Summary

The idea of this project is to design a distributed system which enables different types of robots to share their knowledge or experience by accessing one centralized server via www.

In the past, most of robots gain knowledge regarding the environment based on their own past experience. This obsolete approach is very inefficient, especially for two different robots which are located in distance want to acquire the knowledge collectively. As the development of internet technology, it is feasible to design a standardized platform for all types of robots to access no matter where they are currently at. Based on this assumption, a possible solution is suggested in this project.

Robots are expected to be associated with a camera to capture the objects' images. The digital data captured by the robot can be sent to the centralized server and the server will return a set of possible commands to that robot, so the robot can determine what action should be taken in the next step. One typical example is when the robot encounters a can with an arrow logo on it. It can understand the meaning by sending that arrow logo to the remote server, and probably it would be taught by the intelligent server to follow the direction which the arrow points to.

The system is not only large and complicated, but also applies a wide variety of programming technologies. It contains many modules and has roughly 5500 lines excluding the third party codes, and the main contribution and achieves involve:

1) I implemented a PHP web server based on Linux and Apache platform. A Mysql database schema was also set up to record all the necessary data. See chapter 2 and chapter 5 for more details.
2) designed a distributed system based on HTTP and TCP to instruct the communication between various components. The transferred data package is based on XML, so several xml parsers are written. See chapter 5.
3) I did a lot of research on object recognition area and understood the theory of SURF(Speeded-Up Robust Features)[10], in order to utilize it in this system. See chapter 3.
4) I read the specification of iRobot Create[1], and wrote code to manipulate its behaviors.
5) The integration of the above codes are essential to make it a complete network system.

Content

Chapter 1 Introduction

As the development of Robot technology, the robot learning ability has been increasingly emphasized. However, most of the robots today are learning from their own experience, which is relatively inefficient and restricted. The emerging WWW technology potentially provides a superb platform for all kinds of applications to make use of. Obviously, robots can also take advantage of it to share their experience via the internet.

The main idea of this project is to design and implement a standardized internet platform for robots to share their knowledge. A set of centralized servers are established to analyze incoming information from robots and send corresponding instructions back based on its internal logic reasoning mechanism. Besides, this project is closely associated with the theory of image matching, which enable the robot to request possible instructions from the server by identifying the object it is coming across. The process involves image capturing and objects recognition processes which are crucial topics in computer vision area.

On top of that, the overall architecture and the internet communication between the server and robots are the most important issue in this report. The design of overall architect follows the concept of distributed system [13]. The overall function set is divided according to where the component locates in. In terms of user cases, two typical scenarios Collecting Experience Case and Learning Knowledge Case underlie the total system. In order to clearly identify the intermediate data and permanently store them in the file system, the logic data structure and Database schema are applied widely. There are some necessary predefined communication steps which are taken sequentially inside the system and it also provides a well-designed interface to other application developer.

Several experiments are also taken place in the real environment to test the performance of the whole system. Unfortunately, because of the hardware limitation, the testing is only constrained to iRobot Create robot. In order to add more advanced features, more testing processes is also required.

This dissertation follows a rough structure shown below:

Chapter 2 discusses the overall architecture of the underlying system. Firstly, it will introduce the physical layout of the system, concerning about the different physical network components. After that, a rough outline of the system module will be given to clarify the functionality of different components. Finally, the intermediate data structure and database scheme is illustrated by Class diagram and E-R diagram.

Chapter 3 demonstrates image processing part of the platform. It reveals the basic requirements of the system and suggests a solution SURF [12] to go through this kind of work. Both the basic theory and the advanced implementation are mentioned in this section. Besides, the matching strategies used by this system are also involved.

Chapter 4 describes the running robot iRobot Create [1]. The physical characteristic

and open interface are the main issues in this chapter. Because of the speciality of this robot, the system utilizes a specialized procedure to control its behaviors.

Chapter 5 asserts the most essential part of the whole system, Internet Communication. The details of the Internet Communication model will be further discussed here. Each component's responsibility is also clarified in this chapter. On top of that, three common user cases are illustrated by sequence diagram to make them clear for users.

Chapter 6 is the experiment and result section. It fully focuses on the testing cases in the iRobot Create scenario. Moreover, this section talks about the experiment process in detail, in order to let the user know the running status of the system. The analysis is also mentioned in the following section.

Chapter 7 introduces another robot-based platform roboearth. A simple comparison between it and this system will be referred.

Chapter 2 Architecture Overview

To implement such a large-scaled system, it is worthy to consider more about the architecture design, which significantly influences the extendability, usability and performance of the target system. The chosen strategy divides the system into three different components, which are the centralized server cluster, the local computing unit and the underlying robot. The physical layout follows the principle of this approach, and it needs several functional servers in the server cluster and some local laptops as the local computing unit. (See Section 2.1) At the software design level, different modules are aiming to achieve different functions in the corresponding component. (See Section 2.2) Compared with other typical internet platforms, it puts more emphasis on the system performance, therefore, utilizing a bunch of low level programming technologies instead of the existing framework. Data storage and data structure is specialized in this system. Good database schema and intermediate data representation can increase the overall efficiency enormously. Thus, it is useful to discuss this issue in Section 2.3.

