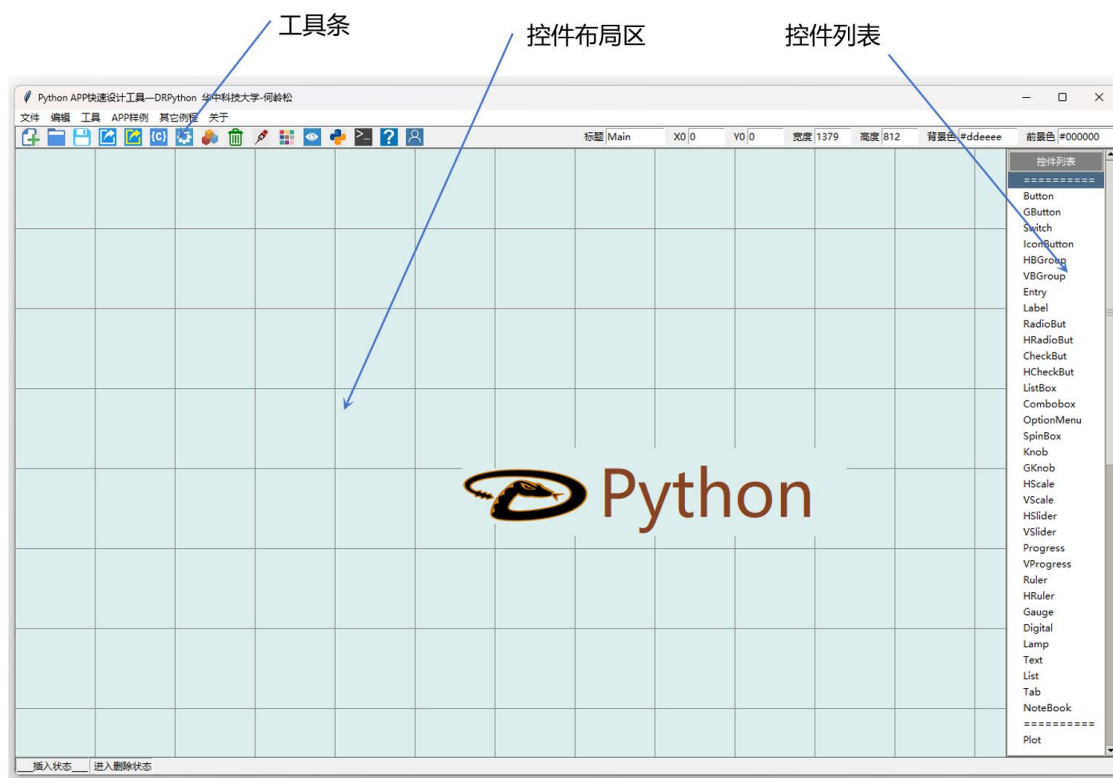


DRPython 图形化信号分析程序开发工具

1、简介

由于有 NumPy、SciPy 和 Matplotlib 等强大的第三方工具包，Python 在科学计算功能上与 Matlab 不相上下，但从 GUI 程序设计的角度，因缺少 Matlab AppDesigner 这样一个灵活的 GUI 程序框架开发工具，Python GUI 编程不算方便，需要耗费许多时间。

为此，我们以 DRVI 库的虚拟仪器控件为 GUI 编程基础，在 Python 内嵌的 IDLE 编辑器的基础上，设计了一个 Python GUI 程序框架开发工具，称为 DRPython。DRPython 的作用与 Matlab 中的 APPDesigner 有些类似，可以采用拖拽的方式将 drvi 库中的控件布置到 GUI 布局区，并调整其位置、大小等参数，进行 GUI 布局；然后使用 DRPython 中的连线功能，在控件间进行直连，建立数据流驱动的运行关系。最后，按所需的 Python 程序结构，导出可运行的 Python GUI 程序框架源码，完成 GUI 程序框架的图形化编程和快速设计；最后，用户在导出的 Python 框架程序代码的基础上进行修改和完善后，就可以快速形成自己的 Python 应用程序。



