

**PRÁCTICA 4: EV\_3\_1\_PSOC 5LP ENVIO DE MENSAJES**

Alvarado Conteras Cesar Omar

8° A Ing. Mecatrónica

Dinámica y control de robots

Carlos Moran Garabito

**INTRODUCCIÓN:**

La práctica se desarrolla en recibir mensajes en la terminal de ROS con la tarjeta Psoc l5

**MARCO TEÓRICO:**

Sistema Operativo Robótico (en inglés Robot Operating System, ROS) es un [framework](https://es.wikipedia.org/wiki/Framework) para el desarrollo de software para [robots](https://es.wikipedia.org/wiki/Robot) que provee la funcionalidad de un [sistema operativo](https://es.wikipedia.org/wiki/Sistema_operativo) en un [clúster](https://es.wikipedia.org/wiki/Cluster_(inform%C3%A1tica)) heterogéneo. ROS se desarrolló originalmente en 2007 bajo el nombre de *switchyard* por el [Laboratorio de Inteligencia Artificial de Stanford](https://es.wikipedia.org/w/index.php?title=Laboratorio_de_Inteligencia_Artificial_de_Stanford&action=edit&redlink=1) para dar soporte al proyecto del Robot con [Inteligencia Artificial](https://es.wikipedia.org/wiki/Inteligencia_Artificial) de Stanford ([STAIR](http://stair.stanford.edu/)[2](https://es.wikipedia.org/wiki/Sistema_Operativo_Rob%C3%B3tico#cite_note-2)​). Desde 2008, el desarrollo continua primordialmente en [Willow Garage](https://es.wikipedia.org/w/index.php?title=Willow_Garage&action=edit&redlink=1), un instituto de investigación robótico con más de veinte instituciones colaborando en un modelo de desarrollo federado.

ROS provee los servicios estándar de un sistema operativo tales como abstracción del hardware, control de dispositivos de bajo nivel, implementación de funcionalidad de uso común, paso de mensajes entre procesos y mantenimiento de paquetes. Está basado en una arquitectura de [grafos](https://es.wikipedia.org/wiki/Teor%C3%ADa_de_grafos) donde el procesamiento toma lugar en los nodos que pueden recibir, mandar y multiplexar mensajes de sensores, control, estados, planificaciones y actuadores, entre otros. La librería está orientada para un [sistema UNIX](https://es.wikipedia.org/wiki/Unix-like) ([Ubuntu](https://es.wikipedia.org/wiki/Ubuntu) (Linux)) aunque también se está adaptando a otros sistemas operativos como [Fedora](https://es.wikipedia.org/wiki/Fedora_(distribuci%C3%B3n_Linux)), [Mac OS X](https://es.wikipedia.org/wiki/Mac_OS_X), Arch, [Gentoo](https://es.wikipedia.org/wiki/Gentoo), [OpenSUSE](https://es.wikipedia.org/wiki/OpenSUSE), Slackware, [Debian](https://es.wikipedia.org/wiki/Debian) o [Microsoft Windows](https://es.wikipedia.org/wiki/Microsoft_Windows), considerados como 'experimentales'.

ROS tiene dos partes básicas: la parte del sistema operativo, *ros*, como se ha descrito anteriormente y *ros-pkg*, una suite de paquetes aportados por la contribución de usuarios (organizados en conjuntos llamados *pilas* o en inglés *stacks*) que implementan las funcionalidades tales como localización y mapeo simultáneo, planificación, percepción, simulación, etc.

**DESARROLLO:**

En Linux, descargamos minicom donde conectaremos los puertos Rx y Tx de nuestra Psoc con lo cual podremos enviar mensajes.

