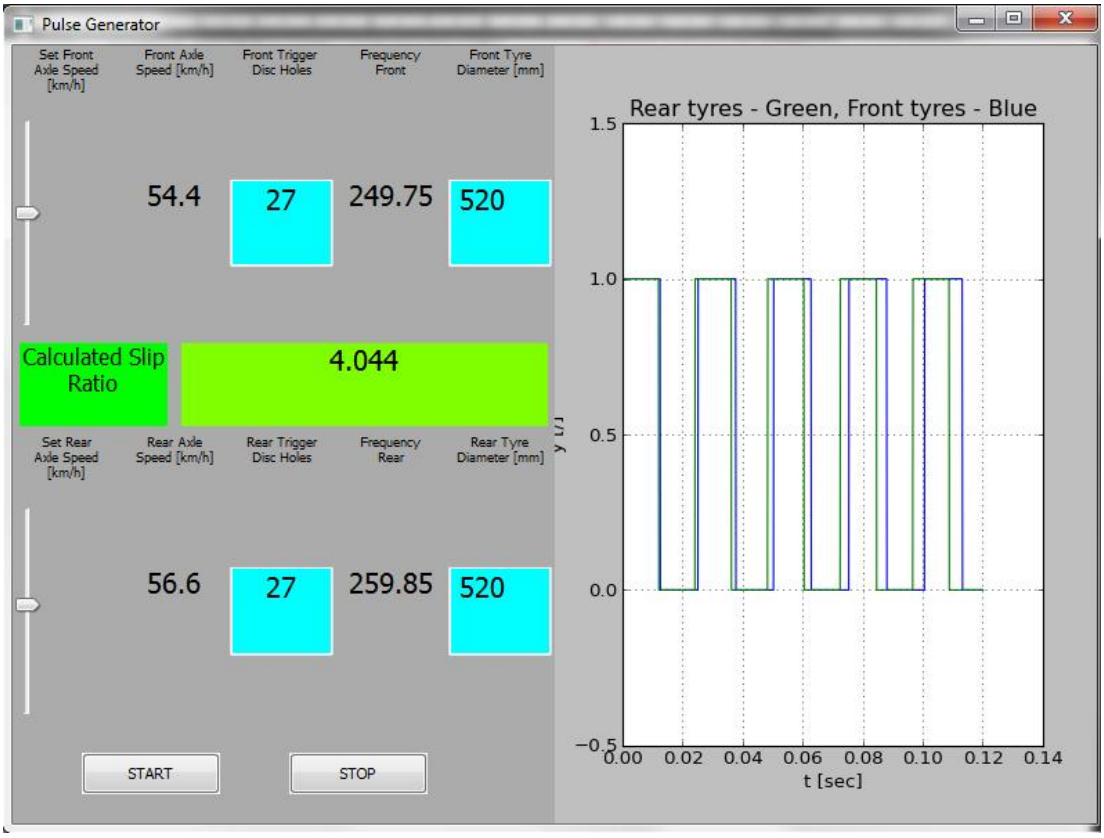
Celá část programu v event handleru tlačítka start pak vypadá takto:

```
#### Event handler pro tlacitko start
def but_start(self, event):  # wxGlade: Pulse generator.<event_handler>
    d = u3.U3()             #inicializace LabJacku
    d.debug = True          #zapnuti Debug
    d.getFeedback(u3.LED(State = True))     #zapnuti diody labjacku aby bylo videt jestli je funkcni

    #### Inicializace Threadu - jsou zde proto ze vystupni pulzy muzou mit ruznou frekvenci
    #### proto potrebuji 2 paralelne bezici while cykly kazdy ve svem vlakne
    class PulseThread(threading.Thread):
        def __init__(self,wait,pin):           #inicializace
            threading.Thread.__init__(self)     #wait je 1/2*frekvence
            self.wait=wait
            self.pin=pin                        #pin je zvoleny GPIO port LabJacku

        def run(self):                          #obsah threadu
            is_run = True
            while is_run:
                d.getFeedback(u3.BitStateWrite(IONumber = self.pin,State=1))    #zapnuti pinu
                time.sleep(self.wait)                                           #cekani o dobe pulperiody
                d.getFeedback(u3.BitStateWrite(IONumber = self.pin,State=0))    #vypnuti
                time.sleep(self.wait)

    trigfr=self.frTriggerdiscHoles.GetValue()   #cteni hodnot GUI
    FrTyre=self.FrDiameter.GetValue()
    trigrr=self.rrTriggerdischoles.GetValue()
    RrTyre=self.RrDiameter.GetValue()
    frspeed=self.frspdtrig.GetValue()/10.
    rrspeed=self.rrspdtrig.GetValue()/10.
    if not FrTyre or not trigfr or not RrTyre or not trigrr:    #generovani se spusti jen
        d.getFeedback(u3.LED(State = False))                    #pokud jsou v gui zadane vsechny
        d.getFeedback(u3.BitStateWrite(IONumber = 4,State=0))   #hodnoty
        d.getFeedback(u3.BitStateWrite(IONumber = 5,State=0))
    else:
        d.getFeedback(u3.LED(State = True))                     #spusteni generatoru pulzu
        FrTyre=int(FrTyre)
        trigfr=int(trigfr)
        RrTyre=int(RrTyre)
        trigrr=int(trigrr)
        waitfr=(3.6*pi*FrTyre/1000)/(2*frspeed*trigfr)          #vypocet pulperiody pulzu
        waitrr=(3.6*pi*RrTyre/1000)/(2*rrspeed*trigrr)          #puperioda druheho pulzu
        ####
        threadfr = PulseThread(waitfr,4,True)                   #vytvoreni threadu na pinu 4
        threadrr = PulseThread(waitrr,5,True)                   #vytvoreni threadu na pinu 5
        threadfr.start()
        threadrr.start()
    event.Skip()
```

