XRemoteBot: Un servicio para programar robots en forma remota

Tesina de grado de la Licenciatura en Informática

Fernando Esteban Mariano López

Licenciatura en Informática

2015

Directoras: Claudia Banchoff

Claudia Queiruga

Índice general

Índice de figuras

Índice de códigos

Prólogo

Esta tesina se enmarca en el proyecto "Programando con robots y software libre" del LINTI[1] y el desarrollo realizado es una contribución a dicho proyecto. Se implementó un servidor denominado XRemoteBot y tres clientes compatibles con el mismo escritos en tres lenguajes distintos. El motivo y las elecciones hechas en el desarrollo del servidor y los clientes serán relatados a lo largo de este trabajo.

A continuación se describe la forma en que se ha organizado este informe y en el CD adjunto se entrega tanto una copia del mismo como los fuentes desarrollados. Aunque cabe aclarar que todo el trabajo implementado ha sido desarrollado con software libre, será liberado bajo licencia MIT[2] y se puede descargar desde GitHub[3].

El capítulo 1 relata la motivación para el desarrollo de XRemoteBot.

El capítulo 2 presenta un relevamiento de desarrollos y proyectos similares.

El capítulo 3 describe el modo funcionamiento de las bibliotecas usadas hasta el momento en el proyecto "Programando con robots y software libre" para interactuar con los robots, estas bibliotecas son la base de XRemoteBot.

El capítulo 4 describe el modo de funcionamiento y tecnologías utilizadas en el servidor XRemoteBot desarrollado para esta tesina y de su antecesor RemoteBot.

El capítulo 5 describe el modo de uso, modo de funcionamiento y tecnologías utilizadas en los tres clientes desarrollados para XRemoteBot, especialmente el cliente Javascript que es el más complejo en su implementación.

El capítulo 6 describe el protocolo de capa de aplicación diseñado para el servidor

[1] `http://linti.unlp.edu.ar`

[2] `http://opensource.org/licenses/MIT`

[3] `https://github.com/fernandolopez/xremotebot` y `https://github.com/fernandolopez/xremotebot-clients`

desarrollado y las modalidades de operación del mismo.

El capítulo 7 describe el entorno en el que fue probado XRemoteBot, detallando algunas de pruebas realizadas y algunas modificaciones realizadas al proyecto en base a los resultados de estas pruebas.

El capítulo 8 contiene las conclusiones del trabajo y el trabajo a futuro.

Capítulo 1

Motivación del diseño y desarrollo de XRemoteBot

En este capítulo se describen los motivos que llevaron a implementar XRemoteBot como parte de esta tesina, como así también las decisiones iniciales de diseño y la decisión del lenguaje de programación en el cuál se implementará la mayor parte de la aplicación.

1.1. Motivación

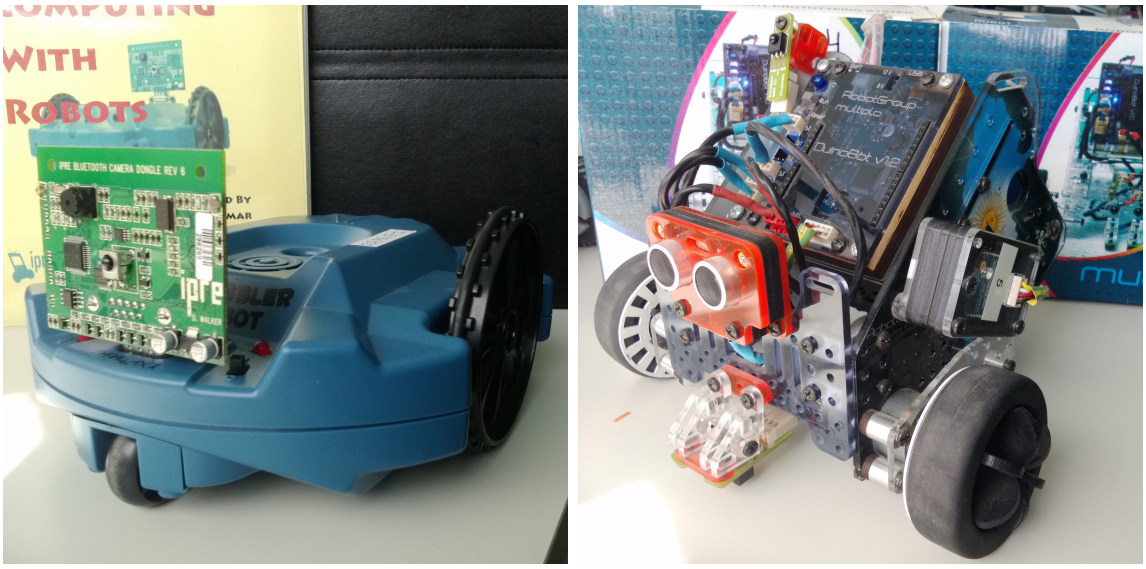
Desde el año 2009, en el marco del proyecto "Programando con robots y software libre" del LINTI[1], se utilizan robots para enseñar programación en Python a alumnos de escuelas secundarias. Estos robots son la herramienta didáctica que permite enseñar las estructuras básicas del lenguaje Python.

En un comienzo se utilizaron los robots Scribbler de Parallax, mostrados en la figura 1-1a. Estos robots pueden programarse directamente a través de un cable utilizando el lenguaje PBASIC[2] y funcionar de forma autónoma, o pueden ser controlados de forma inalámbrica incorporando un dispositivo extra y conectándose por Bluetooth.

En las experiencias realizadas se los utilizó de la última forma, y de esta manera

[1] `http://robots.linti.unlp.edu.ar`

[2] `http://homepages.ius.edu/RWISMAN/C490/html/PythonandRobots.htm` y `http://learn.parallax.com/teach/pbasic-programming-basic-stamp`



(a) Robot Scribbler de Parallax (b) Robot Multiplo N6 de RobotGroup

Figura 1-1: Robots usados en el proyecto

se controla el robot usando el lenguaje Python (cuyo intérprete no es posible ejecutar directamente sobre el microcontrolador del mismo).

Los robots Scribbler se importan desde Estados Unidos en volúmenes bajos por las restricciones a la importación en nuestro país, lo que dificulta su adquisición tanto por el equipo del proyecto como de posibles escuelas que quisieran comprarlos, dejando, además al equipo sin ninguna garantía ante posibles averías. Por estos motivos, se buscaron alternativas de fabricación nacional, que pudieran ser programados en lenguaje Python y tuvieran especificaciones abiertas. De esta manera, a fines del año 2011, se adquirieron dos robots Multiplo N6 de fabricación nacional de la empresa RobotGroup, mostrados en la figura 1-1b. Tanto el software necesario para controlar estos robots como sus especificaciones se distribuyen bajo una licencia abierta[3].

Los robots Multiplo N6 están basados en Arduino[4] por lo que pueden ser programados usando el IDE Arduino en C++. Además RobotGroup ofrece el entorno de programación MiniBloq que permite programar el robot de forma gráfica usando bloques. Dado que estas modalidades no se adecuaban a las propuestas del proyecto

[3] `https://www.kickstarter.com/projects/1689254125/multiplo-create-your-own-robot/description`

[4] `http://www.arduino.cc/`

se pidió a la empresa una versión modificada del robot para controlarlo de forma inalámbrica (ya que no es posible ejecutar un intérprete CPython en una placa de esas características) y se desarrolló una biblioteca Python que permite la programación del robot en dicho lenguaje. Este desarrollo fue realizado en forma conjunta entre el equipo del proyecto "Programando con robots y software libre" y miembros de la empresa RobotGroup, y actualmente está siendo mantenido por el LINTI [Díaz et al., 2012].

La biblioteca Python que permite controlar los robots Multiplo N6 se denomina DuinoBot. Se eligió este nombre por coincidir con el de la placa controladora de los robots, sin embargo en este informe DuinoBot hace referencia a la biblioteca Python desarrollada en conjunto entre el equipo del proyecto "Programando con robots y software libre" y la empresa RobotGroup.

En el año 2012, a través de un convenio de cooperación con la Fundación YPF, que brindó un subsidio, y bajo el auspicio de la Dirección de Escuelas Técnicas de la Provincia de Buenos Aires se realizó una capacitación en 10 escuelas técnicas de la provincia de Buenos Aires. La Fundación YPF brindó a cada una de estas escuelas, entre otras cosas, 20 robots Multiplo N6 que quedaron en cada escuela. En estos cursos se capacitaron más de 140 docentes y 40 alumnos avanzados, en programación en el lenguaje Python utilizando estos dispositivos [Díaz et al., 2012] como herramienta didáctica.

Esta experiencia consolidó el uso de los robots Multiplo N6 en el proyecto "Programando con robots y software libre", reemplazando a los robots Scribblers, tanto por su confiabilidad como por su facilidad de adquisición.

1.2. Los robots

Como se mencionó anteriormente, se utilizaron a lo largo del proyecto dos modelos de robots distintos: el Scribbler y el Multiplo N6. Esta sección provee una breve descripción de sus características técnicas.

1.2.1. Robot Scribbler de Parallax

Los robots Scribbler (versión 1) cuentan con un microcontrolador *PIC16F57*[5] que viene programado con un intérprete de Basic. El producto completo es denominado por Parallax "Basic Stamp 2"[6]. Para su locomoción este robot cuenta con dos ruedas conectadas a través de cajas reductoras a sus respectivos motores de DC (corriente continua) y una tercer rueda no conectada a ningún motor que sirve como tercer punto de apoyo para el cuerpo del robot.

La alimentación eléctrica del robot es provista por 6 pilas AA.

En cuanto a los sensores, este robot está equipado con:

- Dos emisores infrarrojos a los lados de su parte frontal y un sensor infrarrojo entre ellos que permite sensar (de acuerdo a si se refleja el haz de luz infrarroja de alguno de los laterales) si hay algún obstáculo a la derecha o a la izquierda del robot.

- Tres fotorresistores ubicados dentro de 3 cavidades al frente del robot que permiten sensar la presencia de fuentes de luz e identificar (si la luz no es muy difusa) si las mismas se encuentran directamente enfrente, a la izquierda o a la derecha del robot.

- Dos sensores de línea que consisten cada uno de un emisor y un sensor infrarrojo (IR). Los mismos detectan cambios de contraste en el piso de acuerdo a la cantidad de luz IR reflejada y generalmente se utilizan para que el robot siga un camino demarcado por una línea de un color sobre una superficie de un color contrastante.

Si bien las anteriores son las características generales de estos robots, usándolos de esta manera es necesario programarlos a través de un cable serial usando el lenguaje PBASIC basado en BASIC. Para las actividades planteadas en el marco del proyecto "Programando con robots y software libre", se utilizó una versión que agrega un

[5]Esta información se obtuvo con una inspección visual de uno de estos robots.

[6]`http://www.andybrain.com/extras/scribbler-robot-review.htm`

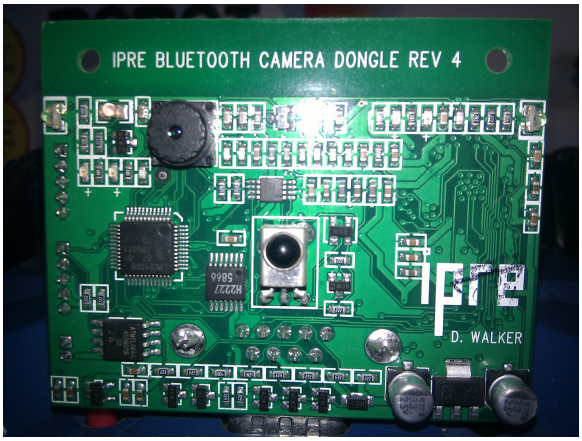


Figura 1-2: IPRE Fluke

dispositivo extra: el IPRE Fluke 1-2. Este dispositivo se conecta al puerto serial del robot y provee la capacidad de controlarlo de forma inalámbrica usando Bluetooth. Conectar el IPRE Fluke invierte el sentido de giro de las ruedas, por lo que el frente del robot pasa a ser el lugar donde se conecta el IPRE Fluke. De aquí en adelante cuando se mencione la parte delantera del Scribbler, será la parte donde se encuentra conectado el IPRE Fluke y la parte trasera será la opuesta.

El IPRE Fluke es comandado por con un microcontrolador *LPC2106F*[7] basado en la arquitectura *ARM7*[8]. Posee un módulo de comunicaciones Bluetooth que utiliza como medio de comunicación con la computadora controlante y provee una cámara de baja resolución con la que se pueden tomar fotografías, que luego serán transmitidas por Bluetooth. Además agrega un sensor infrarrojo que permite detectar la presencia de obstáculos a la izquierda, al centro o a la derecha (está compuesto de 3 emisores y un receptor)[9]. Con esta incorporación el robot queda equipado ahora con sensores en la parte frontal y en la parte trasera, pero más importante aún el módulo Bluetooth permite controlarlo de forma inalámbrica siempre que se grabe un firmware especial en la placa IPRE Fluke. Esto es lo que permite que estos robots puedan ser programados con Python usando una biblioteca denominada *Myro*[10], que será descripta en la sección 3.1.

[7]Información obtenida con una inspección visual de la placa.
[8]`http://pdf1.alldatasheet.com/datasheet-pdf/view/83950/PHILIPS/LPC2106FHN48.html`
[9]`http://wiki.roboteducation.org/Myro_Hardware`
[10]`http://wiki.roboteducation.org/Myro_Reference_Manual`

1.2.2. Multiplo N6

El robot Multiplo N6, fabricado por la empresa RobotGroup, cuenta con un microcontrolador Atmel *AVR ATMega32U4*[11] con el bootloader de Arduino precargado. Para su locomoción cuenta con dos ruedas conectadas a través de reductores de velocidad a sus correspondientes motores de corriente continua y una tercer rueda no conectada a ningún motor que sirve como tercer punto de apoyo. La alimentación eléctrica de estos robots está dada por tres pilas AA en serie (o bien dos packs de tres pilas cada uno en paralelo).

En cuanto a los sensores el Multiplo N6 está equipado con:

- Un sensor ultrasónico para detectar obstáculos al frente del robot.
- Un sensor infrarrojo que puede ser utilizado en conjunto con un control remoto universal para enviar comandos al robot (siempre que se programe este comportamiento previamente).
- Dos sensores detectores de línea, compuestos de un emisor y un receptor infrarrojo cada uno. Opcionalmente estos sensores pueden desatornillarse y ensamblarse para funcionar como "encoders"[12] de las ruedas y de esta manera es posible calcular la velocidad de giro de las ruedas. Este uso no es recomendable si se controla el robot de forma remota ya que la latencia de las comunicaciones puede hacer que se pierdan datos haciendo que el cálculo de la velocidad sea impreciso.

En esta configuración estos robots son programables usando el entorno de Arduino y desde MiniBloq[13], pero dado que el objetivo del proyecto es la enseñanza de programación en el lenguaje Python se encargó a la empresa una versión modificada del Multiplo N6 que permitiera controlarlo de forma inalámbrica, de manera tal que una computadora con un intérprete de Python instalado pudiera enviar instrucciones y recibir los valores de los sensores del robot tal como hace la biblioteca Myro con

[11] `http://www.robotgroup.com.ar/index.php/productos/131-robot-n6#especificaciones`
[12] `http://machinedesign.com/sensors/basics-rotary-encoders-overview-and-new-technologies-0`
[13] `http://minibloq.org`

los robots Scribbler. Esta modificación consistió en la adición de un módulo de comunicación inalámbrica XBee que permite enviar y recibir datos del robot a través del protocolo ZigBee[14], la modificación de un firmware basado en una implementación existente para Arduino del protocolo de control `Firmata`[15] y el desarrollo del módulo Python `DuinoBot` para controlar a los robots usando este protocolo, el cuál fue adaptado por los miembros del proyecto "Programando con robots y software libre" para soportar nuevas funcionalidades y proveer una API similar a la de la biblioteca `Myro`[Lanfranco et al., 2012].

Básicamente los robots Multiplo N6 utilizados en el proyecto "Programando con robots y software libre" contienen un firmware basado en Firmata, modificado para soportar las características propias del robot. Este firmware recibe instrucciones desde un un puerto serie conectado a una placa XBee y envía las respuestas a la misma utilizando una variación del protocolo Firmata. Desde el lado de la computadora controlante, el módulo DuinoBot provee una interfaz de alto nivel para los alumnos, donde cada instrucción para el robot programada en Python se convierte a una secuencia de bytes del protocolo Firmata y se envía al robot a través de una placa XBee conectada por USB a la computadora, y cada respuesta del robot se almacena para ser utilizada como valor de respuesta en los métodos asociados con los sensores.

1.3. Características comunes

Aunque se han utilizado a la fecha estos dos modelos distintos de robots, en esencia, las características principales de ambos modelos son las mismas y coinciden con las de otros robots usados en la Argentina para enseñar a programar. En general, todos ellos incluyen:

- Un medio de locomoción con 2 motores continuos que mueven cada uno una de las ruedas laterales.
- Sensores que permiten detectar obstáculos.

[14] `http://www.digi.com/xbee/`
[15] `http://firmata.org/`

- Sensores que permiten detectar líneas.

- Operación inalámbrica.