DRPython 程序运行界面图

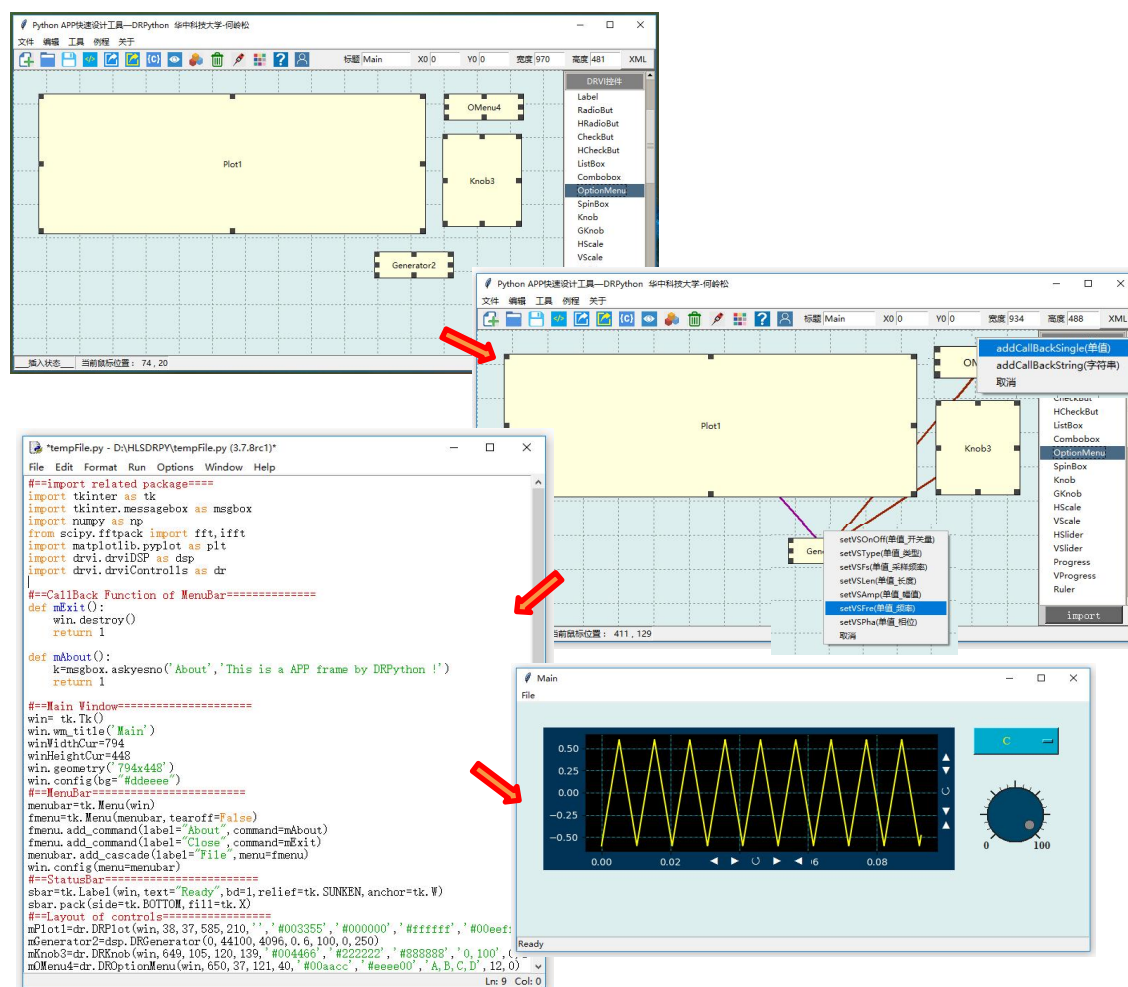
程序设计时，先从右侧的控件区将所需的控件拖入布局区，调整位置和大小；然后点击工具条上的连线按钮，进入连线状态，此时就可以在两个控件间进行运行关系连线；所有控件间的连线完成后，就形成了一个数据流驱动的 Python GUI 程序框架。这时，可以预览运行，看看效果；也可以导出标准 Python 程序框架或导出基于 Class 类的 Python 程序框架。这样就完成了 GUI 程序框架编写工作，可以把重心放在程序的功能实现上。

下面是一个简单的信号发生器程序的设计示例。首先，选择新建文件，建立一个空白的 GUI 设计区，然后拖入一个 Plot 曲线控件，用于显示信号曲线波形；再拖入一个 OMenu 弹出式菜单控件，用于选择信号类型；再增加一个 Knob 旋钮控件，用于设定信号频率；最后，拖入 Generator 信号发生器控件。至此，就完成了第一步的控件布局工作。

第二步，点击工具条上的连线按钮，进入连线状态。点击“OMenu 弹出式菜单控件”，在输出绑定菜单中选“addCallBackSingle”；然后点击“Generator 信号发生器控件”，在输入绑定菜单中选“setVS Type”，就完成了这两个控件的直连绑定，运行时弹出式菜单的选项变化，则信号的类型就变化。同理，完成“Knob 旋钮控件”输出与“Generator 信号发生器控件”信号频率设定方法的输入绑定，实现用旋钮改变信号发生器频率。最后，完成“Generator 信号发生器控件”曲线输出与“Plot 曲线控件”信号设定输入方法的绑定，实现信号发生器控件输出信号的波形显示。这样，一个可以运行的数据流驱动的 Python GUI 程序框架就完成了。