2.1 Physical layout

Currently, the whole experimental environment is built in my own laptop (Thinkpad T410 with Intel(R) Core(TM) i5 2.40 GHz CPU M520 and 2GB RAM), which runs Ubuntu GNU/Linux operating system. The robot I use for experiment is iRobot Create. (See Chapter 4 for more details)

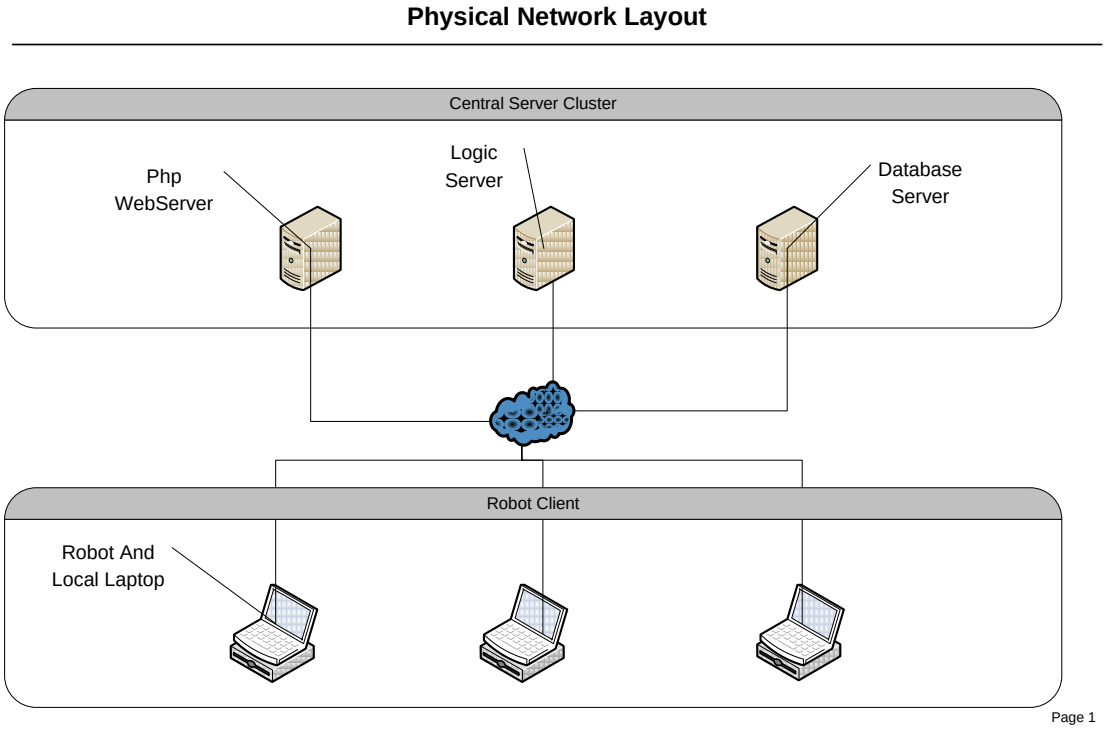

Figure2.1 Physical Network Layout

Figure 2.1 illustrates the physical network layout of the system. Physically speaking, the system consists of two different parts – Central Server Cluster and Robot Client, which are connected with each other via the network cloud.

In the Central Server Cluster, there are three different categories of servers. Php Web

Server is responsible for the administration of the whole system, including managing all the local files, maintaining the running the system and guaranteeing security of service for the potential clients. A Database server can be attached with the Php Web Server only. It records all the necessary data about the system or the valuable information. However, it is noticeable that any other components in this system do not have permission to communicate with the database server directly for the security consideration. The last one is the logic server; this server is used to do the logic reasoning work for different types of the robot. (See other chapter for details.)

In the Robot Client side, a local laptop should be attached with the iRobot Create[1] robot, in order to perform the necessary functionality, which means the local laptop would be moved whenever the robots is moving. These two stuffs can be connected with each other either by Bluetooth or USB port. It is system inborn defect to have a laptop attached with each robot. Obviously, this would reduce the usability, because one has to equip every robot with a computer in reality.

The reason for choosing this approach is that, ideally, the robot should have a strong computing element inside itself, so the system does not require an extra laptop to support the robot. Yet in practice, irobot create does not have a solid computing unit for the experiment, so there is no available alternative device to replace the attached laptop. In addition, there is no independent camera which can be directly attached with irobot to capture the video in this case. Thus the system has to make a compromise sacrifice the usability a little bit. However, for the future usage, the other robots which have the camera equipment and a strong computing processor can drop the local laptop to increase the system's flexibility.

2.2 Logical infrastructure and modules

As described before, the logical infrastructure can be shown in the figure 2.2. The whole system takes the network classic model client/server [2] to manage the communication. There should be a cluster of centralized servers and many local computing clients, which communicates through a standardized Internet Open Interface. (See Chapter Internet for more details).

2.2.1 Php Web Server

The database and file system is placed on top of the system, in order to provide access for the php web server. In the centralized server component, Php web server is responsible for providing a unique Web interface to both the system administrator and third-party programmer.

For the third-party programmer, the web interface api is typically used to manage the permanently stored system data resources. More specifically, the database system and file system can only be accessed by this php web server via Data Storage Module. In other words, if any other servers or clients persistent their data, they must present the relevant data package to the web server first. Thus, it can check the validity of the content and maintain the consistency of the database and file system.

For the system administrator, the system affords a web-based interface for routine maintenance. It is much more convenient and targeted to use a web application to

manipulate the file system, rather than directly change the database or file system. Therefore, Administration Module is separated out to fulfill this assignment independently. On top of these, Authentication Module is developed to guarantee the

