Desarrollo de un Software Libre para la lectura, graficación y registro en tiempo real de señales provenientes de ondas cerebrales

Carlos Antonio Bulnes Domínguez

28 de mayo de 2015

Índice general

Introducción

El trabajo presentado describe la realización de un software libre desarrollado con ROS para ser utilizado con un lector de ondas cerebrales. El nuevo software tiene como objetivo la lectura, graficación y registro de los datos que reciba a través de la plataforma ROS. Durante la investigación se encontró que ya existía un software similar, sin embargo este resultaba muy lento lo que provocaba que la información en pantalla no fuera fiable. El software resultante en este trabajo tiene como objetivo resolver ese problema, es por eso que, como se explicará en capítulos posteriores, se escogió integrarlo a ROS.

El nuevo software permitirá controlar el momento en que se quiera Ejecutar, Pausar y Detener la recepción de la información, estas acciones no afectarán al programa que envía los datos pues solo afecta de forma visual al programa receptor para que el usuario tenga control de que quiere ver y en que momentos. Adicionalmente cuenta con la funcionalidad de visualizar y crear un registro de todas las señales que se reciban mientras el software se encuentre en modo de "reproducción". La visualización es e tiempo real mientras que la generación del registro se genera cuando se presiona "detener" y creará un archivo en disco duro con el nombre que se haya ingresado así como la fecha y hora de creación de este, dicho archivo permitirá al usuario consultarlo en cualquier momento independientemente de que el software se encuentre en ejecución o no.

Justificación

La necesidad de la creación de un software libre para la lectura de las ondas cerebrales surge debido a la escaza variedad de programas dedicados a dicha tarea actualmente, donde es el mismo creador del dispositivo físico el que te proporciona el software, el cual, ya cuenta con funciones y operaciones definidas por el fabricante.

En la actualidad los proyectos de investigación requieren interactuar más con la información que obtienen con los dispositivos lectores de ondas cerebrales, y como ya se mencionó, las alternativas actuales se encuentran limitadas, se propone entonces un proyecto de código libre en el cual, partiendo del software desarrollado en este trabajo, los desarrolladores futuros sean capaces de complementarlo e implementarlos a sus necesidades específicas.

Objetivos

Objetivo General

Crear un software libre capaz de leer y graficar en tiempo real a través de la plataforma ROS las señales enviadas por el Emotiv EPOC, las cuales son interpretadas por un software libre llamado Emokit[1].

Objetivos Particulares

- Establecer un canal de comunicación entre el software y el EEG.
- Obtener datos numéricos a partir de las señales obtenidas.
- Tomar esos datos en tiempo real y graficarlos en una relación frecuencia-tiempo.
- Establecer un canal de salida para enviar los datos interpretados.

Antecedentes

El software desarrollado en este trabajo toma como base al emokit[1] el cuál forma parte de OpenYou[1]. El Emokit lee, descifra e interpreta la información enviada por el Emotiv EPOC, el cual es un dispositivo "Interfaz Cerebro-Máquina" (BCI), tales como nivel de batería de la diadema, intensidad de la señal, y las 14 lecturas realizadas por la diadema; este software actualmente solo imprime a nivel terminal dichos datos.

Una "Interfaz Cerebro-Máquina"[2] (BCI) es un medio de comunicación directo entre el cerebro y un dispositivo externo. Los BCI están normalmente dirigidos a asistir, aumentar y reparar la habilidad cognitiva y las funciones sensoriomotoras.

Las inventigaciones con los BCI comenzaron en 1970 en la Universidad de Los Ángeles California(UCLA) bajo el subsidio de Fundación Nacional de Ciencia contratados por la Agencia de Proyectos de Investigación Avanzados de Defensa (DARPA). La publicación hecha después de la investigación marcó la primera aparición de la expresión "Interfaz Cerebro-Máquina" en la literatura científica. En la actualidad existen tres tipos de BCI:

- BCI invasivos: Son implantados en la materia gris del cerebro por medio de una neurocirugía, debido a que se encuentran implantados en el cerebro los BCI invasivos producen la mejor calidad de las señales pero pueden tener consecuencias negativas al portador a futuro.

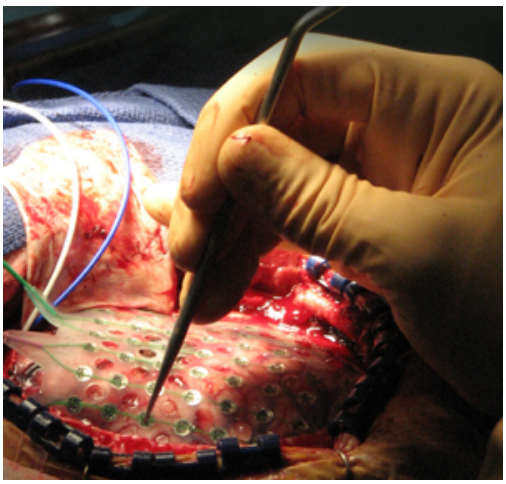

Figura 1: BCI invasivo[3]

- BCI semi invasivos: Son implantados dentro del cráneo pero sin tocar la corteza cerebral. Producen una mejor señal que los BCI no invasivos y tienen menor riesgo de causar daños al cerebo que los BCI invasivos.