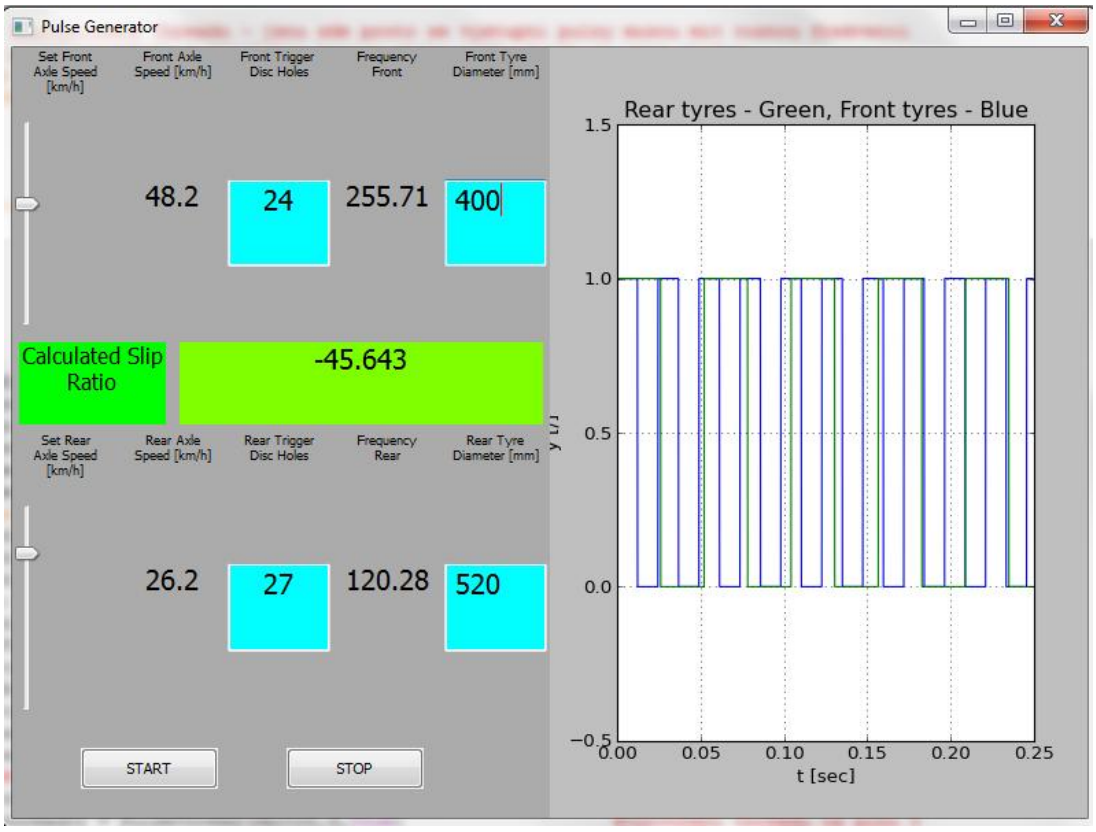
Obr. 5 – Event handler tlačítka pro start generování pulzů

Při použití tohoto kódu však není možné dále ovládat GUI. Pokud se o nějakou interakci pokusím celé GUI spadne. Tento problém by se dal odstranit použitím ještě jednoho threadu, který bude obsluhovat celé GUI.

Hotové GUI



Obr. 6 – Hotové spuštěné GUI



Obr. 7 – Hotové spuštěné GUI pro demonstraci s jinými hodnotami, zde je vidět správně vypočtená záporná hodnota slip ratia, a překreslení rozsahu osy x

Závěr

Cílem této práce bylo vyzkoušet tvorbu uživatelských rozhraní pro jazyk Python, seznámit se s možnostmi měřící karty LabJack a navrhnout zařízení pro testování kontroly trakce studentské formule na motorové brzdě. Seznámil jsem se s prací s thready pro Python. Podařilo se mi zprovoznit potřebné generování obdélníkových pulzů na dvou portech LabJacku. Původní záměr byl, aby bylo celé GUI ovladatelné v reálném čase a tak byla umožněna jednoduše změna parametrů výstupních signálů. Toto se mi nepodařilo splnit. Pro splnění tohoto záměru bych musel celé GUI vložit do třetího threadu a pak by celý program měl fungovat správně.