第三步，导出 Python 程序源代码到 Python IDLE 编辑器中。

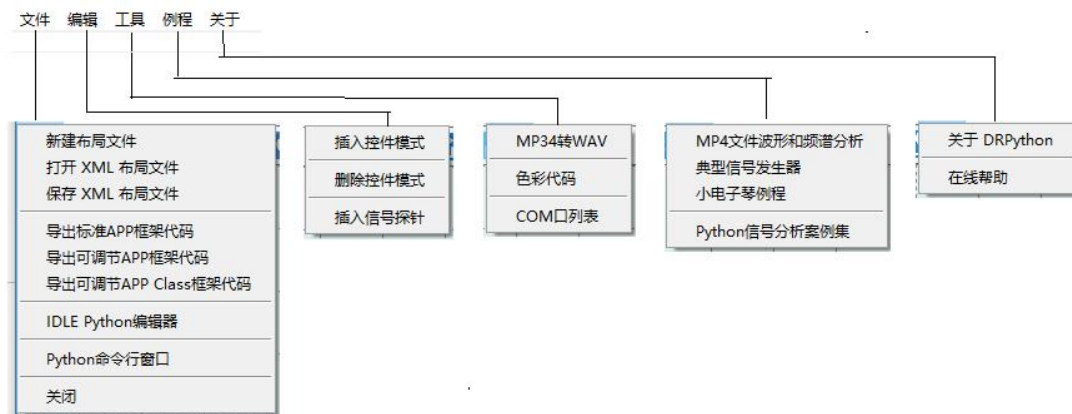
第四步，运行、调试、修改、完善。



DRPython 工作流程示意图

2、菜单

下面是 DRPython 的菜单功能一栏：



2.1、文件子菜单

2.1.1 新建布局文件：

清空当前控件布局区，建立一个新的 GUI 设计文件。

2.1.2 保存 XML 布局文件：

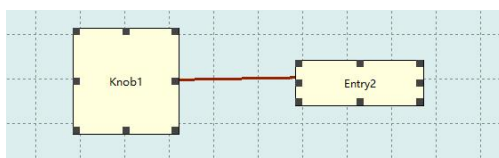
弹出文件保存菜单，将当前控件布局区的 GUI 设计和控件间的连线情况，以 XML 文件形式保存，以便于将来继续设计和修改。

2.1.3 打开 XML 布局文件：

弹出文件读取菜单，选择 XML 文件形式的布局文件，将保存在 XML 文件中的 GUI 设计加载到 GUI 布局区，继续进行编辑。

2.1.4 导出标准 APP 框架程序代码：

根据布局区中插入的控件和控件间的连线关系，自动生成 Pythonk 框架程序代码，包括控件插入代码、控件连接代码和回调函数等代码，用户可以在其基础上进行快速修改，提高 APP 设计效率。例如，下面是一个简单的 GUI 程序设计，插入一个旋钮控件和一个文本框控件，然后通过直连将旋钮的值传递到文字框控件中进行显示。



旋钮控件和文本框控件构成的样例程序布局图

下面是导出的 Python 标准 APP 框架程序代码：

```
import tkinter as tk
import tkinter.messagebox as msgbox
import drvi.drviControls as dr
#==CallBack Function of MenuBar=====
def mExit():
    win.destroy()
    return 1
```

```

def mAbout():
    k=msgbox.askyesno('About','This is a APP frame by DRPython !')
    return 1

#==Main Window=====
win= tk.Tk()
win.wm_title('Main')
winWidthCur=1204
winHeightCur=716
win.geometry('1204x716')
win.config(bg="#ddeeee")
#==MenuBar=====
menubar=tk.Menu(win)
fmenu=tk.Menu(menubar,tearoff=False)
fmenu.add_command(label="About",command=mAbout)
fmenu.add_command(label="Close",command=mExit)
menubar.add_cascade(label="File",menu=fmenu)
win.config(menu=menubar)
#==StatusBar=====
sbar=tk.Label(win,text="Ready",bd=1,relief=tk.SUNKEN,anchor=tk.W)
sbar.pack(side=tk.BOTTOM,fill=tk.X)
#==Layout of controls=====
mKnob1=dr.DRKnob(win,89,42,120,120,'#004466','#222222','#888888','0,100',0,100,20,12)
mEntry2=dr.DREntry(win,342,81,145,52,'#ffffff','#000000','100',12)
#==User defined data probe=====
mKnob1.addCallBackSingle(mEntry2.setValueSingle)
#==Main Loop=====
win.mainloop()
#==End of APPp=====