- BCI no invasivos: Los implantes no invasivos son colocados en el cuero cabelludo. Los BCI no invasivos producen señales débiles porque el cráneo amortigua las señales, dispersando las ondas electromagnéticas creadas por las neuronas.

Figura 2: BCI no invasivo[4]

El Emotiv Epoc es BCI que fue creado con el propósito de ser un periférico para juegos en Windows, OS X y Linux[5], cuenta con 14 electrodos y funciona como dispositivo de entrada. En 2011 Kirill Stytsenko, et al.[6] realizaron un análisis del Emotiv EPOC para la CogSci Conference. Emotiv EPOC es un BCI de bajo costo basado en la tecnología EEG. Cuenta con 14 electrodos montados en una diadema inalámbrica que se coloca sin esfuerzo y se conecta a la computadora. Originalmente fue creada para los juegos de computadora pero la "research edition" permite el acceso a los datos para su análisis lo que abre nuevas posibilidades a la ciencia para realizar nuevos experimentos o integrarlo a los ya existentes. En dicho estudio se someten a diferentes pruebas al Emotiv EPOC y al G-TEC[7]. Al compararlos se obtiene que la información en general es igual, pero la señal es más clara e intensa en el G-TEC. Uno de los desafíos encontrados es la creación de software de grabación para ambos dispositivos.

Job Ramón de la O Chávez en su tesis Interfaz Cerebro - Computadora para el Control de un Cursor Basado en Ondas[8] plantea una interfaz que permita la comunicación entre el usuario y la computadora, haciendo uso de sus ondas cerebrales, para el control de un cursor en pantalla mediante comandos obtenidos de las lecturas de un amplificador de ondas cerebrales.

En EPOC-alypse Mind Controlled Car[9] plantean la construcción de de un carro de control remoto que es controlado por la mente usando el Emotiv EPOC. El proyecto fue desarrollado utilizando el SDK oficial del Emotiv EPOC.

Asim Raza en SSVEP based EEG Interface for Google Street View Navigation[10] analiza los sistemas BCI y su aplicación en el mundo real. También desarrolla un prototipo interactivo que pueda ser controlado en un ambiente controlado para demostrar el funcionamiento de los sistemas BCI. Para el desarrollo decidió utilizar el software libre OpenViBE para la adquisición y procesamiento de las señales.

En ROS: an open-source Robot Operating System[11] se explica el uso de la plataforma ROS para el desarrollo de aplicaciones de robótica y resumen los objetivos filosóficos de ROS en:

- Per-to-per

- Basado en herramientas
- Multilenguaje
- Ligero
- Gratuito y de Código Abierto

En Things that twitter: social networks and the internet of things[12] utilizan ROS aplicado en las redes sociales. ROS permite intercambiar información por medio de servicios con mensaje de request y response definidos. La información es intercambiada por una arquitectura publish/suscribe donde los procesos permiten que sus datos estén disponibles para que otros procesos puedan utilizarlos.

Diseño

La Universidad Veracruzana adqurió un Emotiv EPOC en 2014 para el laboratorio de robótica donde los alumnos de la Facultad de Ingeniería participan en diferentes proyectos tecnológicos. Este trabajo de tesis contribuye a un proyecto donde se controlará un robot por medio de la mente a través del Emotiv EPOC. Con el proyecto resultante de este trabajo se tendrá una interfaz la cual graficará las señales del Emotiv EPOC además de tener la capacidad de grabar los datos obtenidos cada vez que se ejecute un "experimento" obteniendo un archivo separado por comas.

Al momento de la investigación se encontraron algunos software que mostraban la información obtenida de forma gráfica, sin embargo el principal inconveniente de estos era el desfase entre los datos que se estaban obteniendo y su graficación. Para atacar ese problema en este proyecto se utilizó un framework llamado ROS. ROS (Robot Operating System)[13] provee librerías y herramientas para ayudar a los desarrolladores de software a crear aplicaciones para robots. ROS provee abstracción de hardware, controladores de dispositivos, librerías, herramientas de visualización, comunicación por mensajes, administración de paquetes y más. ROS está bajo la licencia open source, BSD. En este proyecto se utilizó ROS para crear un escenario de comunicación en tiempo real, donde el Emokit[1] publica a través de mensajes los datos obtenidos del Epoc así mismo el nuevo software leerá los mensajes publicados en ROS para obtener los datos para graficarlos y generar el "log", todo esto ya en tiempo real.

