

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

Manzo Torres Marcos

8° A Ing. Mecatrónica

Dinámica y control de robots

Profesor: Carlos Moran Garabito

**INTRODUCCIÓN:**

En esta práctica llevaremos a cabo la conexión de la tarjeta PSoC con ROS, con lo cual el resultado será el envió de mensajes uniendo estas dos herramientas.

**MARCO TEÓRICO:**

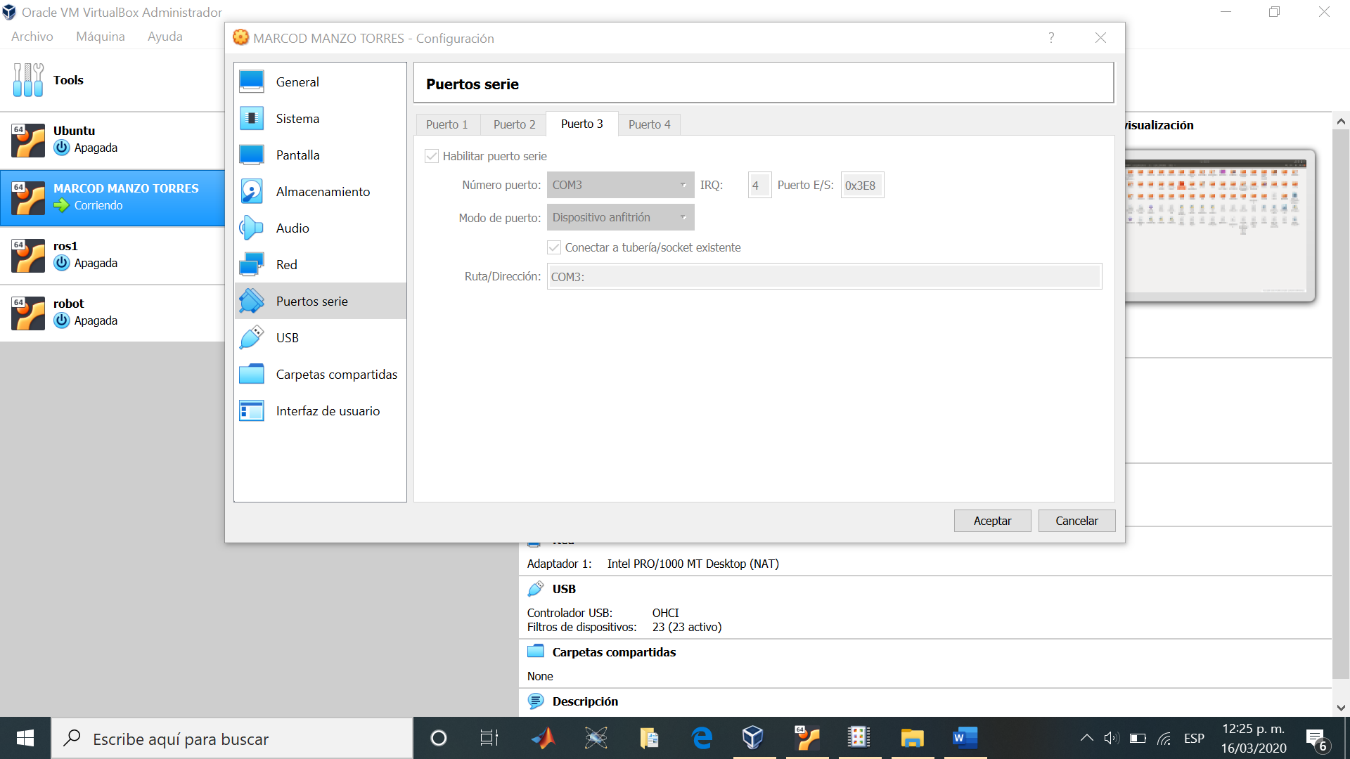
Sistema Operativo Robótico (en inglés Robot Operating System, ROS) es un [framework](https://es.wikipedia.org/wiki/Framework) para el desarrollo de software para [robots](https://es.wikipedia.org/wiki/Robot) que provee la funcionalidad de un [sistema operativo](https://es.wikipedia.org/wiki/Sistema_operativo) 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.

