Épsilon

Volume 1 | Issue 25 Article 4

2015-12-01

Integración de los sistemas embebidos Raspberry Pi y Arduino para el manejo de un brazo robótico mediante una aplicación Android

David Rolando Suárez Mora
Universidad Cooperativa de Colombia, david.suarezm@campusucc.edu.co

Alexandra Aparicio Gómez
Universidad Cooperativa de Colombia, aparicio@campusucc.edu.co

Yuli Katerine Gallego Ibáñez
Universidad Cooperativa de Colombia, yuli.gallego@campusucc.edu.co

Juan Camilo Ramírez Salcedo
Universidad Cooperativa de Colombia, juan.ramirezs@campusucc.edu.co

Integración de los sistemas embebidos Raspberry Pi y Arduino para el manejo de un brazo robótico mediante una aplicación Android

David Rolando Suárez Mora[1]
Alexandra Aparicio Gómez[2]
Yuli Katerine Gallego Ibáñez[3]
Juan Camilo Ramírez Salcedo[4]

RESUMEN
Los sistemas de comunicación permiten hoy en día controlar dispositivos electrónicos que facilitan su uso y desplazamiento. Sin embargo, la integración entre diferentes tipos de sistemas se convierte en un inconveniente porque existen diversas tecnologías y protocolos. En el presente artículo se plasma la integración de los sistemas embebidos, Raspberry Pi y Arduino, mediante el uso de los lenguajes de programación Java y Phyton, teniendo en cuenta comunicación serial de dos formas diferentes: puerto GPIO y cable USB. El sistema de integración que permite la transmisión y la recepción de datos se realiza por medio del *software* Python, y la interfaz de conexión es realizada con Java sobre una plataforma Android. Con el fin de establecer resultados de primera mano, se diseñó e implementó un brazo robótico, el cual es controlado por un Arduino, y la interfaz de acceso al sistema se realiza usando la placa Raspberry Pi. Los resultados de las pruebas que se hicieron determinaron la eficiencia del sistema de comunicación y la confiabilidad de la información recibida por la placa Raspberry Pi y transmitida a la placa Arduino para la conexión directa con cada uno de los servomotores.

Palabras clave: Arduino, brazo robótico, comunicación, Raspberry Pi, sistemas embebidos.

[1] MSc. Network System. Miembro del grupo de investigación Automatización Industrial, Universidad Cooperativa de Colombia. Correo electrónico: david.suarezm@campusucc.edu.co
[2] Ingeniera electrónica. Miembro del grupo de investigación Automatización Industrial, Universidad Cooperativa de Colombia. Alexandra. Correo electrónico: aparicio@campusucc.edu.co
[3] Ingeniera electrónica. Miembro del grupo de investigación Automatización Industrial, Universidad Cooperativa de Colombia. Correo electrónico: yuli.gallego@campusucc.edu.co
[4] Ingeniero electrónico. Miembro del grupo de investigación Automatización Industrial, Universidad Cooperativa de Colombia. Correo electrónico: juan.ramirezs@campusucc.edu.co

Fecha de recepción: 11 de diciembre de 2014 • Fecha de aprobación: 30 de marzo de 2015

Cómo citar este artículo: Suárez Mora, D. R., Aparicio Gómez, A., Gallego Ibáñez, Y. K. y Ramírez Salcedo, J. C. (2015). Integración de los sistemas embebidos Raspberry Pi y Arduino para el manejo de un brazo robótico mediante una aplicación Android. *Épsilon*, (25), 69-96.

Integration of Raspberry Pi and Arduino Embedded Systems for Controlling a Robot Arm through an Android App

ABSTRACT

Nowadays it is possible to control electronic devices with communication systems that facilitate their use and transportation. However, integration between different types of systems has its drawbacks, due to the existence of different technologies and protocols. This paper captures the integration of the Raspberry Pi and Arduino embedded systems by using Java and Phyton programming languages, considering two different types of serial communication: GPIO port and USB cable. The integration system allowing transmission and reception of data is performed through the Python software, and the connection interface is made using java on an Android platform. In order to establish first hand results, a robotic arm controlled by an Arduino was designed and implemented, and the system access interface is made using the Raspberry Pi plate. Test results show the efficiency of the communication system and the reliability of the information received by the Raspberry Pi plate and transmitted to the Arduino Plate for direct connection to each servomotor.

Keywords: Arduino, communication, embedded systems, Raspberry Pi, robot arm.

Integração dos sistemas embebidos Raspberry Pi e Arduino para o manejo de um braço robótico mediante uma aplicação Android

RESUMO

Os sistemas de comunicação permitem hoje m dia controlar dispositivos eletrônicos que facilitam seu uso e deslocamento. Contudo, a integração entre diferentes tipos de sistemas se transforma em um inconveniente porque existem diversas tecnologias e protocolos. Neste presente artigo se plasma a integração dos sistemas embebidos, Raspberry Pi e Arduino, mediante o uso das linguagens de programação Java e Phyton, levando em conta a comunicação serial de duas formas diferentes: porto GPIO e cabo USB. O sistema de integração que permite a transmissão e a recepção de dados se realiza por meio do software Python, e a interfase de conexão é realizada mediante Java sobre uma plataforma Android. Com o fim de estabelecer resultados de primeira mão, se desenhou e implementou um braço robótico, o qual é controlado por um Arduino, e a interfase de acesso ao sistema se realiza usando a placa Raspberry Pi. Os resultados das provas realizadas determinaram a eficiência do sistema de comunicação e a confiabilidade da informação recebida pela placa Raspberry Pi e transmitida à placa Arduino para a conexão direta com cada um dos servomotores.

Palavras chave: Arduino, comunicação, sistemas embebidos, Raspberry Pi, braço robótico.

Introducción

En la actualidad, cada vez más se puede encontrar con mayor frecuencia el uso de sistemas especializados, los cuales fueron construidos y programados para realizar tareas particulares. El objetivo es dividir tareas grandes en mucho más pequeñas y así construir subsistemas que se especialicen en este tipo de tareas. El concepto de *sistemas embebidos* cobra fuerza en este enfoque, y por tal motivo muchos fabricantes construyen placas con mayores capacidades y diferentes tipos de interfaces.

Uno de los grandes desafíos que se imponen en este panorama es la integración de los diferentes sistemas embebidos. Aquí es cuando surgen soluciones que varían desde la construcción de dispositivos hasta el desarrollo de algoritmos con el fin de realizar dicha integración. Sin embargo, cada sistema embebido por separado ha tenido grandes desarrollos y esto se puede evidenciar en los diferentes proyectos y prototipos que se han construido a partir de estas placas.

La propuesta de especializar sistemas con el fin de atacar problemas grandes trae consigo muchas ventajas, entre las cuales se encuentra la segmentación de errores, la división de tareas, el aprovechamiento de los recursos y el mejoramiento del rendimiento de los sistemas. No obstante, como se comentó, al momento de integrar estos subsistemas se puede encontrar con algunos problemas; entre estos encontramos la falta de sincronismo, el *software* propietario, los conectores propietarios y los datos en formato especial, entre otros.