Además de ROS el nuevo software se desarrolló en Python, utilizando las librerías PyQt y matplotlib. Python es un lenguaje de programación interpretado, interactivo y orientado a objetos que provee estructuras de datos de alto nivel como listas y diccionarios, módulos, clases, excepciones, manejo automático de memoria, etc. Además es un lenguaje poderoso y multi propósito que cuenta con una sintaxis simple y elegante[14]. Para la creación de la interfaz se decidió utlizar el framework QT integrado a Python mediante PyQt. PyQt un binding[1] de QT para Python v2 y v3 que puede ser ejecutado en todas las plataformas soportadas por QT incluyendo Windows, MacOS/X y Linux. Está implementada por medio de módulos de Python contenidos en más de 620 clases. Finalmente la graficación se llevó a

[1] Adaptación de una biblioteca para ser usada en un lenguaje de programación distinto de aquél en el que ha sido escrita.

cabo con matplotlib. matplotlib es una librería de Python para graficación 2D que produce figuras de calidad en una variedad de formatos y entornos interactivos a través de distintas plataformas. matplotlib puede ser usado en scripts de Python, MATLAB, Mathematica, servidores de aplicaciones web y en herramientas de interfaz gráfica de usuario.

El criterio para seleccionar los lenguajes, frameworks y bibliotecas mencionados fue el de la filosofía del software libre, siendo las tecnologías seleccionadas libres y además multiplataforma[2]. La aplicación resultante por lo tanto es libre y multiplataforma. EL software cuenta con una interfaz sencilla; cuenta con 14 gráficas donde se mostrarán cada una de las señales recibidas, tres botones para reproducir, pausar y detener la graficación de las señales y un cuadro de texto para escribir el nombre con el que se generará el archivo de registro y un campo de texto que sirve para visualizar cada paquete de señales que se va recibiendo.

Figura 3: Interfaz

Cada gráfica muestra cada una de las señales recibidas. Las gráficas se refrescan cada 30 señales recibidas limitadas principalmente por el tamaño de la interfaz. El espacio de texto para el log por otra parte muestra una lista con las 14 señales recibidas en x tiempo, es decir, cada vez que se recibe un mensaje se mostrará una lista en el cuadro de texto y se dibujará un punto nuevo en cada una de las 14 gráficas. Por último los botones controlan el dibujado en la gráfica y en el cuadro log sin embargo el software seguirá leyendo la información de las señale se encuentre en pausa o detenida, los botones por lo tanto controlan lo que el usuario necesita ver y registrar al momento que está utilizando la interfaz.

Para entender mejor como la interfaz funciona e interactúa con el emokit a través de ROS hay que explicar los términos Topics, mensajes, publisher y subscriber. ROS trabaja como un servidor el cual se ejecuta para que se puedan establecer todas las comunicaciones entre los procesos, dentro de una interacción de procesos intervienen tres factores:

- Mensajes: Son archivos donde se definen que datos y de que tipo serán comunicados

[2]Windows no es soportado por ROS a la fecha de la publicación, ver http://wiki.ros.org/ROS/Installation

por medio de el, cada archivo es un paquete que contiene diferentes tipos y nombres de variables.

- Publishers: Son los programas o procesos que se encargarán de publicar información en los mensajes para que estos sean leídos por uno o varios subcribers.
- Subcribers: Son los programas o procesos que leen los mensajes publicados por los publishers.

Este esquema de funcionamiento es fácilmente comparable con el clásico esquema de comunicación del habla donde contamos con un emisor(publisher), un mensaje y uno o varios receptores(subcribers).

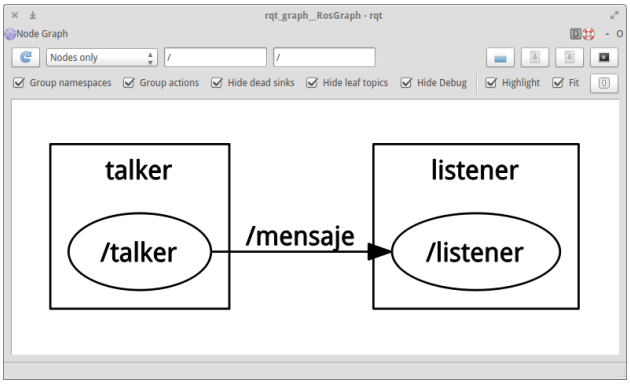

Figura 4: Comunicación ROS

Gracias a ROS el software resultante en este trabajo no se encuentra limitado a trabajar únicamente con el emokit. El software solo lee los mensajes publicados por ROS independientemente de quien los esté enviando. Podemos ver esto como una ventaja ya que permite que el software se use en conjunto con cualquier otro programa o proceso que publique el mensaje que el software lee.