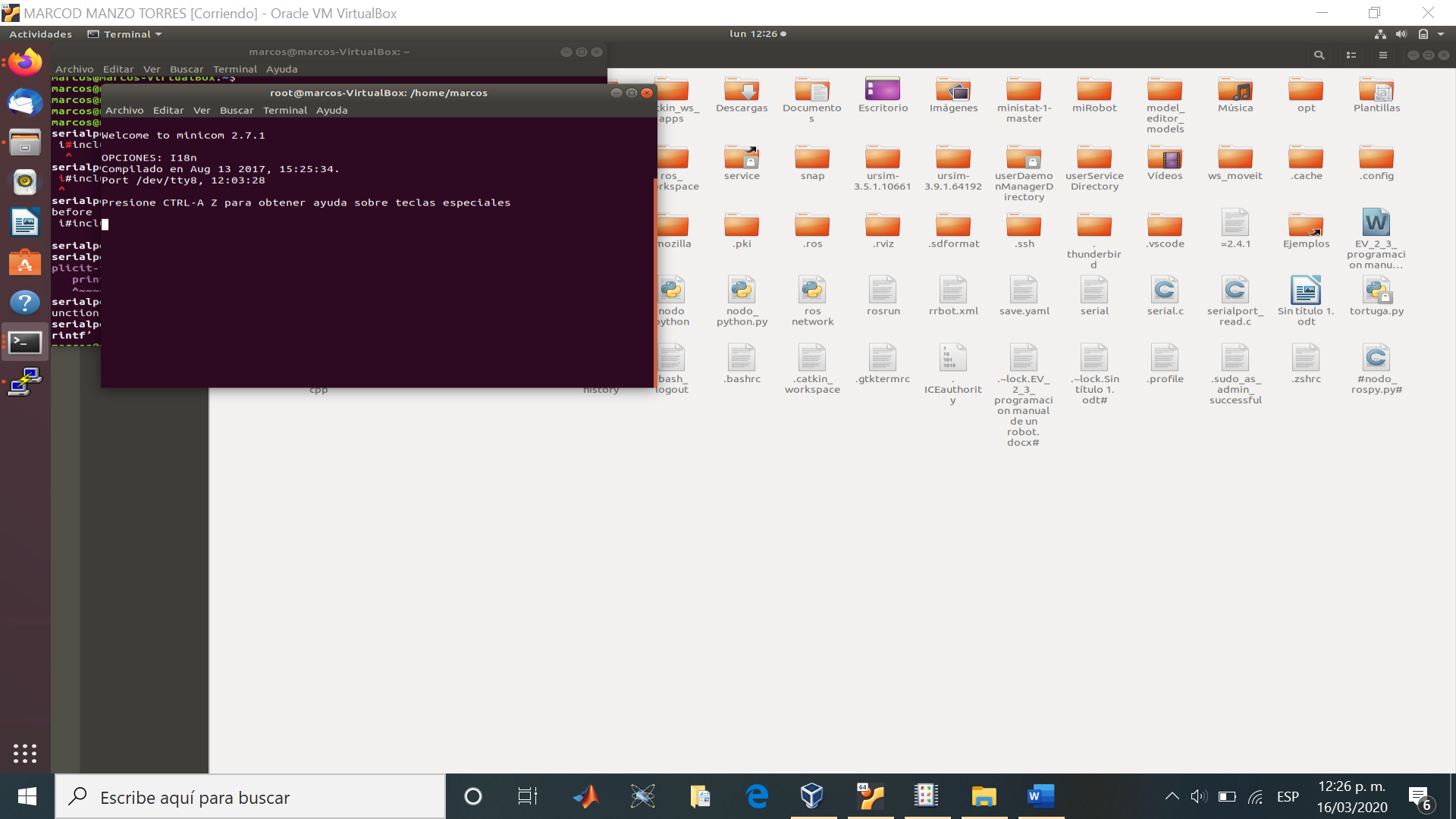
**DESARROLLO:**

Primeramente, habilitamos el puerto COM donde haremos la comunicación entre la PSoC y el sistema Linux, es muy importante observar en el ordenador el número de puerto, para así tener la certeza de una correcta conexión.