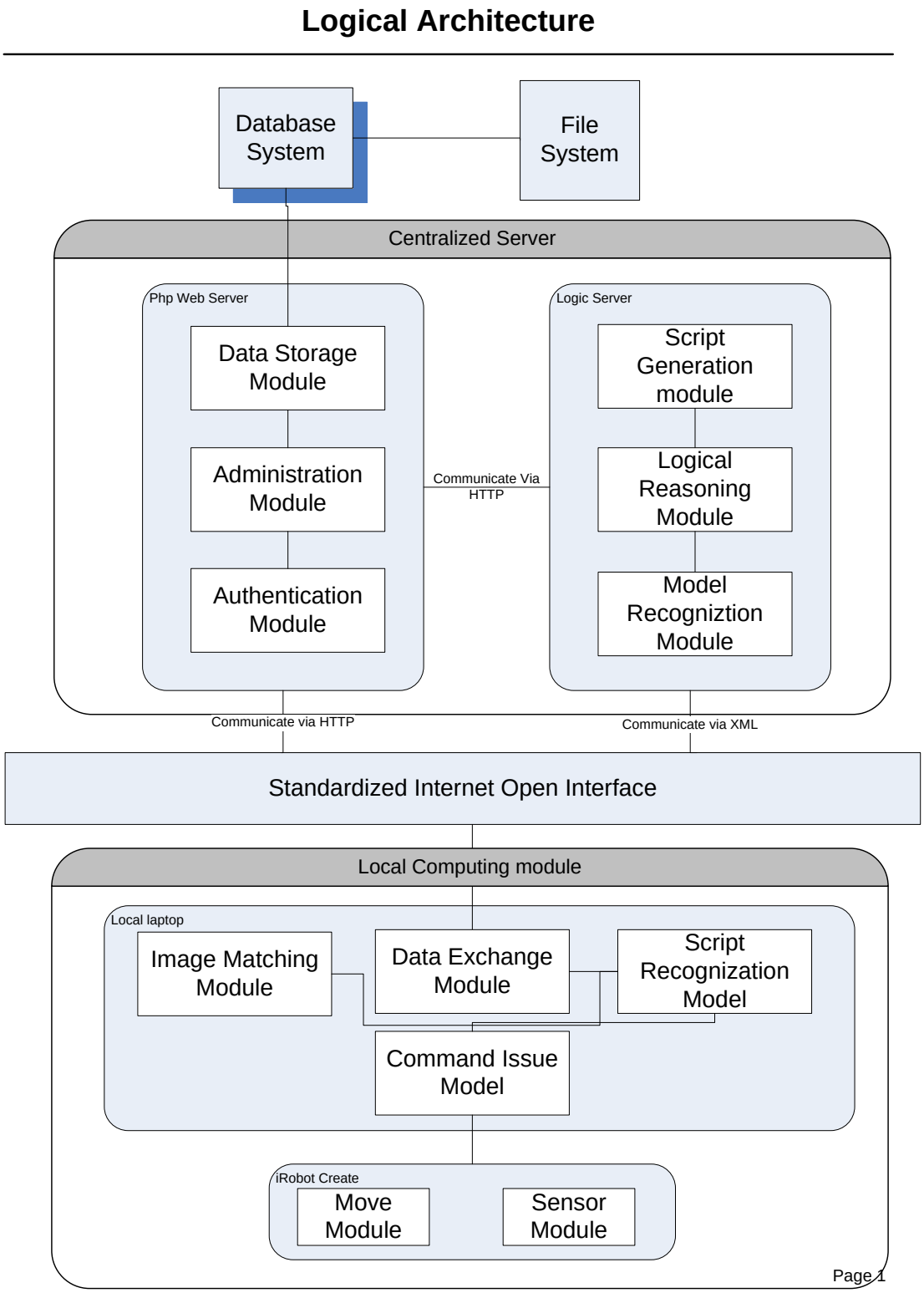

Figure2.2 Logical Architecture

system security by providing the entity checking and access control. Different users are grouped into different categories with different level of privileges. As a result, the storage system can be built in a safer environment without suffering arbitrary attack

from some unknown malicious hosts.

2.2.2 Logic Server

One typical Logic Server aims to receiving the data package from the robot client component, analyzing the valuable data and generating/responding the proper "action script" file.

The logic server is the essential component to achieve the goal of scalability and extendability. One logic server should be specialized to deal with the communication requested from one or several types of robot clients. Because there are so many different types of robots, the communication method between the server and the robot is varied significantly. Accordingly, the logic reasoning functions are totally diverse depending on the robot type. Therefore, it is the third-party programmer's responsibility to design the particular module to meet his own demand.

This system implements a straightforward logic server demo for potential users. This logic server puts the target on the iRobot Create, which can do some basic operations like moving and turning or read the attached sensors and send it back to the local computing unit. It can analyze the data package in the irobot problem set. This part will be discussed more in the chapter 5.

In general, a Logic Server is composed by three modules, Model Recognition Module, Logic Reasoning Module and Script Generation Module. Model Recognition and Script Generation Module are highly standardized. Model Recognition Module accepts the connection from the robot client and tries to extract information from an XML-based data package. Script Generation Module takes the input from the Logic Reasoning Module and reforms it into a system-defined standard "action script". So the web server can recognize the resulting file and make it permanent in the system's file system.

However, Logic Reasoning Module in the middle is highly flexible, and it can be written in whatever ways the client wants. The only restriction is this module must have the ability to accept the standard input and produce the output based on the predefined open standard. More importantly, it should be able to do some logic reasoning task according to the specific robot. In some simple cases, the logic is quite straightforward, like calculating the shortest path from one point to another based on the experience of 10 robots, which only records their random movement until they arrive at the destination. However, for some more complicated cases, the reasoning module may need to tackle with large-scale of data set, utilize the technology or knowledge in other computing field like machine learning or image processing. It depends on the programmer that what kind of the logic reasoning module he want to implement. For the logic server in the demo system, it can make itself acting as a good template for other users to take advantage of.

2.2.3 Local Computing module

As mentioned above, Local Computing Module is made of two components, one local computing unit and one robot. In this case, the local computing unit resides in a local laptop.

It has been divided into four parts. Image Matching Module is used to capture the image from the integrated camera, which is designed for this irobot create only. For the other robot, which does not need the image processing function, it is unnecessary to invoke this module. Then, Data Exchange Module is responsible for sending or receiving the data from either php web server or logic server. It strictly follows the standardized internet open interface, so it can easily interpret the command or share their experience with server. In most cases, the received form is a form of "action script", which means there should be a specialized module to recognize this script. Script Recognition Module is designed for this. More specifically, it also analyzes the data from the Image Matching Module and generates a localized command file to the Command Issue Module, which transforms this command file to the robot-recognizable language.

For the robot modules in the bottom, the functionality highly depends on the features of robots used for experiment. For example, iRobot Create is running in this case, and it only needs two types of action. One is moving, and the other one is reading the sensor. So the Move Module and Sensor Module are developed for controlling the iRobot Create.