Para ilustra mejor la interacción entre los componentes involucrados se presenta el diagrama de secuencias:

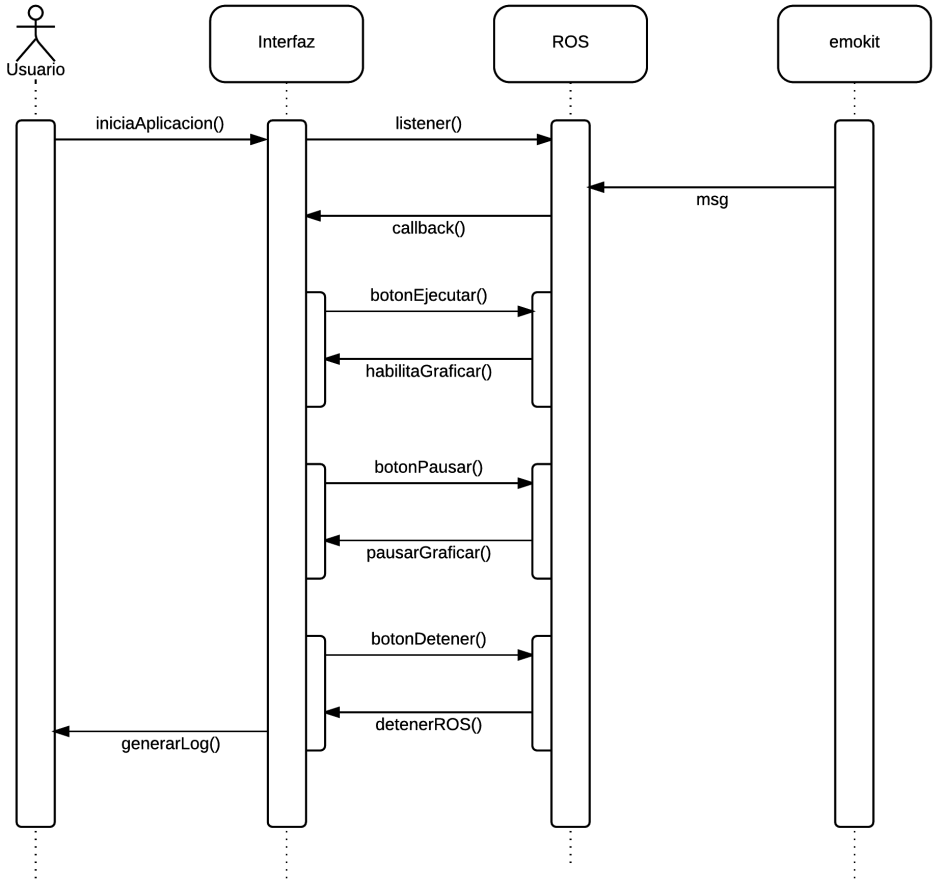

Figura 5: Diagrama de Secuencia

Implementación y Resultados

La impelmentacion es una explicacion del software funcionado e imagenes de esllo

A continuación se presentan los códigos con los que fue posible este proyecto, el proyecto completo puede ser descargado en `http://github.com/carlosbulnes/epoc/`.

Primero tenemos al código del emokit adaptado a ROS.

```
#!/usr/bin/env python

# Librerias de ROS
import rospy
import numpy as np
import roslib; roslib.load_manifest('epoc')
from epoc.msg import Frecuencias

# Librerias de Emokit
from emokit.emotiv import Emotiv
import platform
if platform.system() == "Windows":
    import socket
import gevent

def obtenerDatos(datos):
    packet = headset.dequeue()
    datos.append(packet.sensors['F3']['value'])
    datos.append(packet.sensors['FC5']['value'])
    datos.append(packet.sensors['AF3']['value'])
    datos.append(packet.sensors['F7']['value'])
    datos.append(packet.sensors['T7']['value'])
    datos.append(packet.sensors['P7']['value'])
    datos.append(packet.sensors['O1']['value'])
    datos.append(packet.sensors['O2']['value'])
    datos.append(packet.sensors['P8']['value'])
    datos.append(packet.sensors['T8']['value'])
    datos.append(packet.sensors['F8']['value'])
    datos.append(packet.sensors['AF4']['value'])
    datos.append(packet.sensors['FC6']['value'])
    datos.append(packet.sensors['F4']['value'])
    print datos
```

```
    gevent.sleep(0)

def talker():
    pub = rospy.Publisher('mensaje', Frecuencias)
    rospy.init_node('talker', anonymous=True, disable_signals=False)
    rate = rospy.Rate(20) # 10hz

    while not rospy.is_shutdown():
        datos = []
        hello_str = "%s" %rospy.get_time()
        rospy.loginfo(hello_str)

        obtenerDatos(datos)

        pub.publish(datos)
        rate.sleep()


if __name__ == '__main__':
    headset = Emotiv()
    gevent.spawn(headset.setup)
    gevent.sleep(0)
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
    except KeyboardInterrupt:
        headset.close()
    finally:
        headset.close()
```

