



The Integration of Web Technologies and Robotics: from Robot Application Development to Social Ability

Le Kang

Supervisor

Dr. Benjamin Johnston

Supervisor

Prof. Mary-Anne Williams

*A dissertation submitted on 14 November 2016 in fulfilment of the requirements
for the degree of*

Bachelor of Science (Honours) in Information Technology

Declaration

I, Le Kang, declare that this thesis titled, ‘*The Integration of Web Technologies and Robotics: from Robot Application Development to Social Ability*’ and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Acknowledgement

Foremost, I express my sincere gratitude to my supervisors Dr. Benjamin Johnston and Prof. Mary-Anne Williams for the continuous support of my honours study and research, and for their patience, engagement, and immense knowledge. They also offer me an opportunity to get involved into an amazing social robotics project which fosters my research topic and expands my vision in the field. I could not have imagined having better mentors for my honours study.

Besides my supervisors, I would like to thank:

- Paul Fuller, Xun Wang and Meg Tonkin, for their supervision and guidance during the ATN social robotics project.
- My ATN project teammates Denis Draca, Navi Gunaratne and Jose Gunawarman, for contributing their effort to carry out a successful project.
- Magic lab team, for providing friendly research environment and giving me support, feedbacks and ideas.

Last but not the least, I would like to thank my beloved family and friends, who have been there for me throughout the entire process, keeping me motivated and offering me advices when I am in difficulties. A special thank you goes to my wife and my daughter, for their constant love and faith in me.

Abstract

This dissertation explores the possibility of combining web technologies with robotics to address the current accessibility and usability issues in communicating with robots. Through applications and interfaces developed using web technologies, robots can be accessed by users more freely and conveniently on different devices. More importantly, robots can be exposed to a broader user base as the applications and interfaces are designed to be understood and implemented by people without programming or engineering background. As a result, the users can focus on achieving their various goals with robots, rather than the complicated coding work during the development stage. Only through this way can robots participate more in people's daily life.

The first project in this dissertation presents the development and implementation of a web application for the control and configuration of the motions of a PR2 robot. The application is based on MEAN stack architecture and developed with JavaScript and HTML. A user case is presented where the PR2 robot is trained to catch a thrown ball, under human partner's command sent through via the application. This project illustrates the utilisation of web tools in robot application development which enables web developers to get involved in robot technologies. The project also provides a paradigm for implementing web development framework into robot application development.

The second project focus on improving the human-robot interaction with more user-friendly interfaces. The project involves the REEM robot to conduct promotion activities in a shopping centre. The developed web interface can be used by shop owners to design and book an activity, as well as by customers to enter the promotion activity and provide feedbacks. Various web technologies are adopted,

namely node.js, angular.js, robot web tools, cloud communications services etc. The implementation of web technologies increases the means of the social robot to communicate with its stakeholders, while enabling the stakeholders to fully utilise the robot without being exposed to the development of the interface. This project presents a web-based solution of implementing a social robot.

The dissertation also suggests future work that can be conducted in this area, such as generalising the web application or interface for implementation for various purposes. In particular, those applications and interfaces can be provided as online services to be subscribed by users in different areas. However, exploring these possibilities is beyond the dissertation's scope.

Contents

1	Motivation and Introduction	10
2	Literature Review	13
2.1	Web 2.0 and Beyond: Technologies, Trend and Impact	14
2.1.1	Overview	14
2.1.2	The Cutting-edge Web Development Technologies and Trends	16
2.1.2.1	The Rise of JavaScript	16
2.1.2.2	Representational State Transfer (REST) Web APIs	18
2.1.2.3	Web of Things (WoT)	20
2.1.3	The Future of Web and Its Possible Impacts on Robotics . .	21
2.2	Robot Web Tools: State of the Art	22
2.2.1	Rationale	22
2.2.2	Under the Hood	23
2.2.3	Tools and Applications	24
2.3	Current Applications of Web Technologies for Robotics	25
2.3.1	Robotics Education and Training	25
2.3.2	Programming by Demonstrations	26
2.3.3	Knowledge-based Cloud Robotics	27
2.4	Conclusion	27

3	Research Design	29
3.1	Identify Research Questions	29
3.2	Tools Analysis	31
3.3	Development Environment and Tools Setup	34
3.4	Research Projects Overview	35
4	Web Motion Control for PR2	37
4.1	Software Infrastructure	37
4.1.1	Structure: client-side	38
4.1.2	Structure: server-side	39
4.2	User Interface	41
4.2.1	3D Robot Model Visualisation	41
4.2.2	Joint Control Panel	42
4.2.3	Robot Vision Live Stream	43
4.3	Use Case: Recording Demonstrations for PR2 Learning Ball Catching	43
4.3.1	Design	44
4.3.2	Procedure	46
4.4	Known Limitations	48
4.5	Discussion	49
5	“Chip” in Shopping Centre	50
5.1	System Design	51
5.1.1	Overview of the Subsystems	51
5.1.2	API Framework	53
5.1.3	Authentication and Authorisation	55
5.2	The Complete Workflow	57

5.3	Interaction Between Users and Chip	59
5.4	Discussion and Future Work	61
6	Conclusion and Future Perspectives	64
A	Development Environment Setup	73
B	List of Source Codes	74
B.1	Client Side Routers Configuration in Web Motion Control Application	74
B.2	Script for controlling PR2 arms as a whole	76
B.3	Web Server Script in Web Motion Control Application	78
B.4	Visualisation of URDF Using <i>ros3djs</i>	79
B.5	Initialisation of Joint Control Panel	80
B.6	Script for Ball Detecting and Tracing Using <i>OpenCV</i>	82
B.7	Example of Using Remote Method in Loopback	83
B.8	Example of Defining Model Relationships in Loopback	84
B.9	Example of Controlling the Data Access in Loopback	86

List of Figures

2.1	The evolution of the Web	14
2.2	Most popular technologies in the 2016 StackOverflow’s annual developer survey	18
2.3	HTTP request format	19
2.4	Web of Things	20
2.5	Workflow of an application using robot web tools	23
3.1	Illustration of the MEAN stack	33
3.2	Robots used in this research	35
4.1	Overview of the infrastructure of web motion control	38
4.2	User interface of web motion control	41
4.3	User interfaces for monitor and controller	44
4.4	High-level infrastructure for the use case	45
4.5	Motions for ball catching	46
4.6	Data collection for a ball catch trial	47
4.7	Data feedback of a ball catch trial	48
5.1	UI of <i>Chip for Hire</i>	52
5.2	Loopback API explorer	53

5.3	Loopback models structure	54
5.4	Authentication strategy for Chip to acquire activity content	56
5.5	Workflow of entire system	57
5.6	Entity relationship diagram of Chip for Hire	58
5.7	UI from <i>Chip for Hire</i> for shopkeeper to view an acitivity	59
5.8	Application UI during a promotion activity	60
5.9	Email and SMS notification from Chip	61

Chapter 1

Motivation and Introduction

Robotics have greatly changed human life in various aspects, from modern warfare, hazard works and outer-space study to more traditional industries such as health care and education. As robots are becoming increasingly smart and adaptive, they are expected to be found not only in factories and laboratories but also in more daily-life environments like households and public places.

However, there are still problems to be solved for robots to be more commonplace. Typical problems include poor accessibility and usability. In terms of poor accessibility, robot application development is still limited to the domain of the experts or specialists who have considerable knowledge of robot operating system (ROS). In order to expand robotic applications on various platforms and devices, it is necessary to remove the limitation and make robotics accessible to a broader and more general groups of researchers and developers. As for the usability issue, which refers to the effectiveness and efficiency of the interaction between humans and robots, another challenge presents. When presented to novice users or the general public, the research question is how to improve the social ability of robots to minimise the social stress and difficulties, while making sure that all the communication tasks can be completed satisfactorily.

As one possible way to address the above two issues, the combination of web technologies and robotics has attracted increasing interests. The past two decades witnessed a significant development in web technologies, especially after entering Web 2.0 era, when massive user participation is allowed and a large number of

web service based applications have been widely used. These applications become very commonplace in people's daily life and provide an adequate potential user base for robotics applications.

In addition, the threshold for web application development is decreasing for developers, as the basic client-side technologies such as HTML and JavaScript are easy to get started with. Following this trend, it is suggested that this web-based solution can be used to address the aforementioned issues in robotics. The growing community of web developers in robot-related development is essential to produce applications and interfaces that will help the large user base to access robots freely, conveniently and comfortably.

This dissertation explores how the web technologies extend the impact of robotics from two aspects: robot application development and social ability. The thesis is that, integrated with web technologies, robotics will have a larger developer-based community from the web-based robot application development, and consequently, a user-friendly, cross-platform and interactive environment will be created for robot and human social activities.

The key contribution of this research is exploiting the use of web technologies in the robot-related application or interface development so that robotics are more pervasive to both web application developers and normal users. Although there are some well-developed robot web tools, the utilisation of modern web development framework into robot application has not yet attracted much attention. The projects focus on the seamless integration of web and robot application development to expose robotics to web developers who are non-experts as well as normal people who are not familiar with interacting with robot. The benefit of this research are threefold:

1. *Help web developers get involved with robot application development with web tools*
2. *Provide a paradigm for implementing web development framework into robot application development*
3. *Facilitate human-robot interaction with more user-friendly interface for social robots to provide a wider range of social services*

The first project in this dissertation develops a web application for the purpose of controlling and communicating with robot. The users of the application and robot can use a simple device such as a smart phone to send command to the robot, even when the user does not have physical access to the robot. The significance of the application lies in the fact that the application is easy to use and can be generalised further to accommodate different needs.

In addition, in the first project a use case is provided as an example of implementing such an application. In the use case, the PR2 robot is trained with ball catching skills, while controlled by the human partner through a developed application on a smartphone. By receiving commands from the human partner, the robot learns the best time to perform the grip gesture, according to the ball position and speed captured by its sensor.

The second project in this dissertation introduces the implementation of a social robot in a shopping centre to interact with customers in promotion activities. The project focuses on the communication between the robot and the shop owner, who can design a promotion activity in a simple way, book a time for the robot to conduct the activity, and receive the feedback after the activity is completed. The process is easier by using web technologies, and the communication among the stakeholders of the robot is greatly simplified.

The dissertation is organised as follows: Chapter 2 discusses the current literature on robot and robot web technologies, Chapter 3 introduces the research question of the dissertation and gives an overview of the research projects, Chapter 4 presents the first research project, and the second research project is described in Chapter 5. Finally, Chapter 6 concludes the paper with future prospectives.

Chapter 2

Literature Review

This chapter gives a summary of relevant web technologies in robotics, including their historical development, current applications and future trend. These web technologies help professionals to control and communicate with robots in a more convenient and easier way. Compared to native applications, they offer much more flexibility, are more adaptable across various devices and platforms, thus more efficient and effective for the users.

The focus of this chapter is on the impact of these web technologies on the future development of robotics, in particular how they can help expand the user community so that the robotics are more accessible to those without professional training or expertise. Only through this way can the robotics be applied to more and more industries and increasingly become common in daily life.

The chapter is organised as follows: section 2.1 focuses on the Web 2.0, including the technology itself, its trend and impact. Section 2.2 introduces the popular robot web tools. Section 2.3 discusses the current applications of web technologies for robotics including education and training, programming by demonstrations and knowledge-based cloud robotics. Section 2.4 concludes this chapter.

2.1 Web 2.0 and Beyond: Technologies, Trend and Impact

2.1.1 Overview

World-Wide Web (W3) emerged as “a pool of human knowledge” [3] in early 1990s and undoubtably has become one of the most important inventions in the 20th century. It creates a global information medium for people to share ideas through computers and mobile devices that connected with internet. Since the beginning of the web, it has influenced almost every aspect of human activities and continuously evolved along multiple directions. Nowadays, it is hard to imagine a world without web technologies.

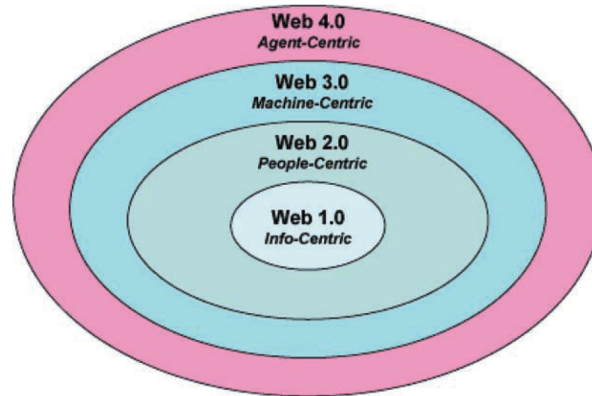


Figure 2.1: The evolution of the Web [35]

Web X.0 refers to the Xth phase in the evolution of the web. Murugesan [35] set the ongoing Web’s evolution into 4 stages (Figure 2.1): Web 1.0, which is information-centric and read-only; Web 2.0, which is people-centric and read-write; Web 3.0, which is machine-centric and semantic; and Web 4.0, which is agent-centric and intelligent. He also described the movement of Web to be “aimed at harnessing the potential of the Web in a more interactive and collaborative manner with an emphasis on social interaction”.

Web 1.0, the first generation of web, was in a one-way communication manner (read-only from sever). Nevertheless, emerged in this phase were some dynamic

features such as CMS (Content Management System) like WordPress, and commercialised sites like Amazon, eBay [29]. To some extent, these features and products accelerated the evolution of the Web in its second stage.

Today Web 2.0 has become a mainstream web technology. It was first introduced by Darcy DiNucci [11] in 1999. He pictured the web as beyond “screenfuls of text and graphics” and could work across multiple devices such as TV set, car dashboard or even microwave oven. The term was then popularised by Tim O’Reilly at the first Web 2.0 conference in 2004. He defined the web as a platform where the users could control their own data and should be trusted as co-developers [37].

The key word of the Web 2.0 is “social” which embodies both usage and development aspects. It not only provides rich and responsive user interface for sharing information, but also facilitates collaborative content creation and modification [34]. As a social revolution in the use of web technologies, Web 2.0 let users actively contribute contents online rather than just passively view web pages created by a small group of developers [28]. In addition, with spin-off technologies such as blog, wiki, mashup, social network and widgets etc, Web 2.0 also features a rapid development of web applications, thus making internet more ubiquitous and accessible.

The evolution of robots follows a similar path as web. At beginning, robots can only take commands from human partner in completing a certain task, where communication is one way (Robot 1.0). Gradually with more features enabled, current robots can sense, act and provide feedbacks to humans in a controlled environment (Robot 2.0). At this stage, both of the web and robotics emphasise on the interaction with humans. The next generation of robots should be able to work in collaboration with other robots regardless of their hardwares and platforms (Robot 3.0). And the future robots are expected to intelligently interact with humans with the ability to understand their speech, recognise their motion and predict their intention. As a result, robots can assist humans in accomplishing complex tasks (Robot 4.0).

2.1.2 The Cutting-edge Web Development Technologies and Trends

2.1.2.1 The Rise of JavaScript

In the dynamic and constantly changing world of web development, the toolbox also keeps growing with emerging new technologies. In particular, dynamic programming languages such as JavaScript, Perl, PHP, python and Ruby have received much attention. Mikkonen and Taivalsaari [32] summarised the characteristics of dynamic languages as *Dynamic typing*, *Interpretation* and *Runtime modification*. Compared with conventional static programming language such as C, C++ and Java, a dynamic languages is more flexible and malleable. More importantly, from the viewpoint of web developers, these advantages boost their effectiveness and productivity [41].

JavaScript has become one of the most commonly used programming languages on the Web and an indispensable feature of every major web browser. It was created to add dynamic contents to web pages through direct embedment into HTML code. Its simplicity and expressive power are attractive and friendly to novice developers, because of which JavaScript had a reputation of targeting "amateurs" audience at the beginning [10]. In more recent years, JavaScript starts to embrace module pattern which enables developers to build reusable and extensible components for big and scalable applications [16].

JavaScript is currently experiencing a "renaissance", with the increasing popularity of single-page application (SPA), server-side implementation and a hybrid of both. A SPA is composed of individual components that can be updated or replaced independently by using AJAX (Asynchronous JavaScript and XML) [31]. The code of the application is retrieved within a single load, so that the application renders and responds as a desktop one, since the page reloading can be avoided while transferring to other pages [6]. SPA brings better user experience by implementing more fluid and interactive web pages, yet without extra plugins or software to be installed. Due to its popularity, many JavaScript frameworks or UI components libraries have emerged to support the modern SPA development. Examples are Google's Angular.js, Twitter's Bootstrap and Facebook's ReactJS.

Another revolution of JavaScript language is Node.js, a server-side JavaScript environment using Google's V8 engine. Node.js has been adopted by over 120 of the fortune 500 companies in their everyday business operations [38]. Unlike most other server-side technologies, Node.js uses an asynchronous I/O eventing model rather than the conventional multi-threading to support concurrent execution of business logic [54]. The single-thread event-driven approach with JavaScript's highly expressive features makes the code block lightweight and efficient without having to sacrifice performance. Research has shown [27] that given the same amount of time, Node.js server can handle more requests than those based on other dynamic languages such as PHP and Python.

Moreover, with Node.js, developers can avoid language context changes when working on multi-tier web applications. One paradigm is the use of the MEAN framework (MongoDB, Express.js, Angular.js and Node.js) for full stack web application development. Compared to the traditional LAMP (Linux, Apache, MySQL, and PHP/Python/Perl) stack architecture, MEAN stack enables developers to create more agile software by using a single language across all layers of application development [17]. Also, it has been proved that a Node.js server significantly outperforms both Apache and Nginx for serving dynamic contents [5]. In addition, Express.js, a flexible and robust server-side web application framework built upon Node.js, encapsulates the lower-level Node.js interface, giving the developer a more convenient way to handle routings and HTTP operations [44].