2.3 Data Structure and Data Storage

Apart from the predefined Internet Protocol, the system needs well-designed representations of both the intermediate data structure and action script. More importantly, because the server is closely coupled with a Database, the system requires a well-rounded schema to record all the necessary information. Therefore, in order to design a robust programming interface, it is essential to define the data representation first.

2.3.1 Data Structure

The data package transmitted between different hosts need to be written under a predefined format, so each component in this system can communicate with others without ambiguity. For the communication taken place between the robot client and logic server, the requirement is even tougher. That is because, due to the consideration of overall performance, the communication between these two end users takes raw socket transmission as the principal implementation measure. However, designing a new internet protocol in the traditional way (like determining the offset and reading the binary data) can achieve a higher level of efficiency, but add more programming difficulties. This system balances the performance over the usability. As a result, the solution is to use XML [3] to define all the intermediate data structure and final data output.

XML stands for Extensible Markup Language. It also can be seen as Standard Generalized Markup Language (SGML). The proliferation of XML as a cross-platform technology in the internet environment is attributed to its powerful ability to handle the structured document. More specifically, a set of customized tags can be easily established to describe data content. Although XML file is more likely to occupy more space than binary stream in the data transmission process, the programmer is more likely to enjoy convenience and usability brought by XML.

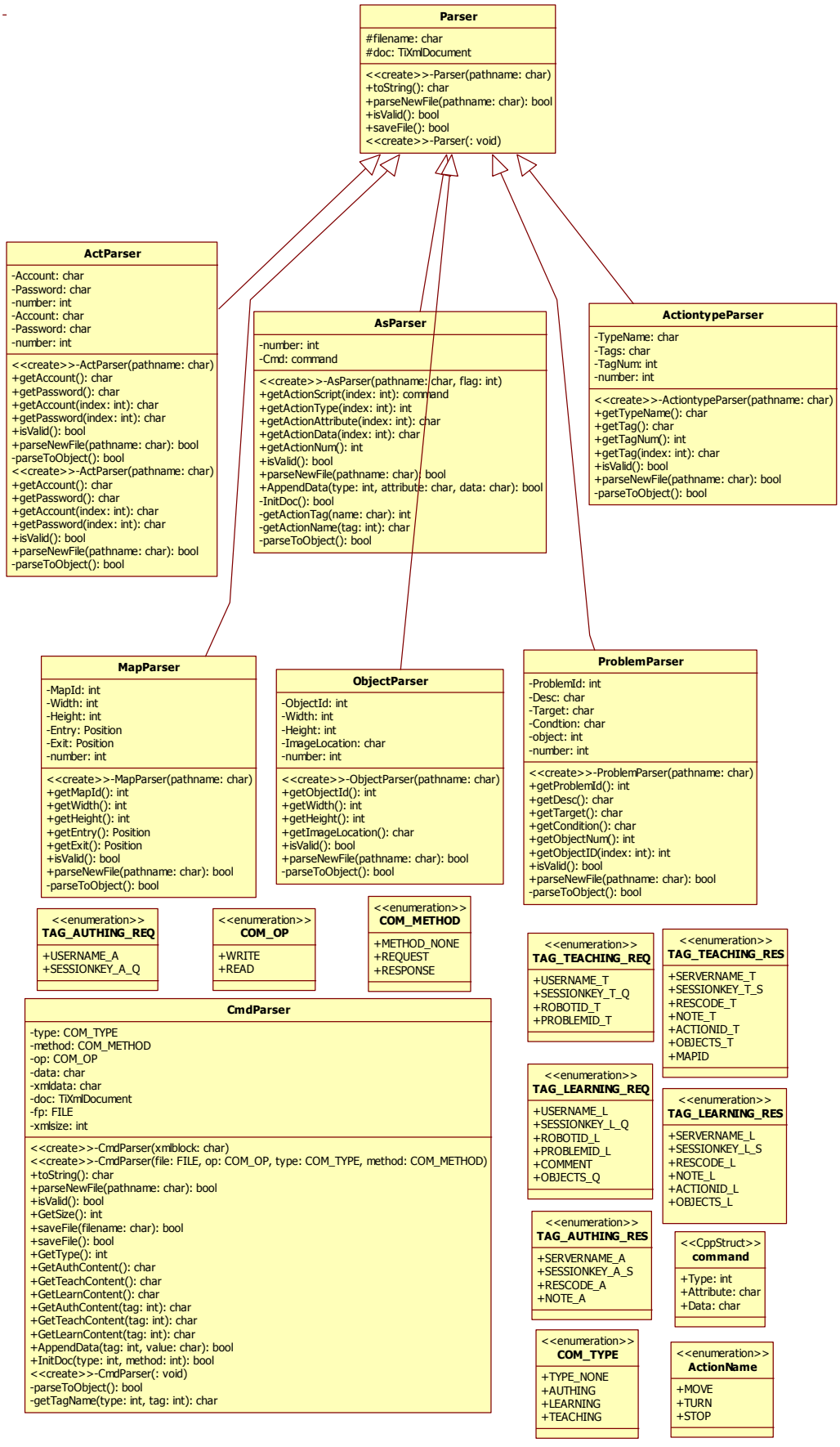

Figure2.3 Class Diagram of Parser Module

One fascinating benefit for using xml is that there are many light weight third party library which can do the xml parsing conveniently and efficiently. TinyXML[4] is a lightweight xml parser for C++ language. It perfectly meets the requirements for xml parser in this system, because of the following features:

1) Its size is quite small, so it consumes less system resources. Speed is particularly important in this system, because there are tons of reading and writing jobs during the daily running. The efficiency and performance is the most essential factor for choosing the right xml parser.

2) It is open source software, which means the programmer can use it more freely without thinking about paying the fees.

3) It is proved to be compatible with C++ and Linux environment. The system needs the library to be stable, especially under the linux/c environment. TinyXML is maintained by an experienced group, and several large open source projects are built on top of it, which indicates that it is fairly reliable.

4) The api it provides is simple and can be easily integrated into the system, which eases the potential programming complexity.