Cabe destacar sobre este último ítem que otros robots no requieren cables para operar ya que son programados a través de algún lenguaje compilado como C, Assembler o Basic, donde se les transfiere el programa a través de un cable (usualmente USB o Serial) y este programa se graba en la memoria del microcontrolador permitiendo que el robot posteriormente ejecute el programa de forma autónoma. Pero los robots usados en el proyecto antes mencionado no requieren cables ya que son controlados a través de señales inalámbricas, lo que permite controlarlos en tiempo real y utilizando un intérprete Python estándar (CPython) instalado en el dispositivo controlante.

Dado el costo y fragilidad de los robots habitualmente los alumnos interactúan con los mismos solamente en el aula, ya sea en su escuela si la misma adquirió los robots, o en las instalaciones de la Facultad de Informática si el alumno realiza alguna pasantía o práctica en la misma. Esto tiene varias connotaciones:

- El alumno, al estar en una situación formal en la escuela compartiendo el robot (recurso limitado) con otros alumnos, posiblemente no tenga el tiempo o el ambiente más apropiado para experimentar de forma lúdica con el mismo.

- La tarea para casa solamente es realizable a través de un simulador que puede ser lo suficientemente completo y fiel como para aprender a programar, pero resulta menos estimulante y realista que manipular un robot real.

- Alumnos de escuelas que no tienen los recursos necesarios para adquirir el equipamiento o de escuelas alejadas de la ciudad de La Plata, que no tienen la posibilidad de acercarse a nuestra Facultad, no pueden interactuar con robots reales (a lo sumo podrán usar un simulador).

Por otro lado, las placas XBee que permiten conectarse con los robots Multiplo N6 son relativamente costosas. El esquema normal de conexionado entre dispositivos controladores y robots descripto en el capítulo 3 requiere 2 placas XBee por robot,

una conectada directamente al robot y la otra conectada por USB al dispositivo controlante. En consecuencia:

- El dispositivo controlante debe tener un puerto USB y los drivers necesarios para detectar la interfaz serial con el XBee, esto deja fuera de juego celulares y tablets.

- El costo de operar cada robot en este esquema es sensiblemente superior al costo que tendría si varios alumnos pudieran controlar varios robots usando un solo dispositivo XBee compartido entre varios dispositivos controlantes.

Tomando estos dos problemas se propuso implementar una solución que permita superarlos de forma simultánea. Esta solución permite controlar los robots a través de una red, como puede ser Internet, permitiendo a los alumnos conectarse a los robots desde sus hogares y a las instituciones a compartir el uso de sus robots. Por otro lado también es posible configurar el servidor para su uso en el aula deshabilitando funcionalidades innecesarias en un ámbito local, como ser la necesidad de autenticar a los usuarios para utilizar los robots. Este modo de operación fue pensado para que múltiples alumnos puedan acceder a múltiples robots contando con un solo adaptador USB a XBee. Esto reduciría sensiblemente el costo de cada robot por alumno ya que de otra manera por cada robot deberían usarse 2 dispositivos XBee (uno en el robot y otro en la computadora del alumno) reduciéndose con este modo de operación a un XBee por robot más un único adaptador USB a XBee conectado al servidor.

Como consecuencia de estos requerimientos el servidor debe tener una interfaz web y proveer acceso concurrente, con relativa baja latencia, para permitir a múltiples alumnos acceder a múltiples robots al mismo tiempo desde distintos dispositivos. Si bien el objetivo del proyecto "Programando con robots y software libre" es enseñar programación usando el lenguaje Python, por lo que sería suficiente que XRemoteBot tenga un cliente Python, tener un servidor web basado en protocolos estándares hace que sea relativamente sencillo desarrollar clientes en cualquier lenguaje que soporte estos protocolos. Esto dio lugar a la posibilidad de implementar otros clientes que permitieran usar los robots para enseñar programación en distintos lenguajes sin re-

implementar el protocolo de bajo nivel. XRemoteBot se diseñó de forma tal que la creación de nuevos clientes en distintos lenguajes resulte sencilla. En especial, la elección de protocolos y tecnologías usadas y la falta de necesidad de acceder directamente al hardware a través de USB, habilita la implementación de un cliente Javascript que se ejecute en el navegador web de los usuarios. Esta implementación, junto con las desarrolladas en Ruby y Python, se describen en el capítulo 5, y permite trabajar con el robot desde un navegador web.

El nivel de abstracción que puede proveer esta API hace que sea una consecuencia natural de este diseño pensar en la posibilidad de manejar distintos tipos de robots que tengan algunas características mínimas en común con los robots ya descriptos utilizando la misma interfaz de programación. Por lo que teniendo ciertos cuidados en la implementación del protocolo y el servidor se puede lograr que ambos sean fácilmente extensibles para agregar soporte para nuevos robots.

En cuanto al lenguaje de programación utilizado en el desarrollo del servidor, dado que las bibliotecas utilizadas para controlar los robots Multiplo N6 y Scribbler se encuentran desarrolladas en Python y dada la experiencia del autor tanto en el uso de estas bibliotecas como en el uso del lenguaje Python en general, éste es el lenguaje elegido para el desarrollo del servidor. En particular la versión 2.7 de Python ya que utilizar la versión 3 requeriría extensivas modificaciones a estas bibliotecas y otras relacionadas [Lutz, 2014].

Capítulo 2

Controlar dispositivos de forma remota

Existen diferentes alternativas para controlar robots o microcontroladores de forma remota. Muchas de éstas se agrupan bajo la denominación "Internet of Things"[1]. En este capítulo se describirán algunas iniciativas similares a XRemoteBot.

2.1. Educabot

El proyecto Educabot es una iniciativa de Diego Ramírez con el apoyo de la Fundación Nahual y la organización El Galpón del Banquito[2]. Educabot tiene por objetivo enseñar tecnología a niños y adultos a través del uso, programación y construcción de robots. En el sitio del proyecto se ofertan cursos orientados a los distintos niveles y en la última conferencia de Python de Argentina (PyConAr 2014) uno de los cofundadores del proyecto mencionó que se llevan adelante actividades con los robots en distintas escuelas de la Ciudad Autónoma de Buenos Aires[3].

En la parte de construcción de robots, este proyecto plantea un modelo de robot denominado "Rolo" orientado a niños de más de 10 años, mientras que para los más chicos se plantean actividades con el robot "elBrian" que incluyen controlarlo a través

[1] `https://www.iotwf.com/resources/1`
[2] `http://minimalart.org/blog/presentando-educabot`
[3] `https://youtu.be/1oCOAtX9OS4`

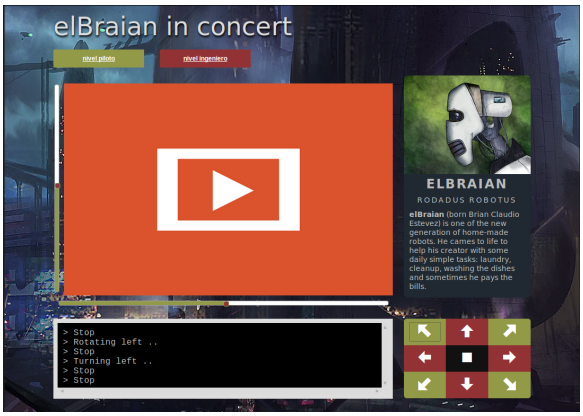


Figura 2-1: Pantalla principal de elBrian (en el recuadro rojo normalmente se muestra la cámara de el robot)

de una interfaz web que muestra las imágenes emitidas por la cámara incorporada en este robot y además permite controlarlo con botones en pantalla que determinan en qué dirección debe moverse el robot (figura 2-1).

Las tecnologías utilizadas en la interfaz web de "elBrian" coinciden en gran parte con las utilizadas en el desarrollo de XRemoteBot, pero el objetivo del servidor de "elBrian" es controlar un único robot en un ambiente local. Además este servidor se instala en el robot utilizado, cuestión que sería imposible en los robots basados en microcontroladores AVR y PIC ("Basic Stamp 2" de Parallax) a los que XRemoteBot se encuentra dirigido.

El servidor web de "elBrian" está implementado usando el framework Tornado[4], WebSockets[5], mjpg-streamer[6], OpenCV[7] y JSON[8]. Este servidor está diseñado para ejecutarse en el robot ya que el mismo se basa en una placa Raspberry Pi, la cuál cuenta con un procesador ARM perfectamente capaz de ejecutar un sistema operativo completo como GNU/Linux y dar soporte al intérprete oficial de Python.

Mientras que este servidor coincide en gran medida en la elección de lenguaje y bibliotecas utilizadas, su implementación es específica para el robot "elBrian" y no puede ser portada a robots con menores capacidades de procesamiento sin una rees-

[4] `http://www.tornadoweb.org/`
[5] `http://www.websocket.org/`
[6] `https://github.com/jacksonliam/mjpg-streamer`
[7] `http://opencv.org/`
[8] `http://json.org/`

critura significativa. Además el protocolo utilizado no contempla el acceso a valores de sensores siendo los únicos mensajes que permite enviar al robot los comandos de movimientos.

Referencias del proyecto:

- Página del proyecto: `http://www.educabot.org/`
- Código fuente de "elBrian": `https://github.com/educabot/elBraian`

2.2. Gobot con cppp-io

Gobot es una biblioteca que permite controlar robots programando en el lenguaje Go[9]. Esta biblioteca soporta el protocolo Firmata para controlar robots conectados directamente a través de una interfaz serial, como es el caso de los robots Multiplo N6, y soporta la API cppp-io[10] que define una API JSON que permite el acceso a la información y control de robots a través de la Web.

El proyecto Gobot surge como una iniciativa de la compañía "The Hybrid Group". El objetivo de este proyecto es facilitar la programación de dispositivos como las placas Arduino y los robots Sphero a personas que no sean expertas en la programación de microcontroladores[11].

Gobot tiene compatibilidad con distintos sensores y robots, además de ser compatible con placas utilizadas normalmente en la construcción de robots como pueden ser Arduino, Raspberry Pi, Intel Edison y Beaglebone Black[12].

Este proyecto es interesante como base para desarrollar un proyecto similar a XRemoteBot en Go, pero requiere, además la reimplementación del módulo de Python DuinoBot que controla, a través de una versión modificada del protocolo Firmata, a los robots Multiplo N6. Por otro lado, los robots Scribbler tampoco aparecen en la lista de robots soportados.

Referencias del proyecto:

[9] `https://golang.org/`
[10] `https://github.com/hybridgroup/cppp-io/`
[11] `http://www.wired.com/2015/04/internet-anything-time-everyone-able-hack-robots/`
[12] `http://gobot.io/#platforms`

- Página del proyecto: `http://gobot.io`.
- Protocolo de su API web: `https://github.com/hybridgroup/cppp-io/`.

2.3. Cylon.js

Cylon.js es un proyecto hermano de Gobot, también es desarrollado por "The Hybrid Group", y soporta la misma variedad de dispositivos y también provee una API cppp-io, con la diferencia de que la biblioteca provista está escrita en Javascript para "Node.js"[13].

Los objetivos de Cylon.js son los mismos que los del proyecto Gobot, pero trasladados al lenguaje de programación Javascript, en lugar del lenguaje de programación Go.

Una característica interesante de Cylon.js es que soporta distintos protocolos para su API remota como ser HTTP[14], socket.io[15] y la capacidad de agregar nuevos protocolos con plugins.

```
1  "use strict";
2
3  var Cylon = require("cylon");
4
5  // ensure you install the API plugin first:
6  // $ npm install cylon-api-http
7  Cylon.api({
8    host: "0.0.0.0",
9    port: "8080"
10 });
11
12 var bots = {
13   "Thelma": "/dev/rfcomm0",
14   "Louise": "/dev/rfcomm1"
15 };
```

[13] `https://nodejs.org/`
[14] `http://tools.ietf.org/html/rfc2616`
[15] `http://socket.io/`

```
16
17 Object.keys(bots).forEach(function(name) {
18   var port = bots[name];
19
20   Cylon.robot({
21     name: name,
22
23     connections: {
24       sphero: { adaptor: "sphero", port: port }
25     },
26
27     devices: {
28       sphero: { driver: "sphero" }
29     },
30
31     work: function(my) {
32       every((1).seconds(), function() {
33         console.log(my.name);
34         my.sphero.setRandomColor();
35         my.sphero.roll(60, Math.floor(Math.random() * 360));
36       });
37     }
38   });
39 });
40
41 Cylon.start();
```

Código 2.1: Cylon.js controlando 2 robots "sphero" usando HTTP (extraído del sitio oficial del proyecto)

En el código 2.1 se puede ver un ejemplo de Cylon.js del lado del servidor exponiendo dos robots de tipo *sphero* a través de la API HTTP. Los robots sphero son pequeñas esferas controladas por Bluetooth que pueden desplazarse rotando o cambiar de color gracias a leds que iluminan la superficie del robot desde el interior del mismo[16].

[16] `http://www.gosphero.com`

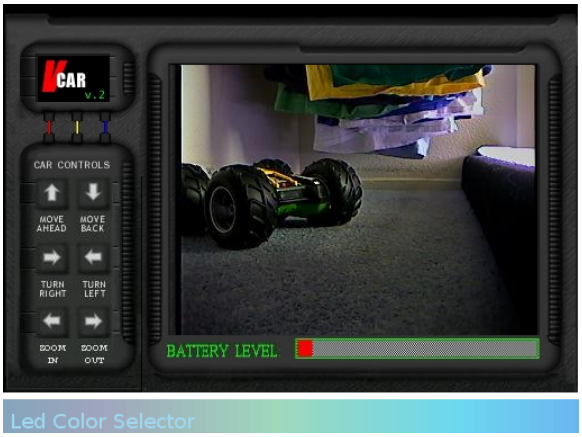


Figura 2-2: Interfaz de VCar

Una vez ejecutado el código 2.1 en el servidor, a través de la API, se les podrá enviar un mensaje a los robots que hará que cambien de color y se desplacen rodando.

Si bien en la documentación del proyecto se detalla como habilitar un sistema de login para acceder a la API de forma remota, el mismo no parece estar diseñado para el acceso concurrente de distintos usuarios ya que de la documentación no surge que Cylon.js soporte un sistema de reserva de robots o de exclusión mutua en el acceso a los mismos.

Referencias del proyecto:

- Página del proyecto: `http://cylonjs.com/`.

2.4. VCar

VCar es un auto a control remoto, cuyo control fue modificado para se manipulado por software desde una computadora con GNU/Linux la cuál permite controlar el robot de forma remota y verlo a través de una cámara de video (figura 2-2).

Este proyecto fue creado por David Mc Anulty con el objetivo de aprender a "traducir código en movimientos reales" (el objetivo no se encuentra expresado de forma explícita en el sitio web de VCar, pero se contactó al autor del proyecto que proporcionó esta información). Originalmente este proyecto consistía de un motor paso a paso que permitía mover una cámara de video de forma remota, eventualmente

ante el éxito del proyecto entre sus amigos, el desarrollador comenzó a investigar distintas formas de interactuar con la cámara (actualmente se puede controlar el zoom de la cámara desde la interfaz de VCar) y finalmente agregó un auto a control remoto similar al que se puede ver hoy en día en el sitio del proyecto.

La interfaz de VCar posee una botonera que permite mover el robot, controlar el zoom de la cámara de video y controlar el color de los LEDs que iluminan al robot. El acceso al robot es público, sin ningún tipo de restricción y la página si bien no tiene detalles de todo el hardware y el software utilizados provee algunas fotos de la construcción del sistema[17].

Referencias del proyecto:

- Página del proyecto: `http://www.hellspark.com/new/css/vcar/vcar.html`.
- Galería de fotos: `http://www.hellspark.com/dm/gallery2/v/projects/robotics/vcar/`.

2.5. Tele Toyland

Tele Toyland es un sitio que provee acceso a varios dispositivos a través de una interfaz web. Este proyecto es una iniciativa personal de Carl Sutter que surgió en 2006 basado en experiencias anteriores, en las que Sutter participó, como el primer robot conectado a la web "Mercury Project" y "TeleGarden" [Goldberg et al., 2000]. El objetivo del proyecto es mostrar una serie de dispositivos creados por el autor del sitio con el fin de inspirar a la gente (especialmente a chicos) a realizar proyectos similares (el objetivo no se encuentra expresado de forma explícita en el sitio web de Tele Toyland, pero se contactó al autor del proyecto que proporcionó esta información).

Desde la interfaz de Tele Toyland, es posible controlar un cabezal con una punta que dibuja sobre una caja de arena. Basta con hacer clic sobre las posiciones sobre las cuales se quiere que pase la punta y presionar el botón "go" para que el cabezal empiece a moverse dibujando lo pedido. En éste y el resto de los experimentos

[17] `http://www.hellspark.com/dm/gallery2/v/projects/robotics/vcar/`

disponibles en el sitio, los resultados se pueden ver a través de un streaming de video.

Entre los proyectos con los que permite interactuar Tele Toyland se encuentran 2 areneros como el descripto anteriormente, una torre de leds, una marioneta y laberintos.