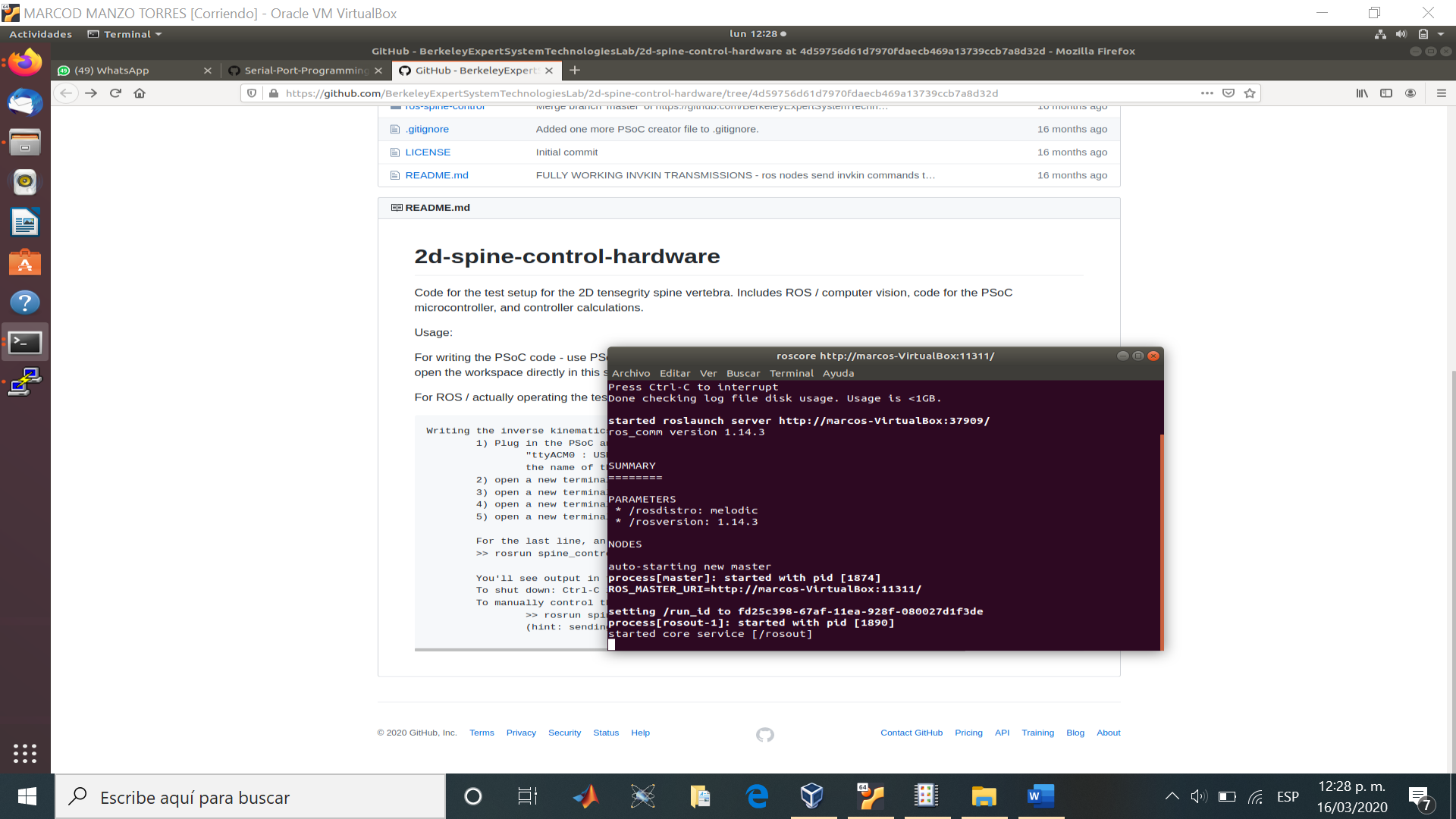


Una vez en el sistema Linux, descargamos minicom donde observaremos los detalles del puerto. Entramos en modo super usuario (sudo su) y damos permisos al puerto, en este caso con el nombre del puerto que nos aparece en minicom.