Listing 1: emokit.py

El código del software realizado en este trabajo es el siguiente. Este está constituido por un programa principal llamado interfaz.py el cuál depende de los códigos GUI.py que es la definición de la interfaz en PyQT y matplotlibwidgetFile.py que permite la integración de matplotlib a PyQt.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import rospy
import roslib; roslib.load_manifest('epoc')
from epoc.msg import Frecuencias
from GUI import *
import sys
from time import strftime
from time import sleep

class GUIForm(QtGui.QWidget):

```

```
14      def __init__(self, parent=None):
15          QtGui.QWidget.__init__(self,parent)
16          self.ui = Ui_EpocGUI()
17          self.ui.setupUi(self)
18          self.grafica = False
19          self.frecuencias = [[],[],[],[],[],[],[],[],[],[],[],[],[],[]]
20          self.listener()
21
22          QtCore.QObject.connect(self.ui.botonEjecutar, QtCore.SIGNAL('clicked()
    '), self.habilitaGraficar)
23          QtCore.QObject.connect(self.ui.botonDetener, QtCore.SIGNAL('clicked()'
    ), self.detenerROS)
24          QtCore.QObject.connect(self.ui.botonPausar, QtCore.SIGNAL('clicked()')
    , self.pausaGraficar)
25
26      def habilitaGraficar(self):
27          self.grafica = True
28
29      def graficar(self, frecuencias):
30
31          self.ui.widget.canvas.ax.plot(frecuencias[0])
32          self.ui.widget_2.canvas.ax.plot(frecuencias[1])
33          self.ui.widget_3.canvas.ax.plot(frecuencias[2])
34          self.ui.widget_4.canvas.ax.plot(frecuencias[3])
35          self.ui.widget_5.canvas.ax.plot(frecuencias[4])
36          self.ui.widget_6.canvas.ax.plot(frecuencias[5])
37          self.ui.widget_7.canvas.ax.plot(frecuencias[6])
38          self.ui.widget_8.canvas.ax.plot(frecuencias[7])
39          self.ui.widget_9.canvas.ax.plot(frecuencias[8])
40          self.ui.widget_10.canvas.ax.plot(frecuencias[9])
41          self.ui.widget_11.canvas.ax.plot(frecuencias[10])
42          self.ui.widget_12.canvas.ax.plot(frecuencias[11])
43          self.ui.widget_13.canvas.ax.plot(frecuencias[12])
44          self.ui.widget_14.canvas.ax.plot(frecuencias[13])
45
46          self.ui.widget.canvas.draw()
47          self.ui.widget_2.canvas.draw()
48          self.ui.widget_3.canvas.draw()
49          self.ui.widget_4.canvas.draw()
50          self.ui.widget_5.canvas.draw()
51          self.ui.widget_6.canvas.draw()
52          self.ui.widget_7.canvas.draw()
53          self.ui.widget_8.canvas.draw()
54          self.ui.widget_9.canvas.draw()
55          self.ui.widget_10.canvas.draw()
56          self.ui.widget_11.canvas.draw()
57          self.ui.widget_12.canvas.draw()
58          self.ui.widget_13.canvas.draw()
59          self.ui.widget_14.canvas.draw()
60
61      def detenerROS(self):
```