El sitio no provee detalles del software, ni el protocolo utilizado. Desde lo funcional provee una especie de control remoto para los distintos dispositivos a los que permite manipular, en donde se puede, incluso, configurar una serie de instrucciones a ejecutar en secuencia. Sin embargo no provee una biblioteca que permita controlar los dispositivos desde un lenguaje de programación.

Referencias del proyecto:

- Página del proyecto: `http://www.teletoyland.com`.

2.6. Algunas reflexiones

Del relevamiento realizado se puede observar que existen diferentes proyectos que permiten controlar un robot u otro dispositivo de manera remota. Sin embargo ninguno de éstos es una solución completa al problema de enseñar a programar usando robots de forma remota y permitiendo a distintos usuarios reservar temporalmente robots de forma que no haya conflicto en la interacción entre los usuarios.

Finalmente en la consigna de "hágalo usted mismo" existen diversas guías para programar servidores que permitan controlar robots o microcontroladores en general, se puede encontrar un caso muy bien explicado en el sitio de Adafruit[18], este es un buen ejercicio de programación, sobre todo para aprender a programar servidores que provean una API web y clientes que la consuman. Sin embargo, estas guías son introductorias y el objetivo es crear un servidor muy simple, similar a lo que fue el servidor RemoteBot que se describe en el la sección 4.1, pensados para ser usados en un ambiente local ya que no proveen autenticación en general.

XRemoteBot aportará una solución novedosa al problema planteado, facilitando

[18] `https://learn.adafruit.com/wifi-controlled-mobile-robot/building-the-web-interface`

la programación de robots de forma remota. Además XRemoteBot es extensible, posibilitando, el soporte de nuevos robots y de otros dispositivos.

XRemoteBot es un proyecto de software libre que contribuirá como herramienta didáctica a la incorporación de la enseñanza de la programación en la escuela.

Capítulo 3

Myro y DuinoBot: Las bases de la propuesta

XRemoteBot tiene una arquitectura compleja que depende de otras bibliotecas y de software instalado en los robots. El objetivo de este capítulo es describir la interacción entre los distintos dispositivos y aplicaciones utilizadas, demarcando el límite entre el desarrollo realizado para esta tesina con el software ya existente con el que interactúa.

3.1. Myro

Como se mencionó al comienzo de este informe, Myro[1] es la biblioteca utilizada para controlar a los robots "Scribbler" desde un script Python. Esta biblioteca fue desarrollada por Institute for Personal Robots in Education (IPRE) para su uso en cursos de computación introductorios [Kumar, 2009] en distintos niveles educativos[2]. Si bien el proyecto original fue desarrollado por Microsoft Research, el software generado es software libre y portable tanto en sistemas Windows como GNU/Linux.

Myro puede interactuar con los robots de dos formas diferentes: a través de un cable serial (RS323) o bien usando Bluetooth con RFCOMM[3]. El funcionamiento de

[1] `http://myro.roboteducation.org/`
[2] `http://www.roboteducation.org/schools.html`
[3] `https://developer.bluetooth.org/TechnologyOverview/Pages/RFCOMM.aspx`

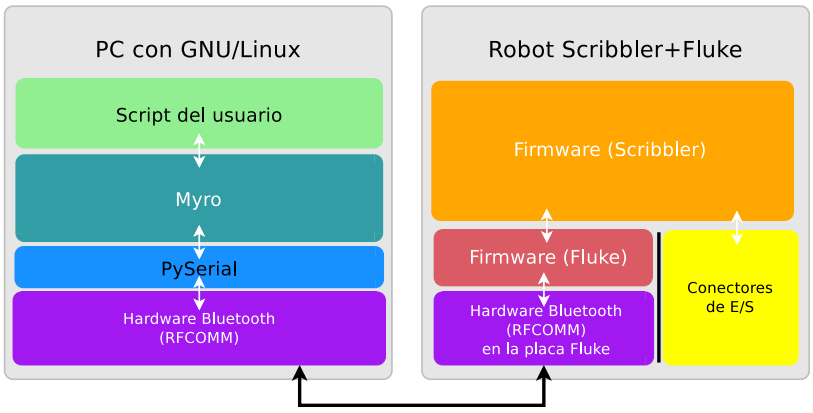


Figura 3-1: Diagrama de bloques de Myro

Myro requiere que el robot tenga cargado un firmware determinado que espera las instrucciones desde el puerto serial con el que cuenta el robot, y una placa llamada IPRE Fluke que se conecta al puerto serial del robot y cuenta con un adaptador Bluetooth que permite enviarle comandos de forma inalámbrica a través de este protocolo.

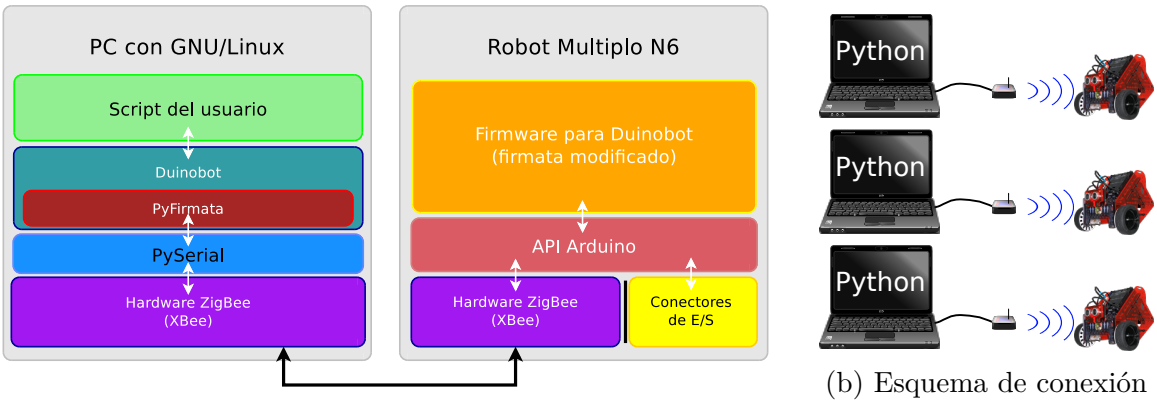
La figura 3-1 muestra la interacción entre los componentes más importantes de software y hardware desde el script del usuario hasta el robot.

Cabe destacar que el código de Myro funciona con Python 2.7 pero no con Python 3. Posiblemente esto se relacione con la decisión del "Institute for Personal Robots in Education" de desarrollar una nueva biblioteca para controlar a los robots que funcione en distintos lenguajes de programación[4]. De todas formas se decidió seguir con Myro por la experiencia del autor de esta tesina con esta biblioteca.

Lamentablemente el servidor SVN oficial donde se publicaban las actualizaciones de Myro a la fecha de escribir este documento no se encuentra en funcionamiento, por lo que se utilizó una copia de este repositorio en GitHub[5] de la cuál se creó un fork y la reorganizó de forma tal que fuera posible instalar esta biblioteca desde un entorno Python con un solo comando. Además se hizo una pequeña corrección para que el uso del paquete IDLE[6] sea opcional ya esa biblioteca sirve para crear interfaces gráficas y eso no es necesario para este proyecto.

Esta biblioteca (técnicamente un paquete Python) consta de múltiples módulos

[4] `http://calicoproject.org/`
[5] `https://github.com/yingted/Myro`
[6] `https://docs.python.org/2/library/idle.html`



(a) Diagrama de bloques de DuinoBot

(b) Esquema de conexión típico

Figura 3-2: Funcionamiento de DuinoBot

y requiere de otros para contar con funcionalidades opcionales como el soporte de joysticks, una interfaz gráfica para mostrar los valores de los sensores en un diálogo y soporte para manipulación de imágenes para poder sacar fotos con el robot y manipularlas. Sin embargo para el alcance de este trabajo con el acceso a las funcionalidades básicas del robot como moverlo y acceder a sus sensores es suficiente.

3.2. DuinoBot

La biblioteca de Python DuinoBot[7] desarrollada en conjunto entre un grupo de becarios del laboratorio LINTI de esta Facultad y la empresa RobotGroup cuenta con un modo de operación similar al de Myro, conectándose con los robots a través de una placa XBee (que utiliza el protocolo ZigBee) conectado a través de un puerto USB con el dispositivo controlante. Para que esta biblioteca pueda controlar a los robots, estos últimos deberán tener instalado también un firmware especial: en este caso, es un una versión modificada de un firmware que implementa el protocolo Firmata de uso amplio para controlar dispositivos Arduino a través de una interfaz serial que habitualmente es el puerto USB. Pero en este caso la interfaz serial está conectada a una placa XBee en el robot, la interacción entre las distintas capas de software y hardware se ilustra en la figura 3-2a.

[7] `https://github.com/Robots-Linti/duinobot`

La modalidad de uso esperada con DuinoBot consiste en que cada usuario debe contar con una computadora, conectada por USB a una placa XBee, y desde esa computadora puede controlar uno o más robots. Con esa modalidad fueron utilizados los robots en los cursos del proyecto "Programando con robots y software libre" y en actividades similares planteadas posteriormente, generalmente además cada alumno controla un único robot, el esquema de conexión típico en estos cursos se encuentra ilustrado en la figura 3-2b.

Como material para los cursos dictados como parte del proyecto "Programando con robots y software libre" se escribió un libro de programación con robots que describe por completo la API de DuinoBot, además de un conjunto de prácticas y diapositivas que pueden ser reutilizados para dictar cursos similares ya que ese encuentran publicados con una licencia Creative Commons y disponibles en el sitio del proyecto[8].

[8] `http://robots.linti.unlp.edu.ar/material_disponible`

Capítulo 4

XRemoteBot: eXtended RemoteBot

Como parte de esta tesina se desarrolló XRemoteBot, cuyo objetivo es permitir la enseñanza de programación en distintos lenguajes usando robots de forma remota. XRemoteBot está compuesto por un servidor escrito en el lenguaje de programación Python y tres clientes escritos en distintos lenguajes de programación.

XRemoteBot es un servidor web que provee una API JSON para interactuar con robots didácticos de forma remota y compartiendo un único enlace físico con los robots. Está implementado en Python usando el framework web Tornado, SQLAlchemy para acceder a la base de datos y los módulos DuinoBot y Myro para la comunicación con los robots.

Además del servidor se desarrollaron tres clientes para el mismo en distintos lenguajes: Javascript, Python y Ruby. El cliente Javascript de XRemoteBot se encuentra integrado con el servidor y la intención es que sea utilizado desde una vista web provista por el mismo. Los clientes Python y Ruby son independientes y la intención es que sean utilizados para crear programas que se ejecuten desde fuera del navegador para enseñar a programar en estos lenguajes usando un entorno de programación tradicional.

XRemoteBot es un rediseño y una reescritura completa de RemoteBot, un servidor simple realizado como aplicación auxiliar de una aplicación Android desarrollada como

trabajo final para la materia Laboratorio de Software.

4.1. RemoteBot

La versión original de RemoteBot fue presentada como trabajo final de la materia Laboratorio de Software[1], de esta Facultad. El objetivo principal del trabajo era desarrollar una aplicación para dispositivos Android que accediera a sensores y posiblemente a la red wireless usando el lenguaje Java [Queiruga et al., 2013].

Para este trabajo se implementó RemoteBot4Android, una aplicación Android que permite controlar robots usando los acelerómetros de los dispositivos móviles donde se ejecuta.

Una vez arrancado y configurado RemoteBot4Android se puede mover el robot seleccionado inclinando el dispositivo en la dirección deseada o bien detenerlo poniendo el dispositivo en posición horizontal. También es posible controlar el robot sin utilizar los acelerómetros con una botonera provista por la aplicación, controlar la velocidad con un barra deslizante y ver la distancia al obstáculo más cercano desde la interfaz.

El producto principal del trabajo de la cátedra de Laboratorio de Software fue el cliente RemoteBot4Android (se puede ver una captura de pantalla de éste en la figura 4-3), pero además del cliente, se desarrolló un protocolo de comunicaciones y un servidor básico que atiende los requerimientos y se los envía al robot correspondiente, respondiendo también a peticiones de datos de los sensores del robot. Esta fue la primer implementación del servidor RemoteBot en cuyo concepto está basado XRemoteBot.

La arquitectura del servidor RemoteBot se muestra en la figura 4-1, su función principal es la de intermediario entre el cliente original implementado en JAVA y el módulo DuinoBot. RemoteBot no es configurable, no cuenta con ninguna forma de visualizar los robots de forma remota, no permite autenticación, no dispone de un sistema de reserva de robots por lo que un cliente puede interferir en la operación de

[1] `http://wiki.labmovil.linti.unlp.edu.ar/index.php?title=RemoteBot:_Android_%2B_Robots`

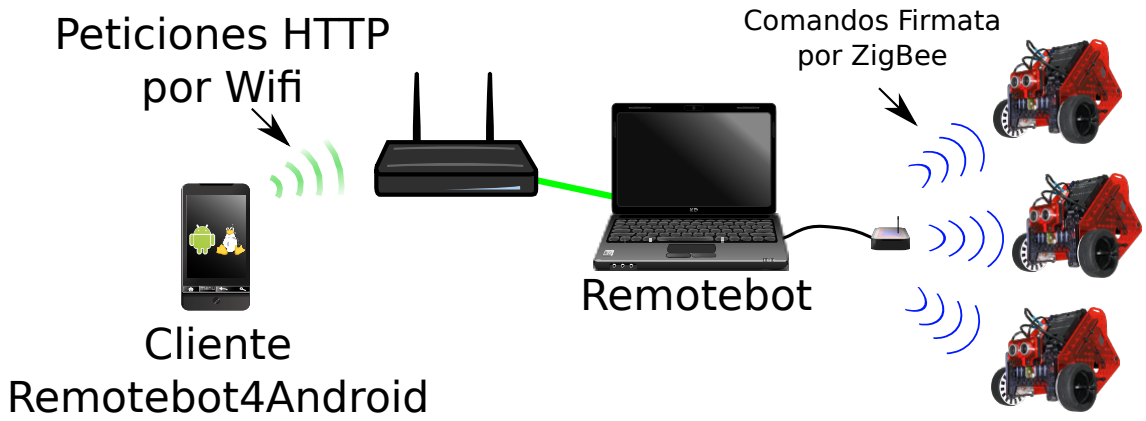


Figura 4-1: Esquema de conexión de RemoteBot

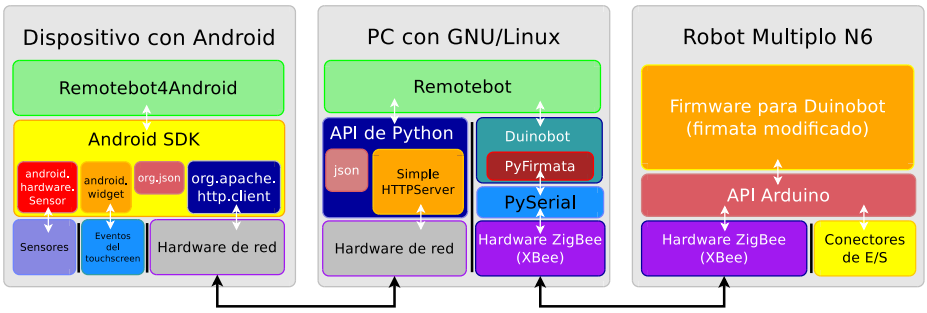


Figura 4-2: Diagrama de bloques de RemoteBot

un robot de otro cliente y las operaciones bloqueantes de un cliente impiden el uso de los robots al resto de los clientes hasta que esa operación termine.

RemoteBot provee una API JSON a través de HTTP que permite controlar a los robots de forma remota. Pero si bien es posible poner el servicio público, el mismo no cuenta con autenticación, no provee un soporte robusto de concurrencia y no provee ningún mecanismo de seguridad por lo que es recomendable solamente usarlo dentro de ámbito de una red LAN. La implementación del servidor usa el módulo "SimpleHTTPServer" de la biblioteca estándar de Python para proveer el soporte del protocolo HTTP e introspección[2] para convertir los mensajes enviados por los clientes en invocaciones a métodos.

Si bien RemoteBot fue creado para ser utilizado desde la aplicación Remote-Bot4Android se diseñó de forma tal de ser lo más genérico posible, dando acceso

[2] `http://www.ibm.com/developerworks/library/l-pyint/`

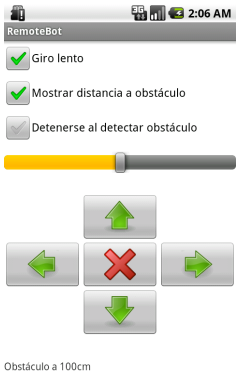


Figura 4-3: Cliente de RemoteBot

a algunas características de DuinoBot no utilizadas por el cliente Android como los sensores de línea con `getLine()` y operaciones bloqueantes como `forward(100, 1)` que no debe retornar hasta pasado un segundo.

Como se ve en la figura 4-2 RemoteBot utiliza la biblioteca DuinoBot para comunicarse con los robots y el cliente para el cuál fue diseñado es una aplicación Android que permite controlar los robots Multiplo N6 y ver los valores del sensor de distancia usando una interfaz con botones, además de poder controlar el robot usando los acelerómetros del dispositivo (inclinar el dispositivo en una dirección dada hace que el robot se mueva en esa dirección).