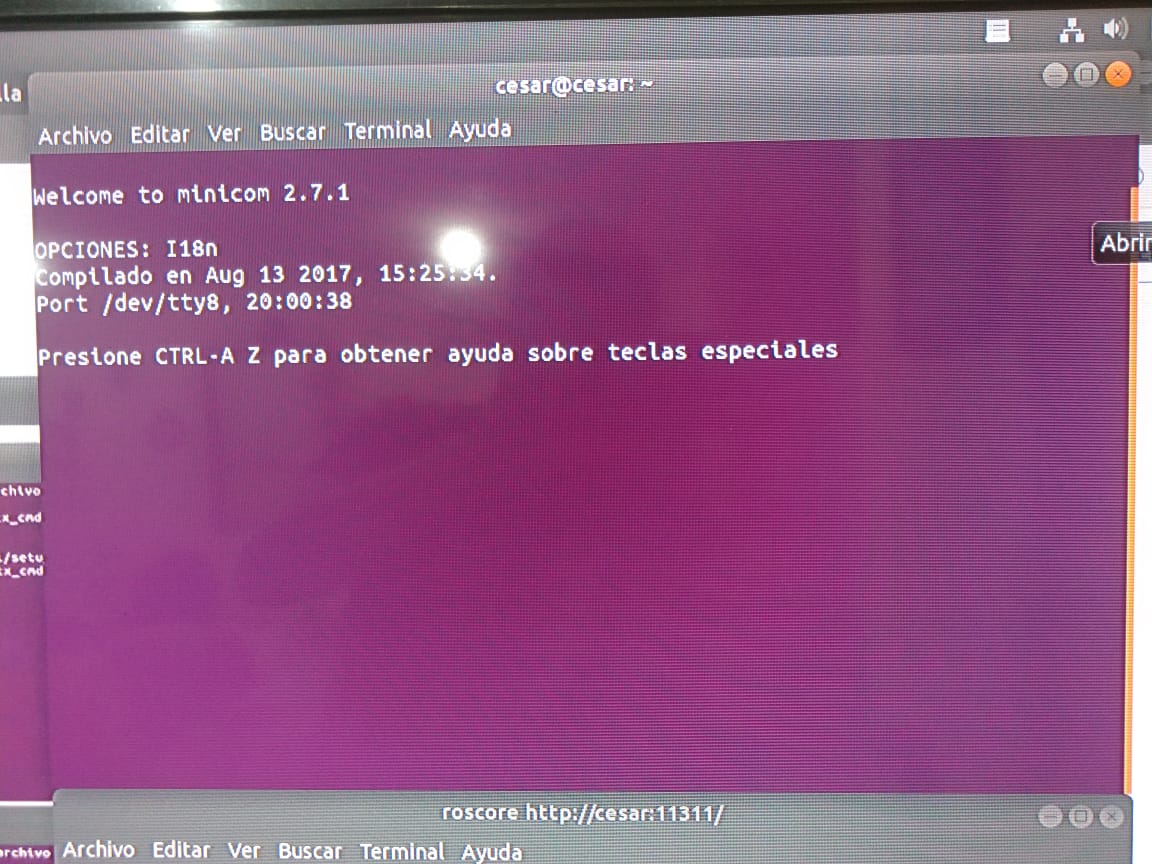
Abrimos desde la terminal de Linux con el comando:

* *minicom*

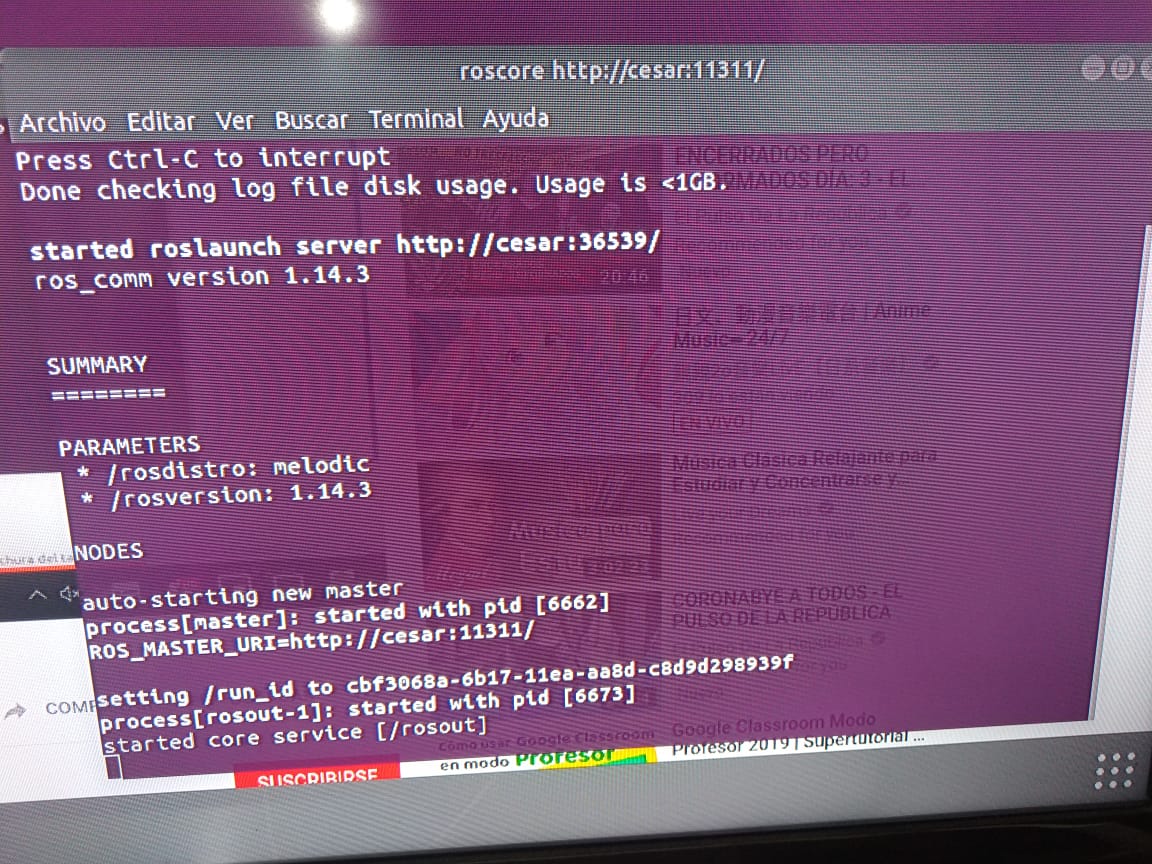
si no abre el programa y pide permisos ingresamos el siguiente comando.

* *chmod 777 /dev/tty8*

volvemos a escribir el primer comando el cual nos abrirá una venta como la siguiente.



Iniciamos el entorno de ROS con el siguiente comando en otra nueva terminal.

* roscore

si todo va bien se vera otra ventana como la siguiente corriendo ROS.

una vez realizado esto procedemos hacer un entorno de desarrollo de ROS, como se muestra en el archivo Ros-con-faldita.doc el cual nos explicara como desarrollar un entorno de ROS para poder operar nuestro programa.

Una vez realizado el entorno de ROS procedemos a copiar la liga de github siguiente: <https://github.com/BerkeleyExpertSystemTechnologiesLab/2d-spine-control-hardware>

Con lo cual la única carpeta importante de la Liga será la que se llama “ros-spine-control” la cual contendrá un código para comunicar nuestra psoc con ROS la cual debe pasar a nuestro entorno de ROS creado en la cual debe ser ubicada en la carpeta SCR para su funcionamiento el cual tiene un código.