According to StackOverflow's annual developer survey in 2016 [51], JavaScript in general remains the most popular programming language, even for back-end developers (Figure 2.2). JavaScript is now under a significant update to the ECMAScript 2015 (also known as ES6 "Harmony") which includes many new features such as native support for module and classes. The implementation of those features in major JavaScript engines is underway now. This evolution, along with the ongoing standardisation of HTML5, will lead to the creation of more advanced web applications [24]. Such a trend towards web-based software will cause a paradigm shift, that is, in the future, the use of conventional binary programs will be limited to system software, whereas the vast majority of end-user software will be developed using web technologies [52].

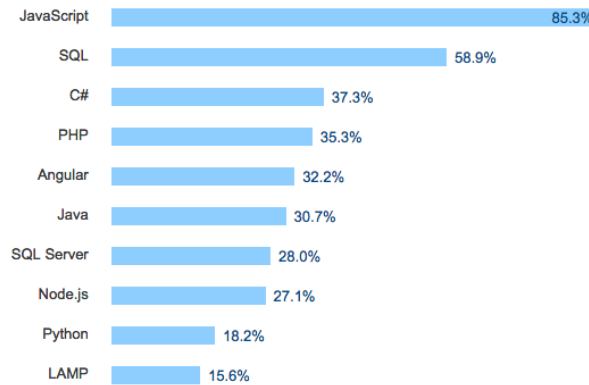


Figure 2.2: Most popular technologies in the 2016 StackOverflow's annual developer survey

2.1.2.2 Representational State Transfer (REST) Web APIs

Web 2.0 not only connects people more closely, it also improves the interoperability of software applications. Web APIs act as connectors of applications and enable distributed and autonomous software services over the Internet. The modern Web APIs, which is also referred to as the RESTful services [48], have been increasingly adopted in providing services on the Web. According to ProgrammableWeb [15], there were more than 9000 Web APIs in 2013, up from 105 in 2005, and more and more services published on Web are embracing REST. By October of 2016, ProgrammableWeb has listed more than 15800 APIs.

REST is a resource-centric HTTP protocol for data communication. The resource represents a piece of information that is identified and addressed in a given format, with XML and JSON being the most frequently adopted [49]. In the REST architecture, a REST client can access or present the resources via URIs (also known as endpoints) provided by a REST server. Software applications in various programming languages or platforms can exchange data, or more technically, perform CRUD (create, read, update and delete) data persistence, using HTTP request with verbs such as GET, PUT, POST and DELETE. Figure 2.3 depicts a HTTP request format. With the continued expansion of Web APIs, developers are more productive now as some complex processes, such as payment

and messaging, become highly reusable with minimum amount of coding.

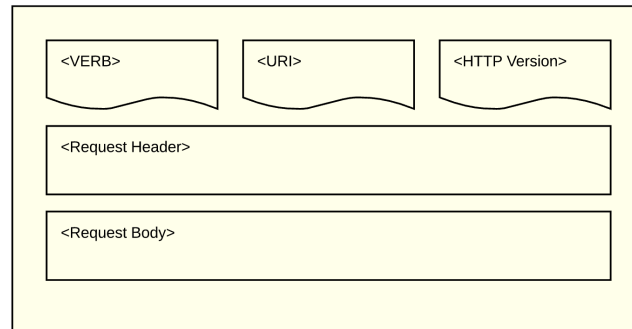


Figure 2.3: HTTP request format

Tan *et al.* [53] expounded two reasons for REST Web APIs to become the mainstream of Web services. First, the CRUD interface greatly improves consumability. Second, JSON with REST results in a much simpler and easier understanding of the communication payload, compared to the traditional WSDL/SOAP-based services. The authors also pointed out that those emerging areas, such as mobile/Internet of Things (IoT), network, Big Data and machine learning which are quickly developing and evolving, will lead to an increasing demand for Web APIs as the delivery channel.

2.1.2.3 Web of Things (WoT)



Figure 2.4: Web of Things

source: <https://www.w3.org/WoT/>

Before discussing the concept and significance of Web of Things (WoT), it is necessary to identify the difference and relationship between WoT and IoT. In terms of network layer, the WoT focuses on the interoperability at the application layer, while the IoT emphasises on the transport-layer [30]. Regarding the scope of the technology, the IoT is much greater as it refers to a global ecosystem of smart things that are able to connect to the Internet and perform computation with their sensors or actuators. However, there is no unique and universal network protocol that supports those smart things to communicate with each other seamlessly across different networking interfaces [22]. To enable the communications regardless of how they are physically connected, we need an application layer protocol that is capable of exchanging structured data. Fortunately, the ubiquitous World Wide Web with REST style architecture can be the solution. Therefore the significance of WoT is that it is a promising way for the IoT to become a reality.

The architecture of WoT is resource-oriented and all things should be exposed via RESTful Web APIs over HTTP. As a consequence, smart things and their services get transportable URIs that one can discover, reference and exchange on the web [21]. The WoT is targeting at a concrete integration of embedded devices into the web where everything can be abstracted as web services, despite their internal operating systems or driven programs. For example, a mobile application can control the home heating and cooling system based on the current and predicted temperature from public sensors connected to internet. This is a paradigm of physical mashups, based on the success of Web 2.0 mashup application, allowing for heterogenous devices to be easily combined with other virtual and physical resources [20]. As WoT is still at a preliminary state, there are some open issues such as scalability, security and privacy [59]. However, the WoT has shown its potential to change the future human life significantly, such as the “as-a-service” paradigm [7, 8] and smart industry environment [14].

2.1.3 The Future of Web and Its Possible Impacts on Robotics

It is commonly mentioned that Web 3.0 is advancing and marching toward a mainstream adoption [35]. Although there is no clear sign to indicate that a new revolution of web has taken place, such as a breakthrough in web technology or a fundamental change of how we use the web, human life in the future will no doubt be greatly changed by the technologies and features of Web 3.0 such as *Semantic Web*, *3D Web*, *Media-centric Web*, *Pervasive Web*, *Database as Web pages* [35]. They will enable people to achieve so many things with ease and convenience. For example, you may be travelling overseas and become worried about your favourite plant at home. You open an mobile application for your housekeeper, a robot, and tells him that you would like to see the plant. Then he goes to the balcony and starts a 3D live stream so you can ensure that the plant is indeed under a good care. That is because your housekeeper can read the whether sensors so that he will put the plant indoors when it is raining and water it when it is dry.

This is a scenario where a robot is able to leverage web technologies to serve people autonomously in a web-based smart environment. As outlined by Mayer [30], the possible impact on robotics from the future web is that, “Web-based smart

environments could support robotic devices in achieving their tasks and, vice versa, robots could extend the capabilities of environments that integrate smart things, i.e. devices with processing and communication capabilities”.

2.2 Robot Web Tools: State of the Art

2.2.1 Rationale

Robot application development is difficult for two main reasons. First, hardware varies for different types of robot, as a result code reuse is non-trivial [45]. Second, the complexity of robot systems makes it difficult for application programmers to be productive unless they are of high expertise [39].

The first issue has been tackled by using robot middleware systems that provide common interfaces for code sharing and reusing. ROS is a commonly adopted open source robot operating system that abstracts low-level device control and provides a communication mechanism across robot components via network protocol. ROS allows researchers and developers to build robot applications with a universal standard for different types of robots. However, the middleware system is quite large and complicated, as a result, the learning curve of ROS is steep especially for novice users [1], which leads to the second issue. Thus, in order to make robot development accessible to a broader user base, a widely used technology as an auxiliary tool is necessary to leverage the middleware framework.

As an attempt to address the above issues, in 2012, the Robot Web Tools (RWT) ¹ team officially introduced the project that enables web interfaces for robotics [39]. Such interfaces allow users to remotely access robots for development purposes via modern web browser that supports HTML5 websocket as the only advanced dependency. RWT provides a convenient abstraction to the core ROS functionality yet harnesses the power of ROS, so that it is more accessible for application programmers who are not themselves ROS users.

¹Robot Web Tools (<http://robotwebtools.org>) is a robotics and JavaScript research organisation for maintaining a collection of open-source modules and tools for building web-based robot applications.

2.2.2 Under the Hood

RWT is built upon *rosbridge*, an abstraction layer in ROS which provides a simple, socket-based programmatic access to robot interfaces and algorithms provided by ROS [9]. More concretely, given that ROS allows robots to be controlled through topic messages, *rosbridge* protocol provides access to the underlying ROS messages and services from a remote client by using serialised JSON objects. As *rosbridge* serves in a client-server paradigm, it is suitable for wide area networks and human-robot interaction at a global scale through modern web browsers [57]. *rosbridge* supports communications through either HTML5 websockets or standard TCP/IP sockets, without restriction to any programming or runtime. Figure 2.5 illustrates the workflow of a web-based application that uses RWT.

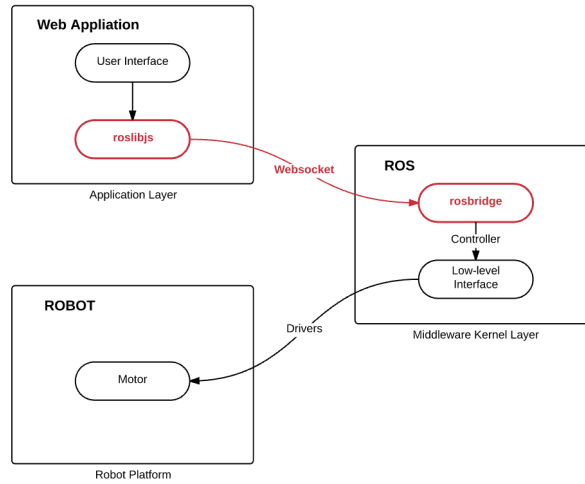


Figure 2.5: Workflow of an application using robot web tools

Another core component of RWT is *roslibjs* (formerly as *rosjs*), which is a JavaScript client library for communication of ROS topics, services and actions between browser and *rosbridge* [39]. The interfaces provided by *roslibjs* allows web developers to publish and subscribe to ROS topics in a JavaScript runtime environment as they do in ROS. It is designed to be event-based in order to make UI more responsive and decouple from other JavaScript modules [1]. In addition, it is fully compatible with server-side JavaScript environment Node.js.

The conjunction of *rosbridge* and *roslibjs* is significant for three reasons: 1)

the system-level complexity of ROS is hidden behind the abstraction layer so that developers can focus more on application level; 2) web-based robot applications are portable across platforms, therefore software update and iteration are made easier; 3) by leveraging ubiquitous web browsers, robots are accessible even to nontechnical users.

2.2.3 Tools and Applications

RWT has been used by a number of projects for developing web-based robot applications. The community of RWT has been growing tremendously along with several libraries, widgets and systems for general operations, such as task execution, visualisation and navigation. A list of popular tools and their related usages are presented as follows:

1) *ros3djs*: a 3D visualisation library for rendering ROS-related robot models on the HTML5 `<canvas>` element. It utilises the power of *three.js* [12] which is built on top of WebGL, a JavaScript API for rendering interactive computer graphics within any compatible web browser without plug-ins. The library includes standard ROS features such as interactive marker [19], Unified Robot Description Format (URDF) [23] and maps. The PR2 Remote Lab project [43] used this 3D visualisation method to display the PR2 robot model and poses, as it can provide intuitive view of robot's states with a relatively low bandwidth cost.

2) *web_video_server*: a ROS node for streaming ROS image topics via HTTP protocol. There are two transport options from ROS: MJPEG and VP8. Although VP8 is more efficient, it has an issue that results in an unavoidable delay or lag in transmission [57]. Lee [26] adopted MJPEG stream to provide vision stream from the PR2 robot in his web application, so that remote users can control the robot to perform simple object manipulation without a physical access to it.

3) *RMS (Robot Management System)*: a remote lab management tool developed by Toris [55], and designed to control ROS enabled robots from the web. It is a content management system built upon the Model-View-Controller framework *CakePHP* and backed by a MySQL databases. RMS offers a web simulation interface for researchers to conduct experiments in parallel without any concern or risk of damaging the robot or its surroundings. Furthermore, Toris [56] deployed

an instance of RMS in the project of *RobotsFor.Me* for researchers to develop, prototype, and run preliminary tests on learning algorithms, interfaces, or other methodologies in a rapid development cycle.

The tools and their implementations present a diverse set of use cases that showcase the efficiency of web-based application development for human-robot interaction. These standalone widgets or libraries help generate robot user interfaces that are more flexible and responsive. Web developers can use them in a “plug-and-play” manner without concerning the underlying technologies. The core ROS middleware requires a considerable learning curve including a general understanding of UNIX systems and languages such as C++ or Python [1]. Such requirements become obstacles for web developers to participate robotics research and development. Robot web tools as an abstraction layer expose the functionality of ROS via common web development tools such as JavaScript.

2.3 Current Applications of Web Technologies for Robotics

2.3.1 Robotics Education and Training

Robots are expensive research equipments. Take the humanoid robot PR2 as an example, it was commercially available at a price of approximately 400,000 US dollars [46]. This leads to a poor availability and accessibility of robotic hardware, making it difficult for students to learn about robot platform and perform experiments.

To expand the impact of robotics education, remote or online robot laboratories that can be accessed through web interfaces are considered as a successful solution. Djalic *et al.* [13] discussed the significance of the remote laboratory as an example of an effective E-learning tool intended for distance education of undergraduate, graduate and PhD students in the field of robotics and automation. Ordua *et al.* [36] presented a web-based framework which supports load balance and transitive sharing, so that the remote laboratories can be shared with other schools or universities. Santana *et al.* [50] introduced the Distance Laboratory System that allows

learning and adjusting predefined controllers, designing new controllers, testing and analysing the performance of the predefined/designed controllers over a set of physical devices through the Internet, where the client and the workstation are connected with a PHP powered web server. Casa *et al.* [4] presented Robotic Programming Network(RPN), a web-based extension for ROS-based remote laboratories and online robots, allowing for rapid and seamless execution of a ROS program in a remote web browser. The system can ease the learning process of the entry-level robot software programming for students.

2.3.2 Programming by Demonstrations

Programming by Demonstration (PbD), also referred to as imitation learning or learning from Demonstration, is a skill development strategy that allows a robot to acquire a new skill through demonstrated examples. As a result, the process of skill transfer can automate the tedious and complex manual programming for the robot. However, The data collection and transformation from human demonstrations often involve operations on a real robot for manual joint trajectories configuration, which makes the process inefficient and time-consuming.

The web-based approach is proved to be a promising way to address this issue. The PR2 Remote Lab Project, which has been mentioned before in 2.2.3, offers opportunities for researchers to compare and evaluate their works through web interfaces during the PbD process. Osentoski *et al.* [40] described the utilisation of the project in the Robot Learning from Demonstration Challenge held in conjunction with the AAAI-11 Conference on Artificial Intelligence. They claimed that “such technologies enable researchers to create a variety of different interactions quickly, efficiently, and in platform-agnostic fashion”

Moreover, Ratner *et al.* [47] proposed a web-based infrastructure for recording large numbers of high-dimensional demonstrations of mobile manipulation tasks. Compared to alternative approaches such as motion capture system [25] or haptic technology [42], the web-based interface coupled with light-weight simulator does not require any additional hardware and software. Their remote lab is capable of gathering demonstrations from non-experts through crowdsourcing platforms such as Amazon Mechanical Turk. Since their server can support 10 concurrent

demonstrators, the demonstration stage becomes more efficient and leverages a greater community effort.

2.3.3 Knowledge-based Cloud Robotics

Although World Wide Web was initially designed for human to share knowledge and information, robots can also take the advantage of such a opportunity to expand their capability of object cognition and task execution. RoboEarth [58] is a system aiming at building a WWW for robots and allow them to store and share knowledge and information independent of specific robot hardware. The system consists of various software components in a cloud-robot architecture. The RoboEarth DB stores object models, semantic task descriptions and maps and encode them in Web Ontology Language using typed links and URIs. Rapyuta [33] as the RoboEarth cloud engine can access the knowledge database and perform heavy computation on the data over the cloud.

Similar to RoboEarth, openEASE platform [2] is a web-based services which is remotely accessible for robot knowledge representation and processing. It aims at facilitating the use of Artificial Intelligence technology to equip robots with knowledge and reasoning capabilities. The platform consists of a big-data database, an encyclopedic knowledge base and software tools for querying, visualising and analysing. It provides a web-based graphical interface where users can retrieve, visualise and analyse the experiment data.

2.4 Conclusion

It is no doubt that the introduction of robotics has changed people's lives in many different ways. Although the development of robotics has been impressive over the last decade, accessibility and usability issues are still need to be address for robotics to be more common and helpful in daily lives in addition to professional industries. This can only be achieved with more flexible technologies compared to conventional methods.

The various web technologies mentioned above are expected to play an impor-

tant role in the expansion of robotics. The existence and development of Web allows robots to be more adaptable and accessed by users more freely and conveniently on different devices. In addition, the emerging development language and technologies provide much potentials in building relevant applications in managing and communicating with robots. The many applications of web technologies for robotics, discussed in the previous section, indicate a promising future of further utilising web tools to use robots more widely in people's daily lives.

The central benefit of the conjunction of web technologies and robotics is that, it generate interest in robotics for a larger developer community of web development. This is crucial in the expansion of robots into more common areas rather than professional industries, as the robotics research and development requires certain amount of expertise when programming on ROS-based application.