Después de hacer esto, volvemos a abrir minicom, y nos debería aparecer una ventana similar a la siguiente, donde se observa la hora a la que el puerto fue conectado y el estado actual.



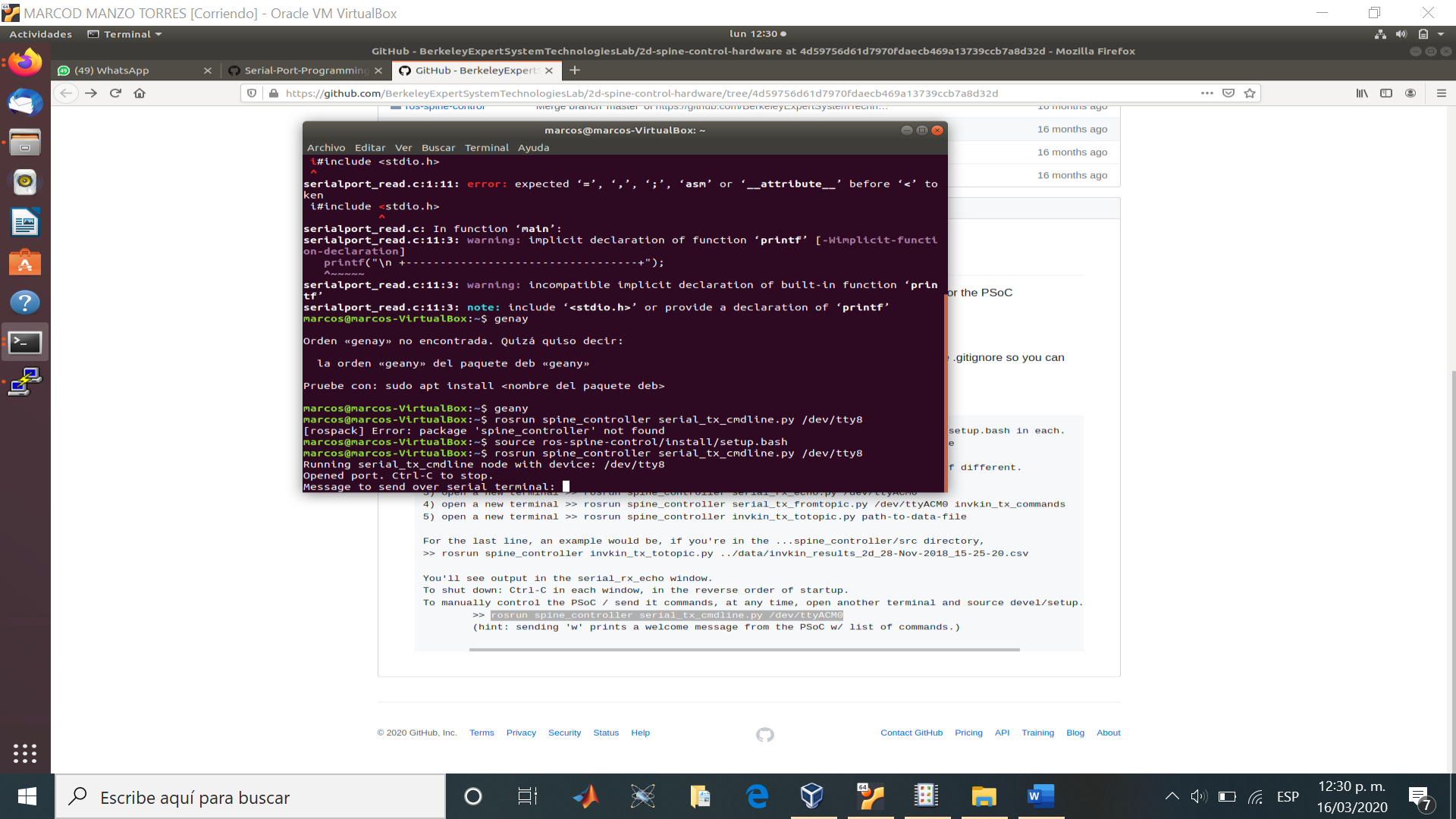
Para iniciar cualquier operación con ROS, necesitamos comenzar el entorno, esto lo hacemos con:



El código empleado fue el siguiente, básicamente hacemos la conexión del puerto serie, y si todo está en orden, lo habilitamos para enviar y recibir datos, como se ha observado por medio de rX y tX.

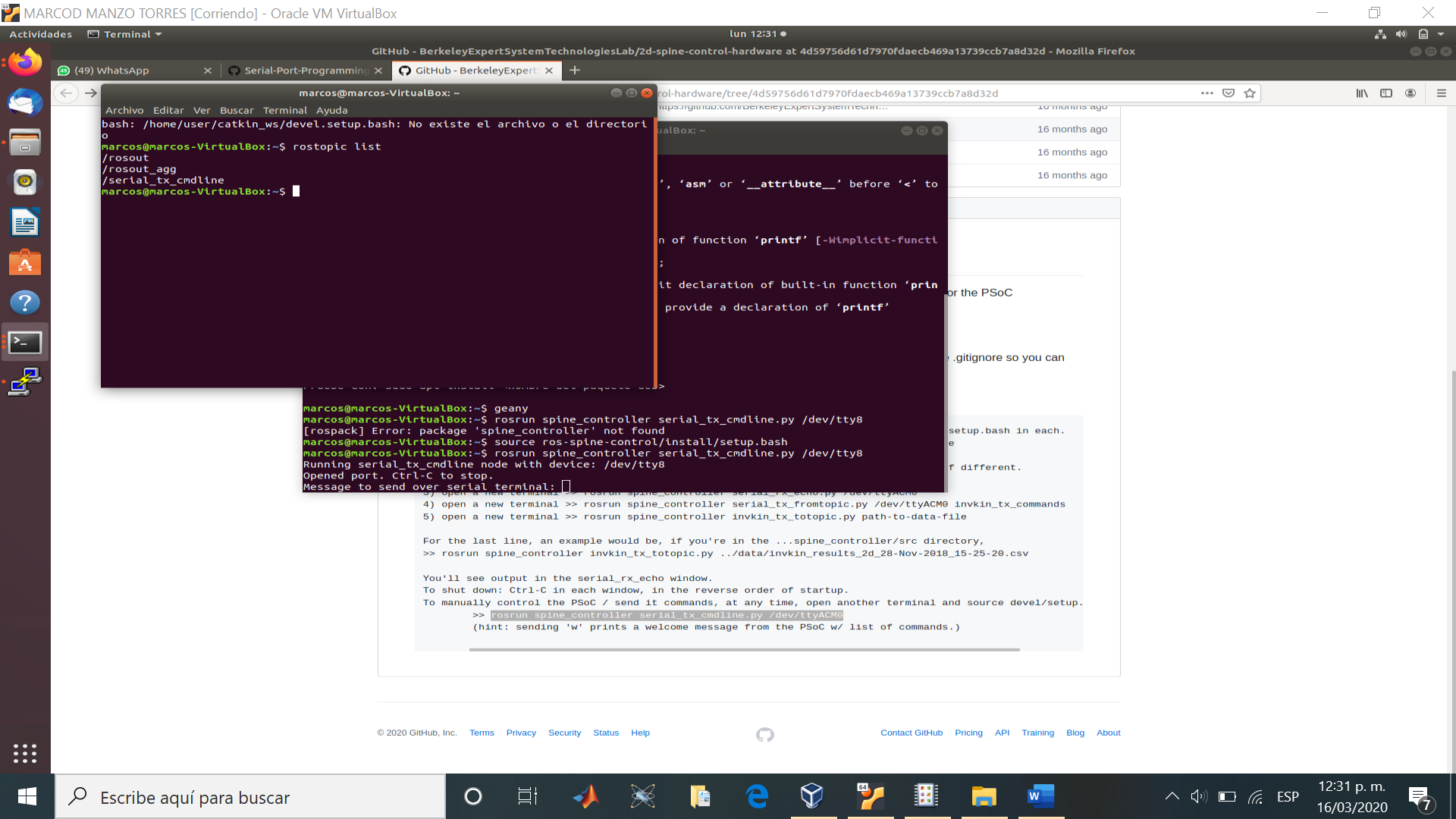
|  |
| --- |
| import rospy |
|  | # because we need the command-line arguments |
|  | import sys |
|  | # and for the serial/usb/uart via pyserial: |
|  | import serial |
|  | # We'll also be echoing messages to a ros topic. |
|  | from std\_msgs.msg import String |
|  |  |
|  | # The primary helper 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.") |
|  |  |
|  | # Instead of an infinite loop, use ROS's shutdown procedure. |
|  | while not rospy.is\_shutdown(): |
|  | # request something to send |
|  | try: |
|  | to\_psoc = raw\_input("Message to send over serial terminal: ") |
|  | # Concatenate a newline so that the psoc calls its command parser |
|  | to\_psoc += '\n' |
|  | serial\_port.write(to\_psoc) |
|  | # and publish to the topic, too. |
|  | pub.publish(to\_psoc) |
|  | except KeyboardInterrupt: |
|  | # Check: if we get the shutdown signal... |
|  | # TO-DO: this doesn't seem to be working. Luckily enough, the PSoC |
|  | # doesn't do Ctrl-C as a command, so it's ugly but safe to just not |
|  | # deal with keyboard interrupts and let rospy handle it. |
|  | rospy.signal\_shutdown("Shutting down, Ctrl-C received.") |
|  |  |
|  |  |
|  |  |
|  | # the main function: just call the helper, while parsing the serial port path. |
|  | if \_\_name\_\_ == '\_\_main\_\_': |
|  | try: |
|  | # the 0-th arg is the name of the file itself, so we want the 1st. |
|  | tx\_to\_serial(sys.argv[1]) |
|  | except rospy. ROSInterruptException: |
|  | pass |