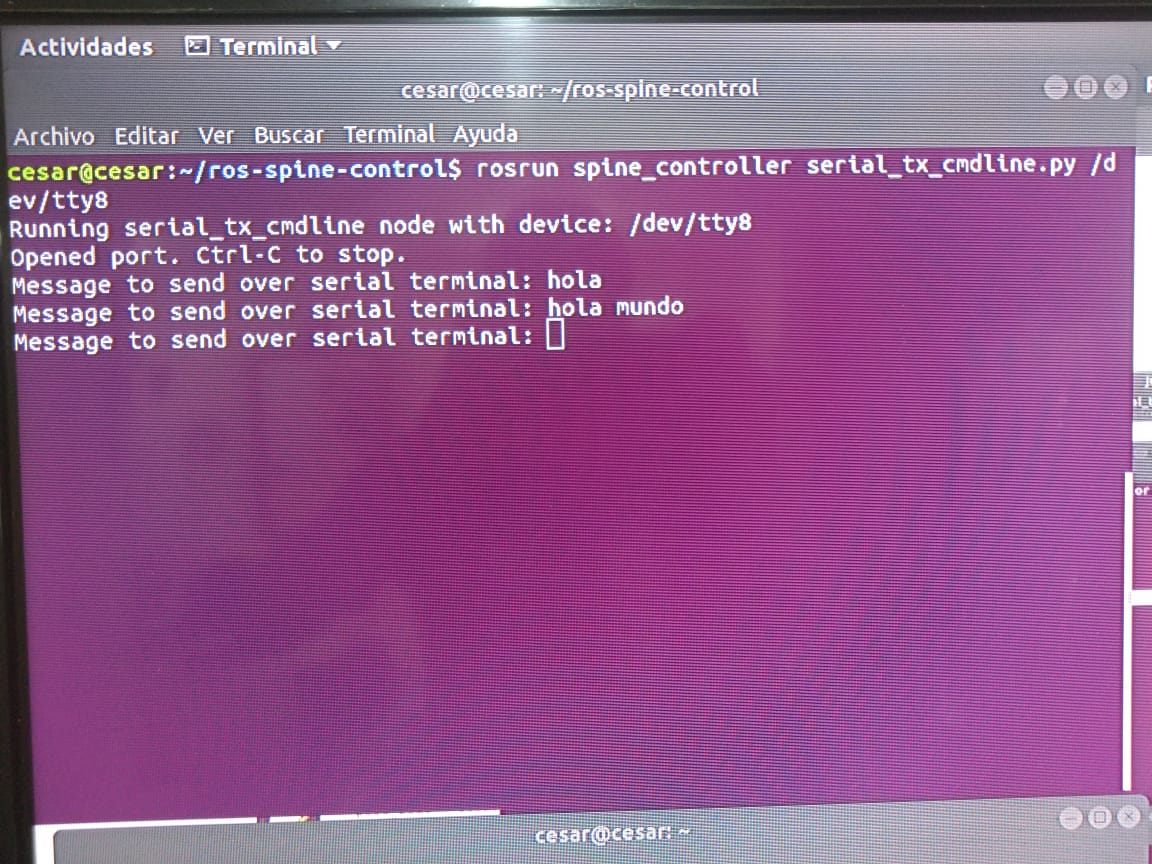
A continuación, se verá un fragmento del código:

|  |
| --- |
| #!/usr/bin/python |
|  |  |
|  | # NOTE: on Ubuntu 18.04, specifying /usr/bin/env python uses python 3, |
|  | # while specifying /usr/bin/python uses python 2.7. We need 2.7. |
|  |  |
|  | # serial\_tx\_cmdline writes lines from the command line to the serial port. |
|  | # Usage is: |
|  | # rosrun spine\_controller serial\_tx\_cmdline path-to-serial-device |
|  | # and then lines are put to the serial device upon carraige return/newline or whatever. |
|  |  |
|  | # Imports: |
|  | import rospy |
|  | # because we need the command-line arguments |
|  | import sys |
|  | # and for the serial/usb/uart via pyserial: |
|  | import serial |
|  | # We'll also be echoing messages to a ros topic. |
|  | from std\_msgs.msg import String |
|  |  |
|  | # The primary helper function here opens the serial device, |
|  | # and writes to it from raw\_input. |
|  |  |
|  |  |
|  | def tx\_to\_serial(device\_name): |
|  | # A welcome message |
|  | print("Running serial\_tx\_cmdline node with device: " + device\_name) |
|  | #print(" and python version:") |
|  | # print(sys.version) |
|  | # Hard-code a timeout for pyserial. Seems recommended, even for tx? |
|  | serial\_timeout = 1 |
|  | # First, start up the ros node. |
|  | rospy.init\_node('serial\_tx\_cmdline', anonymous=False) |
|  | # We'll publish commands to a topic just in case someone else wants to use them |
|  | pub = rospy.Publisher('serial\_tx\_cmdline', String, queue\_size=10) |
|  | # Next, do the serial setup: |
|  | # Hard-coded: our PSoC uses the following baud rate: |
|  | psoc\_baud = 115200 |
|  | # create the serial port object, non-exclusive (so others can use it too) |
|  | serial\_port = serial.Serial(device\_name, psoc\_baud, timeout=serial\_timeout, exclusive=False) |
|  | # flush out any old data |
|  | serial\_port.reset\_input\_buffer() |
|  | serial\_port.reset\_output\_buffer() |
|  | # finishing setup. |
|  | print("Opened port. Ctrl-C to stop.") |

Tras terminar de insertar la carpeta con el código procedemos a ejecutar nuestro archivo con la Psoc conectada cabe recalcar usando el siguiente comando.

* Rosrun spine\_controller serial\_tx\_cmdline.py /dev/tty8

El cual nos mostrara una ventana como la siguiente si todo va bien.