En conclusión, se pueden encontrar muchas bondades que nos presentan los sistemas embebidos, definidos como sistemas diseñados para realizar tareas particulares. De todas formas, es necesario considerar una serie de tareas necesarias para realizar la integración sin perder todas las ventajas que tiene cada sistema por independiente. En este trabajo se realiza la integración del sistema embebido Raspberry Pi y Arduino. Cada uno de ellos tendrá funciones particulares. El primero permitirá la interacción del sistema con su entorno a través de una interfaz gráfica sobre un sistema operativo Android para su uso en un dispositivo móvil; por otro lado, recibirá las peticiones realizadas por parte de los usuarios mediante el uso de la conexión *wireless*. El segundo sistema realizará la conexión con los servomotores de un brazo robótico. En la figura 1 se puede apreciar la arquitectura general del proyecto en la que se encuentran presentes los sistemas embebidos (Raspberry Pi y Arduino), la interfaz gráfica en Android y los servomotores que forman parte del brazo robótico.

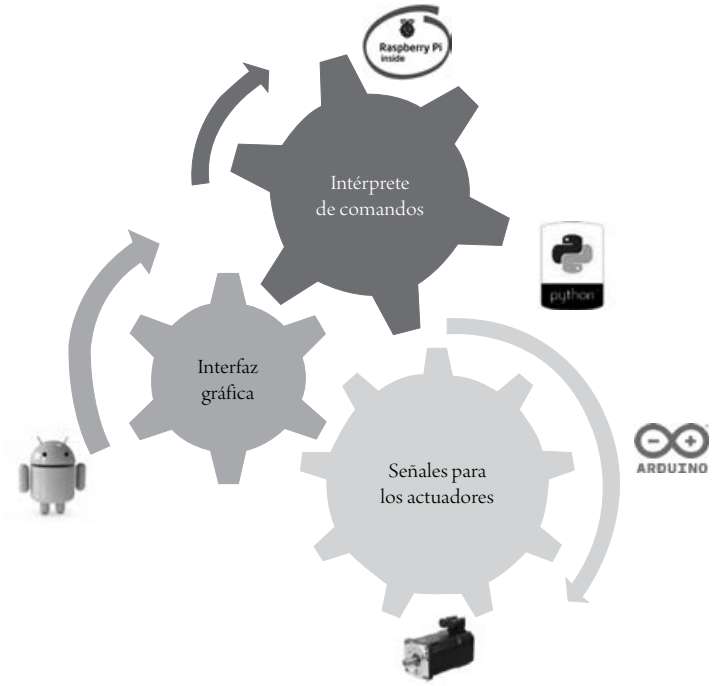

Figura 1. **Arquitectura general del sistema**

Fuente: elaboración propia.

La figura 2 describe el flujo de datos que se establece en el sistema general. Allí se puede apreciar que la información de control es recibida inicialmente por la aplicación Android. Luego se envía a la placa Raspberry Pi mediante una interfaz de conexión *wireless*. Posteriormente, la placa Raspberry Pi envía la información a la tarjeta Arduino, la cual procesa esta información y es enviada a los servomotores del brazo robótico.

Estado del arte

En el siguiente estado del arte se abordan algunos proyectos desarrollados según dos placas que atienden los conceptos anteriormente comentados, las cuales son la Raspberry Pi y la placa Arduino 1.

Existen diferentes campos de acción para este tipo de tecnologías. Uno de ellos es la telemedicina. Así es como Dudas y colaboradores (2014), en el Departamento

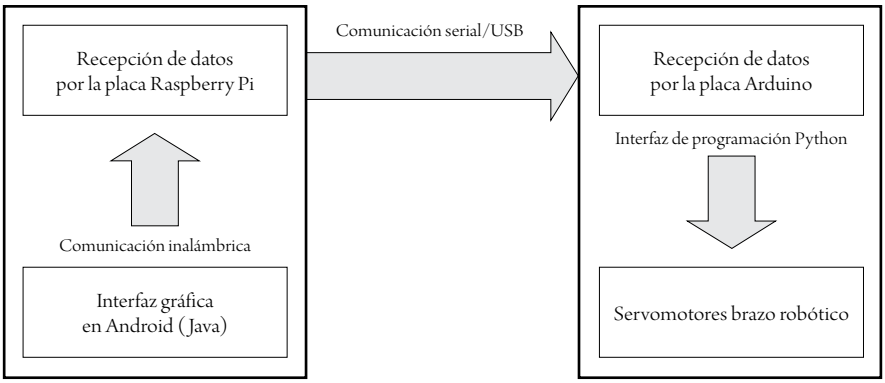

Figura 2. **Flujo de información en el sistema**

Fuente: elaboración propia.

de Patología de la Universidad de Medicina Johns Hopkins, desarrollaron una solución denominada telecitología, que implementa el uso de la placa Raspberry Pi para la recepción de imágenes desde un iPhone. Este proyecto arrojó como resultado que una placa de bajo costo podía dar prestaciones y rendimiento como un dispositivo de gama alta. Además, se encontró la posibilidad de manipular imágenes de buena calidad con la integración de una cámara de video convencional.

Según Siegle (2014), el *hardware* de código abierto como Arduino, Raspberry Pi, y otras placas que han sido desarrolladas para ser usadas en diferentes aplicaciones, ha permitido a campos como la electrofisiología acercarse a soluciones más económicas con las mismas prestaciones y rendimiento que una solución creada por una compañía especializada (Jutila et al., 2014). Otro campo de acción para este tipo de placas es el *geoposicionamiento*. Con este concepto se desarrolló un sistema que permite el seguimiento de niños dentro de un área particular. Al tener un tamaño pequeño, la placa Arduino y muchas placas de su tipo se prestan para ser ubicadas en sitios estratégicos. En este caso fue ubicada en los chalecos que usaba un grupo de niños dentro de una escuela, y mediante el uso de la tarjeta XBEE de Arduino y algunos sensores fue posible hacer un seguimiento dentro de las instalaciones de la escuela a los niños y determinar si cada uno de ellos se encontraba en algún tipo de situación que generara peligro para la integridad de los infantes.

Sin embargo, el tema de las placas especializadas no solo se centra en proyectos que tratan la información para el control de actuadores, sino también en campos

donde la información es recibida y transmitida en un canal de comunicación, que tiene como forma de transmisión medios no guiados. Así, Ferdoush y Li (2014) desarrollaron un sensor *wireless* con la integración de los sistemas embebidos Raspberry Pi y Arduino, como una opción de bajo costo en el amplio campo de los sensores inalámbricos.

Según el artículo anterior, sobre tratamiento y transmisión de la información a través de medios no guiados, las placas especializadas son capaces de implementar algoritmos para el control y seguridad de la información, con el fin de evitar ataques por parte de terceros en la implementación de este tipo de proyectos. De esta manera, Al-Haija y colaboradores (2014) desarrollaron un algoritmo criptográfico sobre la placa Arduino, la cual implementa un esquema de seguridad a través de un proceso criptográfico.

Teniendo en cuenta todos los desarrollos mencionados anteriormente, Koenka, Sáiz y Hauser (2014) crearon un *framework* con el fin de facilitar el desarrollo de aplicaciones en Python como lenguaje de programación para el control y desarrollo de la placa Arduino.