```
62          self.grafica = False
63          self.generarLog()
64
65      def pausaGraficar(self):
66          self.grafica = False
67
68      def generarLog(self):
69          sleep(.1)
70
71          nombre = self.ui.textNombrePrueba.toPlainText()
72          if nombre:
73              nombre = nombre + '_' + strftime("%d-%m-%y_%H:%M") + '.xls'
74              file = open(nombre, 'w')
75              file.write('Senal 1, Senal 2, Senal 3, Senal 4, Senal 5, Senal 6,
    Senal 7,'
76                         ' Senal 8, Senal 9, Senal 10, Senal 11, Senal 12,
    Senal 13, Senal 14\n')
77              file.write(self.ui.textBrowser.toPlainText())
78              self.ui.textBrowser.setPlainText("")
79          else:
80              msgBox = QtGui.QMessageBox(QtGui.QMessageBox.Warning,
81                  "QMessageBox.warning()", "Espeficifica un nombre y presiona
    Detener nuevamente",
82                  QtGui.QMessageBox.NoButton, self)
83              msgBox.exec_()
84
85      def callback(self, data):
86
87          if self.grafica:
88              data = list(data.datos)
89
90              self.frecuencias[0].append(data[0])
91              self.frecuencias[1].append(data[1])
92              self.frecuencias[2].append(data[2])
93              self.frecuencias[3].append(data[3])
94              self.frecuencias[4].append(data[4])
95              self.frecuencias[5].append(data[5])
96              self.frecuencias[6].append(data[6])
97              self.frecuencias[7].append(data[7])
98              self.frecuencias[8].append(data[8])
99              self.frecuencias[9].append(data[9])
100             self.frecuencias[10].append(data[10])
101             self.frecuencias[11].append(data[11])
102             self.frecuencias[12].append(data[12])
103             self.frecuencias[13].append(data[13])
104
105             log = str(str(data[0]) + ', ' + str(data[1]) + ', ' + str(data[2])
    +
106                       ', ' + str(data[3]) + ', ' + str(data[4]) + ', ' + str(
    data[5]) +
107                       ', ' + str(data[6]) + ', ' + str(data[7]) + ', ' + str(
```

```
    data[8]) +
108                         ', ' + str(data[9]) + ', ' + str(data[10]) + ', ' + str(
    data[11]) +
109                         ', ' + str(data[12]) + ', ' + str(data[13]))
110
111             self.ui.textBrowser.appendPlainText(log)
112             self.ui.textBrowser.appendPlainText("")
113
114             if len(self.frecuencias[0]) == 30:
115                 self.frecuencias = [[],[],[],[],[],[],[],[],[],[],[],[],[]]
116                 self.ui.widget.canvas.ax.clear()
117                 self.ui.widget_2.canvas.ax.clear()
118                 self.ui.widget_3.canvas.ax.clear()
119                 self.ui.widget_4.canvas.ax.clear()
120                 self.ui.widget_5.canvas.ax.clear()
121                 self.ui.widget_6.canvas.ax.clear()
122                 self.ui.widget_7.canvas.ax.clear()
123                 self.ui.widget_8.canvas.ax.clear()
124                 self.ui.widget_9.canvas.ax.clear()
125                 self.ui.widget_10.canvas.ax.clear()
126                 self.ui.widget_11.canvas.ax.clear()
127                 self.ui.widget_12.canvas.ax.clear()
128                 self.ui.widget_13.canvas.ax.clear()
129                 self.ui.widget_14.canvas.ax.clear()
130
131             self.graficar(self.frecuencias)
132
133     def listener(self):
134          rospy.init_node('listener', anonymous=True, disable_signals=False)
135         rospy.Subscriber("mensaje", Frecuencias, self.callback)
136
137
138 if __name__ == '__main__':
139
140     app = QtGui.QApplication(sys.argv)
141     myapp = GUIForm()
142     myapp.show()
143     sys.exit(app.exec_())
144     #rospy.signal_shutdown("Se presiono el boton detener")
145     #listener()
```

Listing 2: interfaz.py

```
1 # -*- coding: utf-8 -*-
2
3 from PyQt4 import QtCore, QtGui
4 from matplotlibwidgetFile import matplotlibWidget
5
6 try:
7     _fromUtf8 = QtCore.QString.fromUtf8
8 except AttributeError:
9     _fromUtf8 = lambda s: s
```