In this system, all the intermediate data including the temporary data file and internet data package take the form of XML standard. For every type of the data file, a specialized xml parser would be set up. Figure2.3 gives a class diagram about the outline of the parser module. There are eight individual parser classes which have already been set in this system. Class Parser is the super class of all the other specific document parser (except cmdparser). It contains some common member functions and several virtual interfaces, which can be rewritten by its derived class. ActParser, AsParser, ActionTypeParser, MapParser, ObjectParser, ProblemParser is designed to parse different type of the xml files.

The programmer can extend the super class parser to write his own xml file parser. By writing a parser module for different type of xml files, other modules can have a proper interface to manipulate the data without messing up the whole file. Besides, the set of permitted operations can be limited by class definition.(Thanks to the C++ class encapsulation feature)

The details of the previous six parsers will be discussed later. Yet it is noticeable that cmdParser in the diagram does not have any relations with other parser class. The reason is that this file defined the communication protocol between the robot client and the robot server, rather than a normal xml data file. The data packages transmitted between them can be read or written by this cmdParser. The details of this class will also be given in the chapter 5.

2.3.2 Data Storage

Database system takes all the necessary information and gets it stored permanently. The database schema is the most essential part to establish the desired database. A detailed database schema is shown in figure 2.4

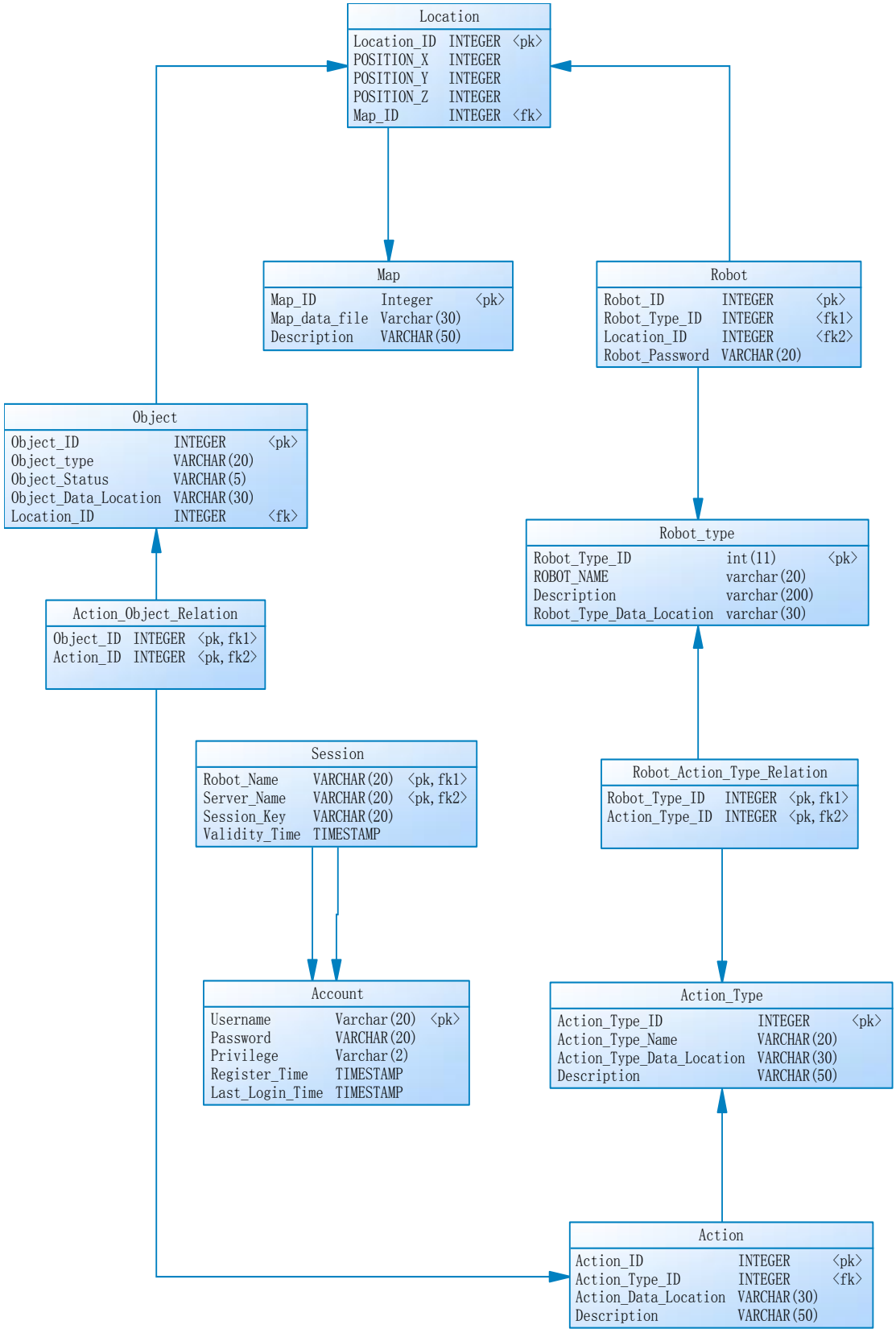

Figure 2.4 Database Schema

In this project, mysql[5] is chosen to be the DBMS. Mysql is an open source database management system, which is suitable for small-scale application. The speed of Mysql is very effective and it does not need the user to pay any fees to buy

it. More importantly, because the system is developed under linux operating system and the web server chooses php+apache platform, mysql has inborn advantage in working with these technologies.

Figure 2.3 gives a top view of the database schema. Many tables are built to record the necessary information. It is noticeable that the schema is probably to be changed in the future and it can be adjusted to fit the other demands.

The details of tables are illustrated below:

1) Robot and Robot_Type tables record the data about the running robot. The column can be extended to contain more information.

2) Action and Action_Type tables describe the action script, they have a column pointing to the real action xml file which can be parsed by the predefined action parser class.

3) Robot_Action_Type_Relation table shows what action can be taken by which robot type. The system can check the validity of the learning request from one robot by reading this table.