Tanto el servidor RemoteBot como el cliente RemoteBot4Android pueden descargarse en GitHub[3].

4.2. XRemoteBot

La interacción entre los distintos componentes de hardware y software usados por XRemoteBot es similar a la provista por RemoteBot, pero más compleja como puede verse en la figura 4-4. Uno de los motivos es que XRemoteBot soporta, además de los robots Multiplo N6, los robots Scribbler y puede ser ampliado para interactuar con otros robots. Desde el punto de vista del cliente XRemoteBot tiene varias diferencias, entre ellas la elección del protocolo, usándose WebSockets en lugar de peticiones

[3] `https://github.com/fernandolopez/remotebot` y `https://github.com/fernandolopez/remotebot4Android`

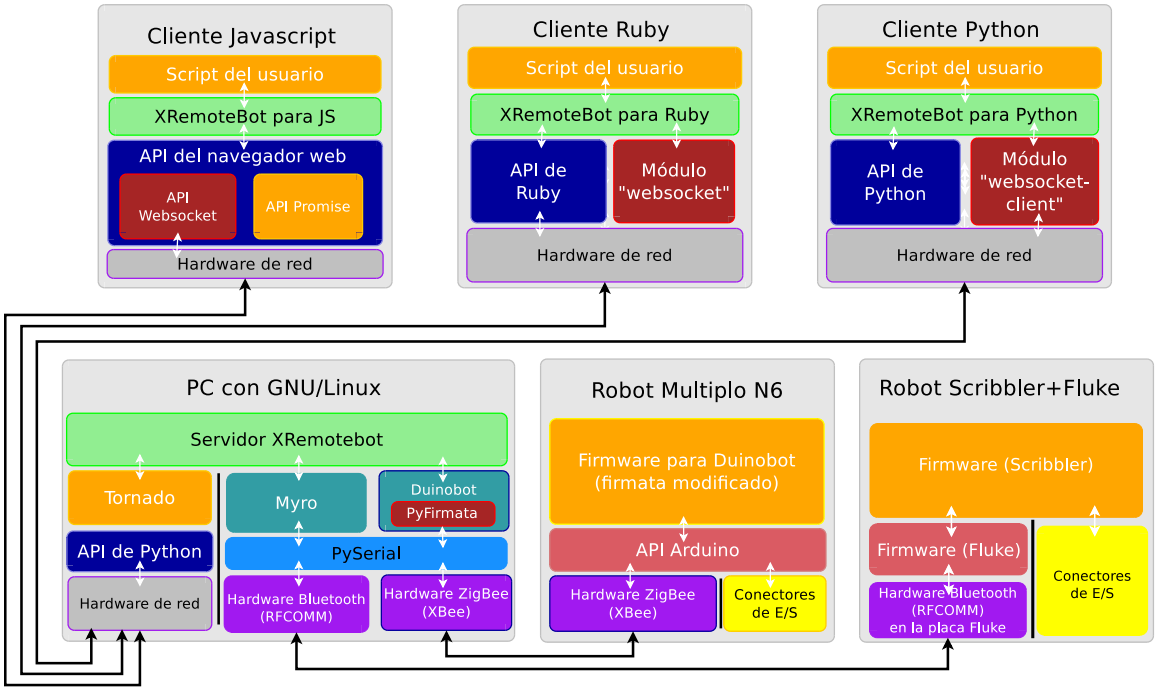


Figura 4-4: Diagrama de bloques de XRemoteBot

HTTP y la definición del protocolo de capa de aplicación que si bien es similar ahora posee un sistema de autenticación opcional con una “API key”.

Las figuras 4-5a y 4-5b describen el modo de conexión de los clientes con el servidor y del servidor con los robots en distintos escenarios. En especial, en la figura 4-5b se destaca que, en un ámbito local, el servidor no necesariamente debe ser un equipo dedicado. Sino que puede ser una estación de trabajo de manera de aprovechar mejor los recursos disponibles.

(a) Esquema de conexión de XRemoteBot en modo “servidor público”

(b) Esquema de conexión de XRemoteBot en modo “LAN”

Figura 4-5: Esquema de conexión de XRemoteBot

4.2.1. Configuraciones

XRemoteBot puede configurarse modificando el archivo `configuration.py` dentro del directorio `xremotebot`. Como su nombre lo indica, el archivo de configuración es un archivo escrito en el lenguaje Python por lo cuál cualquier modificación debe hacerse respetando la sintaxis de este lenguaje.

Algunas de las variables de configuración que se pueden encontrar son:

- `public_server`: Es un booleano, si vale `True` el servidor requiere autenticación con una *API key* para acceder a los robots, en cambio si vale `False` el servidor permite utilizar los robots sin solicitar una *API key* y no utiliza el sistema de reservas por lo que cualquier usuario puede usar cualquier robot.

- `disable_streaming`: Es un booleano, si vale `True` el servidor no hará streaming de video, ahorrando recursos tanto en el servidor como en los clientes que accedan a la interfaz web, en cambio si vale `False` el streaming de video se encontrará habilitado y visible desde la interfaz web.

- `robots`: Es un diccionario de Python, las claves deben ser modelos de robots soportados por XRemoteBot y por cada modelo debe tener como valor una lista con los identificadores de cada robot. Esta configuración determina el stock de robots disponibles a los que podrán acceder los usuarios.

- `hostname`: Es el nombre de dominio del servidor o bien su dirección IP.

- `port`: El puerto en el que esperará peticiones el servidor.

- `video_ws`: Es la dirección y puerto del servidor de streaming de video utilizado. Normalmente la parte de dirección de esta configuración será igual a la variable `hostname`, pero se puede configurar de forma tal que el servidor de streaming esté en un servidor distinto al que provee la interfaz web.

- `api_key_expiration`: Es un objeto de tipo fecha (`timedate`) que determina el tiempo de validez de las *API keys*. Transcurrido este tiempo, el usuario deberá renovar su *API key* para acceder a los robots.

- `reservation_expiration`: Es un objeto de tipo fecha que determina la validez de las reservas de los robots. Una vez transcurrido este tiempo desde la reserva de un robot el usuario no podrá seguir usando este robot a menos que lo vuelva a reservar.

- `log_file`: Es un string con el path donde se guardará el archivo de logs del servidor.

- `log_level`: Es un string que determina en nivel de detalle a guardar en los logs. El valor puede ser cualquiera de los soportados por el módulo logging de Python[4].

La variable `robots` está íntimamente relacionada con la extensibilidad del servidor, ya que si por ejemplo se quiere agregar soporte a XRemoteBot para manejar robots "sphero" basta con agregar la clave "sphero" al diccionario `robots`. Si XRemoteBot encuentra la clave "sphero" en "robots" intentará cargar un plugin en el path "remotebot/robots/sphero.py", el desarrollador debe implementar en este archivo una clase "Robot" que herede de "remotebot.robots.abstract_clases.Robot" implementando los métodos abstractos de esta clase.

Se puede ver un ejemplo de configuración del servidor en el código 4.1. En este ejemplo el servidor cuenta con cuatro robots utilizables, tres Multiplo N6 y un robot Parallax Scribbler.

```
1  hostname = '190.16.204.135'                    # 'xremotebot.example'
2  video_ws = 'ws://{}:8084/'.format(hostname)
3  log_level = 'DEBUG'
4  log_file = 'remotebot.log'
5  port = 8000
6  public_server = False
7  api_key_expiration = days(700)
8  reservation_expiration = hours(1)
9  robots = {
10     'n6': [13, 12, 11],
```

[4] https://docs.python.org/2/library/logging.html#logging-levels

```
11      'scribbler': ['00:1E:19:01:0B:81'],
12  }
```

Código 4.1: Configuración de ejemplo de XRemoteBot en `configuration.py`

4.2.2. WebSockets

Para permitir el uso remoto de los robots a través de Internet sin necesidad de requerir configuraciones ni puertos especiales se eligió implementar el sistema como una aplicación web. Sin embargo XRemoteBot no utiliza HTTP para el intercambio de mensajes y valores de retorno entre los clientes y el servidor, sino que usa el protocolo WebSocket.

El protocolo WebSocket permite mantener conexiones persistentes y no envía encabezados HTTP en cada mensaje, reduciendo así el overhead en los mensajes intercambiados que supondría el uso de HTTP [Wang et al., 2013] y aprovechando al mismo tiempo los puertos TCP que abre el servidor web.

La elección de este protocolo trae como desventaja frente al uso de HTTP la necesidad de tener consideraciones de seguridad especiales. Para la parte web del sistema se utiliza autenticación con cookies pero no resulta seguro utilizar el mismo mecanismo para la sesión con WebSockets ya que el sistema quedaría vulnerable a ataques de tipo CSRF (Cross-Site Request Forgery) [OWASP, 2014][5].

Una de las estrategias recomendadas es verificar el campo `Origin` en los encabezados del hanshake inicial de WebSockets desde el servidor, pero esta técnica no es suficiente ya que los clientes pueden falsear este campo y solamente si el cliente es un navegador envía este encabezado. Otra estrategia es tener un mecanismo de autenticación separado para la comunicación[6]. Esta última fue la estrategia elegida para XRemoteBot, usando una *API key* para identificar a los usuarios al inicio de la comunicación.

[5] `https://www.owasp.org/images/5/52/OWASP_Testing_Guide_v4.pdf`
[6] `https://www.christian-schneider.net/CrossSiteWebSocketHijacking.html`

Otras alternativas a WebSockets

Para decidir que protocolo de comunicaciones utilizar se analizaron las soluciones más comúnmente utilizadas en clientes Javascript, siendo este el cliente más relevante para esta decisión ya que se ejecuta en un entorno restringido a diferencia de los clientes para Python y Ruby donde es posible utilizar cualquier protocolo de red.

Históricamente la forma de realizar requerimientos a un servidor desde un cliente Javascript en un navegador fue usando la API XMLHttpRequest[7], la cual permite realizar peticiones HTTP síncronas y asíncronas al servidor[8].

Este método cuenta con la desventaja de requerir, en el peor caso, una conexión TCP por cada mensaje. Además por cada mensaje es necesario enviar y recibir los encabezados HTTP y esto es muy costoso: "Cada petición XHR es una petición individual HTTP y, en promedio, HTTP tiene 800 bytes de sobrecarga (sin cookies HTTP) por cada petición/respuesta" [Grigorik, 2013].

Una posible solución a este problema es el uso de *Long Polling*, uno de los ejemplos del uso de esta técnica es el protocolo *Bayeux* también conocido como *Comet* por su implementación principal *CometD* [Roden, 2010]. Bayeux puede funcionar con distintos "transportes", en particular CometD soporta "long-polling", "callback-polling" y WebSocket, en lo que resta de esta sección consideraremos solamente los dos primeros. El protocolo Bayeux permite al servidor demorar la respuesta de ciertas peticiones, enviar al cliente respuestas incompletas para reutilizar la conexión abierta en la petición o contestar varias peticiones de una sola vez[9]. Sin embargo cuando el cliente debe hacer una nueva petición enviará nuevamente los encabezados HTTP e incluso el navegador puede llegar a crear una nueva conexión TCP.

Otro uso de *Long Polling* (probablemente el primer protocolo de long polling definido) es el protocolo *BOSH* que tiene características similares a Bayeux[10].

Si bien estos protocolos permiten evitar algunas reconexiones cuando se trabaja

[7] `http://www.w3.org/TR/XMLHttpRequest/`

[8] `https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Synchronous_and_Asynchronous_Requests#Synchronous_request`

[9] `http://docs.cometd.org/3/reference/#_bayeux`

[10] `http://xmpp.org/extensions/xep-0124.html`

de forma asincrónica con el servidor, eventualmente cuando el servidor envíe una respuesta puede cerrarse la conexión TCP si no hay tráfico en una determinada ventana de tiempo, también cada mensaje deberá tener su correspondiente encabezado HTTP, con lo cual estas técnicas no resuelven completamente el problema del overhead del uso de XMLHttpRequests sino que lo hacen parcialmente y sobre todo en aplicaciones asincrónicas donde se supone que el servidor puede enviar datos al cliente en cualquier momento.

La elección de WebSockets resulta más natural para una implementación como XRemoteBot ya que mantiene una conexión permanente y no intercambia encabezados HTTP en cada mensaje (solamente un "handshake" inicial al abrir la conexión).

Capítulo 5

Clientes de XRemoteBot

La implementación de XRemoteBot para esta tesina incluye tres clientes en distintos lenguajes de programación: Python, Ruby y Javascript. Este último diseñado para ejecutarse en el entorno de un navegador web.

Estos clientes tienen distintas características y principalmente el que más difiere de los demás es el cliente Javascript ya que es una implementación a ser ejecutada en un navegador web con las restricciones que esto trae, pero con la ventaja de que se encuentra integrado con la vista web de XRemoteBot por lo que cuenta con una vista de la cámara que integra en una misma pantalla la acción de programar del usuario con el resultado de la ejecución de dicho código.

En general, se intentó mantener el estilo de programación del módulo DuinoBot original de la forma más fiel posible. En el caso de Javascript por limitaciones del entorno no se logró imitar este estilo tan fielmente como en los otros clientes.

5.1. Cliente Python

El uso del cliente Python es muy similar al uso directo de los robots con la biblioteca DuinoBot, aunque la implementación es bastante diferente por el protocolo subyacente.

El cliente Python utiliza un solo módulo que no pertenece a la biblioteca estándar

de Python: el módulo para acceso a WebSockets *websocket-client*[1] que cuenta, con una interfaz asincrónica y con una interfaz de "bajo nivel" sincrónica similar a la interfaz tradicional de Sockets[2]. Esta última resulta la más adecuada para imitar el comportamiento de una biblioteca como DuinoBot.

```
1 from xremotebot import *
2 server = Server('ws://xremotebot.example:8000/api', 'api_key')
3 robot = Robot(server, server.fetch_robot())
4 robot.forward(100, 1)
5 robot.backward(50, 2)
6 print(robot.getObstacle())
```

Código 5.1: Ejemplo con XRemoteBot para Python

En el código 5.1 se muestra un script sencillo creado usando el cliente para Python de XRemoteBot. Este script reserva un robot en la línea 3, luego lo hace avanzar a velocidad máxima por 1 segundo, lo hace girar a velocidad media durante 2 segundos y finalmente imprime en pantalla un valor booleano que indica si existe algún obstáculo enfrente del robot.

Los movimientos, que pueden recibir por parámetro un tiempo de demora, se implementan en el cliente utilizando dos mensajes (uno que comienza el movimiento y otro que lo detiene) y la función `time.sleep()` de Python, esta función suspende la ejecución del hilo actual durante el número de segundos que se envíe como parámetro. De esta forma invocar el método `robot.forward(100, 1)` en el cliente provoca que el mismo envíe al servidor los mensajes del código 5.2 con una pausa de un segundo entre ellos. Como todos los métodos de movimiento proveen esta funcionalidad de demora, la misma está implementada en el *decorator*[3] `xremotebot.timed` a fin de evitar la repetición de código.

```
1 {
2     "entity": "robot",
3     "method": "forward",
```

[1] `https://pypi.python.org/pypi/websocket-client`

[2] `http://pubs.opengroup.org/onlinepubs/7908799/xns/syssocket.h.html`

[3] Esto es un decorator de Python, no confundir con el patrón de diseño. `https://www.python.org/dev/peps/pep-0318/`

```
4       "args": [100]
5   }
6   # Después de 1 segundo...
7   {
8       "entity": "robot",
9       "method": "stop",
10      "args": []
11  }
```

Código 5.2: Mensajes generados al invocar `robot.forward(100, 1)` en XRemoteBot para Python

5.2. Cliente Ruby

Las gemas (en Ruby el formato de habitual de los paquetes de software se denomina "gema" o, en inglés, "gem" [Thomas et al., 2013]) que proveen soporte para WebSockets en Ruby cuentan con una API asincrónica que no es adecuada para este proyecto. Por esto originalmente se analizó incorporar el código del proyecto *web-socket-ruby*[4] al cliente Ruby de XRemoteBot. Este proyecto no ese encuentra empaquetado en forma de gema, pero provee una API sincrónica que permitiría implementar el cliente Ruby de XRemoteBot con una interfaz similar a la biblioteca DuinoBot original. Sin embargo este proyecto no funciona con servidores modernos de WebSockets. Por ello, en su lugar, se utilizó la gema *websocket*[5] que no es una implementación completa del protocolo, sino que es una biblioteca de bajo nivel que solamente permite codificar y decodificar mensajes del protocolo, pero no provee acceso a la red.

Para proveer cierto nivel de abstracción en el acceso a los WebSockets se implementó la clase `WS` que provee la funcionalidad requerida usando la clase TCPSocket[6] de Ruby para el acceso a la red y la gema *websocket* para codificar y decodificar los mensajes intercambiados. Esta implementación se diseñó para que provea una interfaz

[4] `https://github.com/gimite/web-socket-ruby`
[5] `https://github.com/imanel/websocket-ruby`
[6] `http://ruby-doc.org/stdlib-2.1.0/libdoc/socket/rdoc/TCPSocket.html`

similar a la de *websocket-client*[7].