```
10
11 class Ui_EpocGUI(object):
12     def setupUi(self, EpocGUI):
13         EpocGUI.setObjectName(_fromUtf8("EpocGUI"))
14         EpocGUI.resize(1101, 897)
15
16         self.centralwidget = QtGui.QWidget(EpocGUI)
17         self.centralwidget.setObjectName(_fromUtf8("centralwidget"))
18         self.widget = matplotlibWidget(self.centralwidget)
19         self.widget.setGeometry(QtCore.QRect(0, 10, 260, 180))
20         self.widget.setObjectName(_fromUtf8("widget"))
21         self.widget_2 = matplotlibWidget(self.centralwidget)
22         self.widget_2.setGeometry(QtCore.QRect(250, 10, 260, 180))
23         self.widget_2.setObjectName(_fromUtf8("widget_2"))
24         self.widget_3 = matplotlibWidget(self.centralwidget)
25         self.widget_3.setGeometry(QtCore.QRect(500, 10, 260, 180))
26         self.widget_3.setObjectName(_fromUtf8("widget_3"))
27         self.widget_4 = matplotlibWidget(self.centralwidget)
28         self.widget_4.setGeometry(QtCore.QRect(0, 180, 260, 180))
29         self.widget_4.setObjectName(_fromUtf8("widget_4"))
30         self.widget_5 = matplotlibWidget(self.centralwidget)
31         self.widget_5.setGeometry(QtCore.QRect(250, 180, 260, 180))
32         self.widget_5.setObjectName(_fromUtf8("widget_5"))
33         self.widget_6 = matplotlibWidget(self.centralwidget)
34         self.widget_6.setGeometry(QtCore.QRect(500, 180, 260, 180))
35         self.widget_6.setObjectName(_fromUtf8("widget_6"))
36         self.widget_7 = matplotlibWidget(self.centralwidget)
37         self.widget_7.setGeometry(QtCore.QRect(0, 350, 260, 180))
38         self.widget_7.setObjectName(_fromUtf8("widget_7"))
39         self.widget_8 = matplotlibWidget(self.centralwidget)
40         self.widget_8.setGeometry(QtCore.QRect(250, 350, 260, 180))
41         self.widget_8.setObjectName(_fromUtf8("widget_8"))
42         self.widget_9 = matplotlibWidget(self.centralwidget)
43         self.widget_9.setGeometry(QtCore.QRect(500, 350, 260, 180))
44         self.widget_9.setObjectName(_fromUtf8("widget_9"))
45         self.widget_10 = matplotlibWidget(self.centralwidget)
46         self.widget_10.setGeometry(QtCore.QRect(0, 520, 260, 180))
47         self.widget_10.setObjectName(_fromUtf8("widget_10"))
48         self.widget_11 = matplotlibWidget(self.centralwidget)
49         self.widget_11.setGeometry(QtCore.QRect(250, 520, 260, 180))
50         self.widget_11.setObjectName(_fromUtf8("widget_11"))
51         self.widget_12 = matplotlibWidget(self.centralwidget)
52         self.widget_12.setGeometry(QtCore.QRect(500, 520, 260, 180))
53         self.widget_12.setObjectName(_fromUtf8("widget_12"))
54         self.widget_13 = matplotlibWidget(self.centralwidget)
55         self.widget_13.setGeometry(QtCore.QRect(0, 690, 260, 180))
56         self.widget_13.setObjectName(_fromUtf8("widget_13"))
57         self.widget_14 = matplotlibWidget(self.centralwidget)
58         self.widget_14.setGeometry(QtCore.QRect(250, 690, 260, 180))
59         self.widget_14.setObjectName(_fromUtf8("widget_14"))
60
```

```
61          self.botonPausar = QtGui.QPushButton(self.centralwidget)
62          self.botonPausar.setGeometry(QtCore.QRect(780, 180, 85, 27))
63          self.botonPausar.setText(_fromUtf8(""))
64          icon = QtGui.QIcon()
65          icon.addPixmap(QtGui.QPixmap(_fromUtf8("img/pause.png")), QtGui.QIcon.
    Normal, QtGui.QIcon.Off)
66          self.botonPausar.setIcon(icon)
67          self.botonPausar.setObjectName(_fromUtf8("botonPausar"))
68
69          self.botonDetener = QtGui.QPushButton(self.centralwidget)
70          self.botonDetener.setGeometry(QtCore.QRect(980, 180, 85, 27))
71          self.botonDetener.setText(_fromUtf8(""))
72          icon1 = QtGui.QIcon()
73          icon1.addPixmap(QtGui.QPixmap(_fromUtf8("img/stop.png")), QtGui.QIcon.
    Normal, QtGui.QIcon.Off)
74          self.botonDetener.setIcon(icon1)
75          self.botonDetener.setObjectName(_fromUtf8("botonDetener"))
76
77          self.botonEjecutar = QtGui.QPushButton(self.centralwidget)
78          self.botonEjecutar.setGeometry(QtCore.QRect(880, 180, 85, 27))
79          self.botonEjecutar.setText(_fromUtf8(""))
80          icon2 = QtGui.QIcon()
81          icon2.addPixmap(QtGui.QPixmap(_fromUtf8("img/play.png")), QtGui.QIcon.
    Normal, QtGui.QIcon.Off)
82          self.botonEjecutar.setIcon(icon2)
83          self.botonEjecutar.setObjectName(_fromUtf8("botonEjecutar"))
84          self.textBrowser = QtGui.QPlainTextEdit(self.centralwidget)
85
86          self.textBrowser.setGeometry(QtCore.QRect(760, 290, 331, 500))
87          self.textBrowser.setObjectName(_fromUtf8("textBrowser"))
88          self.textBrowser.setReadOnly(True)
89
90          self.textNombrePrueba = QtGui.QTextEdit(self.centralwidget)
91
92          self.textNombrePrueba.setGeometry(QtCore.QRect(780, 120, 281, 21))
93          self.textNombrePrueba.setObjectName(_fromUtf8("textEdit_2"))
94          self.label = QtGui.QLabel(self.centralwidget)
95
96          self.label.setGeometry(QtCore.QRect(780, 100, 141, 16))
97          self.label.setObjectName(_fromUtf8("label"))
98
99          self.menubar = QtGui.QMenuBar(EpocGUI)
100         self.menubar.setGeometry(QtCore.QRect(0, 0, 1101, 23))
101         self.menubar.setObjectName(_fromUtf8("menubar"))
102
103         self.retranslateUi(EpocGUI)
104         QtCore.QMetaObject.connectSlotsByName(EpocGUI)
105
106     def retranslateUi(self, EpocGUI):
107         EpocGUI.setWindowTitle(QtGui.QApplication.translate("EpocGUI", "Epoc",
     None, QtGui.QApplication.UnicodeUTF8))
```

