Universidad de Costa Rica
Escuela de Ingeniería Eléctrica
IE0117 – Platafomas Abiertas

Cliente Control Inteligente De Energía Para Robot Humanoide Por Red

Ricardo Gutiérrez Piedra, B23105

27 de noviembre de 2016

Índice

Índice de figuras

1. Descripción

Este proyecto consta en el desarrollo de un cliente gráfico el cual que se pueda comunicar con el software principal por medio de red y así apagar y encender partes del robot humanoide.

2. Objetivo General

Desarrollar una interfaz gráfica para el cliente de Yarp del software que permita encender y apagar hasta 8 dispositivos por red utilizando una caja de relay Ethernet.

3. Objetivos Específicos

Desarrollar el software del cliente de Yarp el cual pueda comunicarse con el software. Desarrollar una interfaz gráfica para este cliente de Yarp utilizando sus librerías de Yarp o Python según lo que se determine mejor para el proyecto.

4. Justificación

En el desarrollo de un robot humanoide siempre es importante el poder en cualquier momento deshabilitar alguna de sus partes ya sea en caso de algún

accidente, seguridad o simplemente para mantenimiento parcial del mismo y no dejar fuera de servicio por completo al robot.

Así mismo es primordial tener una forma de poder ver el estado de los relays en tiempo real pues en caso de alguna falla se tendría la información de inmediato y se podría trabajar en ello.

5. Metodología

Primero se investigo a fondo la librería de Yarp para crear este cliente y se realizaron pruebas para determinar el buen funcionamiento con nuestro programa principal. Después de investigar librerías de Python como easyGui, TKInter y algunas directamente de Yarp se estableció que la mejor opción fue utilizar kivy la cual es de python, muy completa para lo que se quiere y aparte vista en clase.

6. Marco Teórico

Se utilizaron diferentes librerías en la programación del cliente tanto para su parte solamente por consola como para la parte de la interfaz gráfica.

6.1. Yarp

Librería enfocada en la robótica, permite crear sistemas de control para los robots y permite la comunicación peer-to-peer. En esta caso la utilizamos para generar la comunicación entre el cliente y el programa principal de encendido de los relays. Para realizar pruebas se emula una red de yarp creando un server local, como se puede ver a continuación.

6.2. Kivy

Librería de python para el desarrollo de aplicaciones que involucren interfaz gráfica. Se utilizó en el desarrollo de la parte gráfica del cliente de yarp,en los botones, en las etiquetas, en la ventana que se despliega. Ademas se hizo uso de una particularidad de kivy la cual nos permite asignar eventos a ciertas interacciones, como por ejemplo en los botones, tienen un estado y al ser oprimidos cambian a otro estado el cual realiza un evento, es decir llama

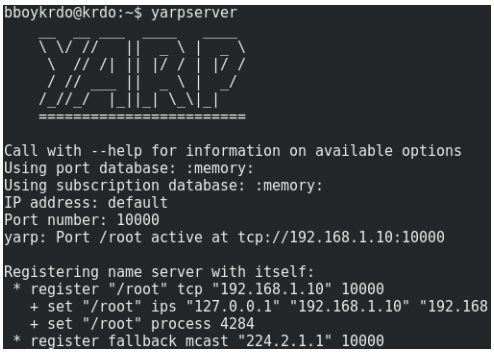

Figura 1: Yarp Server

alguna función que realice alguna tarea, en nuestro caso envía los comandos para encender, apagar los relays. Funciones de kivy:

1. Layout
2. Labels
3. ToggleButtons down/normal
4. Events

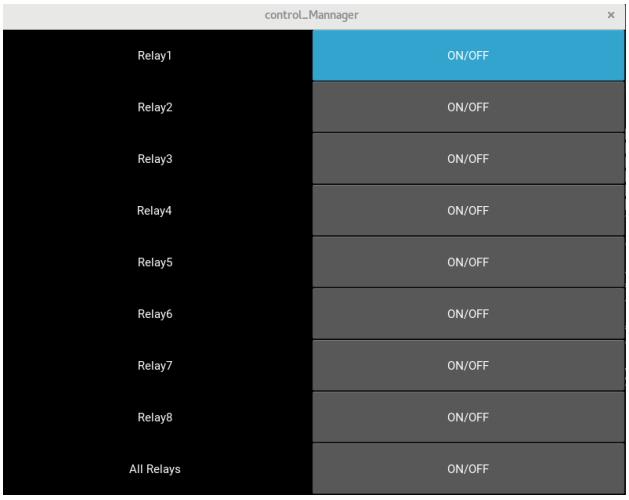

Figura 2: Interfaz Gráfica

6.3. Optparse

Librería de python que se utilizo para enviar los comandos desde la interfaz por medio de la consola al software principal. Se utilizó para enviar los comandos para encender o apagar cualquiera de los relays. Estos son los comandos a utilizar desde la consola:

1. $--help$
2. $-r < numeroderelay >$
3. $--on$
4. $--off$

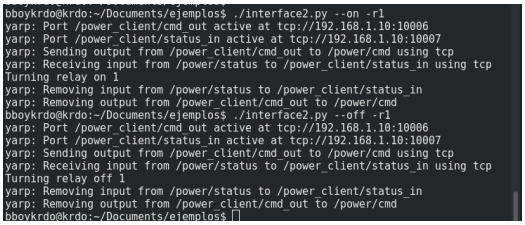

Figura 3: Cliente consola

7. Desarrollo del Cliente

El código quedó como se muestra a continuación, una clase RelayButton que es la encargada de crear lo que son todas las etiquetas y botones en kivy asi mismo realiza los llamados a los eventos según el estado de los botones, una clase MainScreen que permite que se inicie el llamado a la pantalla que despliega la información y se hagan los llamados a las clase RelayButton y por ultimo una clase para inicializar el programa de kivy.

```
#!/usr/bin/python
from yarp import BufferedPortBottle as buffport
from yarp import Value
import yarp
from optparse import OptionParser
from kivy.app import App
```

```
from kivy.uix.button import Button
from kivy.uix.gridlayout import GridLayout
from kivy.uix.label import Label
from kivy.lang import Builder
from kivy.uix.togglebutton import ToggleButton

class RelayButton(ToggleButton):

    def __init__(self, relay_num=0, **kwargs):
        super(RelayButton, self).__init__(**kwargs)
        self.text='ON/OFF'
        self.relay_num=relay_num

    def on_state(self, widget, value):

        if value == 'down':
            print 'Relay on', str(self.relay_num)
            bottle=client_port_out.prepare()
            bottle.clear()
            bottle.addInt(int(self.relay_num))
            client_port_out.write()

        else:
            off_select= 10
            all_relay_select=1
            print 'Relay OFF', str(self.relay_num+off_select)
            bottle=client_port_out.prepare()
            bottle.clear()
            if int(self.relay_num) < 9:
                bottle.addInt(int(self.relay_num)+off_select)
            else:
                bottle.addInt(int(self.relay_num)+
                    all_relay_select)
            client_port_out.write()



class MainScreen(GridLayout):
    def __init__(self, **kwargs):
        super(MainScreen, self).__init__(**kwargs)

        self.cols=2
        relay_buttons=[RelayButton(relay_num=(i+1)) for i in
            xrange(9)]
```

```
        relay_labels=[Label(text='Relay'+str(i+1)) for i in
            xrange(9)]
        relay_labels[-1].text='All_Relays'
        [(self.add_widget(pair[0]),self.add_widget(pair[1])) for
             pair in zip(relay_labels,relay_buttons)]


class control_Mannager(App):
    def build (self):
        return MainScreen()

if __name__ == "__main__":


    yarp.Network.init()
    client_port_out = buffport()
    client_port_out.open("/power_client/cmd_out")

    client_port_in = buffport()
    client_port_in.open("/power_client/status_in")

    yarp.Network.connect("/power_client/cmd_out","/power/cmd")
    yarp.Network.connect("/power/status","/power_client/
        status_in")

    control_Mannager().run()
```

8. Concluciones

Se logró configuarar los dispositivos de KM tronic en la primera parte del proyecto y concluir con el software que envié los comandos necesarios para encender o apagar los relays. Se logro programar un cliente tanto de consola como gráfico con conexión de yarp al software principal. Con todo lo anterior unificado el control de energia por medio de red para el robot humanoide enciende y apaga cada relay por separado y además de ello también tiene como opción apagar o encender todos los relay al mismo tiempo. De igual forma brinda con el software principal el estatus de cada relay en tiempo real por ende muestra cambios de igual forma.

9. Github

En el github mostrado a continuación se puede encontrar todo el código tanto el programa principal como el cliente de consola y la interfaz gráfica.

https://github.com/rkrdogp/Control_Inteligente

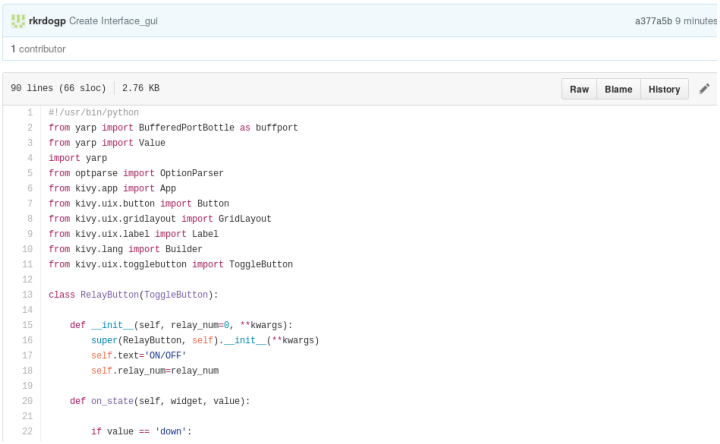

Figura 4: Proyecto en Github

10. Referencias

Referencias

[1] *https://pypi.python.org/pypi/httpretty.*

[2] *http://docs.python-requests.org/en/master*

[3] *https://docs.python.org/3/library/re.htmltodos*

[4] *https://kivy.org/home*

[5] *https://docs.python.org/2/library/optparse.html*