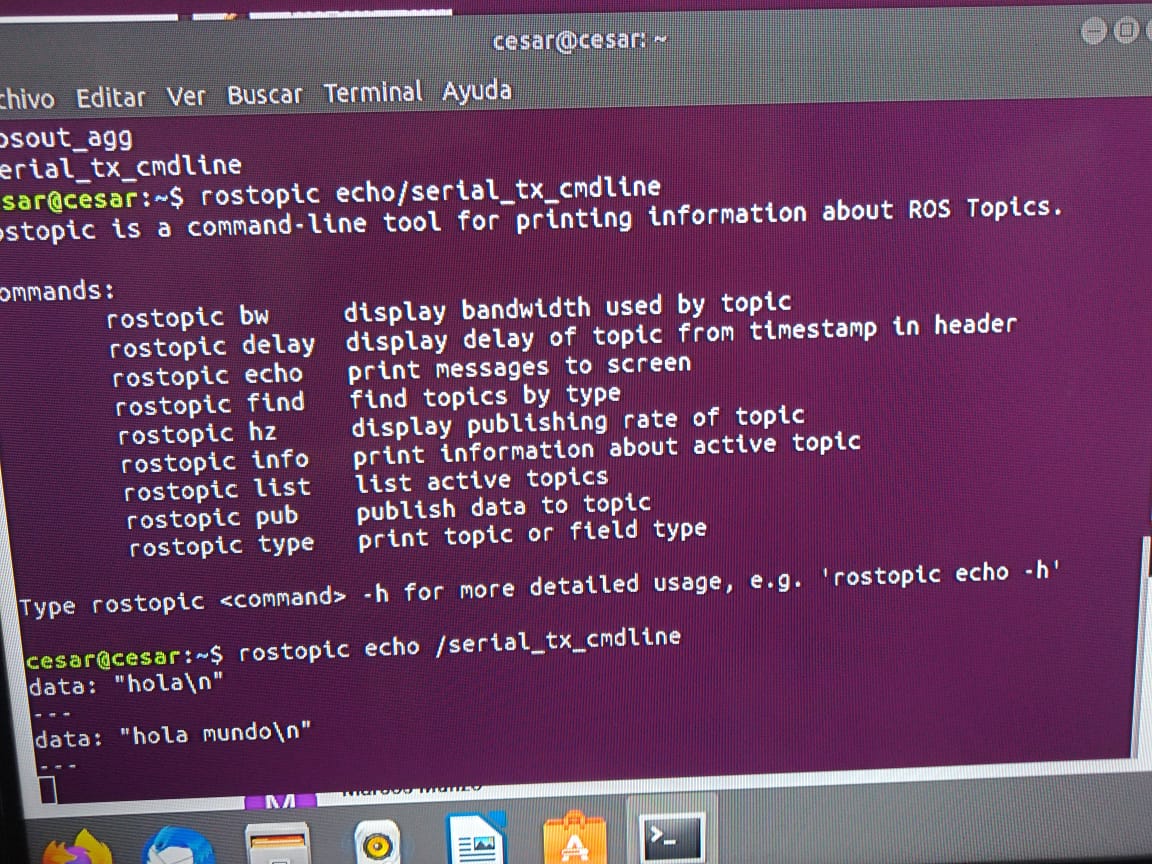
Ya solo falta observar los topic del ROS para ver si este activo con el siguiente comando:

* Rostopic list

Con lo cual veremos el nombre de nuestro archivo .py si va todo bien con lo cual usaremos el comando siguiente para ver los mensajes de nuestra Psoc por UART

* Rostopic echo /serial\_tx\_cmdline

Con lo cual mostrara los mensajes que enviamos a la Psoc y los regresa a ROS como se ve en la siguiente imagen.



Con esto podemos enviar mensajes a la Psoc y ver que se regresan a ROS por UART.

Este método fue desarrollado por el usuario “apsabelhaus” con el cual explica mas o menos el desarrollo de dicho proyecto para la psoc.

**CONCLUSIÓN:**

La práctica en cuestión es complicada por la baja información que se tiene de la tarjeta psoc y muy poco desarrollo, con lo cual toca buscar en el lenguaje Ingles ya que se tiene mas proyecto desarrollados en ese idioma con lo cual se prefiere buscar por ese método para poder dar con proyectos como de este tipo o similar a lo que se busca.