La implementación realizada cuenta con la limitación que por el momento solamente soporta conexiones inseguras, a futuro es posible agregar soporte de conexiones seguras usando el módulo *openssl*[8].

```
1 require 'xremotebot'
2 server = XRemoteBot::Server.new('xremotebot.example',
3                                 8000,
4                                 'api',
5                                 'api_key')
6 robot = Robot.new server, server.fetch_robot
7 robot.forward 100, 1
8 robot.backward 50, 2
9 print robot.getObstacle
```

Código 5.3: Ejemplo usando XRemoteBot para Ruby

El código 5.3 implementa exactamente la misma funcionalidad que el el ejemplo en Python visto anteriormente (código 5.1) pero usando la biblioteca XRemoteBot para Ruby.

5.3. Cliente Javascript

Como se mencionó con anterioridad, gran parte de las decisiones de diseño de XRemoteBot se hicieron para permitir la implementación de un cliente Javascript. Sin embargo, aún tomando estas precauciones, el cliente Javascript resulta ser la más compleja de las tres implementaciones de clientes desarrolladas.

5.3.1. API Javascript y asincronismo

Dado el hecho de que Javascript, dentro del entorno de un navegador web se comporta de forma asincrónica, no fue posible implementar un cliente Javascript cuyo uso se asemeje al de la biblioteca de Python DuinoBot.

[7] `https://pypi.python.org/pypi/websocket-client`
[8] `http://ruby-doc.org/stdlib-2.2.1/libdoc/openssl/rdoc/OpenSSL.html`

Para ilustrar esta problemática se puede tomar en consideración el código 5.4 hecho usando DuinoBot en Python y analizar los inconvenientes para replicar algo similar en Javascript.

```
1  from duinobot import *
2  board = Board()
3  robot = Robot(board, 10)
4
5  robot.forward(100, 1)        # avanza, bloquea la ejecución del script
6                               # 1 segundo y luego detiene el robot.
7
8  robot.backward(50, 2)        # retrocede, bloquea la ejecución del
9                               # script durante 2 segundos y luego
10                              # detiene el robot.
11
12 print(robot.getObstacle())   # imprime True si hay un obstáculo.
```

Código 5.4: Ejemplo típico usando DuinoBot

Las líneas 5 y 8 del código 5.4 no se pueden traducir de forma directa a Javascript ya que demoran la ejecución del programa 1 y 2 segundos respectivamente, esto es imposible de implementar en Javascript en el entorno de un navegador sin utilizar la técnica de "busy waiting" o modificar el código para usar la función de Javascript `setTimeout()`. La línea 12 tampoco puede trasladarse a Javascript sin grandes modificaciones, ya que requiere enviar un mensaje al servidor para pedirle los datos del sensor, esperar la respuesta y retornarla, todo esto no es posible con WebSockets en el navegador ya que su API es asincrónica.

Utilizar la técnica "busy waiting" en el navegador no sería adecuado ya que normalmente los navegadores utilizan un solo thread por pestaña y esto bloquearía al navegador al menos en la pestaña actual. Por otro lado el uso de `setTimeout()` puede servir para implementar las líneas 5 y 8 pero no la línea 12 ya que no es posible determinar el tiempo de respuesta del servidor.

Como se menciona en el la sección 4.2.2, el servidor utiliza WebSockets como protocolo de transporte. Por esto XRemoteBot para Javascript debe utilizar la API

de WebSockets provista por los navegadores. La API de WebSockets disponible en los navegadores web es asincrónica y está basada en el uso de "callbacks" que se ejecutan ante distintos eventos [Wang et al., 2013]. Para utilizar un WebSocket usando esta API es necesario redefinir una serie de atributos de la instancia del WebSocket, asignándoles las funciones que se desea ejecutar ante cada cambio de estado en el WebSocket [W3C, 2014]:

`WebSocket#onopen():` se ejecutará cuando la conexión esté establecida.

`WebSocket#onerror():` se ejecutará ante un error en la conexión.

`WebSocket#onclose():` se ejecutará si la conexión se cierra.

`WebSocket#onmessage():` se ejecutará al recibir un mensaje desde el servidor, recibe como argumento el mensaje enviado por el servidor.

Con el objetivo de ocultar los detalles de implementación a los usuarios que desean controlar los robots usando XRemoteBot para Javascript, se utilizará un objeto `Promise` como se describe en la siguiente sección.

5.3.2. Promises

Para proveer una API lo más parecida posible a DuinoBot sin tener la capacidad de demorar la ejecución del código hasta que llegue la respuesta de las peticiones, se investigó la API *Promise* [ECMA, 2015] de ECMAScript 6 que permite modelar esta espera de una forma razonablemente intuitiva.

Si bien el objeto *Promise* está en proceso de estandarización, los navegadores Google Chrome, Firefox, Internet Explorer, Opera y Safari lo soportan[9].

Los objetos *Promise* se utilizan para obtener valores resultantes de cómputos asincrónicos, como es el caso de la respuesta a una petición hecha con WebSockets. La interfaz de *Promise* no permite programar los robots en Javascript con una interfaz idéntica a la usada al programar los robots con la biblioteca DuinoBot, pero es

[9] `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise`

relativamente fácil de utilizar y ante la imposibilidad de demorar la ejecución del código Javascript provee una alternativa simple en comparación con el uso directo de la interfaz de los objetos `WebSocket`.

Los objetos *Promise* se instancian pasándoles como argumento una función que a su vez recibe 2 argumentos: `resolve` y `reject`. Estos argumentos son funciones, invocar al primero "cumple" o "resuelve" la promesa e invocar al segundo la "rechaza". Las "promesas" tienen dos métodos: `then` y `catch` que se ejecutan cuando la promesa se "cumple" o se "rechaza" respectivamente.

Una primer implementación

Usando la API `Promise` se implementó una primer versión de XRemoteBot para Javascript donde cada mensaje enviado al servidor retorna una instancia de *Promise*. El código 5.5 muestra una implementación en Javascript del ejemplo visto en el código 5.4.

```
1  var server = new Server('xremotebot.example:8000', 'api-key');
2  server.onConnect(function(){
3      server.fetch_robot().then(function(robot_obj){
4          var robot = new Robot(server, robot_obj);
5          robot.forward(100, 1).then(function(){
6              robot.backward(50, 2).then(function(){
7                  robot.getObstacle().then(function(obstacle){
8                      println(obstacle);
9                  })
10             })
11         })
12     });
13 });
```

Código 5.5: Ejemplo de XRemoteBot en Javascript

Como se puede ver, después de cada acción bloqueante, es decir cada método que provoque una demora y de cada método que retorne un valor útil, como `Robot#getObstacle()`, se invoca el método `Promise#then()` con una función anóni-

ma como argumento. Esta función, que se pasa por argumento al método `Promise#then()`, se ejecutará si la *Promise* se "resuelve". En el caso de XRemoteBot la *Promise* se "resuelve" cuando el servidor responde al mensaje que generó la *Promise*.

```
1  this.send_ws_msg = function(msg){
2      var promise;
3      this.msg_id++;
4      promise = new Promise(function(resolve, reject){
5          that.pending_msgs[msg_id] = {resolve: resolve, reject: reject};
6      });
7      msg['msg_id'] = msg_id;
8      that.ws.send(JSON.stringify(msg));
9      return promise;
10 }
11 // ...
12 this.ws.onmessage = function(msg){
13     msg = JSON.parse(msg.data);
14     if (msg['msg_id'] !== undefined){
15         var executor = that.pending_msgs[msg['msg_id']];
16         delete that.pending_msgs[msg['msg_id']];
17         if (msg.response === 'value'){
18             executor.resolve(msg.value);
19         }
20         else{
21             executor.reject(msg.message);
22         }
23     }
24 };
```

Código 5.6: Ejemplo simplificado de la implementación de XRemoteBot con Promises dentro del constructor Server en xremotebot.js

Esta implementación se logró incluyendo en los mensajes un campo `msg_id` que identifica, para cada instancia de WebSocket, a cada mensaje de forma unívoca. A su vez el servidor, al encontrar este campo en un mensaje, lo copia sin modificarlo a la respuesta correspondiente al mismo. Del lado del cliente, por cada instancia

de WebSocket se mantiene una colección de las promesas correspondientes a cada `msg_id` y se las gestiona desde la función `WebSocket#onmessage()` "resolviéndolas" o "rechazándolas" según se reciba como respuesta un valor o un error. El código 5.6 muestra una versión simplificada de esta primer implementación.

Cada método de XRemoteBot para Javascript que involucre enviar un mensaje retornará entonces una "promesa", por lo que habrá que tener en cuenta invocar a "then()" o a "catch()" cuando es necesario utilizar el valor retornado por el servidor o determinar si la ejecución fue exitosa.

Una versión mejorada

Evidentemente la forma de programar del ejemplo visto en el código 5.5 resulta engorrosa y poco práctica. Por esto se modificó el cliente Javascript para que no envíe un nuevo mensaje hasta que el mensaje anterior haya recibido una respuesta. Esto se implementó usando identificadores secuenciales para los mensajes de forma que se pueda verificar si el mensaje previo tuvo respuesta y una cola para demorar a los mensajes que aún no pueden enviarse.

```
1  var server = new Server('xremotebot.example:8000', 'api-key');
2  server.onConnect(function(){
3      server.fetch_robot().then(function(robot_obj){
4          var robot = new Robot(server, robot_obj);
5          robot.forward(100, 1);
6          robot.backward(50, 2);
7          robot.getObstacle().then(function(obstacle){
8              println(obstacle);
9          });
10     });
11 });
```

Código 5.7: Ejemplo de XRemoteBot en Javascript con empleo de una cola para serializar mensajes

El atributo `Server.delayed` en `xremotebot.js` mantiene la cola de mensajes demorados. Cuando el servidor contesta un mensaje previo se toma otro mensaje de

la cola (si lo hubiere) y se lo envía al servidor. De esta manera, se garantiza la demora necesaria entre la ejecución de cada método del robot. Sin embargo, para obtener los valores de retorno de los métodos, como en el caso de los métodos de acceso a los sensores, sigue siendo necesario el uso del método `Promise#then`. A pesar de esto último, como se puede ver en el código 5.7, con estas modificaciones es posible hacer que el cliente Javascript tenga una API más limpia y usable, esta última versión es la definitiva para este trabajo.

Alternativas al uso de promises

Una solución alternativa para lograr imitar tanto como sea posible la API de DuinoBot puede ser utilizar un intérprete Javascript implementado en Javascript como JS-Interpreter[10], pero requeriría modificar el intérprete para lograr una ejecución paso a paso controlada por el flujo de mensajes a través de la conexión con WebSockets. Además de agregar complejidad, este tipo de intérpretes no accede a un entorno completo como lo hace el intérprete incorporado en los navegadores y puede tener incompatibilidades o funcionalidades no implementadas, en este caso por ejemplo JS-Interpreter no soporta excepciones y no puede interactuar directamente con DOM. Otra opción es Hypnotic[11] que utiliza el intérprete Narcissus y provee una función `sleep()` que demora la ejecución del código como se desea para este proyecto, pero el inconveniente de Hypnotic es que por el momento solamente funciona en el navegador Firefox[12].

5.3.3. Interacción con el navegador, DOM y JQuery

Dado que el cliente Javascript se ejecuta en el entorno de un navegador web es posible interactuar con el árbol DOM de la página. Por ejemplo modificando sus nodos. Y dado que la interfaz web utiliza JQuery[13] esta biblioteca también está disponible para los usuarios, pudiendo hacer por ejemplo un script que pide los valores de los

[10] `https://github.com/NeilFraser/JS-Interpreter`
[11] `http://coolwanglu.github.io/hypnotic/web/demo.html`
[12] `https://github.com/coolwanglu/hypnotic/wiki#limitations`
[13] `http://jquery.com`

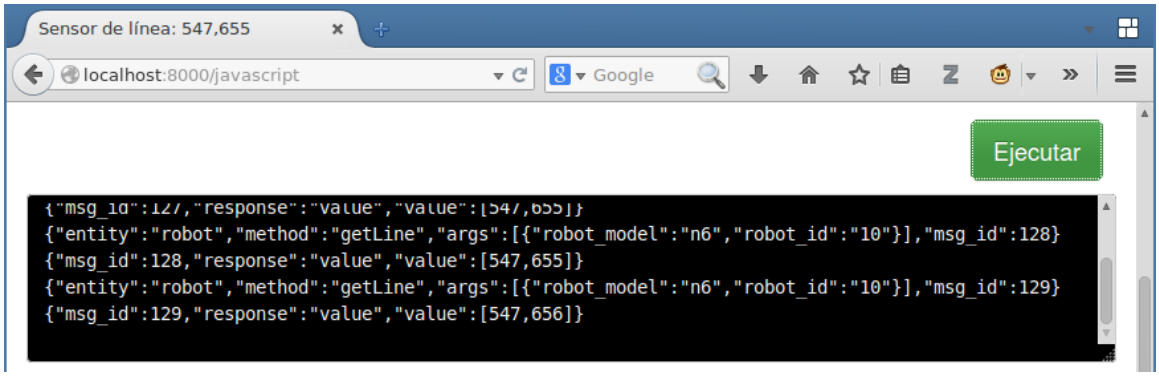


Figura 5-1: Resultado de la ejecución del código 5.8

sensores de línea cada 500 milisegundos y los muestra en el título de la página usando JQuery para manipular el árbol DOM como se ve en el código 5.8, el resultado se puede ver en la figura 5-1.

```
1  var server = new Server('ws://localhost:8000/api',
2                          '97385401-3874-439c-b01b-df94349d888a');
3  server.onConnect(function(){
4      server.fetch_robot().then(function(robot_obj){
5          var robot = new Robot(server, robot_obj);
6          setInterval(function(){
7              robot.getLine().then(function(line){
8                  $('title').text('Sensor de línea: ' + line);
9              });
10         }, 500);
11     });
12 });
```

Código 5.8: Manipulación de DOM con JQuery desde XRemoteBot para Javascript

```
robot.getObstacle().then(function(hay_obstaculo){
    if (hay_obstaculo){
        console.log("Hay un obstáculo al frente");
    }
    else {
        console.log("No hay obstáculos");
    }
});
```

Código 5.9: Implementación de `getObstacle()` con "promesas"

El código 5.9 utiliza `console.log`, para mostrar el valor del sensor de obstáculos en la consola de depuración del navegador.

```
var server = new Server('ws://190.16.204.135:8000/api',
                        '8d4ae089-443b-4bda-bd04-6601d664dd96');
server.onConnect(function(){
    server.get_robots().then(function(robots){
        println(robots);
    });

    server.fetch_robot().then(function(robot_obj){
        println(robot_obj);
        var robot = new Robot(server, robot_obj);
        var inter = setInterval(function(){
            robot.getLine().then(function(line){
                $('h1').text(line);
            }).catch(function(error){
                clearInterval(inter);
                alert(error);
            });
        }, 5000);
    });
});
```

Código 5.10: Manejo de errores en XRemoteBot para Javascript

Figura 5-2: Botonera creada en el código 5.11

El código 5.10 muestra un ejemplo del manejo de errores utilizando la función `catch()`. El ejemplo reserva un robot y le pide periódicamente el valor de su sensor de obstáculos, si la petición tiene éxito pone el valor como título de la página, en cambio si la petición falla (por ejemplo porque expira la reserva) se muestra un mensaje de error. Este ejemplo combina el acceso a las funciones del navegador como `alert()` y `setInterval()`, el uso de JQuery y la manipulación del árbol DOM.

En un ejemplo más avanzado de uso de DOM y JQuery podría plantearse como actividad, para enseñar programación en Javascript, agregar una botonera para controlar a los robots haciendo clic. El código 5.11 muestra una posible solución a esa actividad, agregando los botones debajo del área de streaming de video y la figura 5-2 muestra el resultado de ejecutar este código.

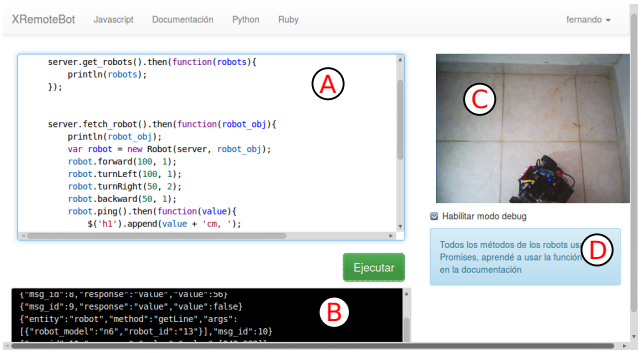
```
1  var server = new Server('ws://localhost:8000/api',
2                          '97385401-3874-439c-b01b-df94349d888a');
3  server.onConnect(function(){
4      server.fetch_robot().then(function(robot_obj){
5          var robot = new Robot(server, robot_obj);
6          $('#videoCanvas').after("<div>\
7              <input id='izq' type='button' value='Izquierda'>\
8              <input id='atr' type='button' value='Atr&aacute;s'>\
9              <input id='ade' type='button' value='Adelante'>\
10             <input id='der' type='button' value='Derecha'>\
11             </div>");
12         $('#izq').click(function(){ robot.turnLeft(50, 0.5); });
```

```
13          $('#atr').click(function(){ robot.backward(100, 0.5); });
14          $('#ade').click(function(){ robot.forward(100, 0.5); });
15          $('#der').click(function(){ robot.turnRight(50, 0.5); });
16      });
17  });
```

Código 5.11: Agregar botones debajo del área de video usando JQuery

5.3.4. Interfaz web y streaming de video

La interfaz web de XRemoteBot que está pensada principalmente para los casos de uso donde los clientes están en una ubicación geográfica distinta a la de los robots, provee login, acceso a la visualización y renovación de una clave alfanumérica única por cada usuario denominada *API key* que permite reservar y controlar los robots sin la necesidad de exponer el nombre de usuario y contraseña, y una página que permite ver los robots en vivo por video opcionalmente controlándolos con un script Javascript (figura 5-3).