Motivated by the many advantages of web technologies, I propose to use web tools to develop robot applications, with a user-friendly interface that can be understood and implemented by people without programming or engineering background. The application will act as an important communication means between the robot and the user. Most importantly, the user is protected from the complication of the coding work hidden behind the interface, and instead can focus on achieving his goals to control and communicate with the robot.

Moreover, in addition to the development of robot applications, I also seek opportunities to operate and provide the robot application as online web services. The applications will be adapted to perform different tasks and interact with robots in distinctive ways. As a result, they can be adopted by people for various purposes. The realisation of the accessible web services will enable more people to benefit from robotic technologies, and help the development of robotics so that robots will become closer and common to ordinary people.

Chapter 3

Research Design

This chapter introduces the overall design of the research. In order to address the research gap discussed in the literature review, three research questions are identified. As the attempt to answer the research questions, two independent projects will be carried out with Rapid Application Development (RAD) Methodology. The first project is small and simple, which aims at prototyping and proof of concept; while the second project is larger, more complex and more production-ready.

The chapter is organised as follows: Section 3.1 presents the three research questions, Section 3.2 performs a tools analysis to address the first research question before looking into the research projects. The instruction to set up the tools and the environment of the research project is provided in Section 3.3, with the aim to build a solid underlying infrastructure for both of the projects. At last, Section 3.4 gives an overview of the two research projects with their background and objective.

3.1 Identify Research Questions

As mentioned and discussed in the literature review, many applications for robotics development and research are based on web technologies. Commonly those applications are design to achieve some specific goal, which can be summarised as “how to use Web technology X (such as robot web tools, cloud platform or web ser-

vices) to facilitate the production or result of Y (such as remote robots access, skill acquiring or knowledge sharing)". Although those applications are successful in fulfilling their objectives, the implementation of modern web application framework into robot software development as a general paradigm has not yet attracted much attention.

To take advantage of the many merits of web-based applications, the focus of this research is to seamlessly integrate various web technologies into robot software application, with a purpose to generalise the use of Web-applications so that they are not just simply used to solve one certain problem. I expect a generalised implementation of web-based robot application where a robot or a instance of robot environment can serve and/or consume web services from different sources with various resources, regardless of its lower level program stack in its operating system.

In order to realise this, three research questions are to be addressed:

1. *What are the tools (such as programming languages and frameworks) selected for building the web-based robot application?*

The question is tied to the underlying infrastructure of the research projects. Programming languages and frameworks for web application vary in speciality, performance and complexity. Some of them are better suited to work in a ROS-enabled environment than others. The selection of the components for the application infrastructure comes down to its ease of development, UI usability and compatibility with ROS. More specifically, since web application architecture such as MEAN stack has demonstrated success for web development, will it be suitable for robot software development as well?

2. *Will the web-based robot application improve the end-user-programmability for robotics research and development?*

Given that the potential users of the applications have limited background of ROS and restricted access to ROS-enabled robots or environment, the objective of the application is to provide intuitive UI and programmability for robot control.

3. *How can the Web-based robot application leverage other Web services to benefit the human-robot interaction and social activities?*

In order for ordinary people to be more familiar with using robots on a daily basis, one of the challenges we face is to create effective and efficient communication interfaces between humans and robots. Since many web services provide programmable APIs for communication tools such as SMS and email, how to leverage those services rather than create a new one is crucial.

3.2 Tools Analysis

As discussed in the literature review, JavaScript is one of the most popular programming languages for Web application development. Compared with other programming languages, JavaScript has following advantages to be applied in this research projects:

- *Easy implementation and testing:*

There is no requirement of platform or runtime for running client-side JavaScript application. Any web browser can execute and debug JavaScript code.

- *High customisation and interactivity:*

Thanks to AJAX (Asynchronous JavaScript and XML), SPAs (Single Page Applications) allow the users to create interactive and dynamic features as well as highly responsive interfaces without having to wait for the server to send new pages.

- *Full-stack development:*

With Node.js, Javascript is capable of performing back-end tasks. Thus, during application development process, the change in programming language contexts can be avoided.

- *Large community:*

There are many open-source libraries and frameworks for building scalable, reusable and maintainable Javascript code. Utilised well, these sources can help significantly improve the productivity, which is also a good practice for rapid application development. For developers, this is less error-prone as the code style will be consistent.

- *ROS integration:*

The majority of robot web tools are written in JavaScript, allowing for seamless interfacing with ROS from modern web browsers.

Given the above advantages, JavaScript is selected for both client-side and server-side developments. On the client side, Angular.js¹ is used as the backbone of the front-end SPA to make the UI more interactive and user-friendly. Angular.js, developed by Google, is one of the most used JavaScript framework for developing SPA. It extends HTML by defining directives in order to build dynamic views. It also supports two-way data binding so that the view and JavaScript object model are synchronised.

On the server side, Express.js² is adopted as the framework. Express.js is a minimal and flexible Node.js web application framework which is designed for building JavaScript Object Notation (JSON) based APIs. It facilitates the rapid development of Node based applications by leveraging various types of middleware, and it takes less coding and time to create complex server-side functions.

The database is enabled by MongoDB³, a NoSQL database. As the data structures for different types of robots are often dynamic (such as joint parameters and sensor messages), NoSQL database is ideal for storing data in heterogeneous format. MongoDB uses JSON-like documents with schemas and the syntax of making queries and updates is based on JavaScript. Instead of configuring local MongoDB instance, *mLab*⁴, a MongoDB hosting platform is used to avoid unnecessary development configurations.

¹<https://angularjs.org/>

²<http://expressjs.com/>

³<https://www.mongodb.com/>

⁴<https://mlab.com/>

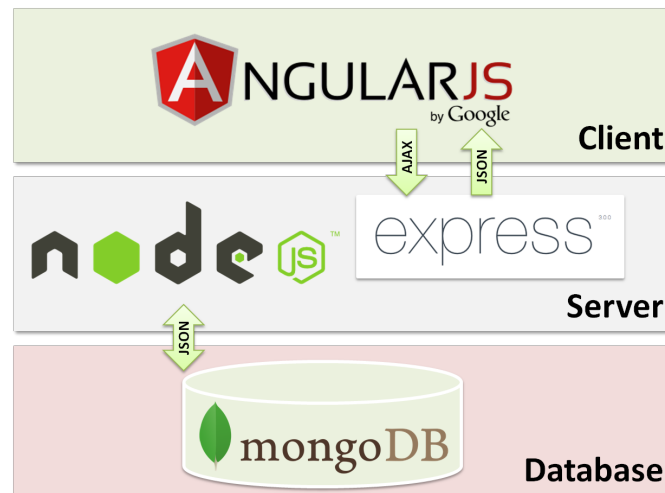


Figure 3.1: The MEAN stack

source: <http://joaopsilva.github.io/talks/End-to-End-JavaScript-with-the-MEAN-Stack/#/2>

Therefore, the underlying architecture for both research projects is the MEAN (MongoDB, Express.js, Angular.js and Node.js) JavaScript stack, as illustrated in Figure 3.1. The reasons for adopting this architecture are summarised in the following:

- **Isomorphic code:** JavaScript runs in both client-side and server-side. It is much easier for code maintenance.
- **Unified data communication format:** JSON as a common data format for communication through web application, ROS and database.
- **Simplified server layer:** Express.js is a flexible, robust yet lightweight server-side framework that exposes node functionalities via simple APIs.
- **Rich UI features:** Angular.js provides features for supporting versatile and dynamic views for SPA, such as two-way data binding, view template, dependency injection and programmable directive.

3.3 Development Environment and Tools Setup

The development environment is the Unix-like *Macintosh operating systems* with latest version (v6.8.0) of Node.js installed. Node.js provides a package manager called *npm*⁵ for installing, upgrading, configuring and removing packages. A package is just a directory of reusable code which is also referred to as modules. Every npm package has a file called *package.json* with meta data about the package. A typical Node-based application is initialised as a npm package and will depend on dozens or hundreds of packages, and those dependencies are stored in the *node_modules* folder.

The first package used is another package manager *bower*⁶. While *npm* is mainly for handling development tools (such as *bower*) and back-end packages, *bower* is solely for managing the front-end dependencies, which are kept in the *bower_components* folder and tracked with a manifest file *bower.json*. The utilisation of both packages helps to set a clear boundary for package management where client- and server-side are divided, so that application tiers can be more isolated and decoupled.

With *npm* and *bower*, consuming dependencies will be more convenient. However, to make them more organised and avoid cumbersome manual configuration in a large project, an automated task runner is necessary. Gulp.js⁷ is a development tool that fulfils the requirement. Gulp.js is a file streaming framework for automating repetitive tasks such as bundling and minifying JavaScript libraries and Cascading Style Sheets (CSS), running code analysis and copying files to an output directory. As the second research project is very complex, Gulp.js will help to solve the problem of repetition during product building and deploying process.

The final part is to create an isolated ROS environment. It is hosted on Ubuntu 12.04 (Precise) operating system in a virtual machine. The ROS version for this research is **Hydro**, which is commonly used by various robot platforms. To integrate with web applications, *roslaunch*, *web_video_server* and *tf2_web_republisher* packages are installed.

⁵<https://www.npmjs.com/>

⁶<https://bower.io/>

⁷<http://gulpjs.com/>

The detailed setup instruction is listed in Appendix A.

3.4 Research Projects Overview

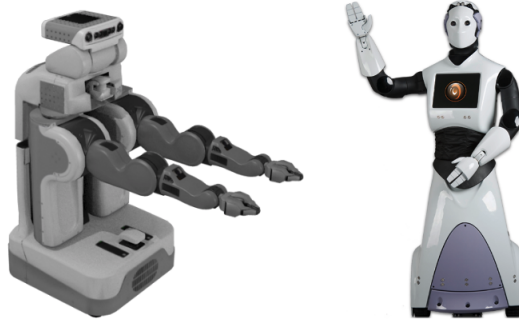


Figure 3.2: Robots used in this research: PR2 (left) and REEM (right)

source: <http://openrobots.org> and <http://pal-robotics.com/>

The first research project is a simple web-based motion control application for PR2 humanoid robot, which addresses research question 2. It presents a application that integrates modern web application framework with a ROS environment for robot software development. The design helps web developer get involved with robotics with pure JavaScript and HTML.

Application “Web motion control for PR2” aims to give end-users an intuitive web interface to control and configure gestures or motions for high degree-of-freedom (DOF) robot PR2. Although ROS packages such as *RViz* and *MoveIt* provide more advanced functionalities for motion planning, they are platform dependent and the configurations are complex. This application aims to cut off those requirements and allow the users to program with simple UI tools. The end-users are isolated from the complicated ROS environment so that they can focus on getting the desired results. The application also provides 3D visualisation of the robot model so users can receive real-time feedback. This project is a prototype of web-based robot application development environment. As a proof of concept, it

also examines the feasibility for implementing more advanced robotics development into a web-based environment.

The second research project gives a complete web-based software solution for enabling REEM humanoid robot to promote products for shopkeepers and collect customer feedback in a shopping centre, which addresses research question 3. The web-based distributed system broadens the social means for service robot to interact with ordinary people. Compared to the first project, the second project has more ambitious objectives. It attempts to explore the potential use of robots in a business context and presents a distributed web system running on various platforms including the REEM robot platform called *Chip*, a full-sized humanoid service robot. The web system offers a commercially viable solution that allows Chip to work in a shopping centre to distribute product sample and conduct surveys autonomously. Chip serves the tenants of the shopping centre as their employee and interacts with customers as a sale person. The main communication tools are web applications.

To facilitate its functions, Chip has its own website called *Chip for Hire*, where the shopkeepers can build their product portfolios, design survey questions and hire Chip to promote the product and collect customer feedbacks via a survey. As Chip has a touch screen in its torso, it is able to present interactive web interface to customers. Multiple web services are used to improve Chip's social ability. In addition, Chip itself is also a service provider. This project aims to integrate various web technologies into the robot platform so that Chip can get involved in a daily-life environment and realise commercial values.

In order to accelerate the development process and still guarantee the software quality, rapid application development methodology is utilised for delivering solutions of the research projects, as RAD often embraces object-oriented programming methodology and emphasises on software reusability. Therefore, the development will focus on leveraging existing tools to achieve the goals, rather than creating new ones.

Chapter 4

Web Motion Control for PR2

This chapter introduces the application designed for the purpose of motion control for PR2. PR2, developed by Willow Garage, is a high DoF (Degrees of Freedom) robot capable of performing various tasks. It is equipped with a full sensor system as well as backdriveable arms and grippers for it to be more flexible in completing tasks. The application in this chapter acts as the communication tool between the human instructor and PR2, so that the PR2 can understand the commands sent by the instructor in a quick and accurate way via a web interface.

Section 4.1 presents both the client side and server side structures, which form the basis of the application. Section 4.2 introduces the user interface of the application, including 3D visualisation, joint control and vision live stream. Section 4.3 is an use case of recording demonstration data for robot learning ball catching skill through the web motion control. Section 4.4 looks at the limitation of the current version of the application. Section 4.5 discusses the impact of the application.

4.1 Software Infrastructure

Figure 4.1 presents an overview of the software infrastructure. The software has independent structures on its client side and server sider, which are described in the following two subsections respectively.

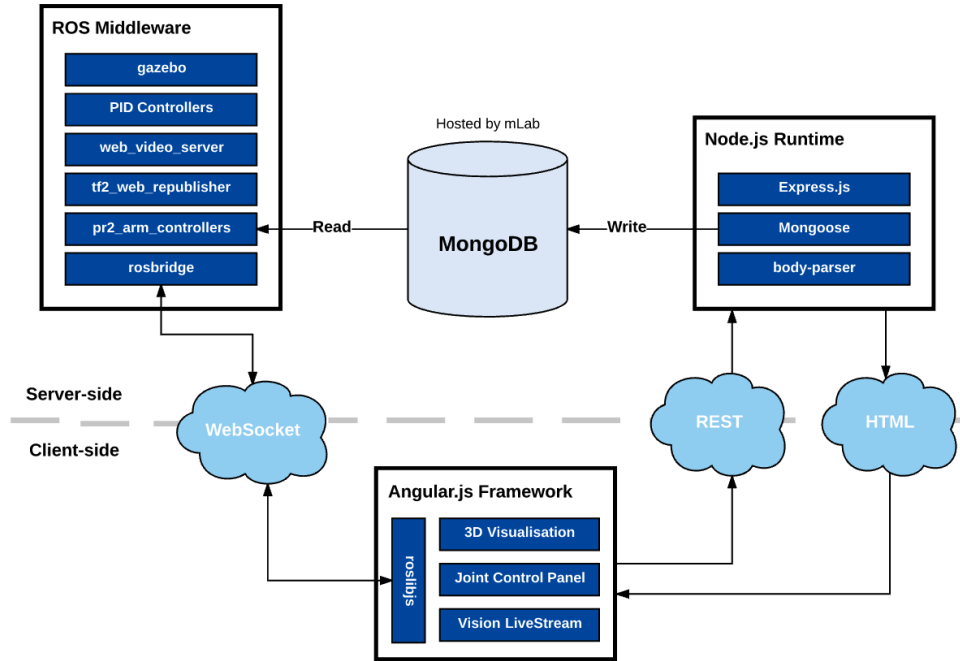


Figure 4.1: Overview of the infrastructure of web motion control

4.1.1 Structure: client-side

The software on the client side is built upon the Angular.js framework as an single page application. As Angular.js offers a lot of flexibility to separate presentation logic from business logic and presentation state, the client side is more close to a MVVM (Model-View-ViewModel) structure. The ViewModel is decorated by a Angular's build-in function called *Controller*, which handles all the application behaviour. Meanwhile, the two-way data binding feature helps to automatically update the views whenever the model changes. Therefore, application views can be abstracted by templates that embed Angular-specific elements, attributes and expressions.

The client-side also leverages many other resources and they are managed by *bower*. Two of those techniques, *Angular Material*¹ and *Angular UI-Router*² are es-

¹<https://material.angularjs.org/>

²<https://github.com/angular-ui/ui-router>

pecially important. Angular Material is a UI framework providing a set of reusable, well-tested and accessible UI components based on *Google's Material Design Specification* [18]. All implementations of UI features on the client-side totally rely on Angular Material.

Angular UI-Router is a routing framework for Angular.js. It updates the browser's URL as the user navigates through the app via state machine, which manages the transitions between those application states in a transaction-like manner. The states of the application have a hierarchical tree structure. Appendix B.1 presents the configuration of client-side routers, where states are modelled according to the function scopes as well as the devices that access the application.

Robot web tools are also key modules on the client-side. The implementation will be described with more details in Section 4.2.

4.1.2 Structure: server-side

The server side of the software is divided into three layers: ROS middleware, web services and database. The ROS middleware layer runs ROS nodes for controlling the PR2 robot and exposing ROS to client-side via *rosbridge*. The following launch file will launch the necessary nodes:

```

1 <launch>
2   <include file="$(find gazebo_ros)/launch/empty_world.launch">
3     <arg name="gui" value="false"/>
4     <arg name="headless" value="true"/>
5   </include>
6   <include file="$(find pr2_gazebo)/launch/pr2.launch"/>
7   <include file="$(find rosbridge_server)/launch/rosbridge_websocket.launch"/>
8   <node name="tf2_web_republisher" pkg="tf2_web_republisher" type="tf2_web_republisher"/>
9   <include file="$(find pr2_web_motion_control)/launch/l_joint.launch"/>
10  <include file="$(find pr2_web_motion_control)/launch/r_joint.launch"/>
11  <include file="$(find pr2_web_motion_control)/launch/head.launch"/>
12  <include file="$(find pr2_web_motion_control)/launch/torso.launch"/>
13  <node name="pr2_arms_controller" pkg="pr2_web_motion_control" type="pr2_arms_controller.py"
14    output="screen"/>
15 </launch>

```

Line 2-6 start a PR2 robot simulation with *Gazebo*. To improve the performance of the simulator, the graphical UI is disabled since the visualisation will be shifted to the web interface. Line 7 launches the *rosbridge* server. The node

tf2_web_republisher in the following line is used to throttle and precompute TF transform information which will be sent to web client via *rosbridge*. Line 9-12 load the lower level PID (Proportional-Integral-Derivative) controllers to replace the default joint trajectory controllers for robot arms, head and torso. The reason of using PID controllers is that they supports faster simulation and the structure of control parameters are simpler. Since the arms have 14 joints in total, the python script *pr2_arms_controller* in line 13 will create a topic called “arms_motion_control” to control both of the arm as a whole rather than invoking each joint separately. The source code is shown in the Appendix B.2.