4) Object table records information about all the stuffs which have distinct meaning. The recorded object can be with some action scripts for clients to choose from. Accordingly, Action_Object_Relation table shows the linking between objects and actions.

5) Both object and robot should have a location, so that is the reason why the system uses a separate Location table. They all logged in a Map table, which documents the environment of a Map.

6) To provide the authentication and access control, session and account tables are set up for system usage. Account table stores the information of a registered system user, including his privilege and right. Session table records the temporary session for every established session between logic server and robot client, in order to guarantee the security of the communication.

The database system plays a very important role in this system. It stores so much confidential information, so it cannot be opened to the public. As a result it must be only accessed by the php web server, and reject all the other requests. Apart from the consideration of the security, the consistency and integrity of the data can also be preserved by taking this approach. If many other components, like logic servers, have the permission to access the data content, all the data there may be messed up by these two operations. Although mysql database has the ability to deal with the concurrent transaction, one third-party application may unconsciously damage the valuable data which is required by other peers. By incorporating a intermediate web server layer, the database system do not need to care about all these things, but concentrates on executing the simple retrieve or modify operations.

2.4 Conclusion

This chapter discusses the system architecture from a general view. A cluster of

server, local computing units and a set of various robots are required to comprise the physical environment. All the servers can also be divided into two categories, web server and logic server. They can communicate with Robot Client freely. But they differ in the functionalities, which is implemented by internal module. The intermediate data representation is defined by xml file to reduce the programming complexity. Finally, mysql database schema is determined to store all the necessary information about both the system and the clients.

Chapter 3 Object recognition

Some of The functionalities in the project rely on the image analysis process done by either the iRobot or Server. The object recognition is one of the basic functions for iRobot. In order to complete some complicated tasks, iRobot is expected to have the ability to match the sampled image with the image set of current known objects.

The image analysis process involves two major parts. When an iRobot first encountered one object of which it is told to record the relevant information for, it should start capturing its static image and uploading the image to the particular server running on the internet.

After the initial collection stage, other iRobots can retrieve information about the known object from the Internet easily. At this point, if iRobot just meets some objects that may already have been documented in the web server, it can do the matching between the image set and target object's image. If successful, server can immediately learn what kind of object it is and what options it have to cope with the current object.

Therefore, by internet communication, iRobots can get a way to share the knowledge via Internet and do the following operations more precisely.

The following section will briefly describe the basic theory and technology of the above concept.

3.1 Video capturing and uploading

The video capturing process is done by the external camera which is attached with the local laptop. Therefore, the whole process is performed by the laptop, rather than the robot. The image with the specialized logo (see section 4.2 for more details) will be recorded then uploaded to the server for the future comparison.

The implementation of capturing process relies on OpenCV library. The captured picture will be stored temporarily on the local disk. Its representation is defined by OpenCV as well, namely IpIlmage. OpenCV also provides some other useful utilities for handling the image. This project would take OpenCV as the third-party library for handling images.

On top of that, this process requires a low level of image analysis work, because images should be filtered by the default logo. The filtering process is in the next chapter.

3.2 Image analysis and matching

Finding the correspondences between different images is always a key research topic in the computer vision area.

One typical process of image matching takes three steps. First, a bunch of "Interest Point" is picked out from images. Second, the orientation information of each Interest Point is recorded in a feature vector. Finally, the iRobot can do the matching

by comparing the vectors.

SURF (Speeded-up Robust Features) is a stable algorithm aiming at recognizing different images. It utilizes a detector to extract the Interest Points and a descriptor to record the neighborhood of every Interest Point. The essential issues of SURF algorithm are how to find the appropriate Interest Points and what information to record for each selected Interest Points. One implementation of SURF, namely OpenSURF, has been released, which is supported by OpenCV. Thus, this project takes advantage of OpenSurf as major third-party library to do the image matching work.

In the following sections, the basic theory and implementation of Opensurf detector and descriptor will be discussed. For more details, please read the original paper"Speeded - Up Robust Features" [2].

3.2.1 Underlying theory of SURF.

SURF is a speeded up version of SIFT (Scale-invariant feature transform). Algorithms in SURF can be used to detect and describe the local characteristics inside images. Object recognition is just one of its general applications.

The recognition algorithm in SURF follows the general three steps mentioned above. In most scenarios, the database running in the server stores images which contain the fully detected and described objects. When the robot comes across an object in reality, it would send the captured image to the home PC. Then, the analysis module will compare the captured image with the stored images on the server.

In order to match objects under different environment, SURF designers believe that once finding more than three pairs of matching points in the same objects. It is accessible to establish their correspondences according to the theorem of Projective geometry.

However, be aware that the same object in the different images is usually varied in terms of rotation and scale. Because the images typically are captured under distinct conditions, the brightness and contrast should be taken into account as well. Therefore, the first task of SURF is to detect some kind of "Interest Points", such as corner points, boundary points, bright points in a dark area or dark points in a bright area, which are so outstanding that they would not be disappeared due to change on illuminance. Since two images have the same object, after extracting the Interest Points from both two source images, there should be some points in one image closely related with their counterparts in the other image. Based on such assumption, the fundamental part of SURF is to collect Interest Points. The algorithm of the SURF detector is calculating the local extremum of greyscale digital image by the assistance of a Hessian matrix. [11]

However, because the representation of the digital image is not continuous but discrete, it is inevitable to use filter (like haar wavelet filter) to calculate derivative and extremum. Yet unfortunately, the size of filters can affect the above calculation. More specifically, using filters with the alike size to calculate local extremum of two images with different scale of objects could probably lead to the situation that two

filters get different local extremums, although they should be same under the same scale.

SURF introduces the concept of Image Pyramid (figure 3.1) in reaction to such a problem. In that scheme, the original image can be imagined as the base of a pyramid, and higher level of the pyramid can be gotten by smoothed with a Gaussian and repeatedly sub-sampling the original image. As a result, two pyramids must have two sub-images containing the same object with the same size. Therefore, the detector can be invariant to scale. However, because the image is discrete, number of layers is finite. Obviously, the more layers the user divides, the more precise result he will get. Nevertheless, accordingly, the operation time for a larger number of layers will be augmented. Thus, it is necessary to balance the accuracy over the performance when determine the number of sub-sampled layers.