Tras terminar el código, ahora procedemos a habilitar el .bash de nuestro archivo, para ello ejecutamos:

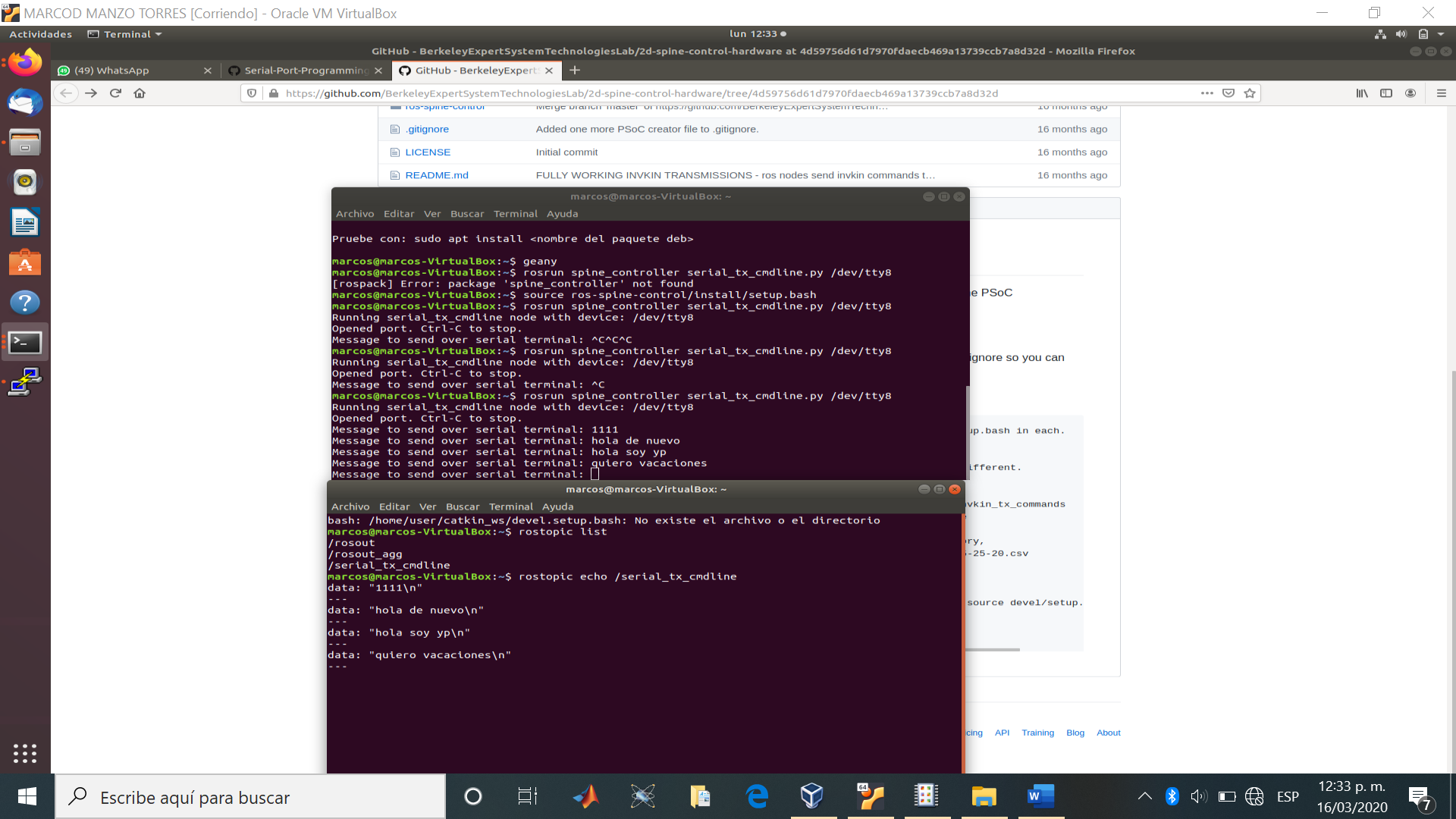


Posteriormente, abrimos nuestro archivo y colocamos el nombre del puerto a donde está conectada la PSoC.