La tabla 1 muestra una descripción de otros trabajos relacionados con la integración de las placas Arduino y Raspberry Pi, con el fin de controlar *hardware*. En la tabla 1 se presenta un resumen de cada proyecto, se revisa si existe una integración entre las placas y si se logró el control del *hardware* mediante el híbrido.

Tabla 1. **Síntesis de integración de placa y control del hardware a través de la integración de las placas**

Título	Integración entre placas	Resumen	Control de *hardware*
A tiny RSA cryptosystem based on arduino microcontroller useful for small scale networks (Al-Haija et al., 2014)	No	En este artículo se muestra el desarrollo de un sistema de encriptación para una red mediante el protocolo Rivest, Shamir y Adleman; para ello se usa un microcontrolador Arduino.	No
Instrumentino: An open-source modular Python framework for controlling Arduino based experimental instruments (Koenka, Sáiz y Hauser, 2014)	Sí	En este proyecto se diseñó un *framework* que facilita el desarrollo de aplicaciones en Python con el fin de controlar una placa Arduino. Este lenguaje de programación es usado para el control de flujo de datos en entrada y salida para placa Arduino.	No

épsilon, nº 25 • julio-diciembre 2015 • pp. 69-96 • ISSN 1692-1259-ISSN-E 2389-8828

Título	Integración entre placas	Resumen	Control de *hardware*
Inexpensive telecytology solutions that use the Raspberry Pi and the iPhone (Dudas et al., 2014)	No	Este artículo muestra el desarrollo de una aplicación, mediante la cual se utiliza la cámara de un teléfono celular, en este caso un iPhone, con el fin de analizar las imágenes en un examen médico. La placa Raspberry Pi trata las imágenes y las muestra en una interfaz con el fin de ser analizadas por el experto.	Sí
Wireless sensor network system Design using Raspberry Pi and Arduino for environmental monitoring applications (Ferdoush y Li, 2014)	Sí	Este artículo presenta un sensor inalámbrico para monitorear variables de una aplicación particular; en este se observa la integración de las placas Arduino y Raspberry Pi con el fin de tomar datos y tratarlos en una aplicación particular.	No
Open-source approaches to hardware for large-scale electrophysiology (Siegle et al., 2015)	No	En este artículo se presenta un estudio sobre cómo las placas de código abierto podrían ser una excelente solución en el campo de la electrofisiología y particularmente en la neurobiología con el fin de evitar adquirir costosos desarrollos propietarios.	No
Implementation of a wearable sensor vest for the safety and well-being of children (Jutila et al., 2014)	No	En este artículos se muestran los resultados encontrados del desarrollo y la implementación de un chaleco, el cual tiene sensores para determinar temperatura, aceleración y posición.	Sí

Fuente: elaboración propia.

Metodología

Para la integración de las placas y la conexión con el brazo robótico se realizaron los siguientes pasos: a) definición del problema, b) comunicación entre placas, c) criterios de selección de los lenguajes de programación, d) diseño de los algoritmos, e) pruebas, f) creación de la aplicación móvil y g) conclusiones.

Definición del problema

A pesar de tener el concepto de que cada placa es una plataforma embebida, las cuales tienen sus puertos de comunicación y sus dispositivos para el procesamiento

de datos, cada una de ellas trabaja sobre mundos distintos y se ha especializado en diferentes campos de acción. Para el caso de la placa Raspberry Pi, es concebida como un computador diminuto en el cual corre un sistema operativo Linux, que ofrece todas las bondades de interfaz gráfica, procesamiento y análisis de datos. Por otro lado, permite correr servicios particulares. En otras palabras, se podría decir que esta placa trabaja en los lenguajes denominados de alto nivel.

Sin embargo, cuando se trabaja a bajo nivel, señales como voltajes o corrientes provenientes de actuadores o pulsadores, la placa no cuenta con las herramientas que permitan hacer fácilmente esta labor. Esto no significa que no sea capaz de realizar este tipo de tareas, sino que se deben implementar más pasos y procedimientos para realizar la misma tarea. En contraste, las placas como Arduino tienen todo el desarrollo de *hardware* y *software* que facilitan la integración con aplicaciones a bajo nivel. Por lo tanto, con el fin de extraer las mayores prestaciones de cada placa y especializar las tareas, aprovechando las ventajas que ofrecen cada una de ellas, se plantea la integración entre estas; no obstante, tienen problemas de conectividad, velocidad y protocolos de transmisión.

Por otro lado, la interfaz de comunicación para el usuario debe ser capaz de recibir los datos y realizar la validación y envío de dichos datos a la placa que realizará la manipulación inicial. De todos modos, todo este proceso debe ser transparente para el usuario. La integración de las entidades anteriormente mencionadas es el problema que se presenta y el cual será resuelto en las siguientes secciones.

Comunicación entre la placa Raspberry Pi y Arduino

La comunicación entre Arduino y Raspberry Pi se realiza por medio de comunicación serial usando el puerto 'COM' de Arduino y los puertos GPIO de la Raspberry Pi. Estos últimos permiten la comunicación de la placa con dispositivos conectados a través de los puertos comentados. Cabe resaltar que estos puertos manejan voltajes de 3 V, y por ello es necesario realizar una etapa de acople de voltajes, debido a que la placa Arduino maneja voltajes del orden de los 5 V. En la figura 3 se pueden observar los pines GPIO en la tarjeta Raspberry Pi. Estos puertos están físicamente diseñados para ser conectados a través de pines de conectores, con voltajes de salida de 3 V.

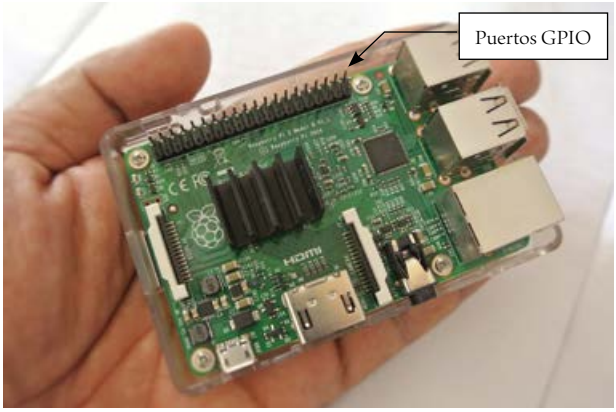

Figura 3. **Diagrama general de la placa Raspberry Pi**

Fuente: Pixabay (2016).

La forma más común de establecer dicha la transmisión de datos entre las placas es a través del protocolo comunicación serial. Esta consiste en la transmisión y recepción de pulsos digitales a una misma velocidad, ya que el transmisor envía pulsos que representan el dato enviado a una velocidad determinada y el receptor escucha dichos pulsos a esa misma velocidad. Para realizar la conexión entre las placas en el *software*, se implementó en cada una de ellas un algoritmo con el fin de establecer los protocolos de comunicación y las políticas de velocidad. En la placa Raspberry Pi se instaló el lenguaje de programación Python y en la Arduino se utilizó el IDE propietario para establecer la configuración necesaria para los puertos.

En Python es necesario declarar una variable que establezca la velocidad de comunicación. En este caso se estableció el valor de 9600. La siguiente línea muestra la sintaxis para declarar la variable.