```

2.1.5 导出可调节 APP 框架程序代码:

可调节 APP 框架程序和标准 APP 框架程序的区别就是增加了一个 Python GUI 界面运行时可以随用户对主窗口大小的调整而变化。因此，增加了一个窗口变化回调函数，通过该函数对主窗口中的 GUI 控件进行展缩。下面是“旋钮+文本框”样例导出的可调节 APP 框架程序代码，与前者的区别是增加了蓝色斜体代码部分的自适应窗口大小调整。

```

import tkinter as tk
import tkinter.messagebox as msgbox
import drvi.drviControlls as dr
#==Widgets Resize=====
def winResize(event):
    w=win.winfo_width()
    h=win.winfo_height()
    mWidSize.resize(w,h)

#==CallBack Function of MenuBar=====
def mExit():
    win.destroy()
    return 1

def mAbout():
    k=msgbox.askyesno('About','This is a APP frame by DRPython !')
    return 1

#==Main Window=====
win= tk.Tk()
win.wm_title('Main')

```

```

winWidthCur=1204
winHeightCur=716
win.geometry('1204x716')
win.config(bg="#ddeece")
win.bind("<Configure>",winResize)
mWidSize=dr.DRResize(winWidthCur,winHeightCur)
#==MenuBar=====
menubar=tk.Menu(win)
fmenu=tk.Menu(menubar,tearoff=False)
fmenu.add_command(label="About",command=mAbout)
fmenu.add_command(label="Close",command=mExit)
menubar.add_cascade(label="File",menu=fmenu)
win.config(menu=menubar)
#==StatusBar=====
sbar=tk.Label(win,text="Ready",bd=1,relief=tk.SUNKEN,anchor=tk.W)
sbar.pack(side=tk.BOTTOM,fill=tk.X)
#==Layout of controls=====
mKnob1=dr.DRKnob(win,89,42,120,120,'#004466','#222222','#888888','0,100',0,100,20,12)
mEntry2=dr.DREntry(win,342,81,145,52,'#fffff','#000000','100',12)
#==User defined data probe=====
mKnob1.addCallBackSingle(mEntry2.setValueSingle)
#==Add sizeable widgets=====
mWidSize.addDRWidget(mKnob1)
mWidSize.addDRWidget(mEntry2)
#==Main Loop=====
win.mainloop()
#==End of APPp=====

```

2.1.6 导出可调节 APP Class 框架程序代码:

Python 支持类 Class，因此，可以将整个 GUI 布局设计作为一个类 Python Class 导出。这样导出的 Python 框架程序的结构更好，独立性更强。下面是“旋钮+文本框”样例导出的可调节 APP Class 框架程序代码，注意与前者的区别。

```

import tkinter as tk
import tkinter.messagebox as msgbox
import drvi.drviControlls as dr
#==Start APP Class=====
class myAPPClass:
    def start(self):
        self.tMain=tk.Frame(self.win,bg=win.cget("bg"))
        self.tMain.pack(side=tk.LEFT,fill=tk.BOTH,expand=True)
        self.innerOpen(self.tMain);

    def destroy(self):
        self.tMain.destroy();

    def winResize(self,event):
        w=self.tMain.winfo_width()
        h=self.tMain.winfo_height()
        if (w<100) or (h<100): return
        if hasattr(self,"mWidSize"):
            self.mWidSize.resize(w,h)
        else:
            self.mWidSize=dr.DRResize(w,h)
            self.addSizeWidget()

    def addSizeWidget(self):
        #==Add sizeable widgets=====
        self.mWidSize.addDRWidget(self.mKnob1)

```

```

        self.mWidSize.addDRWidget(self.mEntry2)

def __init__(self,win=None):
    #=vars=====
    self.win=win

def innerOpen(self,win):
    self.tMain.bind("<Configure>",self.winResize)
    self.mKnob1=dr.DRKnob(win,89,42,120,120)
    self.mEntry2=dr.DREntry(win,342,81,145,52)
    self.mKnob1.addCallBackSingle(self.mEntry2.setValueSingle)

#==Main Program=====
def mExit():
    win.destroy()

#==About==
def mAbout():
    k=msgbox.askyesno("About","This is a APP Class frame by DRPython !")

#==Main Window=====
win= tk.Tk()
win.wm_title("APP Class  样例")
win.geometry("1204x716")
win.config(bg="#ddeeee")
#==MenuBar==
menubar=tk.Menu(win)
fmenu=tk.Menu(menubar,tearoff=False)
menubar.add_cascade(label="About",command=mAbout)
menubar.add_cascade(label="Exit",command=mExit)
win.config(menu=menubar)
myApp=myAPPClass(win);
myApp.start();
#==Main Loop==
win.mainloop()
#==End of Main Program=====