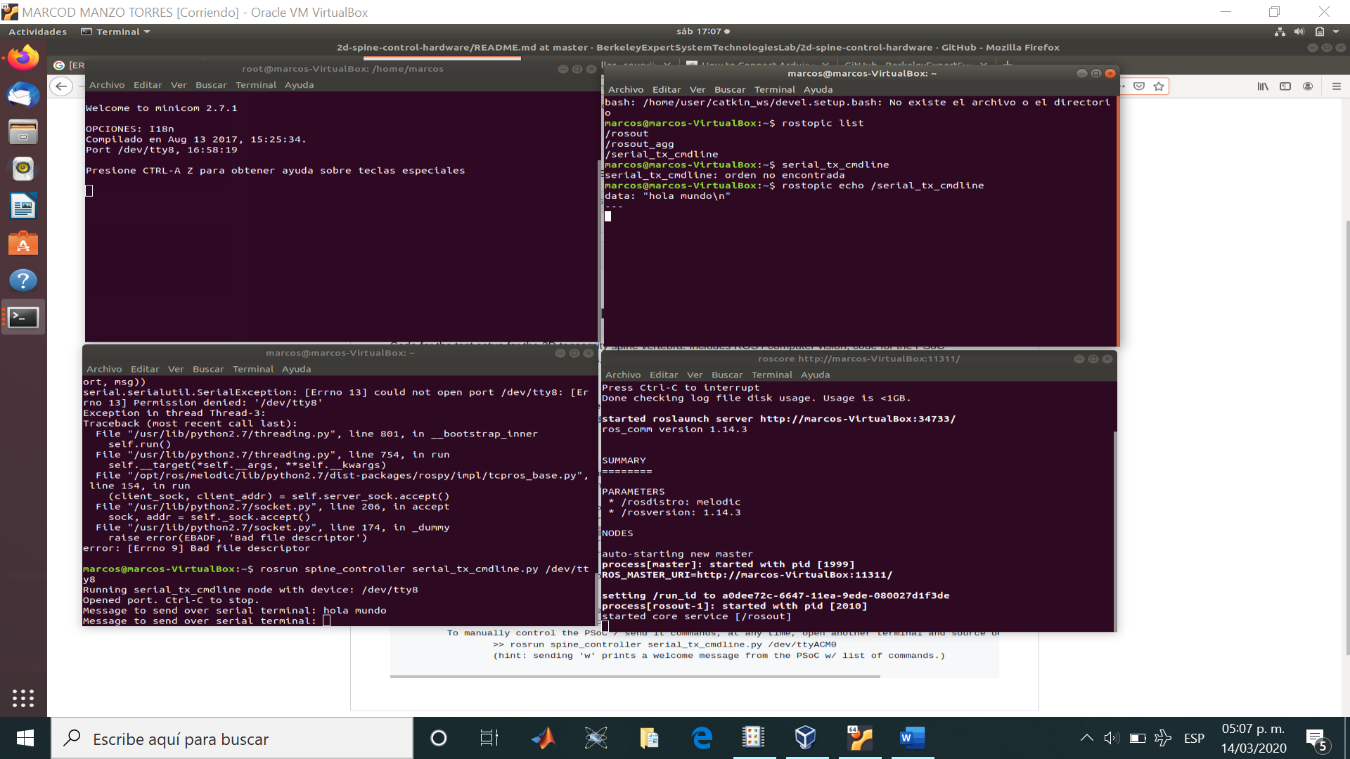
Aparece que el programa está corriendo, ahora basta con ir a rostopic y observar los mensajes emitidos tanto en Publisher como subcriber.



Se observa serial\_tx\_cmdline en rostopic list, por lo que nos dirigimos a el con el siguiente comando para ver las emisiones de la PSoC.



Ahora se ha configurado la PSoC para enviar mensajes de manera directa, como se observa, los enviamos con “ROSRUN” y los recibimos en rostopic.



Como se observa, es un método simple, aunque un poco inusual, algunos puntos importantes a resaltar se encuentran en la comunicación de la PSoC con el sistema operativo Linux mediante la maquina virtual, así como el conocer el nombre del puerto que queremos leer y la velocidad del mismo.

CONCLUSIÓN:

Esta práctica fue complicada, debido a que no existe información alguna que nos facilite la elaboración.

Por mencionar el nombre de una tarjeta usada encontramos “ARDUINO”, aunque Arduino maneja una programación diferente, nos muestra una idea de cómo realizar las conexiones y el cómo trabajar con ellas, si bien el puerto serial sigue siendo el mismo, al trabajar con dos sistemas operativos como Linux y Windows es un mundo totalmente diferente y del que tenemos que tener un conocimiento previo de lo que queremos realizar.

Lo más importante aquí fue el aprendizaje, el saber cómo se hace la comunicación y en todo lo que nos puede llegar a servir en un futuro y sobre todo pensando en el proyecto que estamos elaborando.