The use of PID controllers and the script *pr2_arms_controller* are not necessary if the developer does not have the knowledge of ROS programming. This is purely for the sake of simplifying the control process, and merely using the web interface can achieve the same goal as the script does. The *pr2_arms_controller* also builds a connection to MongoDB to read the configuration of predefined motions by using *pymongo*, which is a python driver for working with MongoDB.

The web server is in a Node runtime environment and uses Express.js as its API framework. The server script is simple yet efficient, as shown in Appendix B.3. There is only one RESTful API on the web server just for users to save their designed robot motions.

The *mongoose* module is an object modelling tool for Node that essentially works like an ORM (Object Relational Mapping). It defines the attributes of motion schema and returns a constructor which can be saved as a document into the database. With the middleware *body-parser*, the motion data can be extracted from request body and transformed to motion model directly.

The web server does not have any connection to ROS, as it just handles the requests from client-side. As depicted from the infrastructure overview (Figure 4.1), the web server serves as a publisher to save motions to database, while ROS subscribes those motions configured by different users. Therefore, the cloud database is like a shared knowledge pool where any PR2 robot that connected into the system is able to perform motions designed by different users.

4.2 User Interface

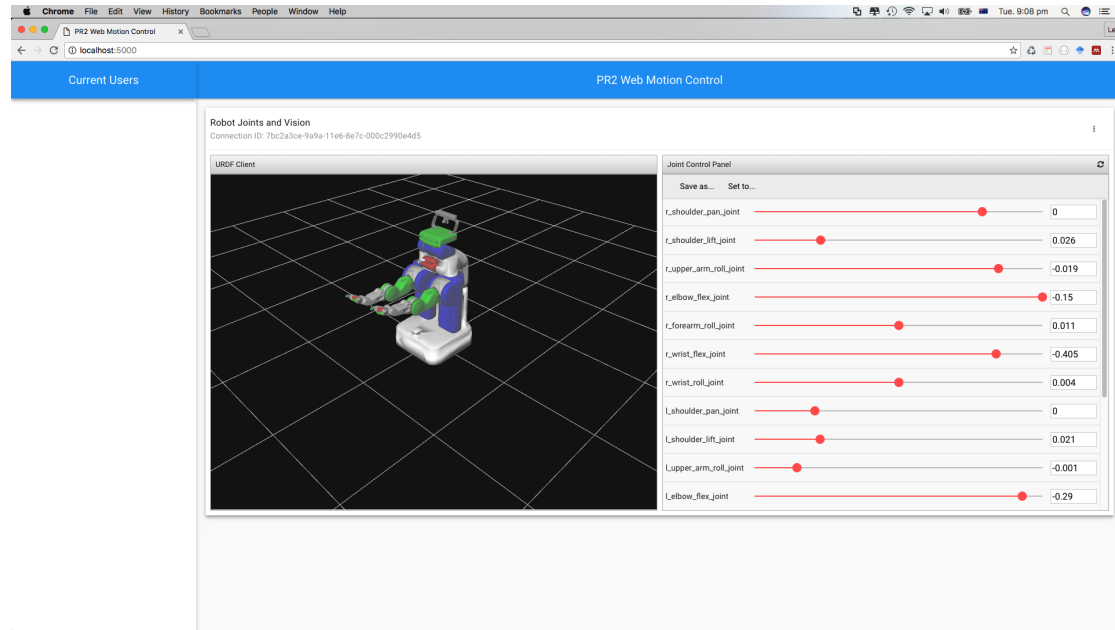


Figure 4.2: User interface of web motion control

As users may not be local or do not have physical access to robot, the goal of the UI design is to provide convenient and intuitive widgets to control a real robot or perform simulation with a ROS instance. An screenshot of the web interface is shown in Figure 4.2. The web interface of the system offers three features with the robot web tools: 3D robot model visualisation, joint control panel and vision live stream.

4.2.1 3D Robot Model Visualisation

As shown on the left in Figure 4.2, the 3D visualisation interface gives users a real-time feedback of robot's state during operation, as the message used for updating the robot model and transferred from ROS to the web client requires low bandwidth. On the ROS side, *tf2_web_republisher* computes and sends the transform information via *rosbridge*. Then in the web client, the URDF is loaded by *ros3djs*, with 3D JavaScript library *three.js* for rendering.

Appendix B.4 presents the source code for the whole process:

1. Create a 3D viewer and setup the dimensions and the element id where the viewer is placed;
2. Initialise the TF client which subscribes the information from *tf2_web_republisher*;
3. Load URDF files from assets folder and render the client by using the robot description parameter from the ROS parameter server.

4.2.2 Joint Control Panel

The most useful interface for this software is the joint control panel since it provides the programmability to the end-users regardless their programming knowledge of any kind. The control panel uses Angular Material slider UI component as the configuration tool for setting the position of each joint from robot's arms, head and torso.

Once the URDF is loaded, all the control sliders will first set the minimum and maximum joint limits to fulfill the safety check by querying the robot description parameters from ROS. Then the control sliders will subscribe the *sensor_msgs/JointState* topic to get the current joint positions. Appendix B.5 is the source code for the initialisation of joint control panel. It fetches the robot description into a DOM parser and filter each node by its type and name to find the desired joint information. After initialisation, Angular will listen to the change of each slider and publishes the new joint positions via *roslibjs* once change is detected.

Users can save the arms' joint positions as a gesture with a chosen name, which can be used with other gestures as a plan of robot motion. For example, to configure a wave hand motion, the whole process can be divided into several way points during the hand movement, and each of the way point is a static gesture with certain joint positions.

4.2.3 Robot Vision Live Stream

Robot vision live stream provides a natural mean for remote users to interact with a real robot and its surrounding environment as well as checking the progress. The ROS node *web_video_server* provides a video stream of a ROS camera image topic which can be accessed via HTTP. It does not depend on *rosbridge* or *roslibjs* since it opens a separate port (default 8080) for incoming HTTP requests. The default stream type is MJPEG, in which each frame is compressed as a JPEG image. Therefore, the video stream can be embedded into HTML ** element with URL in the following pattern: *http://<ROS address>:8080/stream?topic={ros-topic}*, where the *ros-topic* is the corresponding topic that publishes the image data from the target camera. Users can also configure the quality via URL to accommodate different connection speeds. With the vision live stream, the risk of damaging the robot or its surroundings can be reduced when controlling a real robot by remote users.

4.3 Use Case: Recording Demonstrations for PR2 Learning Ball Catching

This section presents a use case implemented using the web motion control to record demonstrations data for PR2 learning ball catching. Catching a thrown ball is a complex skill that requires fast, dextrous manipulation as well as seamless human-robot interaction. This skill is not difficult for most humans to acquire. However, it is complicated for a robot as the variables for ball catching skill are difficult to model in a fast changing environment. Compared with traditional approaches such as trajectory calculation or prediction and 3D ball position reconstructing, learning from demonstrations doesn't require complex task modelling and explicit programming. By investigating the relationships between the initial ball positions in the air and the catching time, it is possible to decide the catching time before the ball reaches the target. Hence, efficiently collecting data during demonstrations is very essential to achieve the goal.

4.3.1 Design

Although PR2 web motion control features joint control via sliders, it is noted that using just a mouse and keyboard is difficult for controlling PR2 to perform real-time manipulation tasks like ball catching. Therefore, to create more effective controlling interface, smartphone is used to solve the real-time control problem. Smartphones like Androids and iPhones provide motion sensors to detect how fast the device is moving along its three axes respectively: X (side-to-side), Y(forward/backward) and Z(up/down). This establishes a direct control from human to robot by mapping the smartphone motion triggered by the human user and the joint controller of the robot. The new implementation of DeviceMotion events in HTML5 enables developers to access the sensor data using javascript within mobile browsers such as Mobile Safari, Chrome, Opera Mobile and Firefox. Therefore, the smartphone controller can be integrated into the PR2 web motion control without installing any extra software.

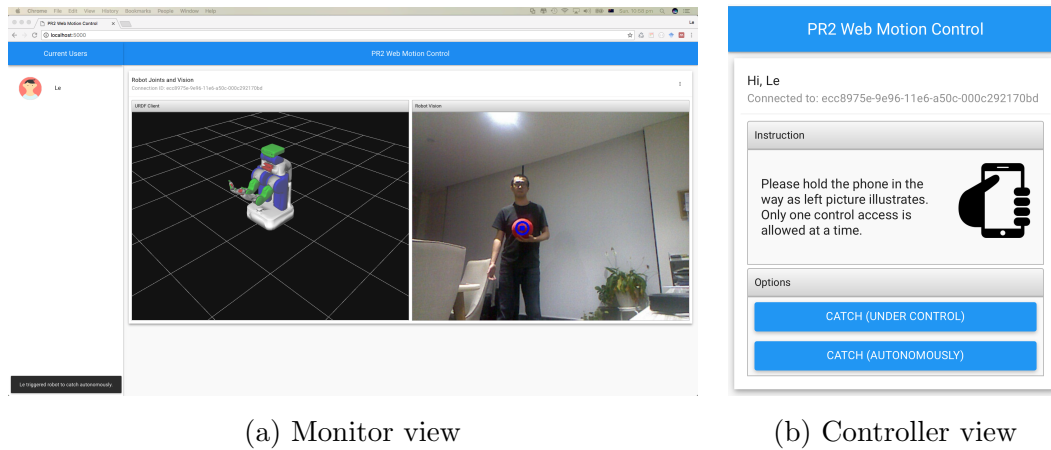


Figure 4.3: User interfaces for monitor and controller

The PR2 web motion control which is running on PC in this experiment plays a monitor role. It provides a live vision stream from PR2 for remote users to conduct the experiment (Figure 4.3 (a)). While, the application provides different user interfaces (Figure 4.3 (b)) when using smartphone as the motion controller. Instead of monitoring the robot state and controlling its joint position, the smartphone interface offers “catch” buttons for commanding PR2 to do the task. Before tele-

operating the robot, the user is required to register in order to select an existing ROS connection established by the monitor. This is for the user management as the system allows multiple users to connect to the same ROS instance for the control purposes. To avoid control conflict, it only allows one control process at a time. Both server-side and client-side are extended with *socket.io*³ framework to notify users if someone is in control or the control position is currently vacant. *Socket.io* is a real-time engine that features instant data communication between back-end and front-end. It also helps to build a communication tunnel across ROS, client-side and server-side. Every experiment dataset is sent by ROS via *rosbridge* to the client-side, and it is pushed further to the server-side by *socket.io*, where the dataset is finally stored into MongoDB for data analytics in the future. The following figure illustrates the high-level infrastructure design for the experiment.

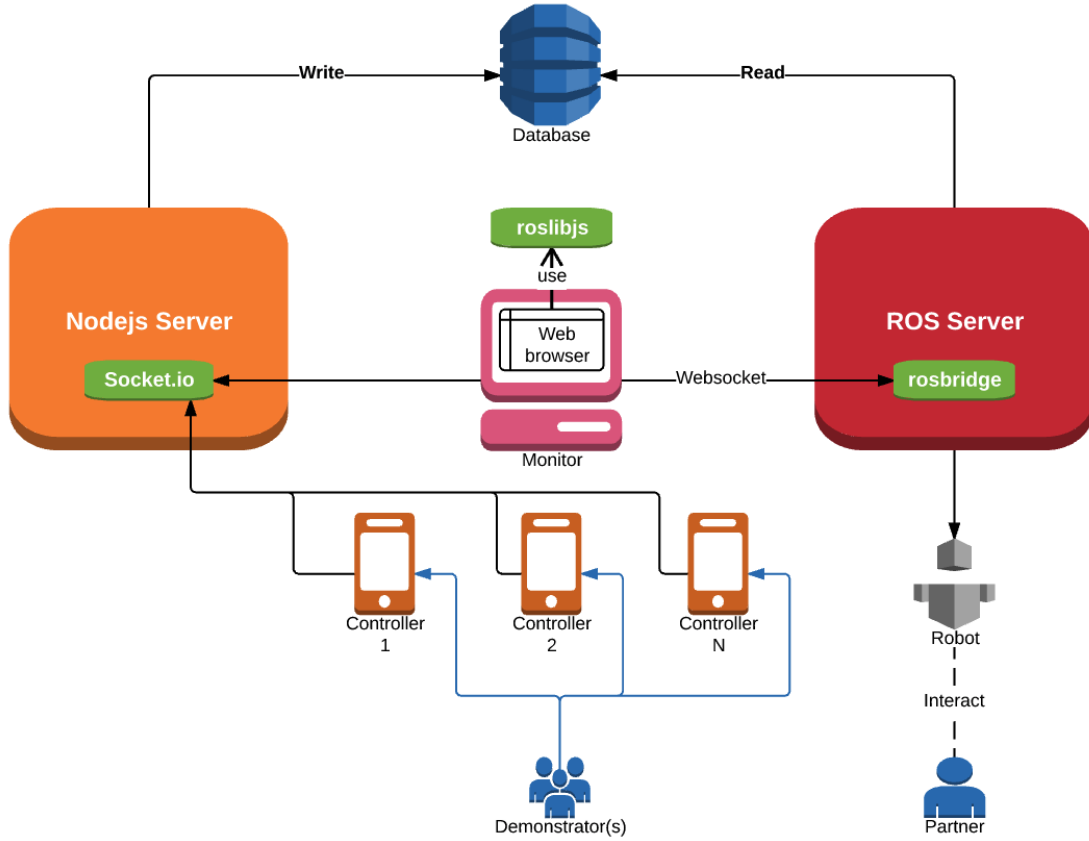


Figure 4.4: High-level infrastructure for the use case

³<http://socket.io/>

4.3.2 Procedure

By using the joint control panel, two motions for preparing for the catch (Figure 4.5 (a)) and finishing the catch (Figure 4.5 (b)) can be defined respectively. During the learning process, a human partner is needed to make a fairly good ball throw to the PR2 robot. At the same time, the demonstrator, onsite or offsite, will hold a smartphone in one hand and conduct a fetch motion, as if to catch a ball thrown to him or herself. The fetch motion will be detected by the smartphone and signalled to PR2 via the web motion control application, so that instantly PR2 will perform a catch motion as well.

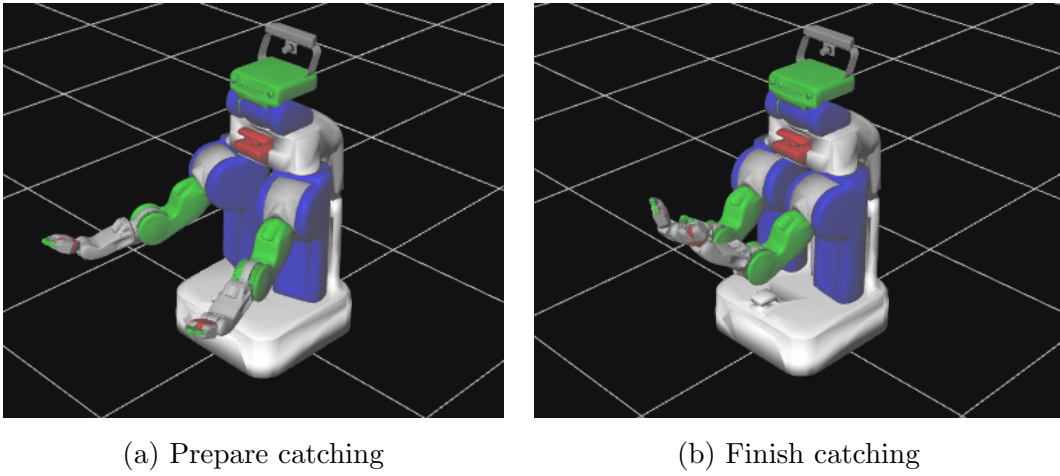


Figure 4.5: Motions for ball catching

Ball detecting and tracking is based on a computer vision technique, namely OpenCV, which is integrated into ROS program stacks. The ball used in the experiment is purely red, so it can be tracked by defining the lower and upper boundaries of the red colour with OpenCV. Source code in Appendix B.6 detects the ball and marks its centre and boundary in every video frame.

The data collection for each catch trial occurs on the ROS side. The PR2 robot is equipped with Microsoft Kinect sensor which is able to acquire the object's Cartesian coordinates as well as the depth value for distance. When a trial begins, PR2 will detect the ball held by the human partner and records its initial depth value, then it poses the gesture to preparing for a catch. The ball is perceived to

be thrown once the change of the ball image depth exceeds a pre-defined threshold. At this time, the timestamp is set to 0 and the ball position is recorded as the first waypoint in the trajectory. The ball positions will be captured in the next 3 image frames as well as the timestamp when PR2 tries to catch the ball. Figure 4.6 illustrates the process.