```

2.1.6 导出可调节 pyqt5 框架程序代码:

DRVI 也支持用 pyqt 作为底层 GUI 框架，从便携的角度，只移植了 QT5 对应的 pyqt5，试过 QT6 的 pyside6 也行，切换下 import 即可，但对应的包从 100M 增大到 800M，做课堂上随身携带的便携版本时不方便；同时因为基本用不到 QT6 改进的 3D 显示部分，故没移植 pysid6 版本；但也不是多大事，以后需要时就是头文件换一下，再重新用 cython 编译一下就完成升级。注意，DRVI 的 pyqt 版本基于 pyqt5，那一定要先安装 pyqt5：

```

python.exe -m pip install PyQt5
python.exe -m pip install pyqtchart

```

这里通过控件的 DRVI 数据流驱动封装屏蔽了 Tkinter GUI 库与 pyqt GUI 库的编程区别，因此编程方法完全相同，只是引用的基础 GUI 库不同而已。点击工具条中的带 qt 的图标按钮，就可以将导出基于 pyqt5 的类结构的 Python 程序框架。下面是“旋钮+文本框”样例导出的 pyqt 框架程序代码。

```

from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
from PyQt5.QtCore import *
import sys

```

```

import numpy as np
import drvi.drviDSP as dsp
import drvi.drviQControls as dr

#===Start of MainClass=====
class drviMain(QMainWindow):
    def __init__(self):
        super(drviMain,self).__init__()
        self.initGUI()

#===Callback of menu=====
def myFile(self, q):
    if q.text() == "Close":
        sys.exit(app.exec_())
    if q.text() == "About":
        QMessageBox.information(QWidget(),"About", "This is a APP frame by DRPython", QMessageBox.Yes)

#===Init GUI=====
def initGUI(self):
    self.setWindowTitle("APP Frame by DRPython")
    self.resize(1204,716)
    self.win=QWidget()
    self.win.setStyleSheet("background-color : #ddeeee; ")
    self.setCentralWidget(self.win)

#Add a statusBar&Menu=====
self.statusBar=QStatusBar()
self.setStatusBar(self.statusBar)
self.statusBar.setStyleSheet("background-color : #eeeeee; ")
self.statusBar.showMessage("Ready",0)
self.menuBar=self.menuBar()
self.menuBar.setStyleSheet("background-color : #eeeeee; ")
self.fileItem=self.menuBar.addMenu("File")
self.fileItem.addAction("About")
self.fileItem.addAction("Close")
self.fileItem.triggered.connect(self.myFile)
#Add controls
self.mKnob1=dr.DRKnob(self.win,85,59,120,120,'#004466','#222222','#888888','0,100',0,100,20,12)
self.mEntry2=dr.DREntry(self.win,319,92,100,30,'#ffff','#000000','100',12)
self.mKnob1.addCallBackSingle(self.mEntry2.setValueSingle)

#===Sizer =====
def resizeEvent(self, event):
    w=event.size().width()
    h=event.size().height()
    if (w<100) or (h<100): return
    if hasattr(self,"mWidSize"):
        self.mWidSize.resize(w,h)
    else:
        self.mWidSize=dr.DRRresize(w,h)
        self.addSizeWidget()

def addSizeWidget(self):
    self.mWidSize.addDRWidget(self.mKnob1)
    self.mWidSize.addDRWidget(self.mEntry2)

#===Main Loop=====
app=QApplication(sys.argv)
app.setStyle("Fusion")
win=drviMain()
win.show()
sys.exit(app.exec())

```

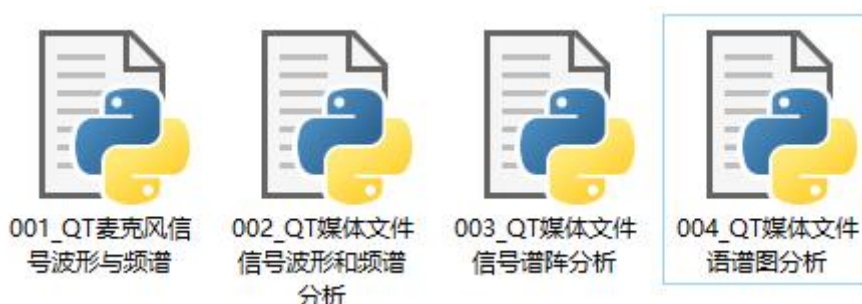
除了底层 GUI 由 Tkinter 切换到 pyqt 外，在显示形式上并无区别。使用 pyqt 代替 Tkinter 的主要优