Nombre_Variable=serial.Serial('/dev/ttyACM0',9600)

En el caso de la placa Arduino, la línea necesaria para la conexión es la siguiente:

Serial.begin(9600);

Después de configurar los dos sistemas embebidos a la misma velocidad de transmisión, se procede a ejecutar los comandos de escritura y lectura de los puertos

seriales (READ, WRITE), para asegurar el intercambio de datos. Es indispensable que uno de los sistemas embebidos escriba los bits en el puerto serial y el otro esté dispuesto a leerlos.

Envío y recepción de datos en Python

Nombre_Variable.write(dato a enviar) Comando para enviar dato.
Dato = Nombre_Variable.read(parámetro) Comando para leer dato.

Envío y recepción de datos en Arduino IDE

Serial.write(dato a enviar); Comando para enviar dato
Char Dato = Serial.read(); Comando para leer bit
String Dato = Serial.readByte(buffer,long) Comando para leer byte

En Arduino se tienen dos formas de leer los datos: una es por *bit*, en la que solo es posible recibir un carácter, y la otra es por *byte*, en la cual se puede recibir una cadena. En esta última los parámetros son dos: *buffer* y *Long*; el primero es el vector que guarda cada uno de los caracteres de la cadena, y el segundo es la longitud o tamaño de la cadena. Es necesario tener en cuenta que al momento de enviar enteros por el puerto serial este los envía en formato ASCII.

Criterios de selección de los lenguajes de programación

Para lograr la implementación del sistema fue necesario el uso de ciertos lenguajes de programación que permitieron realizar la integración en la parte lógica para cada una de las tarjetas embebida. En la tabla 2 se muestra un resumen de los lenguajes y sus funciones dentro del sistema.

Tabla 2. **Resumen de los lenguajes de programación y sus funciones dentro del sistema**

Lenguaje de programación	Funciones	Capa arquitectura
Java	• Aplicación en Android • Interacción con el usuario a través de la interfaz gráfica • Conexión entre la aplicación Android y la placa embebida Raspberry Pi • Envío de los parámetros para la selección de los diferentes modos	Capa de presentación

épsilon, nº 25 • julio-diciembre 2015 • pp. 69-96 • ISSN 1692-1259-ISSN-E 2389-8828

Lenguaje de programación	Funciones	Capa arquitectura
Python	• Recepción de la información para el movimiento de los motores • Conversión de la posición a grados • Conversión de la información dada en grados al formato hexadecimal • Manejo de los puerto GPIO del sistema embebido Raspberry Pi • Envío de los parámetros para la placa Arduino	Capa de negocio
C	• Algoritmo para el manejo de la placa Arduino • Envío de los parámetros a los servomotores • Interpretación de los parámetros enviados por la placa Raspberry Pi mediante el lenguaje de programación Python	Capa de datos

Fuente: elaboración propia.

Los criterios para la selección de los lenguajes de programación anteriormente nombrados se determinaron a partir de las siguientes necesidades:

1. Lenguaje de programación orientado a objetos con el fin de proveer escalabilidad e integración. Aplicaciones móviles propietarias para el caso particular Android. Estos criterios permitieron seleccionar el lenguaje de programación Java.
2. Lenguaje para el manejo y control de los puertos GPIO, manejo de funciones y escalabilidad, ejecución de rutinas desde consola, recibiendo parámetros y manejo de programación orientada a objetos. Según estos criterios el lenguaje de programación seleccionado es Python.
3. Para el diseño del algoritmo en Arduino se selecciona el lenguaje C, debido a que es usado por este sistema propietario.

Descripción de la aplicación

En la figura 5 se muestra la aplicación final (control de un brazo robótico) de la integración de las dos plataformas, a través de la interfaz de comunicación de la tarjeta embebida Arduino.

El servomotor seleccionado para la implementación del brazo robótico es el Herkulex DRS-0101 para las rotaciones del dispositivo, ya que está diseñado principalmente para aplicaciones de robótica por su capacidad de controlar velocidad,

Figura 5. **Brazo robótico**

Fuente: elaboración propia.

posición, temperatura y torque. Una vez definidas las características del dispositivo para pruebas, se realizan los respectivos planos para hacer los cortes de las piezas en acrílico. Luego se ensambla cada uno de los elementos para obtener el dispositivo para pruebas esperado, el cual se expone a continuación.

Diseño de los algoritmos

Como se comentó anteriormente, se diseñó e implementó el código necesario para la integración en cada una de las placas. En la figura 6 se puede apreciar el diagrama de flujo para el código en Python, el cual envía los parámetros necesarios a la placa Arduino para que esta mueva los motores en la dirección solicitada por el usuario. En el código se puede apreciar que en la primera parte del algoritmo se cargan las librerías de conexión para el canal de comunicación serial. Después de esto se almacena en un vector de tres posiciones el valor que se le envía al programa por argumento. Con la primera posición del vector se toma la decisión de correr el programa en modo individual o modo rutina; si se selecciona el modo individual se lee la siguiente posición del vector, el cual determina la parte del brazo robótico que se desea mover. La tercera posición del vector determina el punto hasta donde se debe mover la parte del brazo que se seleccionó en el paso anterior.

Si el valor almacenado en la primera posición del vector es 2, el algoritmo toma el modo rutina. La segunda posición del vector determina la rutina que debe seguir el algoritmo. La diferencia entre los dos modos es que el primero mueve las partes

épsilon, nº 25 • julio-diciembre 2015 • pp. 69-96 • ISSN 1692-1259-ISSN-E 2389-8828

del brazo robótico de forma independiente, mientras que el segundo mueve el brazo robótico siguiendo algunas rutinas previamente almacenadas.