As the next step, the positions of the 4 waypoints and the catch timestamp since the first waypoint will be sent to the demonstrator's smartphone for further evaluation. As depicted in Figure 4.7, the demonstrator will give further feedback according to the PR2's performance or reject the trial if the data is obviously invalid. The collected data will be used for training to find out the relationship between those waypoints and the catch time. The algorithm for training is beyond the scope of this dissertation. Nevertheless, the experiment provides an intuitive and efficient paradigm for collecting demonstrations with web based solution, in which physical access to the robot is not necessary for the demonstrator.

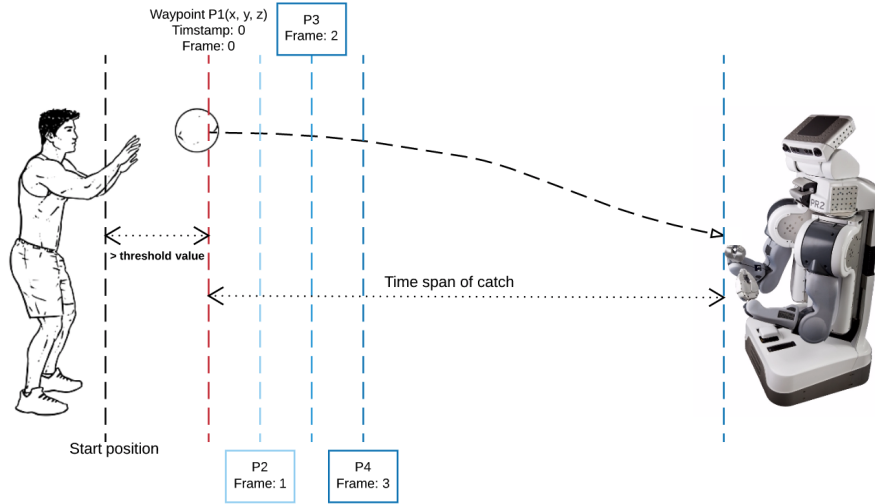


Figure 4.6: Data collection for a ball catch trial

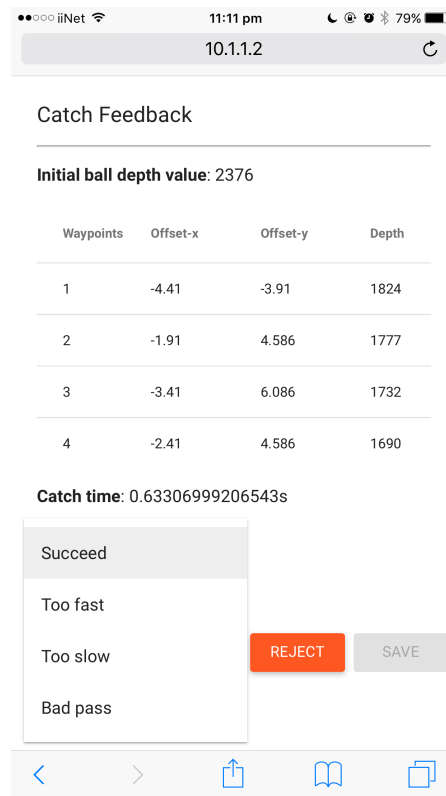


Figure 4.7: Data feedback of a ball catch trial

4.4 Known Limitations

There are some known limitations with the web motion control application. First, the 3D visualisation is very simple and not able to simulate the interaction with other objects. Also, the position of the robot is fixed and moving the robot is not applicable. As a result, numerous aspects for robot control and manipulation cannot be addressed.

Second, it is lack of efficient and real-time control method via the web interface. Network latency issue always presents in the teleoperation especially when it is involved in high bandwidth cost processes such as video streaming. Furthermore, for high DoF robots, mapping joint positions and movements from control devices is difficult. The utilisation of motion detection from the smartphone in the above use case can only provide guidance for simple motion. For higher level movements,

it requires more sophisticated wearable devices to map joints from human to robot to maximise the benefit of learning from demonstrations.

It should also be noted that in order to run the application on different robot platforms other than PR2, it will require explicit programming from the source code. Thus, scripting interface is needed in the application so that the system can be more configurable and flexible.

4.5 Discussion

Despite its limitations, the application successfully demonstrates a paradigm of the implementation of modern web application framework into robot software development. Web motion control application offers end-users with intuitive control interface which supports simulation of the robot arms' kinematics. More importantly, such an application can be programmed entirely via JavaScript and HTML, and requires little or even none expertise of ROS programming via either C++ or Python. By leveraging robot web tools, which establishes well-defined abstractions and interfaces between ROS and the web application, web developers can efficiently manipulate ROS topics, services and actions as JavaScript objects. As a result, the reach and accessibility of robot application development is broadened, which will generate more interest in robotics in a larger population.

In addition, the use case of recording demonstrations through web interface presents a novel approach for training robot with new skills. In the learning framework, it utilises the smartphone as a control bridge between the web server and robots for instant trial feedback and automated data persistence. This gives novice users a opportunity to get involved in the robot learning process by using devices that they are familiar with. On the other hand, through web interface, it connects robot and the smartphone into a small web of things, where both of them exchange their sensor information. Thus, the robot can be remotely connected and controlled.

Chapter 5

“Chip” in Shopping Centre

This chapter focuses on the second project in this dissertation, which is the management and utilisation of “Chip” in a shopping centre. “Chip” is a humanoid service robot with a touch screen on its chest, and it is capable to move around and make speech as well as simple gestures. It can be applied to conduct social activities with human for various goals in a defined environment.

The purpose of the project is to implement the robot to communicate with customers through promotion activities, such as distributing production discount code, introducing stores and collecting feedbacks. Meanwhile, a web interface is developed for the shopkeepers to design and book such promotion activities, view feedback results as well as watch the activities from live stream.

The chapter is organised as follows: Section 5.1 introduces the overall design of the system, Section 5.2 presents the complete workflow of the promotion activities, Section 5.3 focuses on the interactions between the users and Chip, and finally Section 5.4 discusses the findings and thoughts from implementing the project as well as identifies the upcoming challenges in the future.

5.1 System Design

5.1.1 Overview of the Subsystems

This project has two subsystems: one is the website called *Chip for Hire* for shopkeepers to book Chip as their products representative and manage their profiles, products and surveys content; the other one is a local web application called *Chip on Duty* running on Chip’s platform for interacting with customers during the promotion activity. Both of the two subsystems are based on MEAN stack (see 3.2) as in the project 1.

Since *Chip for Hire* features content management and secure user authentication and authorisation, the web server will handle more complicated business logic and intensive requests than the previous project of PR2. Also, unlike in project 1, the web server is not totally isolated from the robot platform. It builds a indirect relationship with the robot platform by providing web service APIs for *Chip on duty* to fetch activity content and save survey results. Besides shopkeeper, *Chip for Hire* has another two types of users: administrator and Chip the robot. Administrator is responsible for maintenance related activities of Chip, releasing time slots for bookings according to Chip’s availability, and releasing news about Chip from time to time. Shopkeepers and administrator have direct access to the website with registered email and corresponding password. However the user interfaces are different, as shown in Figure 5.1. While the robot, Chip, as a special user does not have user credentials for logging into the system via web interface, it uses pre-defined token for subscribing the activity content under the permission from shopkeeper instead.

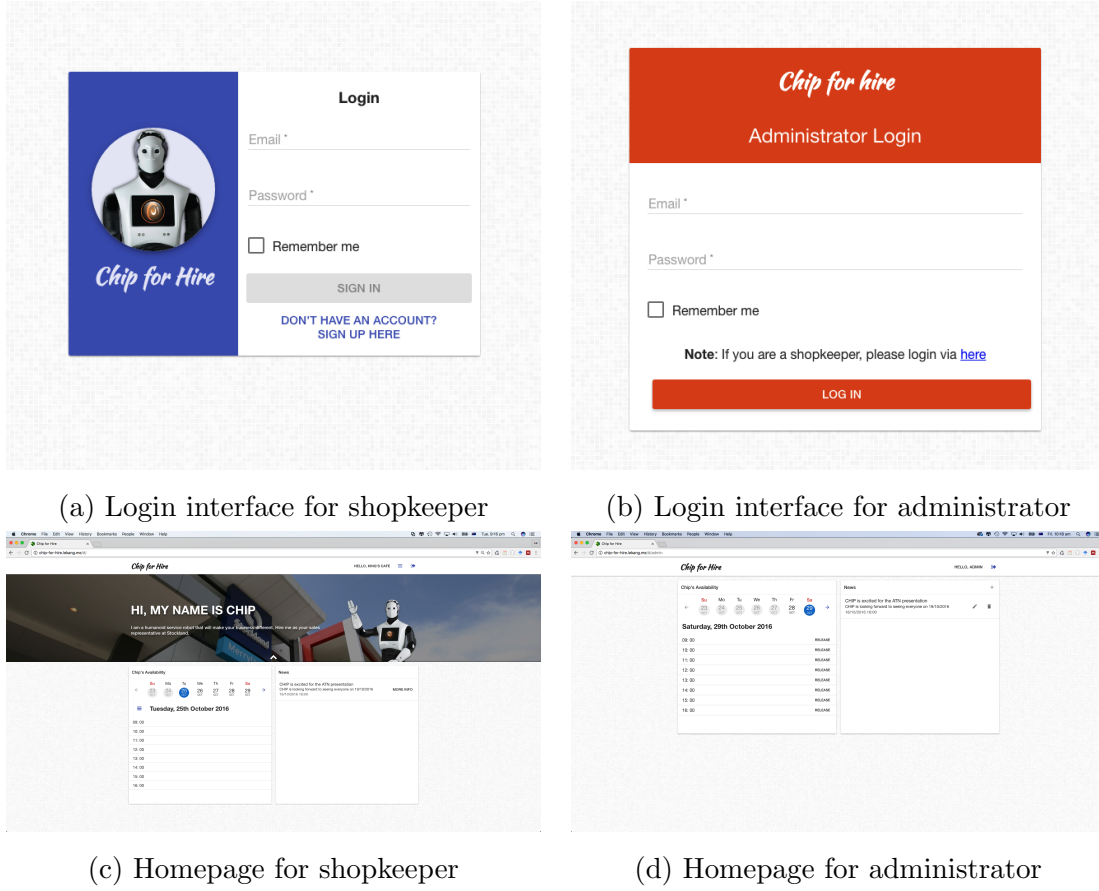
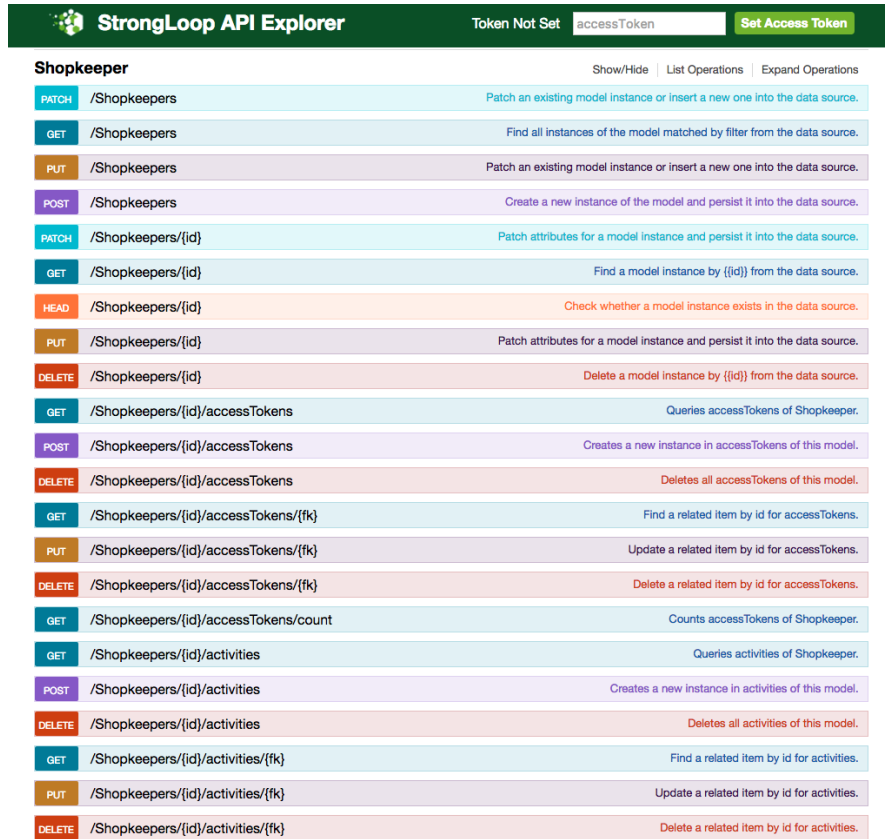


Figure 5.1: UI of *Chip for Hire*

Chip on Duty is a self-contained web application, which is hosted by the robot platform for running promotion activities that booked by a shopkeeper. During the activity, the application will be in various states which are controlled by ROS messages. For example, it displays the photos of the products for promotion on the screen by default. Once a customer is approaching and detected, ROS will send a message via *rosbridge* to inform Chip so it will display greeting text. Chip can also conduct survey for the customers who have tried the product and as a reward, Chip will send a special offer from the shopkeeper to those who give feedbacks. All the assets and resources for the application working in the promotion activity are temporary and will be deleted after activity ends. Therefore, to some extent, Chip offers the software (*Chip on Duty*) as a service (SaaS) for shopkeepers. The integration of web application and ROS enables Chip to interact with people in a

more natural way.

5.1.2 API Framework



StrongLoop API Explorer		
Token Not Set <input type="text" value="accessToken"/> Set Access Token		
Shopkeeper Show/Hide List Operations Expand Operations		
PATCH	/Shopkeepers	Patch an existing model instance or insert a new one into the data source.
GET	/Shopkeepers	Find all instances of the model matched by filter from the data source.
PUT	/Shopkeepers	Patch an existing model instance or insert a new one into the data source.
POST	/Shopkeepers	Create a new instance of the model and persist it into the data source.
PATCH	/Shopkeepers/{id}	Patch attributes for a model instance and persist it into the data source.
GET	/Shopkeepers/{id}	Find a model instance by {id} from the data source.
HEAD	/Shopkeepers/{id}	Check whether a model instance exists in the data source.
PUT	/Shopkeepers/{id}	Patch attributes for a model instance and persist it into the data source.
DELETE	/Shopkeepers/{id}	Delete a model instance by {id} from the data source.
GET	/Shopkeepers/{id}/accessTokens	Queries accessTokens of Shopkeeper.
POST	/Shopkeepers/{id}/accessTokens	Creates a new instance in accessTokens of this model.
DELETE	/Shopkeepers/{id}/accessTokens	Deletes all accessTokens of this model.
GET	/Shopkeepers/{id}/accessTokens/{fk}	Find a related item by id for accessTokens.
PUT	/Shopkeepers/{id}/accessTokens/{fk}	Update a related item by id for accessTokens.
DELETE	/Shopkeepers/{id}/accessTokens/{fk}	Delete a related item by id for accessTokens.
GET	/Shopkeepers/{id}/accessTokens/count	Counts accessTokens of Shopkeeper.
GET	/Shopkeepers/{id}/activities	Queries activities of Shopkeeper.
POST	/Shopkeepers/{id}/activities	Creates a new instance in activities of this model.
DELETE	/Shopkeepers/{id}/activities	Deletes all activities of this model.
GET	/Shopkeepers/{id}/activities/{fk}	Find a related item by id for activities.
PUT	/Shopkeepers/{id}/activities/{fk}	Update a related item by id for activities.
DELETE	/Shopkeepers/{id}/activities/{fk}	Delete a related item by id for activities.

Figure 5.2: Loopback API explorer

In order to build robust application APIs, the web server adopts *Loopback.js*¹ as the API framework. Loopback is a highly extensible Node.js framework for creating dynamic end-to-end REST APIs. It is built on top of Express.js and supports various data sources including MongoDB. Loopback offers powerful tools such as visual API explorer (as shown in Figure 5.2) and CLI code generators. The back-end data are represented by Loopback models (as shown in Figure 5.3), which come with pre-defined REST APIs with a full set of CRUD operations by default.

¹<https://loopback.io/>

The model can be defined via JSON file or command line. Such a programming via configuration approach make the development process more efficient and less error-prone.

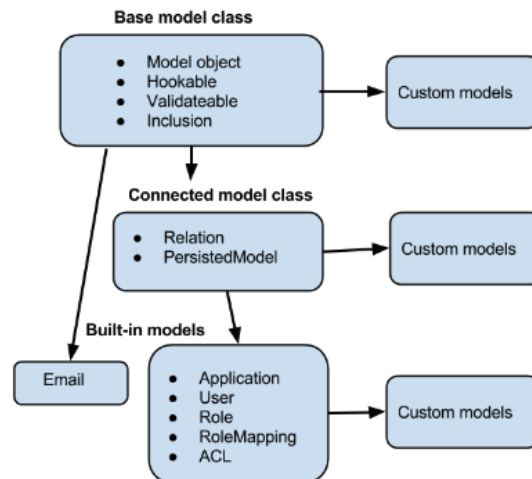


Figure 5.3: Loopback models structure:

source: <http://loopback.io/doc/en/lb2/LoopBack-core-concepts.html>

Beyond the pre-defined REST APIs, Loopback allows custom REST endpoints by adding application logic to models through remote methods. A remote method is a static method of model which performs operations not provided by standard model REST APIs. For instance, typical CRUD operation is not suitable for file uploading, therefore adding a customised method is more appropriate. Appendix B.7 demonstrates how to use the remote method to handle product photos uploaded by shopkeepers. As a result, server can receive product photos via the customised endpoint `/Products/uploadImages`. Remote methods can also be used for handling special data access control rules, providing additional security on top of default the authentication in Loopback framework. In short, remote methods extend basic endpoints and offer flexibility to expose additional application data and logic.