点是基于 C++ 的 pyqt 比基于 Python 的 Tkinter 绘图速度明显快很多。这样调用 Matplotlib 绘制波形、频谱时，程序不容易卡顿。另外，在绘制二维曲线时，还可以完全用 C++ 写的 Qchart 来代替 Python 写的 Matplotlib，那显示速度快的就不是一点半点了。

但 Qchart 是一堆需要 DIY 的绘图散件，集成度远不如 Matplotlib 高。因此，只在 Qchart 的基础上封装了实时曲线显示控件 DRPlotFast、实时谱阵图显示控件 DRArrayX 和实时语谱图控件 DRMeshFast。其余曲线显示控件仍然是基于 Matplotlib。有实时显示需求时，尽量使用这三个基于 Qchart 的控件来代替基于 Matplotlib 的 DRPlot、DRPlotSurface 和 DRPlotContour 控件。

- 1、DRPlotFast(win,x0,y0,w,h,st=0,xlow=0,xhigh=0,ylow=0,yhigh=0,xarray=0,yarray=0)
- 2、DRArrayX(win,x0,y0,w,h,cb='#003355',cf='#ffffff',cst=0,dx=1,dy=3,xlow=0,xhigh=10000,ylow=0,yhigh=0.4)
- 3、DRMeshFast(win,x0,y0,w,h,cb='#003355',cf='#ffffff',M=400,xlow=0,xhigh=5000,ylow=0,yhigh=1)

显示效果可参略案例集中的四个演示实验：



另外，pyqt 的媒体播放功能很强，因此基于 QtMultimedia 的麦克风声音信号采集控件、MP3/MP4 媒体文件音频信号采集控件，功能也远强于 Tkinter 版本的控件。特别是 MP4 视频文件播放控件，不再需要外挂 FFMPEG。