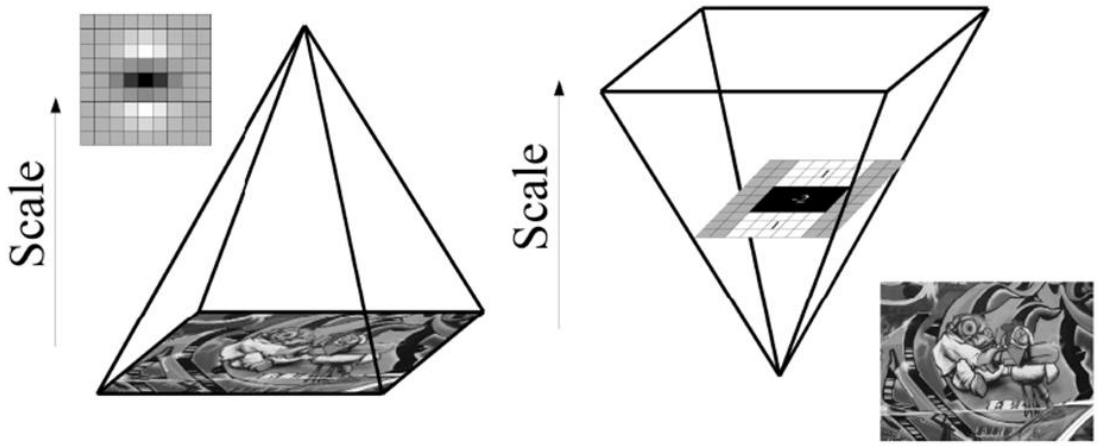

Figure 3.1 Image Pyramid referenced from[12]

Detecting Interest Points is just the first step of object recognition. Once the programme gets the whole set of Interest Points, how to let it do the points matching process precisely is the major task. The designer of SURF comes up with an idea that it is better to not only detect the each single Interest Point, but also describe its neighborhood. More specifically, SURF records and analyzes the nearby information around each Interest Point in the first place. Then SURF tries to discover some features based on the previous data and attach these features with the corresponding Interest Point. Adding the features with the Interest Point is just like one tree with strong root closely seized the ground. This process makes the Interest Point better represent the features around it.

However, there is another problem needed to be considered. Rotation may exist in the matching process because the robot may capture the image from different viewing angle. SURF should be invariant to rotation in this project. Otherwise, it could not be the applicable technology here. The solution for this in SURF is to assign a reproducible orientation to each Interest Point first so that the further extraction of these descriptor components can be based on this direction. Thus, images would be adjusted before matching.

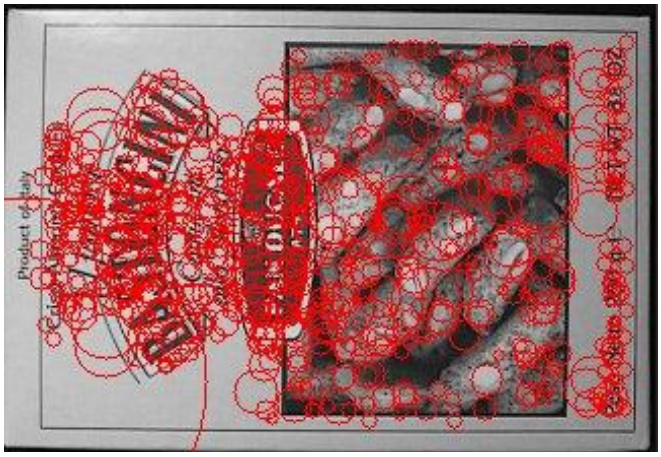

Figure 3.2 Interests Points and its neigbourhood

Figure 3.2 describes a system-generated image example in which all the interest points and their neighourhoods are labeled by red circles. The distribution of circles manifests location of the interest points and how it expands to seize the surrounding information for neighourhoood. From the image, it is obviously that the interest points are more likely to appear in the more characteristic area as mentioned before.

It is noticeable that SURF only takes advantage of grayscale digital image, which means the colour information would be useless in this case. So any object recognitions, which needs colour information could not work on this project.

3.2.2 The specification and open interface of OpenSurf library

As mentioned before, this project relies on third party library OpenSurf to achieve the object recognition work. This library has provided a set of well designed APIs for programmers. The library is written by C++. The whole function of the library has been decomposed into several prominent modules and classes.

Major modules are listed below:

1) Integral Image

Its function is to create the integral image representation of the imput digital image. This module should be invoked before any other opensurf functions are called.

Interface: IplImage *Integral(IplImage *img);

2) Fast-Hessian

This module acts as a detector. Its function is to calculate the set of Interest Points by fast hessian algorithm. This module takes the integral image as input and output the set of Interest Points.

Interface: FastHessian{

FastHessian(IplImage *img, Vector<Ipoint> &ipts, Int octaves,
Int intervals, Int sample, Float thres);

Void getIpoints();

};

3) SURF Descriptor

This module acts as a descriptor. Its function is to calculate the description of any given Interest Point and its image. This module return a vector of “described ” Interest Points.

Interface:

Surf {

Surf(IplImage *integral_img); Void setIpoint(Ipoint *ipt); Void getOrientation();

Void getDescriptor();

};

From the above process, the programmer can call the modules one by one to do the matching.

For the matching phase, the programmer can get the match points by using the OpenSurf API
GetMatches(Vector(Ipoint)P1,Vector(Ipoint)P2, vector<pair<Ipoint, Ipoint>> Match) to compare the interest points of the different images. One issue here is how to define the standard of the matching criterion.

In figure 3.3, the system takes an object image (on the left) and a scene image(on the right) for experiment. It tries to find all the interest points and their neighbourhoods of both images. After that, the system utilizes the above function to do the matching work and link all the similar pair of interest points with a line. The result output picture is shown in figure 3.4.