Another reason to use Loopback framework is its ability to manage model relationships. *Chip for Hire* involves multiple models, however, NoSQL database does not resolve relationships explicitly. Therefore, model relations need to be

created and handled at the application level. Loopback creates relationships via model definition, and exposes them as a set of APIs to interact with each of the model instances, so the related information can be queried and filtered based on the client’s needs. The JSON objects in Appendix B.8 define the shopkeeper’s relationship to product, survey and activity.

The relationships configuration indicates that a shopkeeper can have many products, surveys and promotion activities, while every single of them belongs to a specific shopkeeper. The *foreignKey* is left empty so by default it will be *shopKeeperId*, which will be a property in the models that declares a *belongsTo* relationship. As shown in Figure 5.2, by sending requests to the endpoint */Shopkeepers/{id}/activities* with the shopkeeper’s id, we can find all the promotion activities that belong to the shopkeeper. This also works in a reverse manner, for example, the endpoint */Activities/{id}/shopkeeper* will return the shopkeeper’s details if the activity id is provided.

5.1.3 Authentication and Authorisation

Since *Chip for Hire* will collect some sensitive data such as shopkeeper’s contact details, product portfolios and customer feedback, it is crucial not to overlook the critical issue of security. The website implements authentication for both server-side and client-side. As LoopBack provides built-in token-based authentication, the server-side will generate an access token once a user is logged in and send the token to the client-side. The token is required when making subsequent REST requests from the web interface for the access control system to validate that the user can invoke methods on a given model.

At the client-side, there are two ways to store the token according to the user’s preference. If the user trusts the device for automatic login at every visit, the token will be stored into HTML5 *localStorage* and valid for authentication until this token expires on server-side. Otherwise, the token will be stored in *sessionStorage*, in which all data is removed when the page session ends.

As for Chip, a special user of the web APIs for acquiring activity content including details of shopkeeper, products for promotion and survey questions, a different authentication strategy is implemented. As Chip makes the REST requests from

the web application *Chip on Duty* which is hosted by itself and used for interacting with customers, it is not safe to keep an access token for a non-private usage. Therefore, a “double-key” validation rule is applied. As depicted in Figure 5.4, shopkeeper will input an activation code via the user interface of *Chip on Duty* to start the promotion activity, then Chip will send a request to web server with the activation code and a pre-defined token. The pre-defined token is stored in a configuration file at the server-side for validate Chip’s identity. Once it is confirmed that the request is from Chip, the system will look up the activity within all scheduled activities on that day by the activation code. The found activity content will be returned to Chip, otherwise the request will be denied.

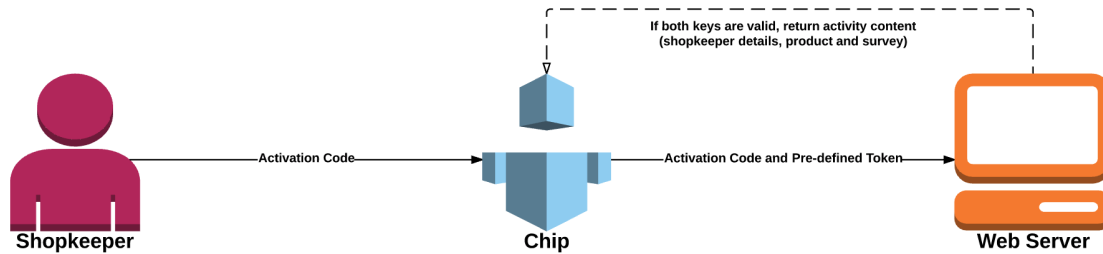


Figure 5.4: Authentication strategy for Chip to acquire activity content

Authorisation is also important for data access especially for writing and deleting operation. Loopback provides configurable ACLs (access control lists) which is defined as a part of model definition. Appendix B.9 demonstrates how to define the data access of activity model for each role and model method via *acls* property of the model definition. It will firstly deny any access to all REST endpoints of activity model, and then set up the access control according to the user role. For the authenticated users, administrator is allowed to read any activity details, while only the owner (shopkeeper) of the activity can perform any default CRUD operation or remote methods. Noticeably, the remote methods *start*, *addSuveryResult*, *sendOffer* and *end* are open to unauthorised or anonymous users. Those are designed for Chip to communicate with web server during the activity. As mentioned before, the authorisation of data access is implicitly implemented during the method invocation as a part of “double-key” validation strategy.

5.2 The Complete Workflow

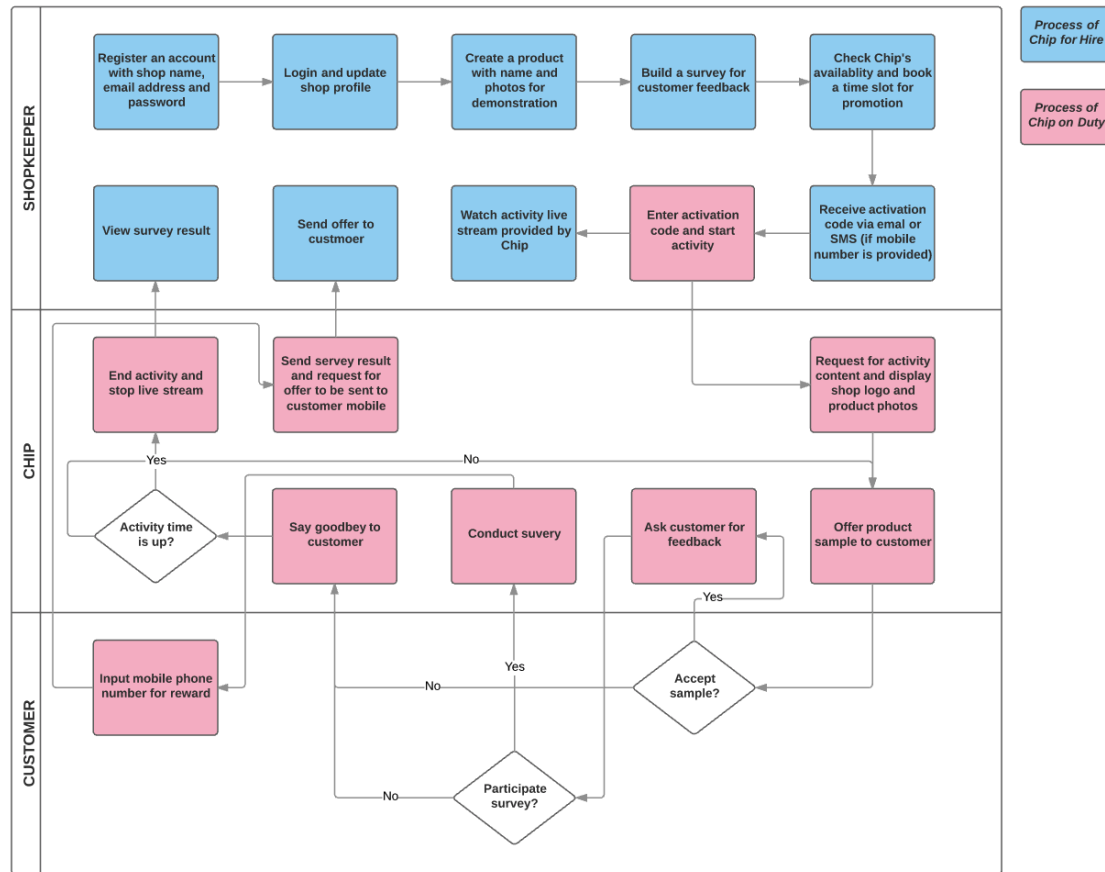


Figure 5.5: Workflow of entire system

Figure 5.5 depicts the complete workflow of the entire project. A shopkeeper from the shopping centre where Chip servers needs to register an account on the *Chip for Hire* website with the shop name, email and password in order to hire chip as sales representative. The shopkeeper can manage the shop profile such as name, introduction, logo and contact details. Next, the shopkeeper creates a product portfolio for promotion. Adding product photos is encouraged as Chip can display them on the screen for better marketing result. To get customer feedback, the website offers a survey builder where the shopkeeper can design a list of multiple choice questions. The survey is also editable and sortable so that it is reusable for different promotion activity. Finally, once the products and surveys are ready,

the shopkeeper can check Chip’s availability via the calendar on the homepage, and reserve a time slot (1 hour per activity). The reservation form will ask the shopkeeper to add the product from the portfolio and choose a customised survey.

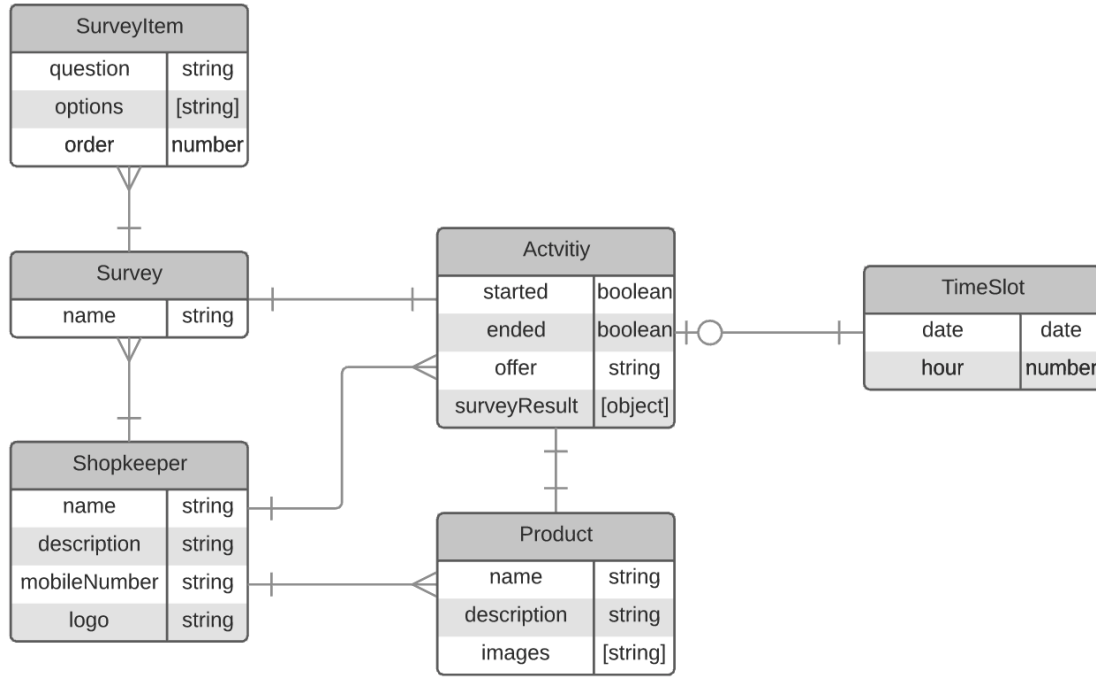


Figure 5.6: Entity relationship diagram of Chip for Hire

In order to attract customers to give feedbacks, the shopkeeper will provide a special offer as a reward for customers who complete survey. Once the booking is successful, the shopkeeper will get an activation code via email and SMS (if mobile number is provided in shop profile) for triggering the promotion activity. The entity relationship diagram in Figure 5.6 illustrates the models involved in the workflow.

Before the promotion activity, Chip will start the web application *Chip on Duty* and bring activation page for the shopkeeper to input the code. The activity promotion starts once the activation code is verified by web server and activity content is loaded by Chip. During the activity, Chip will provide live stream to the activity owner (shopkeeper), as shown in Figure 5.7 (a). While Chip navigating around in the shopping centre, it offers product samples to customers and asks

them to participate the survey. The customers who give feedback can use Chip’s touch screen to input their mobile number for special offers. And Chip will make a request to the web server with the mobile number and survey result from the customer. Subsequently, the web server of *Chip for Hire* will process the request and make a another call to a cloud communication API which sends the special offer to the customer via SMS. After the activity ends, the shopkeeper can view the survey result instantly from the web interface, as shown in Figure 5.7 (b).

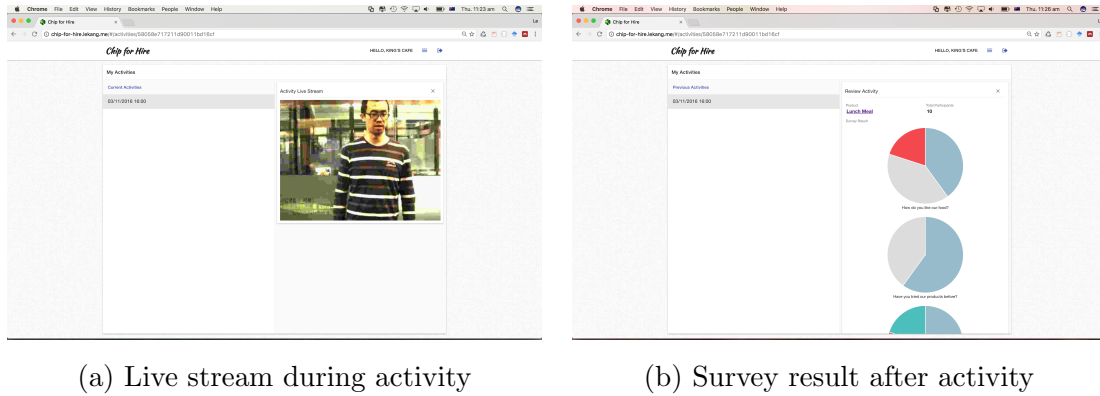


Figure 5.7: UI from *Chip for Hire* for shopkeeper to view an acitivity

5.3 Interaction Between Users and Chip

This project emphasises on human-robot interaction. The web-based solution builds implicit relationships between Chip and users. For Chip and shopkeepers, an employee-to-employer relationship is formed through the *Chip for Hire* website. By using the website, shopkeepers is able to hire Chip for their business without any technical overhead. All they need to do is managing the shop information and contents on the website. More importantly, they can choose a suitable time according to their need.

For Chip and shopping centre visitors, the *Chip on Duty* web application connects them just like a salesperson to customers. Chip can speak, however, it is very difficult for Chip to accurately recognise the customers speech because of the the noisy environment of shopping centre as well as people’s accent. Thus, Chip’s touch screen (Figure 5.8) plays an important role during the interaction, since it

is used to receive customers feedback. The web application is a great way to take advantages of the touch screen due to its interoperability across platforms and popularity for various groups of users. As a result, a mutual and unobstructed communication can be established between Chip and customers.

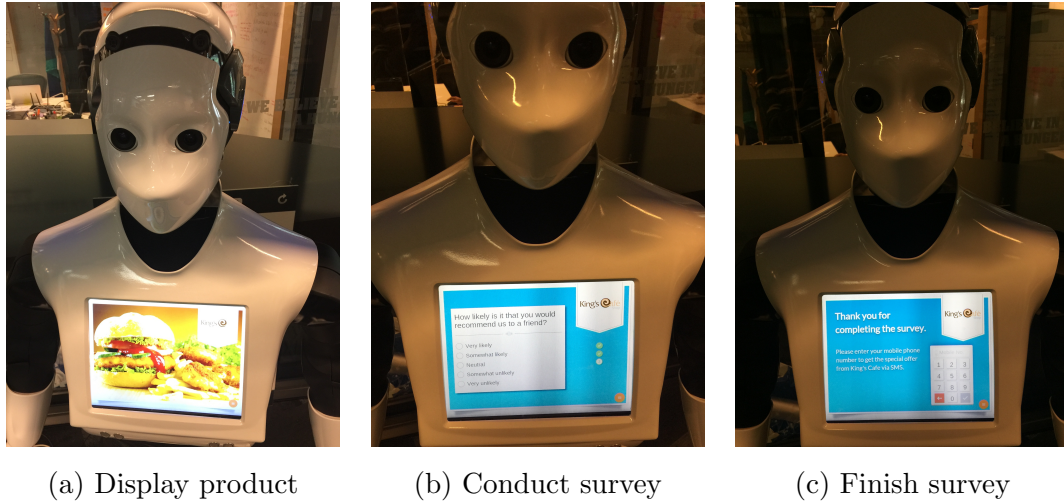


Figure 5.8: Application UI during a promotion activity

To improve the human-robot social experience, the web server also makes use of cloud-based communication APIs offered by *MailGun*² for email and *Twilio*³ for SMS. For example, emails and messages written in Chip’s tone notifies the shopkeeper a activity confirmation with activation code or a cancellation of a booked activity, as shown in Figure 5.9. Therefore, Chip is able to keep in touch with offline users.

