Creacion de aplicaciones Arduino con Phyton y Comunicacion Serial

Alisson Guachamin, Jorge Cruz, and Kevin Mejia

[1] Universidad de las Fuerzas Armadas "ESPE"
[2] alissontorres-2000@hotmail.com
https://cec.espe.edu.ec
[3] aguachamin@espe.edu.ec

Abstract. Es infrecuente encontrar en un ordenador de escritorio moderno puertos serie y de existir tampoco podrian usarse directamente con Arduino, ya que trabaja con niveles TTL, por lo que habria que adaptar los que provee el puerto serie EIA232 o RS232, el mas usado en un ordenador personal.
Los puertos USB si que estan muy presentes en los ordenadores modernos y la mayoria de las placas Arduino disponen de un conector USB que se usa como un puerto serie (coloquialmente llamado TTL por los niveles de voltaje comentados arriba) Este puerto serie, desde un punto de vista funcional, esta clonado en un par de pines de la placa (TX/RX) Ademas, algunas placas como Arduino Mega, o Arduino Due, disponen de ms puertos serie UART aparte del correspondiente al conector USB.
Para conectar con el puerto USB del ordenador podra usarse directamente el conector USB de la placa Arduino o un conversor TTL en el ordenador, que conectara a los pines TX/RX de la placa Arduino.

Keywords: Arduino · Puerto Serie EIA232 o RS232 · Conversor TTL.

1 Configuracion de las comunicaciones serie entre Arduino y Python

1.1 Configurar el puerto serie en Python

La libreria pySerial permite acceder a un puerto serie por su nombre o por su numero. La segunda forma es independiente de la plataforma: en todos los sistemas operativos el primer puerto serie sera el 0, el segundo el 1, el tercero el 2 pero, como se ha visto en los apartados anteriores, en Linux puede llamarse /dev/ttyUSB0, en Windows COM1 y en OS X algo como /dev/tty.usbmodem7d22, por lo que para realizar una aplicacion multiplataforma, en la que se deba elegir el puerto a cada uso, habra de contar con esas diferencias. Segun se describe en la documentacion de la API de la libreria PySerial, para configurar el puerto serie se puede elegir entre hacerlo al crearlo pasando directamente los datos de configuracion o crearlo y configurarlo mas adelante. Si se configura directamente al crearlo no sera necesario usar la funcion open() para abrirlo antes de empezar

a enviar o recibir datos por el puerto serie recien asignado. El constructor de la clase espera los siguientes parametros opcionales. port Nombre o numero de puerto al que se conecta. baudrate Velocidad expresada en baudios. bytesize Numero de nits de datos. Pueden usarse las constantes FIVEBITS, SIXBITS, SEVENBITS, EIGHTBITS para referirse a 5, 6, 7, u 8 bits respectivamente. Parity Tipo de control de paridad. Se pueden usar las constantes:

PARITY$_N ONE(sinparidad)$,
PARITY$_E VEN(par)$,
PARITY$_O DD(impar)$,
PARITY$_M ARK(marca)$
PARITY$_S PACE(espacio)$
stopbits Numero de bits de parada.
Estn disponibles las constantes
STOPBITS ONE (u bit de parada),
STOPBITS ONE POINT FIVE (un bit y medio)
STOPBITS TWO (dos bits)timeout

Tiempo de espera de lectura antes de lanzar una excepcion si no se reciben datos. Activar el control de flujo por software. rtscts Activar el control de flujo por hardware DSR/DTR. dsrdtr Activar el control de flujo por hardware DSR/DTR. writeTimeout Tiempo de espera entre escritura. interCharTimeout Tiempo de espera entre caracteres. Si el objeto para acceder a las comunicaciones serie se crea con algo como puerto=serial.Serial() es decir, sin parametros de configuracion, posteriormente se podr configurar asignandole la configuracion a la instancia que se ha creado. Para asignar el numero de puerto se usa puerto.port=0, para asignar la velocidad (en baudios) algo como puerto.baudrate=9600, puerto. timeout=2 para el tiempo de espera y asi sucesivamente. La configuracion por defecto corresponde a una velocidad de 9600 baudios, 8 bits de datos, sin paridad y con un bit de parada (algo, por cierto, bastante comun) que suele expresarse como 9600,8,N,1 o 9600 8N1. Si se cambia el valor de port la conexion se asocia a ese puerto de comunicaciones serie de forma que, si estubiera abierta se cierra y se vuelve a establecer con el nuevo valor.

Configurar el puerto serie en Arduino En la parte de Arduino tambin habr que elegir el puerto serie con el que se trabaja, si es que hay varios disponibles; en este caso es ms sencillo ya que existe un objeto Serial para cada uno de los disponibles. En una placa Arduino Uno, por ejemplo, solo existe el objeto Serial (aunque siempre pueden inplementarse por software comunicaciones serie usando GPIO) mientras que en una placa Arduino Mega puede usarse tanto el objeto Serial (que corresponde con el puerto USB) como Serial1, Serial2 y Serial3 a los que se accedes por los pines correspondientes del a placa.