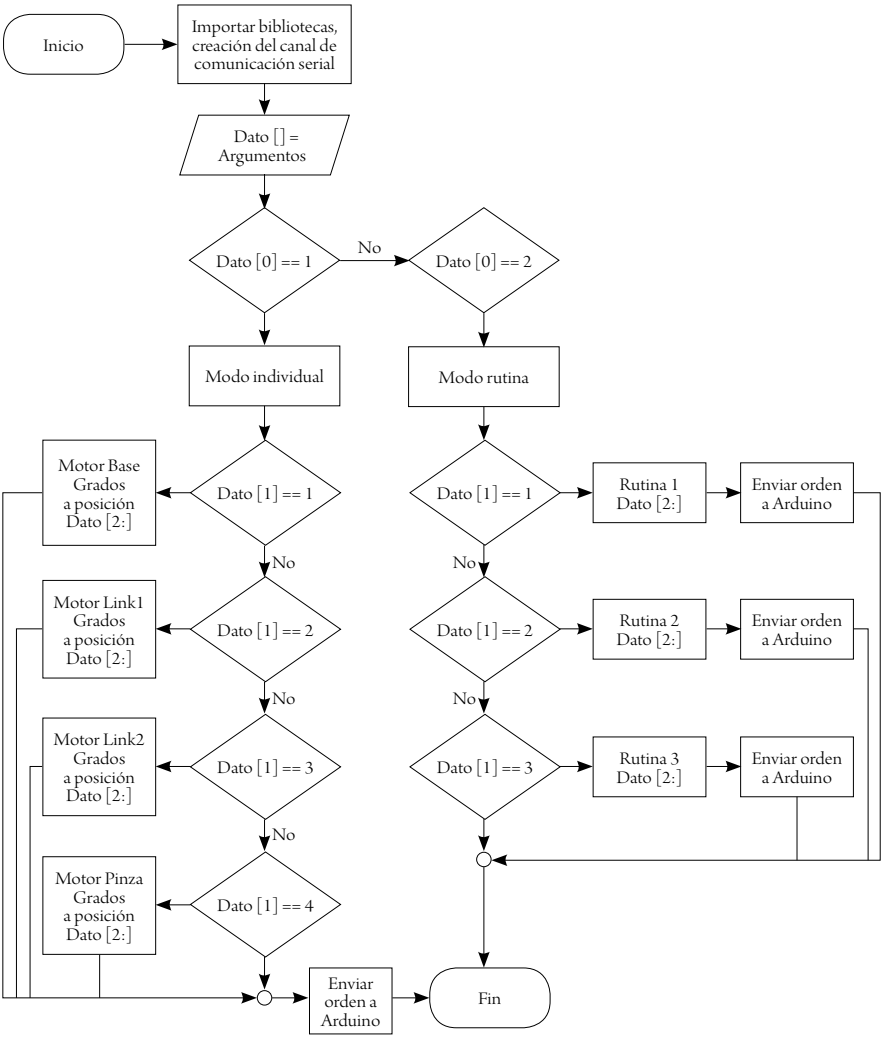

Figura 6. **Código de Python**

Fuente: elaboración propia.

En Python se crea un programa de lectura de argumentos por línea de comando. Lo primero que se realiza es el llamado de las bibliotecas, así como la creación del canal de comunicación del puerto serial, también la lectura de los argumentos del mismo programa (el argumento está compuesto de información como modo de

ejecución, rutina o parte a mover y grados). Una vez hecho esto, el programa descompone este argumento y toma una serie decisiones; la primera es el modo de ejecución del dispositivo para pruebas:

1. Modo individual. En este modo el algoritmo tiene preseleccionadas unas rutinas de movimiento específicas que realiza sin la intervención del usuario. El objetivo de este modo es permitir realizar pruebas de control sobre el brazo; sin embargo, no permite realizar movimientos particulares que el usuario desee.
2. Modo rutina. En este modo el usuario enviará la información de cuál parte del brazo robótico desea mover y a cuántos grados, a diferencia del modo anterior, en el que la rutina se realizaba independiente. Cabe indicar que con ayuda de un condicional ingresa en alguna de estas opciones. En el modo rutina se hace un llamado a otro condicional, mediante el cual se selecciona la parte del dispositivo que desea mover: 1) Base, 2) Link 1, 3) Link 2 y 4) Pinza. Al momento de ingresar alguna de estas opciones, el programa convierte grados a posición y, por último, se envía una cadena de cinco *bits* a Arduino: un *bit* para el modo de ejecución, un *bit* para la parte que se desea mover y tres *bits* para los grados. En la figura 7 se muestra la forma en la que son enviados los datos; en esta se incluyen las posiciones en el vector.

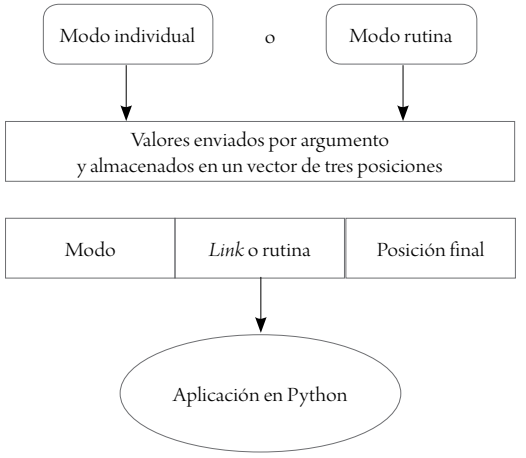

Figura 7. **Arquitectura para la selección de rutinas mediante el envío de argumentos**

Fuente: elaboración propia.

El siguiente es el algoritmo para el enviar la información al Arduino. Este atiende al diagrama de flujo que se mostró en la figura 6. Sin embargo, se debe tener en

cuenta que en este código se implementan las ecuaciones para convertir de grados a posición y viceversa, esto con el fin de capturar valores en grados dados por el usuario y convertir esta información en posiciones para ser enviadas al dispositivo a través del Arduino.

```
import sys
import serial
import time
arduino = serial.Serial('/dev/ttyACM0',9600)//determina la conexión y la velocidad de la conexión
time.sleep(2) //Retardo en la conexión
dato=sys.argv[1]// recibe el valor de las posiciones
//En esta parte del código dependiendo del valor de la posición, convierte a los valores en grados
//Existe un vector donde se almacenan los parámetros necesarios para determinar el movimiento de los //servomotores
//La posición 0 del vector define el modo de rutina, 1 para modo individual y 2 para modo rutina
if dato[0]=='1':
//En el Modo individual se hace un llamado a otro condicional donde se selecciona la parte del //dispositivo que desea mover, 1. Base, 2. Link1, 3. Link2 y 4. Pinza, al momento de ingresar //alguna de estas opciones el programa convierte grados a posición y por último se envía una //cadena de 5 bits a Arduino, 1 bit para el modo de ejecución, 1 bit para la parte que se desea //mover y 3 bit para los grados
 if dato[1]=='1':
 grados=int(dato[2:])
 if grados > 159:
 grados=159
 if grados < -159:
 grados=-159
 posicion=int((((1002-512)/159.8)*grados)+512)
 if posicion <100:
 posicion='0'+str(posicion)
 envio=dato[0]+dato[1]+str(posicion)
 arduino.write(envio)
 elif dato[1]=='2':
 grados=-1*int(dato[2:])
```

```
if grados > 159:
grados=159
if grados < -159:
grados=-159
posicion=int((((1002-512)/159.8)*grados)+512)
if posicion <100:
posicion='0'+str(posicion)
envio=dato[0]+dato[1]+str(posicion)
arduino.write(envio)
elif dato[1]=='3':
grados=int(dato[2:])
if grados > 159:
grados=159
if grados < -159:
grados=-159
posicion=int((((1002-512)/159.8)*grados)+512)
if posicion <100:
posicion='0'+str(posicion)
envio=dato[0]+dato[1]+str(posicion)
arduino.write(envio)
elif dato[1]=='4':
decision='00'+dato[2:]
envio=dato[0]+dato[1]+str(decision)
arduino.write(envio)

elif dato[0]=='2':
if dato[1]=='1':
envio=dato[0]+dato[1]+'000'
arduino.write(envio)
elif dato[1]=='2':
envio=dato[0]+dato[1]+'000'
arduino.write(envio)
elif dato[1]=='3':
envio=dato[0]+dato[1]+'000'
arduino.write(envio)
arduino.close()
```


épsilon, nº 25 • julio-diciembre 2015 • pp. 69-96 • ISSN 1692-1259-ISSN-E 2389-8828

El algoritmo para administrar el Arduino primero configura e inicializa los servomotores, después recibe los datos que provienen de Python; estos datos son procesados y se convierten en tres variables, las mismas que se usaron en el programa Python: modo, parte y grados. Por último, ejecutamos las instrucciones de movimiento del motor con el ángulo correspondiente. Esto se muestra en la figura 8.