2.1.7 IDEL 编辑器：

弹出 IDEL 编辑器，可以打开一个 Python 源码程序进行编辑和调试。

2.1.8 Python 命令行窗口：

弹出 Python 命令行窗口，可单步执行 Python 命令或代码。

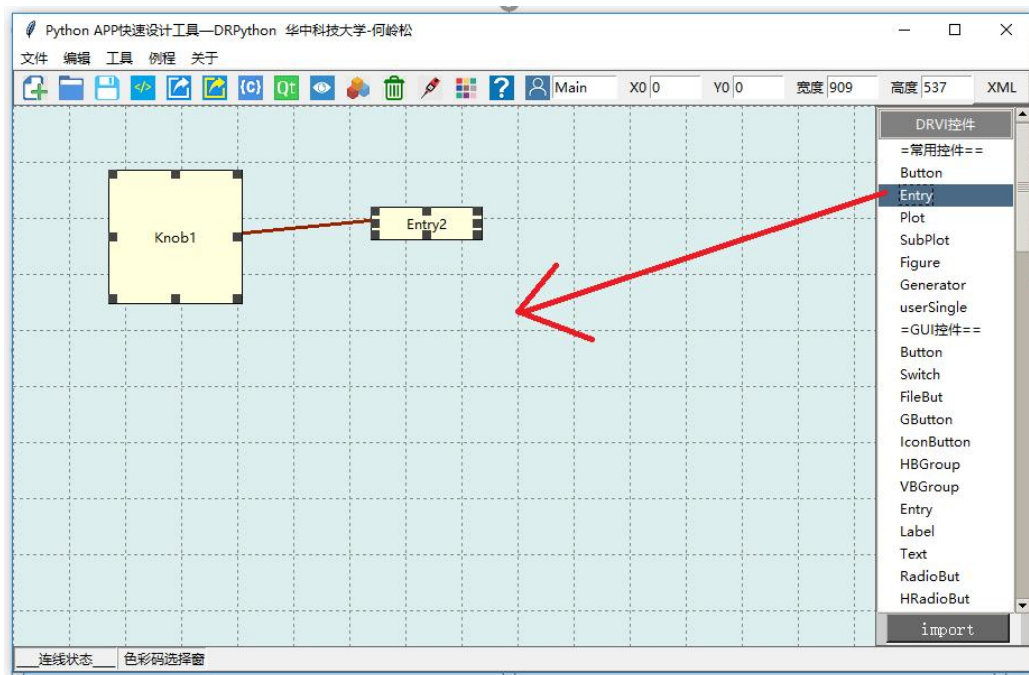
2.1.9 关闭：

退出 DRPython 程序。

3、编辑子菜单

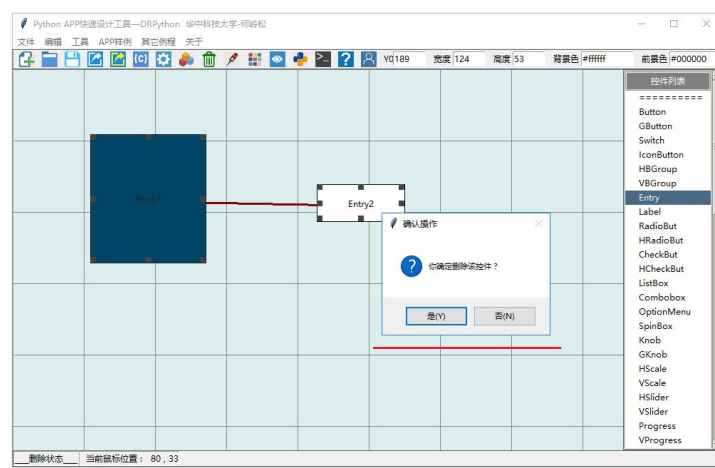
3.1 插入控件模式：

把 DRPython 设定为插入模式，这时可以将右侧控件栏中的控件添加到布局区，也可以调整已插入控件的位置和大小。



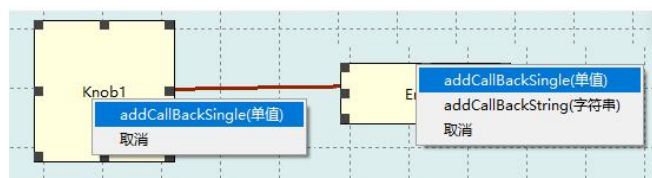
3.2 删除控件模式:

把 DRPython 设定为删除模式，这时点击布局区中已插入的控件，就可以对这些控件进行删除。



3.3 插入信号探针:

进入连线模式，可以在两个控件间进行连线。方法是将前一个控件的输出与后一个控件的输入相连，注意所选的输出端变量类型和数量必须与输入端一致。



4、工具子菜单

4.1 COM 口列表:

列出计算机上当前连接的串口设备。如下图所示。



4.2 色彩代码:

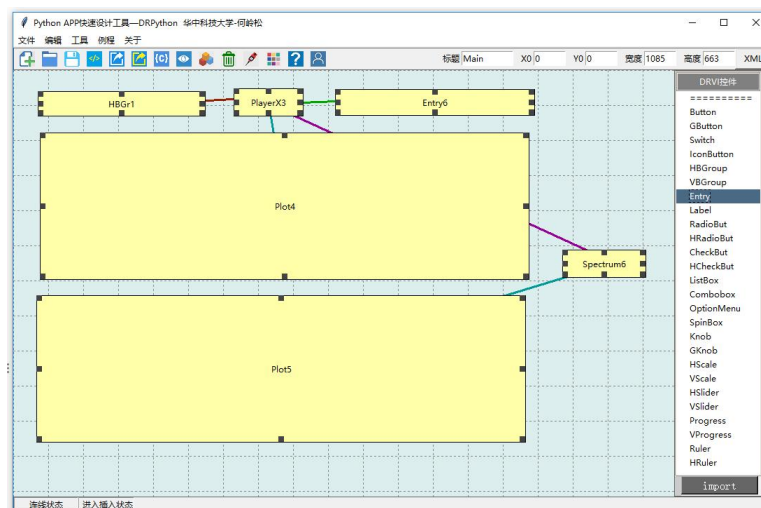
Tkinter 采用 16 进制的色彩码，为便于色彩选择，该功能弹出色彩选择窗，并将当前选择的色彩码贴在剪贴板上返回。例如，若选“绿色”，剪贴板返回值为“#00FF00”。