Para configurar el puerto serie en Arduino basta con llamar a la funcin begin() del objeto Serial que se vaya a utilizar, indicando la velocidad en baudios y una palabra de configuracion que indica los bits de datos, la paridad y los bits de parada en el formato SERIAL ABC siendo fijo el texto "SERIAL" y correspondiendo A al numero de bits de datos, B a la paridad (que puede ser

N, E u O para referirse a sin paridad, par o impar respectivamente) y C a los bits de parada. La configuracion ms habitual es SERIAL 8N1 (8 bits de datos sin paridad y un bit de parada y es la que se usara si se omite este parametro).

Encontrar el puerto serie en uso :

Para poder acceder a un puerto serie del ordenador desde Python es necesario conocer el identificador que corresponde al que conecta con Arduino.

La libreria pySerial incluye algunas utilidades que simplifican el trabajo con los puertos serie. Una de ellas muestra una lista de los puertos que hay disponibles en el ordenador; para lanzar esta utilidad y ver la lista de puertos se utiliza la orden python -m serial.tools.list ports desde la consola.

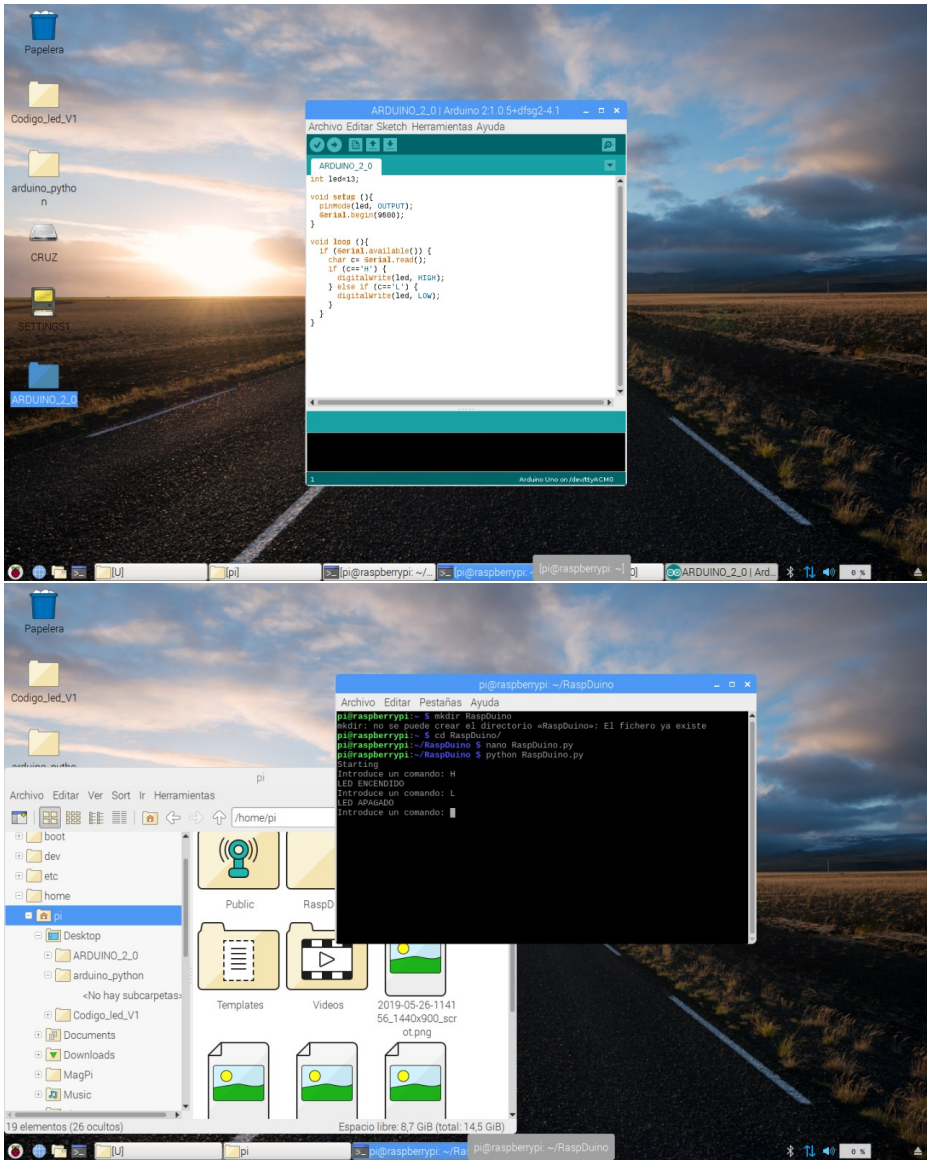

Fig. 1. Programacion en Arduino y Raspberry Pi