A: Area de texto para codificar.

B: Area que simula una consola. Los programas puedan imprimir texto en este área.

C: Transmisión de video. Muestra los robots en tiempo real.

D: Caja de "tips" o consejos.

Figura 5-3: Interfaz web de XRemoteBot

Esta interfaz provee además acceso a la documentación básica para trabajar con XRemoteBot desde cualquiera de los tres clientes implementados.

La interfaz web fue pensada principalmente para ser utilizada en conjunto con el cliente Javascript, aunque puede ser accedida sin problemas aún si se usa otro de los clientes para poder visualizar los robots a través del streaming de video. Provee un área de texto para escribir los scripts a ejecutar. Esta área de texto es creada con

CodeMirror[14] que provee resaltado de sintaxis y manejo de los eventos de teclado para que, al presionar *Tab*, el editor indente el código en lugar de saltar a otro elemento de la página como harían los navegadores por defecto.

Otra área de texto en la parte inferior simula la salida de una terminal, en la misma se pueden ver mensajes de log generados con la función `rblog` que pueden ser habilitados o deshabilitados con la opción "habilitar modo debug" y mensajes impresos con la función `println`. Ambas funciones pueden imprimir, además de cadenas de texto, objetos. Estos últimos se convierten a cadenas de texto usando la función `JSON.stringify()`.

Un área de video destinada a emisiones en vivo que muestra la posición del robot. La emisión de video no requiere ningún plugin ya que se utilizó `jsmpeg`[15] que permite emitir video en vivo usando WebSockets y renderizarlo en el navegador en un elemento Canvas, de esta manera se logró tener video en vivo utilizando características estándar de HTML5.

En la página "datos del usuario", que se puede acceder desde el menú en la parte derecha de la pantalla, se puede visualizar la API key, se la puede renovar y se puede cambiar la contraseña del usuario (figura 5-4).

[14] `https://codemirror.net/`
[15] `https://github.com/phoboslab/jsmpeg`

Figura 5-4: Página “datos del usuario”

Capítulo 6

Protocolo de capa de aplicación de XRemoteBot

Siendo una aplicación cliente-servidor XRemoteBot requiere algún método de serialización para intercambiar datos entre los clientes y el servidor. Considerando que el servidor de XRemoteBot es un servidor web que usa WebSockets como protocolo de comunicación, el método de serialización propuesto es JSON (Javascript Serialization Object Notation).

6.1. Comparación entre JSON, BSON y CBOR

JSON cuenta con las siguientes características:

1. Es un formato estandarizado por ECMA [ECMA, 2013] y está especificado por la RFC-7159 [IETF, 2014].

2. Soporta los tipos de datos necesarios para intercambiar mensajes con los datos necesarios para controlar los robots usando un nivel de abstracción adecuado.

3. Al ser un formato de texto es simple analizar el tráfico entre los clientes y el servidor para detectar posibles errores.

4. Está soportado de forma nativa por los navegadores más utilizados.

Pero también existen otras alternativas cuyos objetivos son serialización y deserialización rápida de datos como BSON (Binary JSON) y (CBOR Concise Binary Object Representation).

Algunas características de BSON y CBOR son:

1. Codifican la información en formato binario.
2. Son tan fáciles de usar como JSON.
3. Ambos formatos pueden codificar y decodificar los mismos tipos de datos que JSON, además de soportar otros tipos de datos adicionales.
4. CBOR está especificado por la RFC-7049 [IETF, 2013].
5. BSON tiene una especificación informal pero cuenta una implementación bien conocida siendo la representación de datos primaria en MongoDB[1].
6. Por estar en formato binario decodificar una captura de tráfico para depurar un programa es más laborioso.
7. En algunos casos estos formatos binarios generarán mensajes más chicos que JSON, pero no siempre.
8. Requieren usar bibliotecas Javascript en los clientes web ya que los navegadores no lo implementan de forma nativa.

Para tomar un decisión respecto al formato a utilizar se decidió generar 16 archivos `JSON` con datos aleatorios. Estos archivos se cargaron con una cantidad de entradas múltiplo de 1024 desde 1024 hasta 16384, donde cada una de estas entradas es un objeto `JSON` con 5 entradas de tipo numérico, 5 strings, 5 objetos (cada uno con una entrada numérica) y 5 arrays (cada uno con 2 entradas de tipo string).

Estos archivos se cargaron desde scripts similares en Python y Javascript que deserializaron y serializaron estos datos repetidas veces calculando el tiempo promedio que llevó hacer cada una de estas acciones para cada formato.

[1] `http://bsonspec.org/`

Se eligió hacer las pruebas con Python y Javascript porque el primero es el lenguaje de implementación del servidor y del cliente que probablemente tenga más uso en un futuro, mientras que Javascript es el lenguaje de implementación del cliente que se ejecutará en los navegadores web y que puede servir como base para implementaciones, por ejemplo, con Brython[2], Skulp[3] u Opal[4].

Para hacer el experimento repetible, la implementación en Javascript se ejecutó con el intérprete **nodejs** basado en el motor **v8** usado por **Chrome** y además se crearon scripts de apoyo para generar los datos de prueba, ejecutar los scripts con los distintos formatos de forma automatizada y formatear los datos resultantes en archivos CSV.

Todas las pruebas se realizaron sobre Lihuen 6 beta (basado en Debian Jessie) en una notebook con procesador "Intel(R) Core(TM) i3 CPU M 370 @ 2.40GHz" y 4GB de RAM usando Python 3.4.2 y NodeJS 0.10.35[5]

En estas pruebas se tomaron mediciones de los tiempos de serialización y deserialización en Python para las 3 alternativas en relación a la cantidad de entradas (figura 6-1), los tiempos de serialización y deserialización en Javascript para las 3 alternativas también en relación a la cantidad de entradas (figura 6-2) y finalmente la relación entre la cantidad de entradas y el tamaño de los strings generados al serializar los mismos datos en los 3 formatos evaluados (figura 6-3).

De la figura 6-3 se desprende que los distintos métodos de serialización no ofrecen diferencias significativas en el tamaño de los strings generados para los volúmenes de datos probados. Por otro lado, se puede observar en la figura 6-1 que, al menos en Python y con la biblioteca usada, CBOR tiene los mejores tiempos. Sin embargo en las pruebas con Javascript en la figura 6-2 se puede observar que los mejores tiempos son para el formato JSON. Esto es entendible ya que este es el único formato (entre

[2] `http://www.brython.info/`
[3] `http://www.skulpt.org/`
[4] `http://opalrb.org/`
[5] Se usó NodeJS 0.10.35 ya que la versión 0.10.29 distribuida al momento con Debian Jessie tenía un bug que generaba un error de memoria al deserializar strings largos con `JSON.load` (CVE-2014-5256)

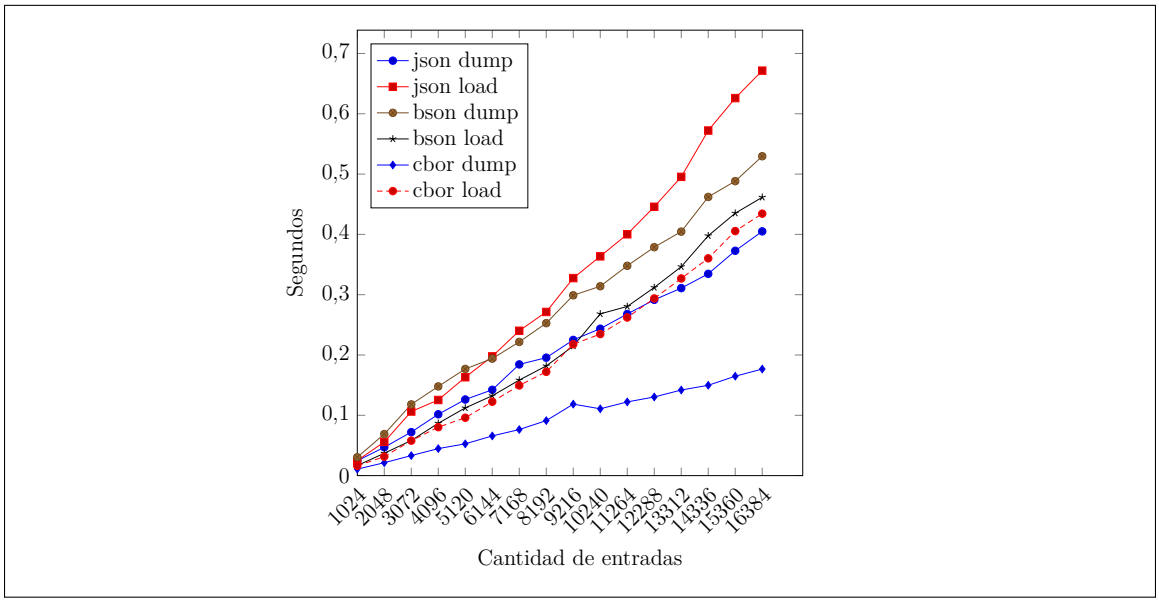


Figura 6-1: Cantidad de entradas versus tiempo de serialización y deserialización en Python

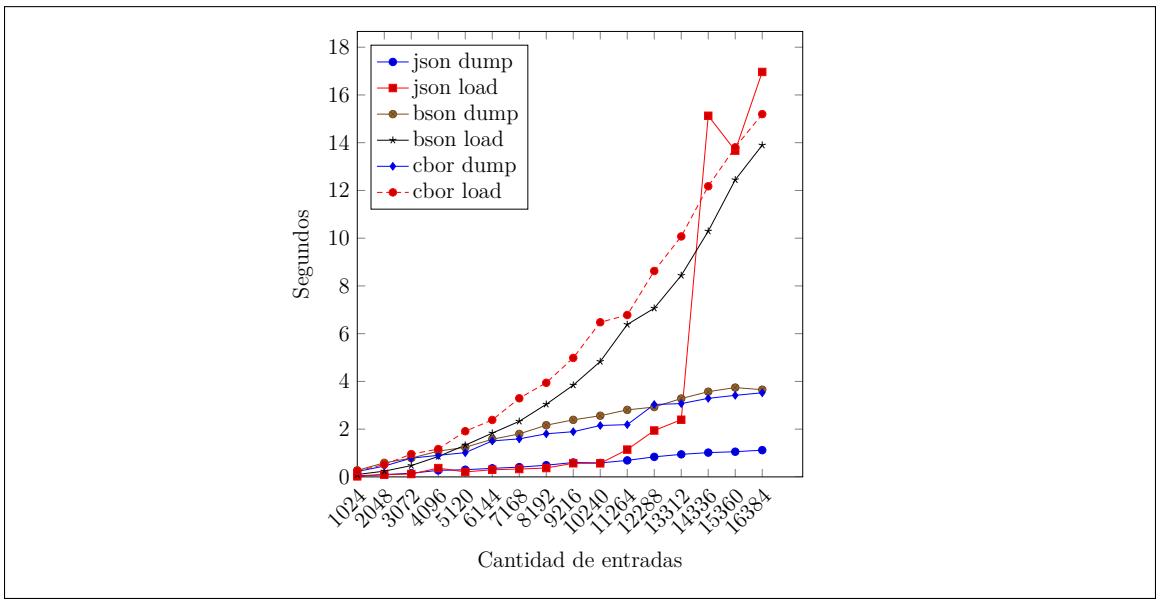


Figura 6-2: Cantidad de entradas versus tiempo de serialización y deserialización en Javascript (nodejs)

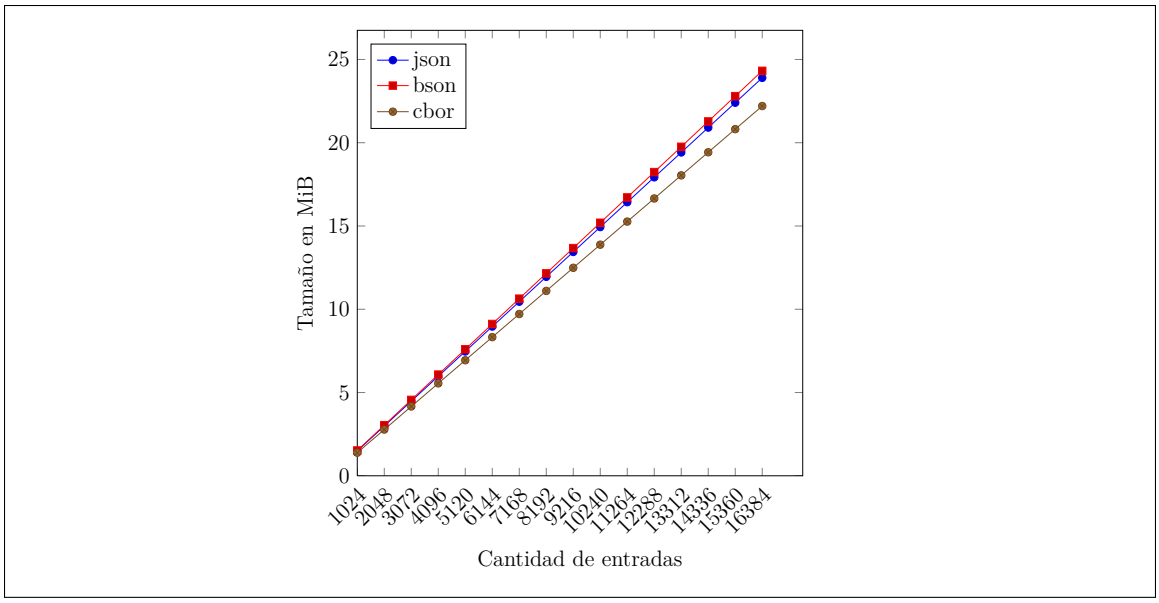


Figura 6-3: Cantidad de entradas versus tamaño del archivo serializado

los evaluados) procesado de forma nativa en este lenguaje[6].

Teniendo en cuenta que las diferencias de tiempo al procesar los datos en Python son, en comparación con la versión en Javascript, marginales (apenas alrededor de 0,25 segundos para 2^{14} entradas entre el mejor y peor tiempo de deserialización) y las diferencias en cuanto a tamaño del string generado al serializar también son pequeñas (aproximadamente 2MiB de diferencia para 2^{14} entradas entre el mejor y el peor método). Se eligió implementar el protocolo de XRemoteBot con JSON, que en estas pruebas cuenta con los mejores tiempos de serialización y deserialización en Javascript, además de estar soportado en los 3 lenguajes sin necesidad de utilizar bibliotecas externas.

6.2. Protocolo diseñado para XRemoteBot

Como se describió anteriormente, XRemoteBot utiliza el protocolo WebSocket como protocolo de la capa de transporte y mensajes codificados con JSON como

[6]Si bien es posible encontrar parsers nativos para los otros formatos y usarlos desde NodeJS no se utilizó esta modalidad porque no sería una prueba realista dado que los motores Javascript de los navegadores no permiten esto.

protocolo de la capa de aplicación. El propósito de este capítulo es describir este protocolo de capa de aplicación y detallar los motivos por los cuales se modeló el protocolo de esta manera.

6.2.1. Consideraciones generales

El protocolo fue diseñado pensando que el mismo debería representar los distintos comandos enviados a los robots como si fueran mensajes a objetos del paradigma de programación orientado a objetos. Este diseño por un lado mapea de forma muy directa los mensajes de la API con los métodos que un programador utilizaría para controlar a los robots usando la biblioteca DuinoBot, y por otro lado, provee un alto grado de flexibilidad a la hora de agregar nuevos tipos de mensajes a la API sin necesidad de reescribir grandes porciones de código.

En algunos de los lenguajes orientados a objetos más populares como Python o Ruby, al ejecutar un método pueden ocurrir básicamente dos cosas: se puede obtener un valor de resultado o bien un error en tiempo de ejecución en la forma de una excepción. Para modelar este comportamiento el servidor XRemoteBot cuenta con dos tipos de mensajes de respuesta:

- El mensaje tipo “value” que retorna el valor resultante de ejecutar el método invocado.

- El mensaje tipo “error” que representa el error en tiempo de ejecución que típicamente se representa con excepciones.

Los mensajes de error de XRemoteBot tienen como contenido un mensaje de texto que describe el error producido. Los clientes pueden tomar este mensaje de error y mostrarlos en pantalla o usarlos como descripción de una excepción como se hace en el cliente Python por ejemplo.