²<http://www.mailgun.com/>

³<https://www.twilio.com/>

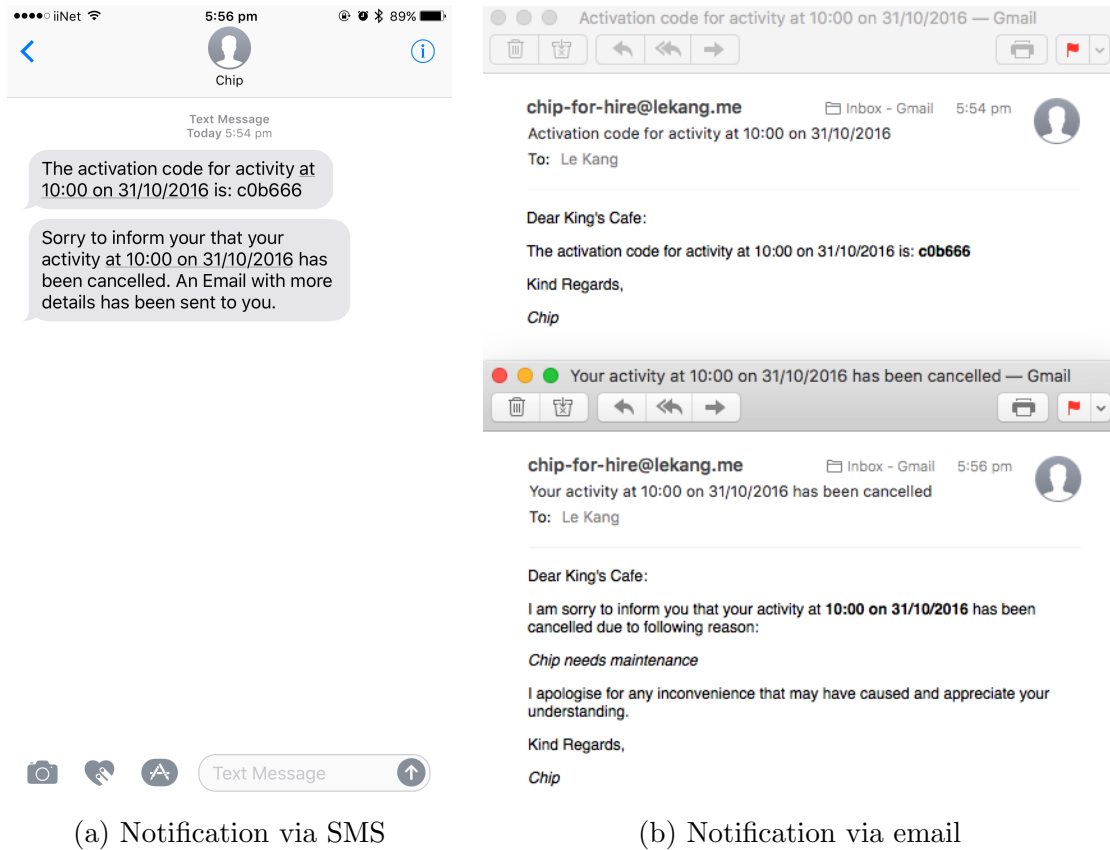


Figure 5.9: Email and SMS notification from Chip

5.4 Discussion and Future Work

The focus of the project is to reduce the social barriers such as poor accessibility and languages issue between Chip and ordinary people, so that Chip can conduct promotion activities and create value for shopkeepers. This research firstly identifies the web as Chip’s prior social means. The ubiquitousness of web helps people establish social network efficiently. Therefore, Chip can take this advantage to be more accessible by its users. That is the reason of building *Chip for Hire* website. Thus, Chip can be employed as a human by shopkeepers via web interface. To interact with customers, this research fully leverages the touch screen built on Chip, creating user-friendly web interface *Chip on Duty* to demonstrate products and gather feedbacks. More importantly, the two web applications can communi-

cate via RESTful web services with security protection for data. This builds an implicit relationship between the shopkeepers and customers through the service robot, and creates novel experience for marketing and shopping. Also, by invoking third part web APIs on cloud communication platforms, Chip is able to keep in touch with shopkeepers offline via email or SMS. To summarise, web technologies expand the social functions of service robots, which are more familiar to ordinary people in their daily life.

The project has been demonstrated on the presentation day of Robotics Program held by Australian Technology Network of Universities (ATN) and received positive feedbacks such as its ease of use and readiness for commercial production. However, there are upcoming challenges in order to further improve the solution.

First, the system needs to be integrated with the navigation, localisation and face recognition modules running on Chip. Currently, the web application is driven by manually inputting ROS messages through console. Chip is expected to be able to localise itself and navigate autonomously within the shopping centre. It is also important for Chip to be aware of approaching customers so it can engage in the communication with them naturally.

Second, the solution needs a real-world trial in a shopping centre to see how people are coping with the service robot. Will the shopkeepers prefer to hire Chip rather than a human? Will the customers be interested to interact with Chip? Will Chip’s survey results be significantly different from a human delivered survey? Those questions require further assessments before a production level of implementation.

Last but not the least, the privacy and security issue needs to be addressed properly. Since Chip will collect large amount of data and some of them are sensitive such as shopkeepers’ business profile and customers’ contact details, to what extent the stakeholders trust Chip is questionable. Also, it is necessary to examine how customers think of the live stream from Chip to the shopkeeper. Although it is a nice feature for shopkeeper to watch the status of activity and observe customers’ reaction to the product, it also exposes customers under the camera which is a serious privacy concern. Thus, the robot’s trustworthiness needs to be tested and at the same time, the protection of privacy for all the stakeholders

can not be ignored.

Chapter 6

Conclusion and Future Perspectives

Accessibility and usability are critical to the success of robot software development and social robotics. At present, novice users can hardly get involved in the robot software design and implementation process due to their lack of knowledge of ROS. The complexity of the robot middleware system needs to be further abstracted so that researchers and developers for other fields and disciplines can participate in robotics study. On the other hand, web development emphasises well established and defined layers of abstractions from low-level system IO to high-level application interfaces. As a result, developers with various levels of expertise are able to contribute to the web development. Therefore, robotics study can take advantages of those features from web technologies to draw more attentions.

The first project in this research attempts to build an abstraction boundary between web interface and robot middleware system to provide robot visualisation and control tools for end-users who are not roboticist themselves. With the robot web tools, robot development is no longer restricted within certain platforms and programming languages. More importantly, the development of effective web-based robot application will facilitate the robotics community to grow beyond the specialised researchers.

The web motion control application for PR2 demonstrates how to use pure JavaScript for manipulating robots so that web developers can get involved with

robot application development. In addition, it integrates the robot web tools into modern web application framework, the MEAN stack, to provide users friendly and intuitive interface as well as programmability for performing experiments. The use case describes a novel approach for collecting demonstration data during robot learning process via a smartphone. As the MEAN stack features scalable architecture and real-time communication, developers are able to add various smart devices for human-robot interaction. This offers the opportunity to put robots into web of things so that they can collect information from different sources rather than relying on their own sensors.

Another benefit brought by web technologies to robotics study is the improved interactivity between robot and human. Web 2.0 broadens the communication tools for people to share information and the ubiquitousness of web applications nowadays connects people more closely. This create opportunities for social robots to engage more into daily-life environment as they can equip and leverage web-enabled devices. Chip, the REEM robot used in the second research project, has a touch screen on its chest, providing a great mean for social interaction with people. By implementing a distributed web system, Chip is able to create commercial value and establish relationships with its stakeholders in shopping centre. The communication between Chip and stakeholders through web interface overcomes some social barriers that may exist if a promotion event is held by humans, such as the social stress when conducting a human-to-human survey.

Web also enriches the social activities for robots as it provides a great range of resources for robots to leverage. The interaction between robots and humans should not rely on speeches and gestures. Social robots are expected to be able to utilise modern communication methods such as email, SMS and online social network so they can maintain relationship with humans while offline. On the other hand, as robot sensors still cannot capture subtle social cues in the way humans do, web applications help humans express their message or intention explicitly through HTML elements such as buttons and text boxes. As a result, it reduces the misunderstandings between humans and robots, thus improve the efficiency of human-robot interaction and experience.

The integration of web technologies and robotics research will be increasing important. The future web and robotics are evolving towards higher levels of in-

telligence. Robots will work with humans in a smart environment where consists of hundreds of heterogeneous devices. They will be required to integrate those devices to achieve more complicated tasks. In such a smart environment, web is a common communication “language” for exchanging information, providing knowledge to robots from various resources. In addition, as Web 3.0 will be semantic and machine-centric, web-enabled and cloud-based robots are expected to understand and discovery information on the web autonomously. Thus, robots are no longer working as stand-alone, instead, they are becoming a part of the Web of Things, serving and consuming web services at the same time. As a result, robot as a service will become more pervasive with sophisticated web-based user interface for remote tele-operating, resource managing and knowledge sharing.

Bibliography

- [1] B. Alexander, K. Hsiao, C. Jenkins, B. Suay, and R. Toris. Robot web tools. *IEEE Robotics and Automation Magazine*, 19(4):20–23, 2012.
- [2] M. Beetz, M. Tenorth, and J. Winkler. OPEN-EASE ? A Knowledge Processing Service for Robots and Robotics/AI Researchers. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1984–1990. IEEE, May 2015.
- [3] T. Berners-Lee, R. Cailliau, A. Luotonen, H. Frystyk Nielsen, and A. Secret. The World-Wide Web. *Communications of the ACM*, 37(8):76–82, Aug 1994.
- [4] G. A. Casañ, E. Cervera, A. A. Moughlbay, J. Alemany, and P. Martinet. ROS-based online robot programming for remote education and training. In *Proceedings - IEEE International Conference on Robotics and Automation*, volume 2015-June, pages 6101–6106. IEEE, May 2015.
- [5] I. K. Chaniotis, K.-I. D. Kyriakou, and N. D. Tselikas. Is Node.js a viable option for building modern web applications? A performance evaluation study. *Computing*, 97(10):1024–1044, Oct 2014.
- [6] W. Chansuwath and T. Senivongse. A model-driven development of web applications using AngularJS framework. In *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*, pages 1–6. IEEE, Jun 2016.
- [7] S. Cherrier and Y. Ghamri-Doudane. The ?Object-as-a-Service? paradigm. *2014 Global Information*, 2014.

- [8] B. Christophe, M. Boussard, M. Lu, A. Pastor, and V. Toubiana. The web of things vision: Things as a service and interaction patterns. *Bell Labs Technical Journal*, 16(1):55–62, 2011.
- [9] C. Crick, G. Jay, S. Osentoski, B. Pitzer, and O. C. Jenkins. Rosbridge: ROS for non-ROS users. In *Proceedings of the 15th International Symposium on Robotics Research*, 2011.
- [10] D. Crockford. JavaScript: The World’s Most Misunderstood Programming Language, 2001.
- [11] D. DiNucci. Fragmented future. *Print*, 53(4):32, 1999.
- [12] J. Dirksen. *Learning Three.js : The JavaScript 3D Library for WebGL*. 2013.
- [13] V. Djalic, P. Maric, D. Kotic, D. Samuelsen, B. Thyberg, and O. Graven. Remote laboratory for robotics and automation as a tool for remote access to learning content. In *2012 15th International Conference on Interactive Collaborative Learning, ICL 2012*, pages 1–3. IEEE, Sep 2012.
- [14] M. C. Domenech, L. P. Rauta, M. D. Lopes, P. H. Silva, R. C. Silva, B. W. Mezger, and M. S. Wangham. Providing a Smart Industrial Environment with the Web of Things and Cloud Computing. *2016 IEEE International Conference on Services Computing (SCC)*, pages 641–648, Jun 2016.
- [15] A. DuVander. 9,000 APIs: Mobile Gets Serious, 2013.
- [16] G. Fink and I. Flatow. Modular JavaScript Development. In *Pro Single Page Application Development*, pages 35–48. Apress, Berkeley, CA, 2014.
- [17] A. German, S. Salmeron, W. Ha, and B. Henderson. MEAN Web Development. In *Proceedings of the 17th Annual Conference on Information Technology Education - SIGITE '16*, pages 128–129, New York, New York, USA, 2016. ACM Press.
- [18] Google. Material design guidelines, 2016.
- [19] D. Gossow, A. Leeper, D. Hersherberger, and M. Ciocarlie. ROS topics: Interactive markers: 3-D user interfaces for ROS applications, Dec 2011.

- [20] D. Guinard and V. Trifa. Towards the Web of Things : Web Mashups for Embedded Devices. *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009)*, pages 1–8, 2009.
- [21] D. Guinard, V. Trifa, and E. Wilde. A resource oriented architecture for the web of things. In *2010 Internet of Things, IoT 2010*, pages 1–8. IEEE, Nov 2010.
- [22] Guinard Dominique. Web of Things vs Internet of Things, 2016.
- [23] L. Kunze, T. Roehm, and M. Beetz. Towards semantic robot description languages. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 5589–5595. IEEE, May 2011.
- [24] K.-I. D. Kyriakou, I. K. Chaniotis, and N. D. Tselikas. The GPM meta-transcompiler: Harmonizing JavaScript-oriented Web development with the upcoming ECMAScript 6 “Harmony” specification. In *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, pages 176–181. IEEE, Jan 2015.
- [25] D. Lee, C. Ott, Y. Nakamura, and G. Hirzinger. Physical human robot interaction in imitation learning. *2011 IEEE International Conference on Robotics and Automation*, pages 3439–3440, May 2011.
- [26] J. Lee. Web Applications for Robots using rosbridge. *Brown University*, 2012.
- [27] K. Lei, Y. Ma, and Z. Tan. Performance comparison and evaluation of web development technologies in PHP, Python and Node.js. In *Proceedings - 17th IEEE International Conference on Computational Science and Engineering, CSE 2014, Jointly with 13th IEEE International Conference on Ubiquitous Computing and Communications, IUCC 2014, 13th International Symposium on Pervasive Systems,,* pages 661–668. IEEE, Dec 2015.
- [28] K.-J. Lin. Building Web 2.0. *Computer*, 40(5):101–102, May 2007.
- [29] S. Maitra and A. C. Mondal. Intelligence in web technology. pages 739–757. IGI Global, 2012.

- [30] S. Mayer. Web-based service brokerage for robotic devices. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing - UbiComp '12*, pages 863–865, 2012.
- [31] A. Mesbah and A. Van Deursen. Migrating multi-page web applications to single-page AJAX interfaces. In *Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR*, pages 181–190. IEEE, 2007.
- [32] T. Mikkonen and A. Taivalsaari. Using Javascript as a real programming language. *Network*, page 17, 2007.
- [33] G. Mohanarajah, D. Hunziker, R. D’Andrea, and M. Waibel. Rapyuta: A Cloud Robotics Platform. *IEEE Transactions on Automation Science and Engineering*, 12(2):481–493, Apr 2015.
- [34] S. Murugesan. Understanding Web 2.0. *IT Professional*, 9(4):34–41, Jul 2007.
- [35] S. Murugesan. Web X.0: A Road Map. In *Handbook of Research on Web 2.0, 3.0, and X.0: Technologies, Business, and Social Applications*, volume 1, pages 1–11. IGI Global, 2010.
- [36] P. Orduña, L. Rodriguez-Gil, D. López-De-Ipiña, and J. García-Zubia. Sharing the remote laboratories among different institutions: A practical case. In *2012 9th International Conference on Remote Engineering and Virtual Instrumentation, REV 2012*, pages 1–4. IEEE, Jul 2012.
- [37] T. O’Reilly. What is Web 2.0?, 2005.
- [38] A. Ortiz. Programming Web Services on the Cloud with Node.js. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, page 719, New York, New York, USA, 2016. ACM Press.
- [39] S. Osentoski, G. Jay, C. Crick, B. Pitzer, C. Du Hadway, and O. C. Jenkins. Robots as web services: Reproducible experimentation and application development using rosjs. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 6078–6083. IEEE, May 2011.

- [40] S. Osentoski, B. Pitzer, C. Crick, G. Jay, S. Dong, D. Grollman, H. B. Suay, and O. C. Jenkins. Remote Robotic Laboratories for Learning from Demonstration: Enabling user interaction and shared experimentation, Nov 2012.
- [41] L. D. Paulson. Developers shift to dynamic programming languages. *Computer*, 40(2):12–15, Feb 2007.
- [42] L. Peternel and J. Babic. Humanoid robot posture-control learning in real-time based on human sensorimotor learning ability. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 5329–5334. IEEE, May 2013.
- [43] B. Pitzer, S. Osentoski, G. Jay, C. Crick, and O. C. Jenkins. PR2 Remote Lab: An environment for remote development and experimentation. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 3200–3205. IEEE, May 2012.
- [44] A. J. Poulter, S. J. Johnston, and S. J. Cox. Using the MEAN stack to implement a RESTful service for an Internet of Things application. In *IEEE World Forum on Internet of Things, WF-IoT 2015 - Proceedings*, pages 280–285. IEEE, Dec 2016.
- [45] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. ROS: an open-source Robot Operating System. *ICRA*, 3:5, 2009.
- [46] M. Quigley, B. Gerkey, and W. D. Smart. *Programming robots with ROS*.
- [47] E. Ratner, B. Cohen, M. Phillips, and M. Likhachev. A web-based infrastructure for recording user demonstrations of mobile manipulation tasks. *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 5523–5530, May 2015.
- [48] L. Richardson, M. Amundsen, and S. Ruby. *RESTful Web APIs*. 2013.
- [49] I. Salvadori and F. Siqueira. A framework for semantic description of RESTful web APIs. In *Proceedings - 2014 IEEE International Conference on Web Services, ICWS 2014*, pages 630–637. IEEE, Jun 2014.