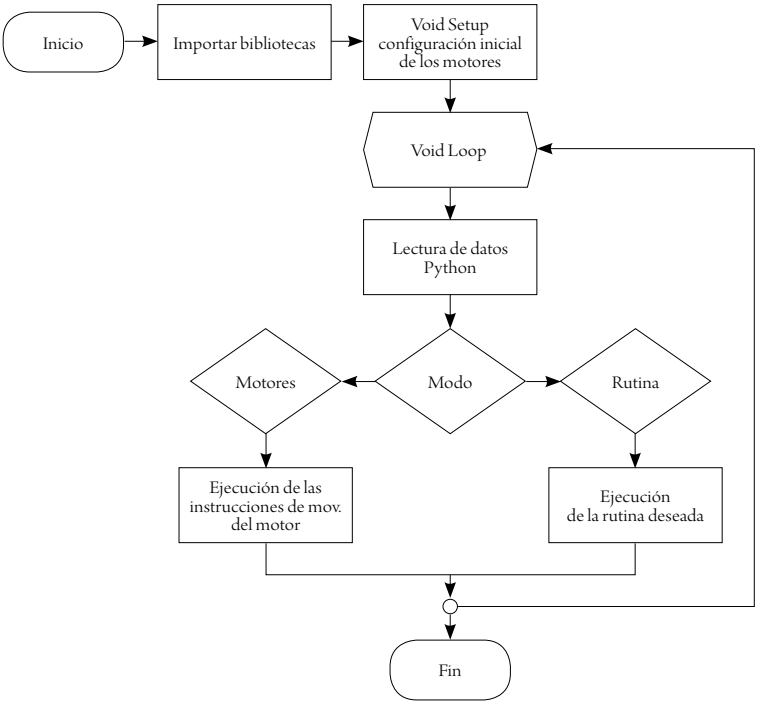

Figura 8. **Código de Arduino**

Fuente: elaboración propia.

El siguiente código muestra el algoritmo implementado en el Arduino a fin de enviar información a los servomotores. Esta información es recibida por el Arduino proveniente de la Raspberry Pi; en este se aprecia que en la primera función (*setup*) hay una rutina previamente establecida; en la segunda función (*loop*) se aprecia el código que recibe la posición a la cual el usuario desea que el brazo robótico se mueva.

//El algoritmo primero configura e inicializa los servomotores, después recibe los datos
//que provienen de Python, estos datos son procesados y se convierten en tres variables,

```
las mismas //que se usaron en el programa Python, Modo, Parte y Grados, por último,
ejecutamos las //instrucciones de movimiento del motor con el ángulo correspondiente
#include <Herkulex.h>
#include <Servo.h>

int modo=0;
int articulacion=0;
int posicion=0;
Servo pinza;

void setup(){
 Serial.begin(9600);
 Herkulex.begin(115200,10,11);
 Herkulex.reboot(1);
 Herkulex.reboot(2);
 Herkulex.reboot(3);
 delay(500);
 Herkulex.initialize();
 Herkulex.moveAllAngle(1, 0, 1);
 Herkulex.moveAllAngle(2, -6, 1);
 Herkulex.moveAllAngle(3, 5, 1);
 Herkulex.actionAll(2000);
 delay(2100);
 pinza.attach(9);
 pinza.write(180);
}
// la función loop
void loop(){
 if (Serial.available() >= 5) {
 char pos[5];
 Serial.readBytes(pos,5);
 int modo=pos[0]-0x30;
 int articulacion=pos[1]-0x30;
 int posicion=(pos[4]-0x30)+(pos[3]-0x30)*10+(pos[2]-0x30)*100;
 Serial.println(modo);
 Serial.println(articulacion);
 Serial.println(posicion);
```

```
if(modo==1){
if(articulacion==1){
Herkulex.moveOne(1, posicion, 1000, LED_BLUE);
}
else if(articulacion==2){
Herkulex.moveOne(2, posicion, 1000, LED_BLUE);
}
else if(articulacion==3){
Herkulex.moveOne(3, posicion, 1000, LED_BLUE);
}
else if(articulacion==4){
if (posicion==001){
pinza.write(180);
}
else{
pinza.write(90);
}
}
}

else if(modo==2){
if (articulacion ==1){
Herkulex.moveAll(1, posicion, 1);
}
else if (articulacion ==2){
Herkulex.moveAll(2, posicion, 1);
}
else if (articulacion ==3){
Herkulex.moveAll(3, posicion, 1);
}
else if (articulacion ==4){
Herkulex.actionAll(2000);
}
}
}
}
```

Pruebas

La primera prueba que se realiza con Raspberry Pi-Arduino consiste en encender o apagar un led por medio de un comando enviado desde la ventana de comando de Raspberry Pi. Con este proceso se comprueba que existe la comunicación serial entre los dos sistemas embebidos gracias a la librería descargada para Python (Python-serial) y la librería propia de Arduino de comunicación serial. En la figura 9 se muestra el código diseñado en Python para enviar señales de encendido y apagado a un led; de esta manera se prueba la comunicación entre las dos placas.

```
import serial

arduino = serial.Serial('/dev/ttyACM0',9600)


while True:
        comando = raw_input('Introduce un comando: ')
        arduino.write(comando)
        if comando == 'H':
                print('LED ENCENDIDO')
        elif comando == 'L':
                print('LED APAGADO')

arduino.close()#FINALIZA LA COMUNICACION
```

Figura 9. **Programación en Python para el envío de información al Arduino**

Fuente: elaboración propia.

El código mostrado atiende la siguiente lógica:

Importación de la librería serial

Creación de la variable llamada Arduino donde se establece la comunicación serial por el puerto ttyACM0 a una velocidad de 9600 baudios,

Ciclo infinito se recibe una variable String (en este caso una H para encender el led o una L para apagarlo)

Envió del bit usando el puerto serial,

Impresión por pantalla de la opción escogida por medio de condicionales.

El siguiente código se implementó en el Arduino para interpretar los datos enviados por la placa Raspberry Pi con el fin se de enviarlos a los puertos.

```
int led = 13;
void setup () {
        PinMode(led, OUTPUT);
        Serial.begin(9600);
}
void loop () {
```

// El ciclo determina si hay alguna comunicación serial, esto se hace con el comando Serial.available, *si esta existe, se lee la información y se guarda en una variable tipo char, la condicional ejecuta la decisión que prender o apagar el led. Cuando se ejecute la aplicación el usuario deberá seleccionar entre algunas de las dos opciones*

```
if (Serial.available()) {
        char c = Serial.read();
        if (c == 'H') {
                digitalWrite(led, HIGH);
        }
        else if (c == 'L') {
                digitalWrite(led, LOW);
        }
} }
```

La siguiente prueba que se desarrolló fue el control de posición de un servo motor de señal PWM (*pulse width modulation*, 'modulación por ancho de pulso'). Para dicha prueba se realiza un programa en Python que solicita un dato; en este caso particular se solicita la posición del servomotor que está reflejada en un ángulo de 0 a 180. Por último, se escribe el dato en el puerto; en Arduino el dato se lee por medio de la instrucción Serial.readBytes, que permite guardar el dato en un vector. Es necesario tener en cuenta que al momento de enviar enteros por el puerto serial, este los envía en formato ASCII, por ello fue necesario usar la siguiente fórmula para convertir un dato ASCII a un entero.

$$n = Dato\,[0] - 0\,x\,30 \qquad (1)\ \text{Transformación de ASCII a entero (unidad)}$$

La fórmula anterior solo sirve para convertir solo una unidad. Si lo que se quiere es convertir una decena o un número mayor, se usa la siguiente fórmula (ejemplo si el dato es de tres cifras):

$$n = (Dato\,[2] - 0\,x\,30) + 10 * (Dato\,[1] - 0\,x\,30) + 100 + (Dato\,[0] - 0\,x\,30)$$

(2) transformación de ASCII a entero (números mayores a la unidad)

Después de convertir el dato, la instrucción de escritura de la librería del servomotor nos ayuda a visualizar la posición del servo. La figura 10 muestra la forma de conexión de la placa Arduino con el servomotor.