6.2.2. Alternativas analizadas

En definitiva cualquier forma de serialización de datos hubiera sido suficiente para este protocolo. Se eligió evaluar protocolos similares a JSON dada su facilidad de uso, facilidad de lectura y la disponibilidad de funciones para manipular JSON en las bibliotecas estándar de los tres lenguajes elegidos. Esto es importante porque es la intención del autor que cualquiera con conocimientos básicos de programación pueda desarrollar clientes en otros lenguajes.

Estas características no se dan en formatos como YAML que son más complejos y solamente no está soportado por la biblioteca estándar de Python ni por Javascript. Tampoco cumple con las características requeridas XML que si bien es posible procesarlo en cualquiera de estos lenguajes sin agregar bibliotecas externas, un posible desarrollador de un cliente tendría más problemas para implementar el cliente ya que las API para manipular XML suelen ser más complejas dada la estructura de estos archivos y además la información codificada en XML es más difícil de leer por ejemplo en una captura de red.

También se descartó crear un mecanismo de serialización específico para este trabajo, ya que de hacerlo debería haber implementado todo el protocolo en tres lenguajes distintos y además cualquiera que quisiera crear un cliente para XRemoteBot debería también implementar el protocolo en el lenguaje deseado, con toda la complejidad que tiene validar y extraer información con distintos tipos de datos de forma correcta y segura.

6.2.3. El protocolo basado en JSON

Como se mencionó anteriormente, tanto para los parámetros, como para los valores de respuesta, los tipos de datos representables de forma nativa con JSON resultan suficientes para implementar una API similar a la de DuinoBot.

Internamente XRemoteBot convierte los valores, objetos y arrays de JSON a distintos tipos de datos de Python de la forma descripta en la figura 6-4, siguiendo, en general las transformaciones realizadas por defecto por las funciones de procesamiento

JSON	Python
`string`	`str`
`number`	`int` o `float`[8]
`object`	`dict`
`array`	`list`
`true/false`	`True/False` (bool)
`null`	`None`

Figura 6-4: Relación entre los tipos y valores de JSON y los usados en Python

de el módulo JSON de la biblioteca estándar de Python[7].

6.2.4. Mensajes del cliente al servidor

Los mensajes al servidor son objetos JSON con dos campos obligatorios y dos opcionales. Los campos que componen a estos objetos son:

`entity:` cuyo valor es un string que describe conceptualmente el receptor del mensaje (obligatorio).

`method:` cuyo valor es un string con el nombre del método a invocar (obligatorio).

`args:` cuyo valor es un array posiblemente vacío con los argumentos que deben pasarse al método invocado (opcional).

`msg_id:` si está presente el servidor incorpora este campo con su valor original en la respuesta. Sirve para identificar a qué petición corresponde cada respuesta en implementaciones asincrónicas como la de Javascript.

En la actual implementación los valores posibles del campo `entity` son `"global"` y `"robot"`. El primero representa acciones globales como el login y el último representa una abstracción de los robots soportados por XRemoteBot.

La entidad "global" soporta los métodos:

`authentication_required:` No recibe argumentos. Retorna el valor `true` si el servidor está configurado para requerir autenticación y `false` en caso contrario.

[7] `https://docs.python.org/3/library/json.html`

`authenticate:` Recibe un string como argumento. Si el string se corresponde con la `api_key` (una clave generada de forma aleatoria que identifica a un usuario específico) de algún usuario y esa `api_key` no ha expirado retorna el valor `true` y `false` en caso contrario.

Entre otros la entidad "robot" soporta los métodos:

`backward:` Recibe un objeto que identifica a un robot específico, una velocidad y un tiempo en segundos. Mueve el robot hacia atrás el tiempo indicado y luego contesta al cliente un valor `null`. Si no se envía el parámetro de tiempo el robot se mueve de forma indefinida.

`forward:` Recibe un objeto que identifica a un robot específico, una velocidad y un tiempo en segundos. Mueve el robot hacia adelante el tiempo indicado y luego contesta al cliente un valor `null`. Si no se envía el parámetro de tiempo el robot se mueve de forma indefinida.

`turnLeft:` Recibe un objeto que identifica a un robot específico, una velocidad y un tiempo en segundos. Gira hacia la izquierda el tiempo indicado y luego contesta al cliente un valor `null`. Si no se envía el parámetro de tiempo el robot se mueve de forma indefinida.

`turnRight:` Recibe un objeto que identifica a un robot específico, una velocidad y un tiempo en segundos. Gira hacia la derecha el tiempo indicado y luego contesta al cliente un valor `null`. Si no se envía el parámetro de tiempo el robot se mueve de forma indefinida.

`getObstacle:` Recibe un objeto que identifica a un robot y, opcionalmente una distancia. Retorna el valor `true` si hay un obstáculo a una distancia menor o igual a la indicada. Si no se envía la distancia se asume un valor por defecto, este valor por defecto depende del modelo de robot usado ya que los sensores tienen rangos diferentes.

`getLine:` Recibe un objeto que identifica a un robot y retorna un `array` con los valores de los sensores de línea del robot.

Opcionalmente un modelo dado de robot puede soportar otros métodos o retornar error en alguno de los métodos anteriores. El protocolo es fácilmente extensible ya que simplemente modela un pasaje de mensajes con argumentos, para agregar nuevos mensajes basta con enviar distintos strings en el campo “method” y para soportar nuevos tipos de entidades simplemente se puede enviar un string diferente en “entity”.

Por ejemplo, si se requiriese implementar un sistema que controle la iluminación de una casa, se podrían definir los métodos: “main_bedroom”, “children_bedroom”, “kitchen”, “living” y “garage”. Si cada uno de éstos controla la luz de la habitación descripta por su nombre, podrían recibir como argumento la intensidad de luz deseada como un número de 0 a 255. Finalmente podemos agrupar estos métodos en una “entity” con el nombre“houselight”.

Podríamos controlar las luces con una secuencia de mensajes como los mostrados en el código 6.1. Por supuesto, luego habría que agregar soporte para estas funcionalidades en el servidor y los clientes pero esto tampoco es un problema significativo en XRemoteBot.

```
{
    "entity": "houselight",
    "method": "main_bedroom",
    "args": [60],
}
{
    "entity": "houselight",
    "method": "garage",
    "args": [0],
}
{
    "entity": "houselight",
    "method": "living",
    "args": [255],
}
```

Código 6.1: Ejemplo de una posible extensión al protocolo para controlar las luces de una casa

Los métodos bloqueantes como `backward` (que con DuinoBot demoran la ejecución del proceso hasta que transcurra el tiempo indicado como parámetro) son modelados desde el servidor demorando la respuesta al método gracias a que el framework web Tornado (usado para dar soporte al servidor para los protocolos HTTP y WebSockets, entre otras utilidades) funciona con un "bucle principal" ("main loop") donde se pueden encolar trabajos para que se ejecuten en el futuro. Esta funcionalidad no es necesaria para la mayoría de los clientes que pueden implementar las demoras de forma local usando la función `sleep()`[9], pero es necesario para la implementación en Javascript donde no es posible demorar la ejecución del script sin impactar de forma negativa el rendimiento del navegador (al menos en la pestaña actual).

6.3. Mensajes del servidor a los clientes

El servidor responde a los clientes con mensajes que contienen una única entrada `response`. Esta entrada identifica el tipo de respuesta que puede ser `value` o `error`. Si el cliente envía un campo `msg_id` en la petición, el servidor incorporará además un campo `msg_id` idéntico en la respuesta correspondiente a esa petición (como se mencionó anteriormente para soportar clientes asincrónicos de forma correcta).

6.3.1. Mensajes tipo value

Los mensajes `value` representan los valores de respuesta de los métodos invocados.

Los campos de los mensajes tipo `value` son:

`response`: identifica el tipo de mensaje. En este caso es un string con el valor `"value"`.

`value`: el valor retornado por el método (puede ser cualquier valor soportado por JSON).

[9] Tanto en Ruby como en Python la función `sleep()` detiene el hilo actual la cantidad de segundos que indique su único parámetro.

`msg_id`: si la petición contiene el campo `msg_id` se copia el mismo campo, sino este campo se omite.

Para los mensajes que requieren un tiempo de demora el servidor contestará con el mensaje `value` una vez transcurrido ese tiempo. Para no demorar la atención de peticiones de otros clientes el servidor utiliza las funcionalidades provistas por Tornado para atender peticiones de forma asíncrona [Dory et al., 2012].

6.3.2. Mensajes tipo error

Los mensajes `error` representan eventos anómalos que generalmente en un sistema no distribuido se modelarían utilizando excepciones, por ejemplo errores de codificación, peticiones a recursos ocupados, etc...

Campos de los mensajes `error`:

`response`: identifica el tipo de mensaje, en este caso es un string con el valor `"error"`.

`message`: un mensaje de error descriptivo.

`msg_id`: si la petición tenía `msg_id` se copia el mismo, sino este campo se omite.

6.4. Ejemplos de interacción entre los clientes y el servidor

Al comenzar la conexión entre el cliente y el servidor, se intercambian los primeros mensajes donde el cliente pregunta al servidor si el mismo requiere autenticación y si es así envía la *API key* correspondiente. En la figura 6-5 se detalla el intercambio de mensajes entre un cliente y el servidor en un caso de autenticación exitoso con un cliente que envía un `msg_id`. Como se puede ver el `msg_id` no tiene por qué ser consecutivo con el del mensaje anterior, incluso puede llegar a ser un string, las únicas restricciones sobre el `msg_id` son que tiene que ser copiado por el servidor en las respuestas y que el valor debe poder ser utilizado como identificador de un

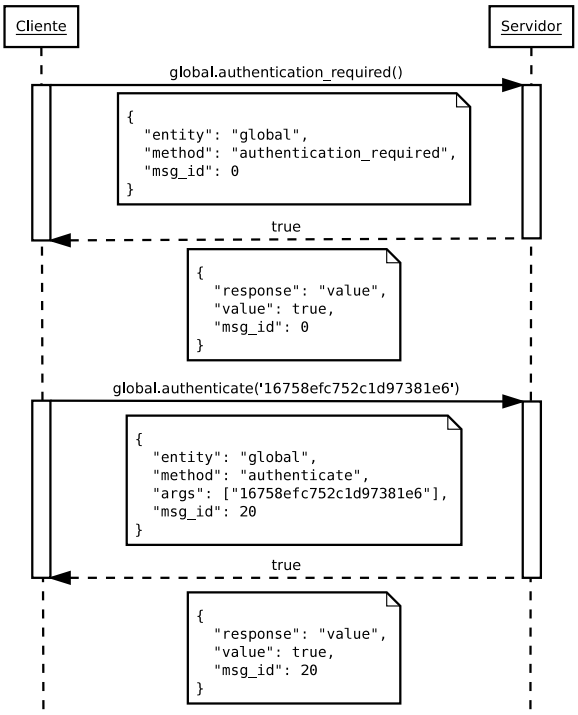


Figura 6-5: Ejemplo de secuencia de autenticación

atributo (usando notación con corchetes) en un objeto Javascript. Es decir puede ser un número o un string arbitrario[10].

Otra secuencia de operaciones típica para XRemoteBot es mover un robot y luego pedir los valores de alguno de sus sensores, en la figura 6-6 se puede ver un caso donde se mueve el robot hacia adelante durante 1 segundo y luego consulta si existe algún obstáculo a 10 centímetros o menos del robot, en el primer argumento de los mensajes enviados a la entidad *robot* se puede ver el objeto que identifica al robot específico a utilizar.

Por último, en la figura 6-7 se puede ver un mensaje de error típico cuando el cliente envía una petición mal formada donde no existe el campo *method* obligatorio, notar que este mensaje a diferencia de los anteriores no tiene un campo *msg_id* pero esto no causaría ningún error ya que este campo no es obligatorio, el único error en la petición es la ausencia de *method*.

[10] `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Property_Accessors`

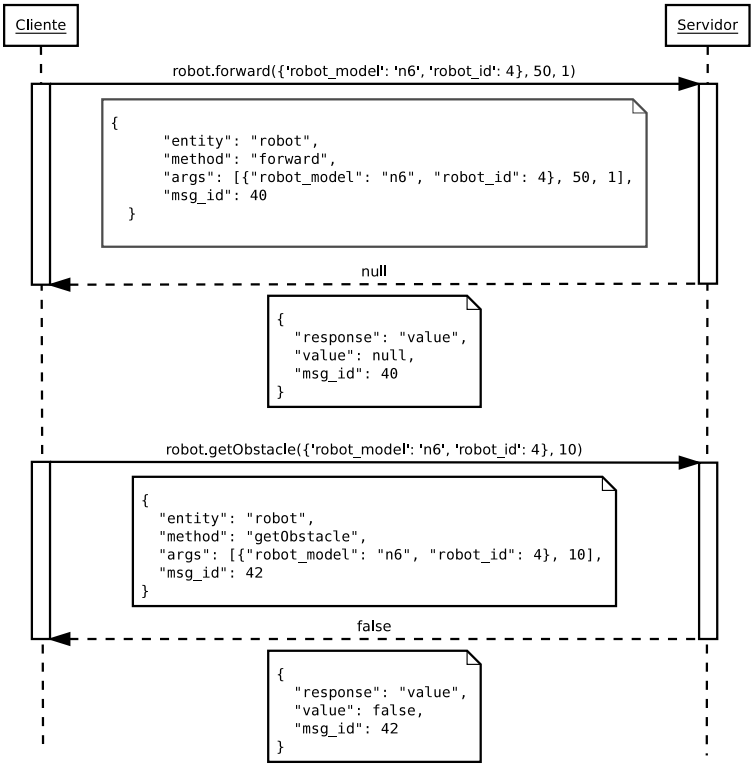


Figura 6-6: Ejemplo de movimiento y acceso a sensores de un robot

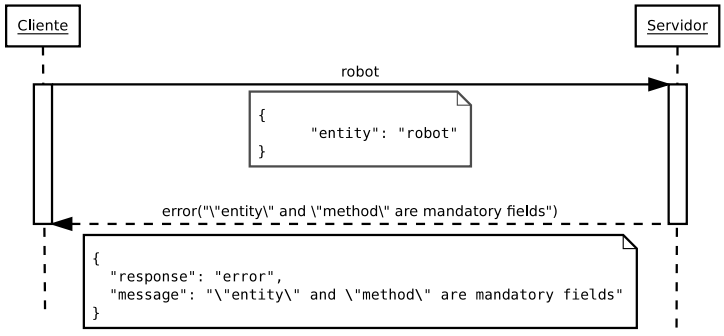


Figura 6-7: Ejemplo de un mensaje de error ante una petición mal formada

6.5. Modalidades del servidor

A fin de hacer que el servidor pueda exponerse al público se cuenta con un sistema con autenticación por *API key*, pero estas claves son largas resultando difícil recordarlas y escribirlas sin errores. En ámbitos locales, como puede ser una red wifi en un aula, este mecanismo de autenticación puede resultar molesto e innecesario, por lo que el servidor es configurable de forma tal que se puede deshabilitar este sistema de autenticación. Para que el servidor opere sin requerir una *API key* a los clientes basta con configurar la opción `public_server` en `False` como se indica en la sección 4.2.1. Esto también deshabilita el sistema de reservas, por lo que cualquier usuario puede usar cualquier robot en este modo.

Dado que existe esta modalidad el protocolo cuenta con un mensaje para preguntar si es necesario proseguir con la autenticación (`authentication_required`), con lo cuál se pueden crear scripts que se adapten a ambos escenarios.

Capítulo 7

Pruebas

El servidor se probó con Python 2.7 (portarlo a Python 3 involucraría portar DuinoBot, Myro y otros módulos asociados), en entornos creados con virtualenv[1] y con una base de datos SQLite[2] sobre sistemas Lihuen 5 y 6 beta[3] (basados en Debian Wheezy y Jessie[4] respectivamente). Las dependencias se instalaron declarándolas en `setup.py`[5] e instalándolas con pip[6], a excepción de los módulos DuinoBot y Myro que fueron instalados desde repositorios Git[7] usando también pip, pero indicando la URL manualmente ya que no fue posible declararlos de esta manera en `setup.py`, el código 7.1 muestra como instalar el servidor y los paquetes necesarios para soportar robots Multiplo N6 y Scribbler.

```
1 git clone https://github.com/fernandolopez/xremotebot.git
2 cd xremotebot
3 virtualenv .
4 . bin/activate
5 pip install -r requirements.txt
6 pip install\
```

[1] `http://www.virtualbox.org/`
[2] `http://www.sqlite.org/`
[3] `http://lihuen.linti.unlp.edu.ar`
[4] `http://debian.org`
[5] `setup.py` es el archivo donde se definen los datos y dependencias los paquetes Python
[6] `pip` es uno de los instaladores de paquetes Python más utilizados y permite instalar paquetes desde repositorios remotos con solo indicar los nombres de los mismos `https://pip.pypa.io/en/latest/user_guide.html`
[7] `http://git-scm.com/`

```
  'git+https://github.com/Robots-Linti/duinobot.git@pygame_opcional'
pip install 'git+https://github.com/fernandolopez/Myro.git'
```

Código 7.1: Instalación de XRemoteBot

```
apt-get install libav-tools nodejs
npm install -g ws
```

Código 7.2: Instalación del soporte de video para XRemoteBot

Para soportar el streaming de video además es necesario instalar `avconv`[8] para capturar y codificar el vídeo, `NodeJS`[9] para ejecutar el script que transmite el vídeo y el paquete `ws`[10] que es una dependencia de este script, se pueden instalar de distintas formas dependiendo del sistema operativo utilizado, en código 7.2 muestra como instalarlo en un sistema basado en Debian GNU/Linux.