- [50] I. Santana, M. Ferre, E. Izaguirre, R. Aracil, and L. Hernandez. Remote laboratories for education and research purposes in automatic control systems. *IEEE Transactions on Industrial Informatics*, 9(1):547–556, Feb 2013.
- [51] Stack Overflow. Stack Overflow Developer Survey 2016 Results, 2016.
- [52] A. Taivalsaari, T. Mikkonen, M. Anttonen, and A. Salminen. The death of binary software: End user software moves to the web. In *Proceedings - 9th International Conference on Creating, Connecting and Collaborating through Computing, C5 2011*, pages 17–23. IEEE, Jan 2011.
- [53] W. Tan, Y. Fan, A. Ghoneim, M. A. Hossain, and S. Dustdar. From the Service-Oriented Architecture to the Web API Economy. *IEEE Internet Computing*, 20(4):64–68, Jul 2016.
- [54] S. Tilkov and S. Vinoski. Node.js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing*, 14(6):80–83, Nov 2010.
- [55] R. Toris. *Bringing Human-Robot Interaction Studies Online via the Robot Management System*. PhD thesis, 2013.
- [56] R. Toris and S. Chernova. Robotsfor.Me and Robots for You. 2013.
- [57] R. Toris, J. Kammerl, D. V. Lu, J. Lee, O. C. Jenkins, S. Osentoski, M. Wills, and S. Chernova. Robot Web Tools: Efficient messaging for cloud robotics. In *IEEE International Conference on Intelligent Robots and Systems*, volume 2015-Decem, pages 4530–4537, 2015.
- [58] R. Waibel, M. and Beetz, M. and Civera, J. and D’Andrea, R. and Elfring, J. and Galvez-Lopez, D. and Haussermann, K. and Janssen, R. and Montiel, J.M.M. and Perzylo, A. and Schiessle, B. and Tenorth, M. and Zweigle, O. and van de Molengraft. RoboEarth-A World Wide Web for Robots. *Robotics Automation Magazine, IEEE*, 18(June):69–82, Jun 2011.
- [59] D. Zeng, S. Guo, and Z. Cheng. The Web of Things: A Survey. *Journal of Communications*, 6:424–438, 2011.

Appendix A

Development Environment Setup

Following commands install client-side package manager *bower*, task runner *gulp* globally and then build client-side framework *Angular* and server-side framework *Express*.

```
1 sudo npm install bower -g
2 sudo npm install gulp -g
3 npm install express
4 bower install angular
```

Following are the commands for setting up ROS in virtual machine with packages that supports robot web tools.

```
1 wget https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -O - | sudo apt-key add -
2 sudo apt-get update
3 sudo rosdep init
4 rosdep update
5 echo "source /opt/ros/hydro/setup.bash" >> ~/.bashrc
6 source ~/.bashrc
7 sudo apt-get install ros-hydro-desktop-full
8 sudo apt-get install ros-hydro-rosbridge-suite
9 sudo apt-get install ros-hydro-web-video-server
10 sudo apt-get install ros-hydro-tf2-web-republisher
```

Appendix B

List of Source Codes

B.1 Client Side Routers Configuration in Web Motion Control Application

```
1 (function() {
2     'use strict';
3
4     angular
5         .module('pr2WebMotionControl')
6         .config(routerConfig);
7
8     /** @ngInject */
9     function routerConfig($windowProvider, $stateProvider, $urlRouterProvider) {
10         var $window = $windowProvider.$get();
11         var isAndroid = $window.navigator.userAgent.match(/Android/i);
12         var isiPhone = $window.navigator.userAgent.match(/iPhone/i);
13
14         if (isAndroid || isiPhone) {
15             $stateProvider
16                 .state('controller', {
17                     url: '',
18                     templateUrl: 'app/views/controller.html',
19                     controller: 'motionController',
20                     controllerAs: 'control'
21                 });
22         } else {
23             $stateProvider
24                 .state('monitor', {
25                     abstract: true,
26                     templateUrl: 'app/views/monitor.html',
```

```
27     controller: 'monitorController',
28     controllerAs: 'monitor'
29   })
30   .state('monitor.views', {
31     url: '',
32     views: {
33       "users": {
34         templateUrl: 'app/views/users.html',
35         controller: 'usersController',
36         controllerAs: 'users'
37       },
38       "robot": {
39         templateUrl: 'app/views/robot.html',
40         controller: 'robotController',
41         controllerAs: 'robot'
42       }
43     }
44   });
45   }
46
47   $urlRouterProvider.otherwise('/');
48   }
49  })();
```


B.2 Script for controlling PR2 arms as a whole

```

1  #!/usr/bin/env python
2
3  import rospy
4  from std_msgs.msg import String
5  from pymongo import MongoClient
6  from std_msgs.msg import Float64
7
8
9  class ArmsController:
10     def __init__(self):
11         self.node_name = 'pr2_arms_controller'
12         self.motions = {}
13         self.get_motions()
14         self.jointPublishers = {
15             'r_shoulder_pan_joint': rospy.Publisher('/r_joint/r_shoulder_pan_joint/command',
16             ↪ Float64, queue_size=1),
17             'r_shoulder_lift_joint': rospy.Publisher('/r_joint/r_shoulder_lift_joint/command',
18             ↪ Float64, queue_size=1),
19             'r_upper_arm_roll_joint': rospy.Publisher('/r_joint/r_upper_arm_roll_joint/command',
20             ↪ Float64, queue_size=1),
21             'r_elbow_flex_joint': rospy.Publisher('/r_joint/r_elbow_flex_joint/command',
22             ↪ Float64, queue_size=1),
23             'r_forearm_roll_joint': rospy.Publisher('/r_joint/r_forearm_roll_joint/command',
24             ↪ Float64, queue_size=1),
25             'r_wrist_flex_joint': rospy.Publisher('/r_joint/r_wrist_flex_joint/command',
26             ↪ Float64, queue_size=1),
27             'r_wrist_roll_joint': rospy.Publisher('/r_joint/r_wrist_roll_joint/command',
28             ↪ Float64, queue_size=1),
29             'l_shoulder_pan_joint': rospy.Publisher('/l_joint/l_shoulder_pan_joint/command',
30             ↪ Float64, queue_size=1),
31             'l_shoulder_lift_joint': rospy.Publisher('/l_joint/l_shoulder_lift_joint/command',
32             ↪ Float64, queue_size=1),
33             'l_upper_arm_roll_joint': rospy.Publisher('/l_joint/l_upper_arm_roll_joint/command',
34             ↪ Float64, queue_size=1),
35             'l_elbow_flex_joint': rospy.Publisher('/l_joint/l_elbow_flex_joint/command',
36             ↪ Float64, queue_size=1),
37             'l_forearm_roll_joint': rospy.Publisher('/l_joint/l_forearm_roll_joint/command',
38             ↪ Float64, queue_size=1),
39             'l_wrist_flex_joint': rospy.Publisher('/l_joint/l_wrist_flex_joint/command',
40             ↪ Float64, queue_size=1),
41             'l_wrist_roll_joint': rospy.Publisher('/l_joint/l_wrist_roll_joint/command',
42             ↪ Float64, queue_size=1)
43         }
44         rospy.Subscriber('arms_motion_control', String, self.do_motion, queue_size=1)
45         rospy.Subscriber('update_motions', String, self.get_motions, queue_size=1)
46         rospy.init_node(self.node_name)
47
48     def get_motions(self, msg=None):

```

```
35     if msg is not None and msg.data == 'update':
36         self.motions = {}
37
38         client = MongoClient('mongodb://<dbuser>:<dbpassword>@ds031607.mlab.com:31607/pr2')
39         db = client.get_default_database()
40         for motion in db.motions.find():
41             self.motions[motion['name']] = motion
42
43     def do_motion(self, msg):
44         if msg.data in self.motions:
45             motion = self.motions[msg.data]
46             for joint_name, joint_publisher in self.jointPublishers.iteritems():
47                 joint_publisher.publish(motion[joint_name.encode()])
48
49     def main():
50         ArmsController()
51         rospy.spin()
52
53     if __name__ == '__main__':
54         main()
```

B.3 Web Server Script in Web Motion Control Application

```

1  var path = require('path');
2  var express = require('express');
3  var app = express();
4  var server = require('http').Server(app);
5  var bodyParser = require('body-parser');
6  var mongoose = require('mongoose');
7  var Schema = mongoose.Schema;
8  var motionSchema = new Schema({
9    name: String,
10   r_shoulder_pan_joint: Number,
11   r_shoulder_lift_joint: Number,
12   r_upper_arm_roll_joint: Number,
13   r_elbow_flex_joint: Number,
14   r_forearm_roll_joint: Number,
15   r_wrist_flex_joint: Number,
16   r_wrist_roll_joint: Number,
17   l_shoulder_pan_joint: Number,
18   l_shoulder_lift_joint: Number,
19   l_upper_arm_roll_joint: Number,
20   l_elbow_flex_joint: Number,
21   l_forearm_roll_joint: Number,
22   l_wrist_flex_joint: Number,
23   l_wrist_roll_joint: Number
24 });
25 var motion = mongoose.model('motion', motionSchema);
26 mongoose.connect('mongodb://<dbuser>:<dbpassword>@ds031607.mlab.com:31607/pr2');
27 app.set('port', (process.env.PORT || 5000));
28 server.listen(app.get('port'), function() {
29   console.log('Application is running on port', app.get('port'));
30 });
31 app.use(express.static(path.resolve('client')));
32 app.use(bodyParser.urlencoded({ 'extended': 'true' }));
33 app.use(bodyParser.json());
34 app.use(bodyParser.json({ type: 'application/vnd.api+json' }));
35 app.get('/', function(req, res) {
36   res.sendFile(path.resolve('client/index.html'));
37 });
38 app.post('/api/motion', function(req, res) {
39   motion.findOneAndUpdate({ name: req.body.name }, req.body, { upsert: true }, function(err) {
40     if (err) {
41       res.send(err);
42     }
43     res.send('Motion (' + req.body.name + ') configured');
44   });
45 });

```

B.4 Visualisation of URDF Using *ros3djs*

```
1  var viewer = new ROS3D.Viewer({
2    divID: 'urdf',
3    width: 800,
4    height: 600,
5    antialias: true
6  });
7  viewer.addObject(new ROS3D.Grid());
8
9  var tfClient = new ROSLIB.TFClient({
10    ros: vm.ros,
11    angularThres: 0.01,
12    transThres: 0.01,
13    rate: 10.0
14  });
15
16  var urdf = new ROS3D.UrdfClient({
17    ros: ros,
18    tfClient: tfClient,
19    path: 'assets/',
20    rootObject: viewer.scene,
21    loader: ROS3D.COLLADA_LOADER
22  });
```

B.5 Initialisation of Joint Control Panel

```

1  // 'vm' is the convention of naming ViewModel in Angular.js
2  vm.armJoints = ['r_shoulder_pan_joint', 'r_shoulder_lift_joint', 'r_upper_arm_roll_joint',
   ↪ 'r_elbow_flex_joint', 'r_forearm_roll_joint', 'r_wrist_flex_joint', 'r_wrist_roll_joint',
   ↪ 'l_shoulder_pan_joint', 'l_shoulder_lift_joint', 'l_upper_arm_roll_joint',
   ↪ 'l_elbow_flex_joint', 'l_forearm_roll_joint', 'l_wrist_flex_joint', 'l_wrist_roll_joint'];
3  vm.bodyJoints = ['head_pan_joint', 'head_tilt_joint', 'torso_lift_joint'];
4  vm.jointLimits = {};
5  vm.armJointStates = {};
6  vm.bodyJointStates = {};
7
8  function getJointParameters() {
9      var param = new ROSLIB.Param({
10         ros: vm.ros,
11         name: 'robot_description'
12     });
13     param.get(function (value) {
14         var parser = new DOMParser();
15         var xmlDoc = parser.parseFromString(value, 'text/xml');
16         var robotDescriptionXml = xmlDoc.evaluate('//robot', xmlDoc, null,
   ↪ XPathResult.FIRST_ORDERED_NODE_TYPE, null).singleNodeValue;
17         for (var nodes = robotDescriptionXml.childNodes, i = 0; i < nodes.length; i++) {
18             var node = nodes[i];
19             if (node.tagName == 'joint' && node.getAttribute('type') != 'fixed') {
20                 var limit, minValue, maxValue;
21                 if (node.getAttribute('type') == 'continuous') {
22                     minValue = -Math.PI;
23                     maxValue = Math.PI;
24                 } else {
25                     if (node.getElementsByTagName('safety_controller')[0]) {
26                         limit = node.getElementsByTagName('safety_controller')[0];
27                         minValue = parseFloat(limit.getAttribute('soft_lower_limit'));
28                         maxValue = parseFloat(limit.getAttribute('soft_upper_limit'));
29                     } else {
30                         limit = node.getElementsByTagName('limit')[0];
31                         minValue = parseFloat(limit.getAttribute('lower'));
32                         maxValue = parseFloat(limit.getAttribute('upper'));
33                     }
34                 }
35                 vm.jointLimits[node.getAttribute('name')] = {
36                     minValue: minValue,
37                     maxValue: maxValue
38                 };
39             }
40         }
41         getJointStates();
42         $scope.$apply();
43     });
44 }

```

```
45 function getJointStates() {
46   var subscriber = new ROSLIB.Topic({
47     ros: vm.ros,
48     name: '/joint_states',
49     messageType: 'sensor_msgs/JointState'
50   });
51   subscriber.subscribe(function(message) {
52     var jointStates = _.zipObject(message.name, message.position);
53     _.forEach(vm.armJoints, function(name) {
54       vm.armJointStates[name] = parseFloat(parseFloat(jointStates[name]).toFixed(3));
55     });
56     _.forEach(vm.bodyJoints, function(name) {
57       vm.bodyJointStates[name] = parseFloat(parseFloat(jointStates[name]).toFixed(3));
58     });
59     $scope.$apply();
60     subscriber.unsubscribe();
61   });
62 }
63
64 getJointParameters();
```

B.6 Script for Ball Detecting and Tracing Using *OpenCV*

```
1  hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
2  lower_red = np.array([156, 43, 46])
3  upper_red = np.array([180, 255, 255])
4  mask = cv2.inRange(hsv, lower_red, upper_red)
5  mask = cv2.erode(mask, None, iterations=2)
6  mask = cv2.dilate(mask, None, iterations=2)
7  contours, hierarchy = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
8  if len(contours) > 0:
9      c = max(contours, key=cv2.contourArea)
10     (x, y), radius = cv2.minEnclosingCircle(c)
11     if radius > 10:
12         self.ball_center = (x, y)
13         cv2.circle(frame, (int(x), int(y)), int(radius), (255, 0, 0), 3)
14         cv2.rectangle(frame, (int(x) - 5, int(y) - 5), (int(x) + 5, int(y) + 5), (255, 0, 0),
            ↪ -1)
```

B.7 Example of Using Remote Method in Loop-back

```
1 Product.uploadImages = function(ctx, id, cb) {
2   ctx.req.params.container = id;
3   var Container = Product.app.models.Container;
4   Container.upload(ctx.req, ctx.result, function(err, fileObj) {
5     if (err) return cb(err);
6     var newImages = [];
7     _.forEach(fileObj.files, function(file) {
8       newImages.push(file[0].name);
9     });
10    Product.findById(id, function(err, product) {
11      if (err) return cb(err);
12      var images = product.images || [];
13      images = _.union(images, newImages);
14      product.updateAttribute('images', images, function(err) {
15        if (err) return cb(err);
16        cb(null, images);
17      });
18    });
19  });
20 };
21
22 Product.remoteMethod(
23   'uploadImages',
24   {
25     description: 'Upload product images',
26     accepts: [
27       { arg: 'ctx', type: 'object', http: { source: 'context' } },
28       { arg: 'id', type: 'string' }
29     ],
30     returns: { arg: 'images', type: '[string]' },
31     http: { verb: 'post' }
32   }
33 );
```


B.8 Example of Defining Model Relationships in Loopback

```

1  {
2    "name": "Shopkeeper",
3    ...
4    "relations": {
5      "products": {
6        "type": "hasMany",
7        "model": "Product",
8        "foreignKey": ""
9      },
10     "surveys": {
11       "type": "hasMany",
12       "model": "Survey",
13       "foreignKey": ""
14     },
15     "activities": {
16       "type": "hasMany",
17       "model": "Activity",
18       "foreignKey": ""
19     }
20   }
21   ...
22 }
23 {
24   "name": "Product",
25   ...
26   "relations": {
27     "shopkeeper": {
28       "type": "belongsTo",
29       "model": "Shopkeeper",
30       "foreignKey": ""
31     }
32   }
33   ...
34 }
35 {
36   "name": "Survey",
37   ...
38   "relations": {
39     "shopkeeper": {
40       "type": "belongsTo",
41       "model": "Shopkeeper",
42       "foreignKey": ""
43     }
44   }
45   ...
46 }

```

```
47 {  
48   "name": "Activity",  
49   ...  
50   "relations": {  
51     "shopkeeper": {  
52       "type": "belongsTo",  
53       "model": "Shopkeeper",  
54       "foreignKey": ""  
55     }  
56   }  
57   ...  
58 }
```

B.9 Example of Controlling the Data Access in Loopback

```
1 {
2   "name": "Activity",
3   ...
4   "acls": [
5     {
6       "accessType": "*",
7       "principalType": "ROLE",
8       "principalId": "$everyone",
9       "permission": "DENY"
10    },
11    {
12      "accessType": "READ",
13      "principalType": "ROLE",
14      "principalId": "Admin",
15      "permission": "ALLOW"
16    },
17    {
18      "accessType": "*",
19      "principalType": "ROLE",
20      "principalId": "$owner",
21      "permission": "ALLOW"
22    },
23    {
24      "accessType": "EXECUTE",
25      "principalType": "ROLE",
26      "principalId": "$everyone",
27      "permission": "ALLOW",
28      "property": "start"
29    },
30    {
31      "accessType": "EXECUTE",
32      "principalType": "ROLE",
33      "principalId": "$everyone",
34      "permission": "ALLOW",
35      "property": "addSurveyResult"
36    },
37    {
38      "accessType": "EXECUTE",
39      "principalType": "ROLE",
40      "principalId": "$everyone",
41      "permission": "ALLOW",
42      "property": "sendOffer"
43    },
44  ]
}
```

```
44     {
45         "accessType": "EXECUTE",
46         "principalType": "ROLE",
47         "principalId": "$everyone",
48         "permission": "ALLOW",
49         "property": "end"
50     }
51 ]
52 ...
53 }
```