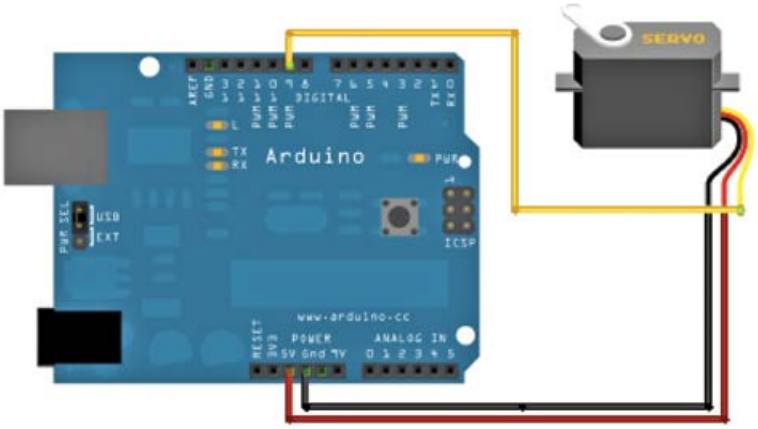

Figura 10. **Montaje de prueba con servomotor**

Fuente: Arduino (2015).

El siguiente código se implementa con el fin de mover un servomotor de prueba. En la función *Setup* se establece la velocidad de comunicación y en la *loop* se envían los datos para realizar el movimiento. Cabe desatacar que esta información es enviada en formato hexadecimal.

```
#include <Servo.h>
char posicion[3];
```


épsilon, nº 25 • julio-diciembre 2015 • pp. 69-96 • ISSN 1692-1259-ISSN-E 2389-8828

```
Servo motor;
unsigned int pos=0;

void setup(){
 Serial.begin(9600);// establece la velocidad de comunicación
 motor.attach(9);
}
// envía los datos para realizar el movimiento
void loop(){
 if (Serial.available() >= 3){
 Serial.readBytes(posicion,3);
//Convierte a hexadecimal la posición para ser enviada al Arduino
 pos=(posicion[2]-0x30)+(posicion[1]-0x30)*10+(posicion[0]-0x30)*100;
 motor.write(pos);
 }
}
```

En el siguiente código se capturan los valores en grados que deben mover el motor y después se envían al código en Arduino para que este mueva el motor a la posición deseada.

```
import serial
arduino = serial.Serial('/dev/ttyACM0', 9600)
print("Starting!")

while True:
//captura los valores en grados que se debe mover el motor
 grados = raw_input('Que posicion desea mover el servo (000-180 grados): ')
//envió el valor en hexadecimal para que el Arduino mueva el motor a la posición
 arduino.write(grados)
arduino.close()
```

La prueba que se ejecutó fue el manejo de un servomotor digital Herkulex DRS-0101; se descarga la biblioteca para el manejo del motor con Arduino. En el siguiente enlace se encuentra la biblioteca junto al listado de instrucciones para el correcto manejo de esta: http://robottini.altervista.org/dongbu-herkulex-arduino-library-2.

El programa de Python es exactamente el mismo que el anterior. La única diferencia es el dato que se va a enviar, ya que no se enviarán los grados, sino directamente la posición. Sin embargo, se puede convertir la posición en grados de la siguiente forma:

$$Grados = \frac{(Posición - 512) * 159.8}{(1002 - 512)} \qquad (3)\ \text{Posición a grados}$$

$$Posición = \left(\frac{(1002 - 512) * Grados}{159.8} \right) + 512 \qquad (10)\ \text{Posición a grados}$$

En el Arduino, se definen los pines de Tx y Rx del motor así como la velocidad de transición 115.200 baudios. Cabe indicar que estos motores se comunican por medio de comunicación serial; por ello se les asigna una velocidad de transmisión diferente a la utilizada en la comunicación entre Raspberry Pi y Arduino (9600 baudios). Después lee el dato del puerto serial; en este caso lee un dato de 4 bit, lo convierte en un entero, este se pasa como argumento a la instrucción de movimiento del servomotor: Herkulex.moveOne(). En la figura 11 se muestra la conexión entre el Arduino y el servomotor Herkulex.

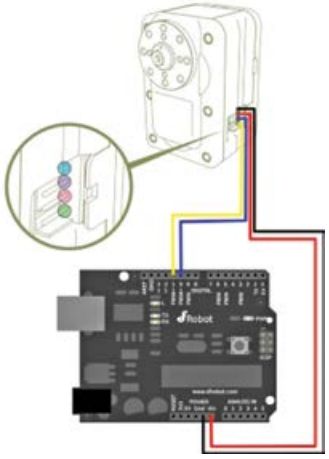

Figura 11. **Montaje de prueba con servomotor Herkulex DRS-0101**

Fuente: Dfrobot (2015).

épsilon, nº 25 • julio-diciembre 2015 • pp. 69-96 • ISSN 1692-1259-ISSN-E 2389-8828

Creación de aplicación móvil

Para verificar la conexión SSH entre la aplicación móvil en Android y la Raspberry Pi, se realizó una prueba enviándole el comando *shutdown - h now* desde la aplicación de prueba creada a la Raspberry Pi, la cual fue exitosa. El siguiente algoritmo envía un comando a la tarjeta Raspberry Pi que permite apagar la placa. Este código se encuentra embebido en la aplicación Android y es enviado a esta a través de la comunicación inalámbrica.

```
public class SSHCliente extends AsyncTask<String, Integer, String> {
    public SSHCliente() {
            jsch = new JSch();
    try {
    JSch.setConfig("StrictHostKeyChecking", "no"); //Tipo de conexión
    sesion = jsch.getSession("pi","192.168.0.150",22); //Usuario y dirección IP
    sesion.setPassword("raspberry"); //contraseña para la conexión
    sesion.connect();
    m= sesion.isConnected(); //creación del objeto M el cual genera la sesión
    x= sesion.getHost();
    consola = (ChannelExec) sesion.openChannel("exec");
//Envió del comando para ser ejecutado en la placa Raspberry Pi
    ((ChannelExec)consola).setCommand("sudo shutdown -h now.py");
    consola.connect();
    } catch( JSchException e) {
    System.out.println("Error de JSCH. Mensaje: "+e.getMessage());
    }
    }
    }
```