Finalmente para ejecutar todo es conveniente utilizar el script `run` provisto con el servidor que se encarga de ejecutar avconv, el servidor de jsmpeg y de asociar las MACs de los robots Scribbler que hubiera configurados con dispositivos `rfcomm` de manera que queden utilizables para `Myro`. Luego de ejecutar esas tareas el script `run` ejecuta el servidor XRemoteBot.

```
./run
```

Código 7.3: Ejecutar el servidor y procesos asociados

Si hubiese algún robot Scribbler en la configuración el script `run` pedirá la contraseña de administrador para asociar su MAC con un dispositivo `rfcomm`.

7.1. Pruebas de uso de recursos

Se dejó el servidor ejecutándose durante aproximadamente 24hs en un equipo con un procesador "AMD Athlon(tm) 64 X2 Dual Core Processor 4600+" y 2GB de memoria RAM. Durante ese tiempo hubo un cliente conectado controlando un robot

[8] `https://libav.org/`
[9] `https://nodejs.org/`
[10] `https://www.npmjs.com/package/ws`

(que recibió energía de una fuente ATX para evitar problemas con las pilas) realizando movimientos cada determinada cantidad de segundos. Esporádicamente se verificó con la herramienta `htop` el uso de recursos general de XRemoteBot y herramientas asociadas. En estas pruebas ni XRemoteBot ni el servidor de jsmpeg mostraron valores significativos en el uso de CPU y memoria en el sistema, en cambio la herramienta `avconv` consumió de manera prácticamente constante un 20 % de un núcleo de la CPU y un 0.3 % de memoria. Al ejecutar un cliente en este mismo equipo en el navegador Mozilla Firefox 40.0a1 (Nightly) el uso de CPU por parte de este navegador estuvo alrededor del 25 % y al realizar la misma actividad en Google Chrome 41.0.2272.101 (Stable) el uso de CPU por parte del navegador fue de alrededor del 40 %.

Eliminando el elemento *canvas* usado para mostrar el streaming de video el uso de CPU se redujo a un 0.5 % en Firefox y 30 % en Google Chrome, por lo que aparentemente ese uso de CPU se encuentra relacionado con la decodificación y renderizado de video.

7.2. Pruebas con la interfaz Javascript

Estas pruebas llevaron a algunas modificaciones de la interfaz web, la API Javascript y el servidor en base a la experiencia de uso, por ejemplo:

- Se incorporó la funcionalidad al servidor que cuando el cliente se desconecta del WebSocket, se liberan todas sus reservas. Esto es una cuestión de gustos, podría mantenerse también la reserva sin problema hasta su vencimiento, pero haciendo pruebas con una cantidad limitada de robots se hizo notorio que un sistema así desperdicia mucho el tiempo ocioso del robot.

- Dado que al cerrar el WebSocket se pierde la reserva se hizo que también se envíe el mensaje `stop()` a cada robot reservado por el usuario, de manera que el robot no se quede en movimiento una vez terminada la reserva.

- Se incorporó una caja con tips debajo de la vista de la cámara, en la misma van rotando una serie de consejos acerca de aspectos de la API Javascript que

pueden resultar poco familiares a una persona que no utiliza este lenguaje con frecuencia y otros muy específicos del control de los robots con XRemoteBot.

- Se modificó la forma de evaluar el código, para que se ejecute en un contexto donde las funciones `setInterval()` y `setTimeout()` se encuentran ocultas por otras implementaciones que garantizan que cada vez que se presione el botón "Ejecutar" se eliminen todos los timeouts e intervalos previamente creados. De otra manera si el usuario escribe un script que usa estas funciones (cuestión que parece ser conveniente en este entorno) y luego escribe otro, los timeouts e intervalos de la versión anterior se seguirían ejecutando interfiriendo con el comportamiento del script actual.

7.2.1. Pruebas de codificación

Dado el uso de Promises en el diseño de este cliente, fue necesario repensar la forma de resolver los ejercicios habituales. Por ejemplo en el código 7.4 se ve un script que mueve el robot, en el código 7.5 se ve un algoritmo para que el robot avance esquivando obstáculos y en el código 7.6 muestra un script que mueve el robot en direcciones aleatorias.

```
1  var server = new Server('ws://xremotebot.example:8000/api',
2                          'api_key');
3  server.onConnect(function(){
4      server.fetch_robot().then(function(robot_obj){
5          var robot = new Robot(server, robot_obj);
6          robot.forward(50, 5);
7          robot.turnLeft(50, 2);
8          robot.turnRight(50, 2);
9          alert('Hola');
10     });
11 });
```

Código 7.4: Secuencia de movimientos

```
1  var server = new Server('ws://xremotebot.example:8000/api',
```

```
                        'api_key');
server.onConnect(function(){
    server.fetch_robot().then(function(robot_obj){
        var robot = new Robot(server, robot_obj);
        var esquivar = function(){
            robot.getObstacle().then(function(obstacle){
                if (obstacle){
                    robot.backward(40, 1);
                    if (Math.random() < 0.5){
                        robot.turnLeft(40, 1);
                    }
                    else{
                        robot.turnRight(40, 1);
                    }
                    robot.forward(40);
                }
                setTimeout(esquivar, 500);
            });
        };
        robot.forward(40);
        setTimeout(esquivar, 500);
    });
});
```

Código 7.5: Ante un obstáculo el robot lo esquiva girando en un dirección aleatoria

```
var server = new Server('ws://190.16.204.135:8000/api',
                        '8d4ae089-443b-4bda-bd04-6601d664dd96');
server.onConnect(function(){
    server.get_robots().then(function(robots){
        println(robots);
    });


    server.fetch_robot().then(function(robot_obj){
        println(robot_obj);
        var robot = new Robot(server, robot_obj);
```

```
12          setInterval(function(){
13              var rand = Math.random();
14              if (rand < 0.25){
15                  robot.forward(50, 1);
16              }
17              else if (rand < 0.5){
18                  robot.backward(50, 1);
19              }
20              else if (rand < 0.75){
21                  robot.turnLeft(50, 1);
22              }
23              else{
24                  robot.turnRight(50, 1);
25              }
26          }, 10000);
27      });
28  });
```

Código 7.6: El robot se mueve en direcciones aleatorias

Al ejecutar el código 7.4 se puede ver que la función `alert()` se ejecuta mucho antes que el robot termine de moverse. De hecho lo hace de manera casi instantánea al presionar "Ejecutar". Esto es porque si bien se implementó un mecanismo donde el cliente antes de enviar cada mensaje espera a que el servidor conteste el anterior para simular una ejecución secuencial y para simular funciones bloqueantes que demoran la ejecución del script, realmente el cliente no deja de ser asincrónico. La ejecución de `forward()` y cada uno de los métodos del robot en realidad solamente instancia y retorna una "promesa", por lo que el tiempo que tardan en ejecutarse estos movimientos es insignificante, pero vemos que el robot se mueve como si cada uno de estos métodos congelara la ejecución del script por este mecanismo que tiene soporte tanto del lado del cliente como del lado del servidor y que demora la respuesta de algunos métodos a propósito.

```
1  var server = new Server('ws://xremotebot.example:8000/api',
2                          'api_key');
```

```
3  server.onConnect(function(){
4      server.fetch_robot().then(function(robot_obj){
5          var robot = new Robot(server, robot_obj);
6          robot.forward(50, 5).then(function(){
7              robot.turnLeft(50, 2);
8          }).then(function(){
9              robot.turnRight(50, 2);
10         }).then(function(){
11             alert('Hola');
12         });
13     });
14 });
```

Código 7.7: Secuencia de movimientos usando `Promise#then()`

Es posible hacer una versión donde `alert()` se ejecute cuando el robot termine de moverse, pero el código de esta versión (código 7.7) es un poco más engorroso ya que tiene muchos niveles de anidamiento.

En el código 7.5 a simple vista se ven diferencias significativas de la forma de codificar con esta API en comparación con DuinoBot e incluso en comparación con los otros clientes de XRemoteBot. Lo más importante es que dado que los métodos retornan "promesas" no se puede usar un `while` o `for` para repetir una acción por ejemplo mientras no hay obstáculos, por lo que hay que recurrir a `setInterval()` o bien a `setTimeout()` para tener más control.

El código 7.6 usa la misma estrategia que el ejemplo anterior usando `setInterval()` para realizar una tarea que con DuinoBot en Python se haría con un `while` y además accede a la API Math del navegador.

7.3. Pruebas con los clientes Python y Ruby

Estos clientes tienen una complejidad menor que el cliente Javascript por lo que fueron rápidamente probados sin mayores dificultades.

La API de estos clientes se asemeja tanto a la API original de DuinoBot que no

fue necesario repensar como hacer las actividades típicas con los robots como lo fue en la versión Javascript.

Capítulo 8

Conclusiones y trabajo a futuro

Esta tesina permitió desarrollar una aplicación web basada en tecnologías abiertas, llamada XRemoteBot, que permite manipular y visualizar en tiempo real los robots Multiplo N6 de la empresa RobotGroup y que se aplicará al proyecto "Programando con robots y software libre". Se implementaron tres clientes, para tres lenguajes de programación: Python, Ruby y Javascript, que permiten ampliar los alcances del proyecto, por un lado incorporando nuevos lenguajes de programación para programar los robots y por otro lado, facilitando la incorporación de nuevos destinatarios del proyecto, entre ellos escuelas que no disponen de los robots Multiplo N6 para el desarrollo de su práctica y aquellas que aún contando con los robots podrían realizar actividades en forma remota.

XRemoteBot está desarrollado en Python y utiliza las siguientes tecnologías:

- Tornado como framework web y soporte de WebSockets del lado del servidor.
- SQLAlchemy para el acceso a la base de datos.
- DuinoBot para controlar los robots "N6".
- Myro para controlar los robots "Scribbler".
- Sniffer[1] y Nose[2] para correr los tests de forma automatizada.

[1] `https://pypi.python.org/pypi/sniffer`
[2] `https://nose.readthedocs.org/en/latest/`

- Diversos módulos de la biblioteca estándar.

El framework Tornado resultó ser adecuado para el uso de WebSockets y el manejo de demoras en las respuestas debido al soporte incorporado para WebSockets; a su vez cuenta con diversas utilidades para la atención de peticiones de forma asincrónica.

El protocolo WebSocket fue elegido ya que está estandarizado y su uso se está popularizando cada vez más, y si bien la API WebSockets está en proceso de estandarización es soportada por los navegadores más usados en la actualidad.

Para disminuir la cantidad de bytes de información enviados sin impactar en el uso de CPU tanto del servidor como de los clientes, se hizo un estudio con múltiples pruebas en Python y Javascript con los mecanismos de serialización JSON, CBOR y BSON, el primero de éstos de texto plano y soportado de forma nativa en los tres clientes implementados y los restantes mecanismos son binarios. Estas pruebas, presentadas en la sección 6.1, dieron como resultado que el uso de JSON no genera mensajes significativamente más grandes que las otras alternativas analizadas (incluso son ligeramente más pequeños que los generados con BSON), a la vez que el tiempo para codificar y decodificar los mensajes con JSON en Javascript es menor que las otras alternativas y en Python no presenta diferencias significativas para mensajes pequeños como los utilizado por XRemoteBot.

La API de XRemoteBot fue probada con 3 clientes distintos incorporados en este trabajo, de los cuales el cliente Javascript se puede ejecutar desde el navegador (inclusive desde dispositivos móviles) y los clientes Python y Ruby se pueden usar en cualquier computadora donde se puedan instalar sus intérpretes (abarcando computadoras con los sistemas operativos GNU/Linux, *BSD, Windows y Mac).

Los clientes Ruby y Python son los más simples y es posible extenderlos fácilmente para agregarles nuevas funcionalidades.

El sistema resultante es fácilmente extensible en tres aspectos:

- Dada su sencillez el protocolo es fácilmente extensible.
- Definiendo un script en “remotebot/robots/<nombre>.py” en el servidor y agregando “<nombre>” en la opción “robots” de la configuración es suficiente

para agregar soporte para un nuevo robot.

- Si se requiere controlar un dispositivo tan distinto de los robots que requeriría métodos muy distintos se pueden definir nuevas entidades, una entidad simplemente es una clase que hereda de "remotebot.models.entity.Entity" y que se registra al final del archivo "remotebot/handlers/ws_handler.py" con el método `WSHandler.register_api_handler()`. En el método anterior recibe como primer argumento el nombre de la entidad y como segundo argumento una instancia de una clase que herede de "remotebot.models.entity.Entity".

XRemoteBot contribuirá como dispositivo educativo a una de las líneas de investigación del LINTI vinculada con acercar la programación a la escuela. Específicamente se utilizará en el proyecto "Programando con robots y software libre" brindando la posibilidad de contar con una herramienta tecnológica que permita trabajar en forma remota, ofreciendo la oportunidad de trabajar con escuelas de puntos geográficos alejados y con aquellas que no cuenten con robots reales.

Todos los desarrollos implementados se han publicado con una licencia MIT y pueden ser descargados desde el sitio GitHub[3].

8.1. Trabajo a futuro

Habiendo alcanzado la etapa final del desarrollo de XRemoteBot se plantean una serie de posibles trabajos derivados y extensiones que permitirían adaptar este sistema para nuevos requerimientos:

- Extender el soporte de lenguajes de programación en la interfaz agregando, por ejemplo, soporte a Python y Ruby con Brython u Opal.

- Analizar la posibilidad de añadir soporte a la programación con visual con la biblioteca `Blockly`[4].

[3] `http://github.com/fernandolopez/xremotebot` y `http://github.com/fernandolopez/xremotebot-clients`

[4] `https://developers.google.com/blockly/`

- Analizar la posibilidad de integrar la interfaz web de XRemoteBot con sistemas de gestión de contenido para educación como Moodle[5].
- Analizar tecnologías alternativas para el streaming de video en redes con anchos de banda relativamente bajos.

[5] `http://moodle.org`

Bibliografía

Dory, M., Parrish, A., y Berg, B. (2012). *Introduction to Tornado*. O'Reilly Media, Inc.

Díaz, J. F., Banchoff Tzancoff, C. M., Martin, S., y López, F. (2012). Aprendiendo a programar con juegos y robots. `http://hdl.handle.net/10915/19307`.

ECMA (2013). *ECMA-404: The JSON Data Interchange Format*. First edition. `http://www.ecma-international.org/publications/standards/Ecma-404.htm`.

ECMA (2015). *ECMA-262: ECMAScript 2015 Language Specification*. Sixth (draft march 17, 2015) edition. `http://wiki.ecmascript.org/doku.php?id=harmony:specification_drafts`.

Goldberg, K. Y., Gentner, S., Sutter, C., y Wiegley, J. (2000). The mercury project: a feasibility study for internet robots. *IEEE Robot. Automat. Mag.*, 7(1):35–40. `http://goldberg.berkeley.edu/pubs/ramag.pdf`.

Grigorik, I. (2013). *High-Performance Browser Networking*. O'Reilly, 1st edition.

IETF (2013). RFC 7049 - Concise Binary Object Representation (CBOR). `http://tools.ietf.org/html/rfc7049`.

IETF (2014). RFC 7159 - the JavaScript object notation (JSON) data interchange format. `https://tools.ietf.org/html/rfc7159`.

Kumar, Deepak Traducido al español por Guichal, C. (2009). El mundo de los robots. Nota: Esta es una traducción de "Learning Computing With Robots" realizada por Celia Guichal y revisada por Claudia Banchoff como parte del proyecto "Programando con robots y software libre". `http://robots.linti.unlp.edu.ar/uploads/docs/learning_computing_with_robots.pdf`.

Lanfranco, E., Bogado, J., David, V., y Da Silva Gillig, J. (2012). Modificaciones realizadas al robot multiplo n6 para permitir programación interactiva. `http://41jaiio.sadio.org.ar/sites/default/files/11_JSL_2012.pdf`.

Lutz, M. (2014). *Python pocket reference*. O'Reilly Media, Inc.

OWASP (2014). OWASP Testing Project - OWASP. `https://www.owasp.org/index.php/OWASP_Testing_Project`.

Queiruga, C. A., Banchoff Tzancoff, C. M., y López, F. (2013). RemoteBot: una aplicación que combina robots y dispositivos móviles. `http://sedici.unlp.edu.ar/handle/10915/27274`.

Roden, T. (2010). *Building the Realtime User Experience: Creating Immersive and Interactive Websites*. O'Reilly Media, Inc., 1st edition.

Thomas, D., Fowler, C., y Hunt, A. (2013). *Programming Ruby 1.9 & 2.0*. The Pragmatic Programmers.

W3C (2014). The WebSocket API. `http://dev.w3.org/html5/websockets/#the-websocket-interface`.

Wang, V., Salim, F., y Moskovits, P. (2013). *The Definitive Guide to HTML5 WebSocket*. Apress, Berkely, CA, USA, 1st edition.