```
108         self.label.setText(QtGui.QApplication.translate("EpocGUI", "Nombre del
        experimento", None, QtGui.QApplication.UnicodeUTF8))
```

Listing 3: GUI.py

```
1  from PyQt4 import QtGui
2  import matplotlib
3  from matplotlib.backends.backend_qt4agg import FigureCanvasQTAgg as
       FigureCanvas
4
5  from matplotlib.figure import Figure
6
7  class MplCanvas(FigureCanvas):
8
9      def __init__(self):
10         self.fig = Figure()
11         self.ax = self.fig.add_subplot(111)
12
13         FigureCanvas.__init__(self, self.fig)
14         FigureCanvas.setSizePolicy(self, QtGui.QSizePolicy.Expanding,QtGui.
       QSizePolicy.Expanding)
15         FigureCanvas.updateGeometry(self)
16
17
18 class matplotlibWidget(QtGui.QWidget):
19
20     def __init__(self, parent = None):
21         QtGui.QWidget.__init__(self, parent)
22         self.canvas = MplCanvas()
23         self.vbl = QtGui.QVBoxLayout()
24         self.vbl.addWidget(self.canvas)
25         self.setLayout(self.vbl)
```

Listing 4: matplotlibwidgetFile.py

Conclusión y trabajo futuro

Se logró crear un software el cual es capaz de leer mensaje publicados en ROS para generar las graficas y el registro. La principal ventaja es que al ser en ROS su funcionamiento en base a mensajes esto permite que sea usado en cualquier escenario que publique el mismo tipo de mensaje que el software lee y no se encuentra limitado a funcionar unicamente con el emokit o incluso para el Emotiv Epoc.

Trabajo Futuro

Al ser libre, es software en este poryecto puede ser mejorado de muchas formas y para distintos propósitos.

Índice de figuras

Bibliografía

[1] C. Brocious and K. Machulis, "Emokit." http://www.openyou.org, May 2015.

[2] K. Roebuck, *Brain-Computer Interface: High-impact Emerging Technology - What You Need to Know: Definitions, Adoptions, Impact, Benefits, Maturity, Vendors*. Emereo Publishing, 2012.

[3] V. Autores, "neurogadget.com." http://neurogadget.com/2011/04/08/implanted-bci-detects-when-patients-think-of-specific-sounds/1780, May 2015.

[4] V. Autores, "emotiv.com." https://emotiv.com/, May 2015.

[5] V. Autores, "Emotiv systems." http://en.wikipedia.org/wiki/Emotiv_Systems, May 2015.

[6] K. Stytsenko, E. Jablonskis, and C. Prahm, "Evaluation of consumer eeg device emotiv epoc," in *MEi: CogSci Conference 2011, Ljubljana*, 2011.

[7] V. Autores, "G-tec biosignal amplifier g.bsamp." http://www.gtec.at/Products/Hardware-and-Accessories/g.BSamp-Specs-Features, May 2015.

[8] J. R. de la O Chávez, "Interfaz cerebro – computadora para el control de un cursor basado en ondas cerebrales," pp. 16–19.

[9] I. Senior Design, "Epoc-alypse mind controlled car,"

[10] A. Raza, "Ssvep based eeg interface for google street view navigation," 2012.

[11] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, 2009.

[12] M. Kranz, L. Roalter, and F. Michahelles, "Things that twitter: social networks and the internet of things," in *What can the Internet of Things do for the Citizen (CIoT)*

Workshop at The Eighth International Conference on Pervasive Computing (Pervasive 2010), pp. 1–10, 2010.

[13] V. Autores, "Ros.org." `http://wiki.ros.org/es`, May 2015.

[14] M. F. Sanner *et al.*, "Python: a programming language for software integration and development," *J Mol Graph Model*, vol. 17, no. 1, pp. 57–61, 1999.