A continuación se realizó otra prueba, la cual consistía en el envío del comando del archivo para encendido y apagado de un led. Esta prueba se implementa en el siguiente código. En la primera parte del algoritmo se realiza la conexión a la dirección IP de la placa y se envían los datos de conexión (puerto y contraseña). Seguido a esto, se le envía el comando de apagado a la placa, dependiendo de la opción seleccionada por el usuario.

```
public class SSHCliente extends AsyncTask<String, Integer, String> {
 public SSHCliente() {
 jsch = new JSch();
 try {
 JSch.setConfig("StrictHostKeyChecking", "no");
 sesion = jsch.getSession("pi","192.168.0.150",22);
 sesion.setPassword("raspberry");
 sesion.connect();
 consola = (ChannelExec) sesion.openChannel("exec");
 if (botonon==1){
        ((ChannelExec)consola).setCommand("sudo ledon.py");
        consola.connect();
 }
 if (botonoff==1) {
        ((ChannelExec)consola).setCommand("sudo ledoff.py");
        consola.connect();
 }

 } catch(JSchException e) {
 System.out.println("Error de JSCH. Mensaje: "+e.getMessage());

 }
 }
```

En la figura 12 se puede apreciar la aplicación en Android para controlar el brazo robótico. Se aprecia el menú principal, el cual tiene un botón que permite realizar la conexión con la placa Raspberry Pi. Después de realizada la conexión se tiene una segunda interfaz que muestra las dos opciones: modo individual o modo rutina; si el usuario selecciona modo individual, aparecerá la interfaz para digitar los grados y el motor que se desea mover; si la opción seleccionada es la modo rutina, aparecerá la gráfica 4, la cual permite seleccionar entre tres rutinas diferentes.

Conclusiones

El proyecto desarrollado permitió establecer la comunicación entre los sistemas embebidos Raspberry Pi y Arduino, por medio de comunicación serial. Las prime-

épsilon, nº 25 • julio-diciembre 2015 • pp. 69-96 • ISSN 1692-1259-ISSN-E 2389-8828

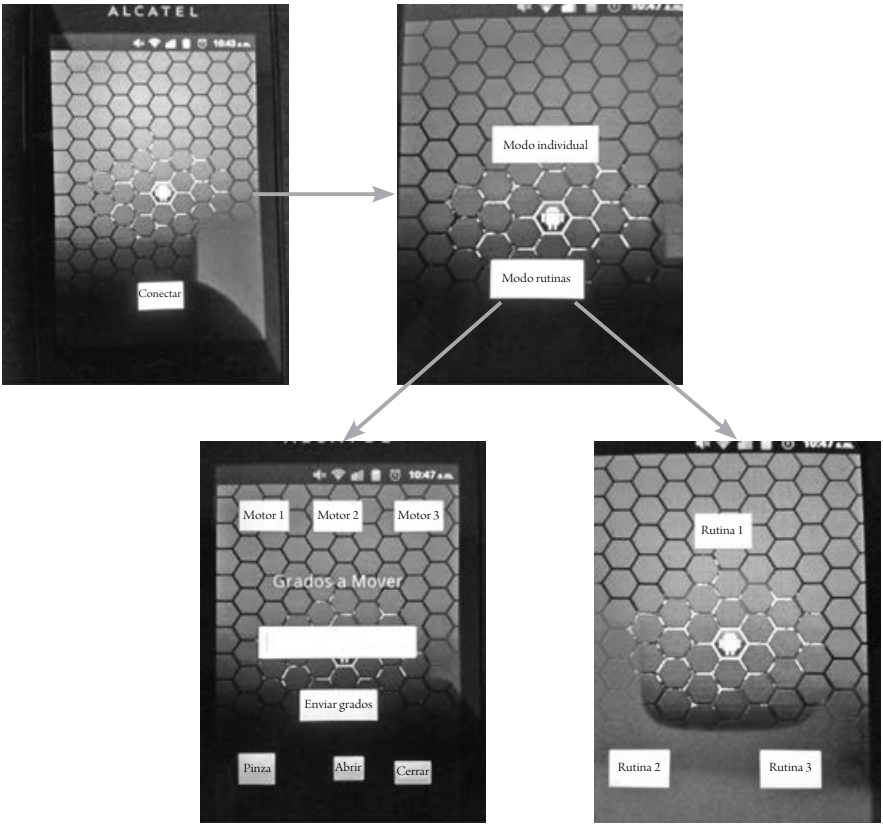

Figura 12. **Aplicación en Android**

Fuente: elaboración propia.

ras pruebas que se ejecutaron fueron el encendido y apagado de leds y el manejo de servomotores. De igual modo, se construyó un dispositivo inalámbrico para pruebas con una estructura en acrílico color negro, utilizando los sistemas embebidos elegidos (Arduino y Raspberry Pi) para acoplarlos a la aplicación móvil diseñada y de esta manera integrar el *hardware* y el *software*.

Una vez realizadas las pruebas de conexión serial entre Arduino y Raspberry Pi, lectura, escritura de datos y manejo del servomotor Herkulex drs-0101 entre los sistemas embebidos, se desarrollaron dos programas para el manejo del dispositivo para pruebas: el programa en Python se encarga de procesar la información dada por el usuario, organizarla y enviarla al Arduino, que recibe dicha información, la interpreta y la ejecuta. Lo anterior es útil en el desarrollo de proyectos en automatización.

Referencias

Al-Haija, Q., Al Tarayrah, M. Al-Qadeeb, H. y Al-Lwaimi, A. (2014). A tiny RSA cryptosystem based on Arduino microcontroller useful for small scale networks. *Procedia Computer Science, 34*, 2014, 639-646.

Arduino. (2015). *Sweep*. Recuperado de https://www.arduino.cc/en/pmwiki.php?n=Tutorial/Sweep

Dfrobot. (2015). *Images*. Recuperado de http://www.dfrobot.com/wiki/images/0/07/HerkuleX_0401_UNO.jpg

Dudas, R., VandenBussche, Ch., Baras, A., Ali, S. y Olson, M. (2014). Inexpensive telecytology solutions that use the Raspberry Pi and the iPhone. *Journal of the American Society of Cytopathology, 3*(1), 49-55.

Ferdoush, S. y Li, X. (2014). Wireless sensor network system design using Raspberry Pi and Arduino for environmental monitoring applications. *Procedia Computer Science, 34*, 103-110.

Jutila, M., Rivas, H., Karhula, P. y Pantsar-Syväniemi, S. (2014). Implementation of a wearable sensor vest for the safety and well-being of children. *Procedia Computer Science, 32*, 888-893.

Koenka, I., Sáiz, J. y Hauser, P. (2014). Instrumentino: An open-source modular Python framework for controlling Arduino based experimental instruments. *Computer Physics Communications, 185*(10), 2724-2729.

Pixabay. (2016). *Frambuesa Pi, RPI Microcontrolador*. Recuperado de https://pixabay.com/es/frambuesa-pi-rpi-microcontrolador-950490/

Siegle, J., Hale, G., Newman, J. y Voigts, J. (2015). Neural ensemble communities: open-source approaches to hardware for large-scale electrophysiology. *Current Opinion in Neurobiology, 32*, 53-59.