5、例程

5.1 MP4 文件波形和频谱分析

点击该菜单条，则加载保存的 XML 设计文件（MP4 文件波形和频谱分析.xml）中的设计内容到 GUI 布局区，如下所示：

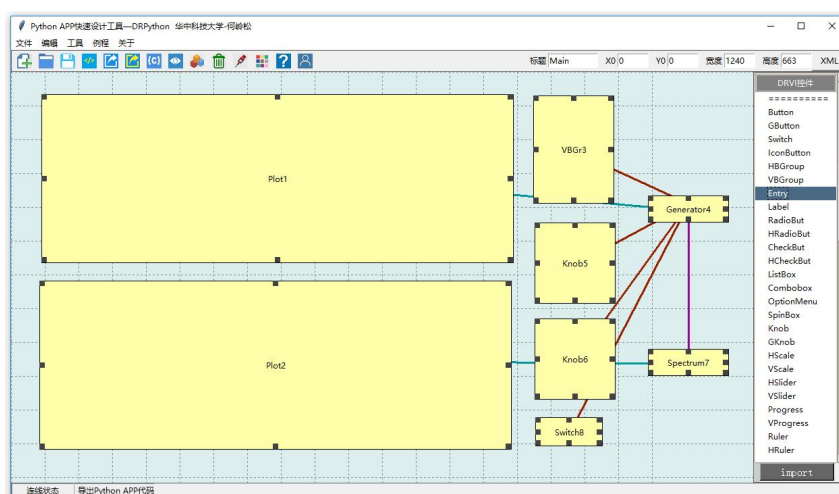


可以导出 Python 源代码，进行编辑、运行；程序运行效果如下：

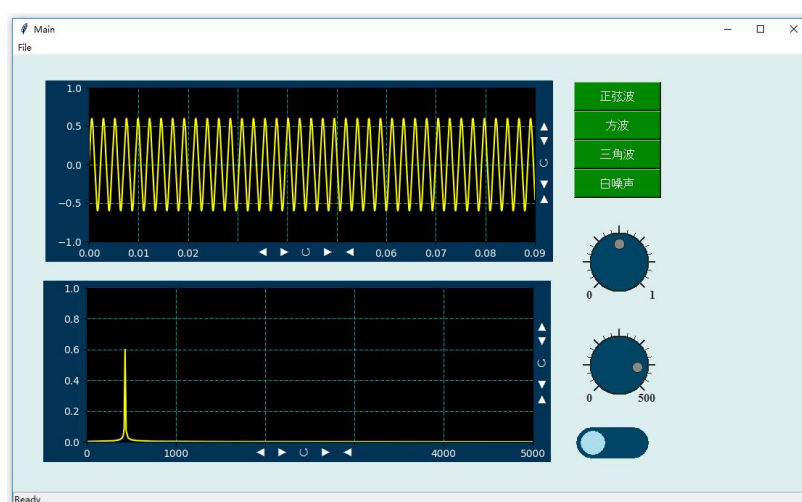


5.2 典型信号发生器:

点击该菜单条，则加载保存的 XML 设计文件（典型信号发生器.xml）中的设计内容到 GUI 布局区，如下所示：

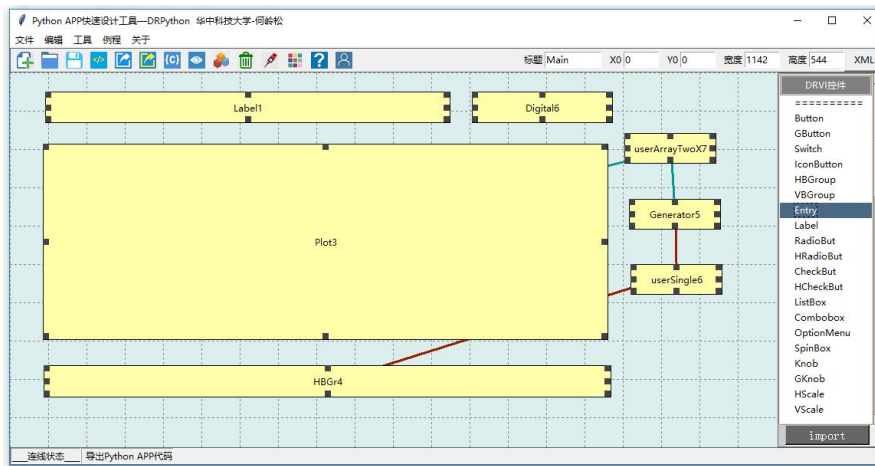


直接导出的 Python 框架程序代码运行效果如下：



5.3 小电子琴例程:

点击该菜单条，则加载保存的 XML 设计文件（小电子琴例程.xml）中的设计内容到 GUI 布局区，如下所示：



直接导出的 Python 框架程序代码运行效果如下：



6、关于子菜单

6.1 关于 DRPython:

弹出一个网页窗口，显示 DRPython 的开发目的、背景等。

6.2 在线帮助:

连接“DRVI 编程助手”智能体，助教小 D，用智能体来进行 drvi 资源包编程应用方面的辅导和问答。在豆包中查找“DRVI 编程助手”就可以找到。



小 D 很专业和优秀，有问必答：



7、工具条

下面是 DRPython 的工具条和快捷按钮，其按钮功能定义与菜单条中相应条目的功能一致，可参略菜单部分。



各按钮的功能依次为：

- 新建布局文件
- 打开 XML 布局文件
- 保存 XML 布局文件
- 编辑 XML 布局文件
- 导出标准 APP 框架程序代码
- 导出可调节 APP 框架程序代码
- 导出可调节 APP Class 框架程序代码
- 导出可调节 pyqt5 APP Class 框架程序代码
- 程序运行效果预览
- 进入控件插入状态
- 进入控件删除状态
- 进入控件连线状态
- 16 进制色彩码查询
- 在线智能体帮助
- 关于