In the real implementation process, the system defined two different criterions for justifying whether the target images contain the specific object or not.

The first way is to simply count the number of correspondence between the two images. If the number exceeds a specified threshold, the system will believes that the object should be contained in this scene. In figure 3.4, this point-to-point relation could be seen as the white lines connecting the two images.

Figure 3.3 object and scene

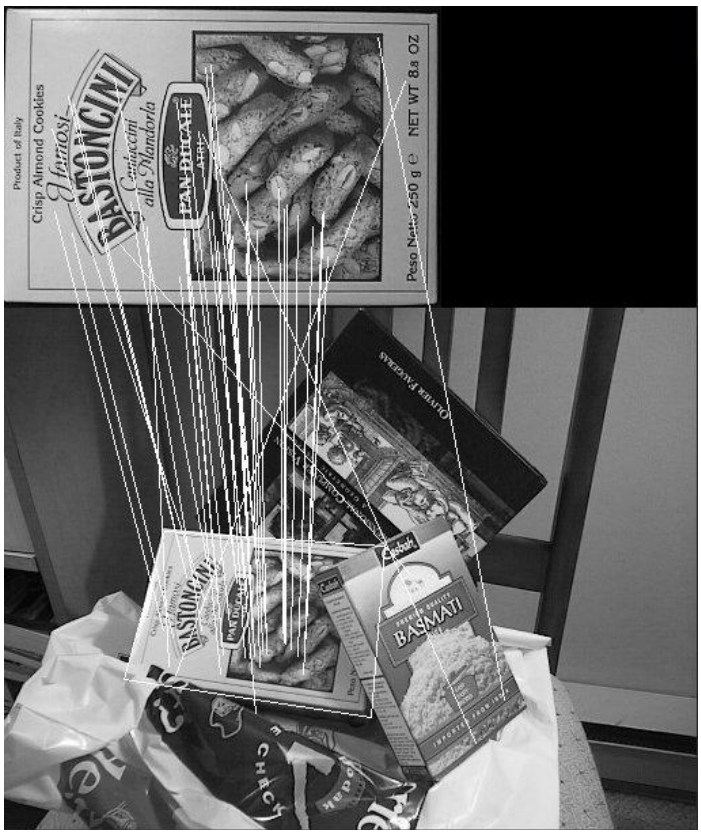

Figure 3.4 Correspondence between object and scene

However, the drawback of this approach is obvious. It needs the object image to be highly featured. Otherwise, the system would miscount the number by linking the object's interest point with some meaningless points in the scene picture. Therefore, this approach is not truly stable, unless the system has gotten a perfect object image (very hard to be mismatched with other objects) and a stable environment.

The second approach tries to locate the object by analyzing the location of the paired interest points in the scene picture. If the scene contains a specified object, the interest points are more likely to appear together within a limited area. In figure 3.4, it is easy to find that most of the white lines is pointing to a limited area in scene picture.

In opencv, the user can calculate the homograph matrix from cvFindHomography(src_pts, dst_pts, homograph), which leads to a flat area that indicates the object location. If the function returns 0, it means the matching process is failed. Otherwise, it is successful. The white square outside the target book object is drawn to indicate its location.

This method behaves much more accurate than the previous one. However, the only problem is this method would mislocate the object in the scene picture, when there are just a limited number of links between the object and the background scene. In that case, the system tends to do every effort to find the corresponding object, so it just analyse several existing links and tries to do the locating based on them.

To reduce the effect of this problem, the system takes the combined solution to solve this problem. Firstly, it will apply the second approach to try to locate the object. If successful, the first method would be used to drop the matching cases which contain too little links.

3.3 Conclusion

To sum up, SURF (Speeded-up Robust Features) together with OpenCV is an efficient tool to do the image matching job. Its accuracy and performance is a considerable plus for the whole system's running.

Chapter 4 iRobot Create

In this project, iRobot Create is chosen to be the foremost type of the robot for experiment. It is a well-supported robot which is distributed by a commercial company.

The client can control the robot flexibly by writing his own code which is well instructed by the developer company. The biggest advantage of iRobot Create is it provides users with a set of Open Interface, which can be utilized by the programmer.(OI will be discussed in section 4.2) This means iRobot is highly programmable and can be precisely controlled by the programme designer. Moreover, iRobot Create can collect data from its various sensors, like bumps sensors, wheels sensor and cliff sensors. (see section 4.3 for more details) Thus, the programmer can get sufficient information about both the internal status and environmental changes.

On top of that, iRobot is really easy to be extended with a wide variety of accessories so that it can be used to achieve more complicated than other robots.

4.1 Physical property of iRobot Create

Figure 4.1 shows the outlook of a typical iRobot Create. It can perform some basic operations like moving or playing songs.

Figure 4.1 Profile of iRobot Create, picture is referenced from [6]

One standard PC can be used to connect with iRobot Create by cable. Because iRobot Create itself have limited computing ability without any processing accessories and iRobot cannot capture the video by itself either, it is necessary to utilize an l to read the sensors' data and send commands to the iRobot. In the project, each iRobot Create is attached with a laptop. The laptop is acted as an intermediate layer between the server and actual iRobot. More specifically, it can also do some analysis work based on both the data packet transmitting from the server and incoming sensor's data packet from iRobot. Also, it can be used to send commands to iRobot or send feedback to the centralized server.

Looking Back to the physical aspect of iRobot Create, the laptop can be connected to a iRobot Create either by Mini - DIN Connector or Cargo Bay Connector. By default, it communicates with external device at 57600 baud. Thus, in order to complete communication, the laptop should set its own baud at 57600 by software.

4.2 Open Interface of iRobot Create

iRobot Create comes up with a set of Open Interface to let the programmer to control the robot's behaviors and read its sensors' data. Typical operations include:

1) mode commands

2) input commands

3) demo commands

4) Actuator commands

5) Scripting commands

6) Wait commands

To start customizing the command for iRobot, the user should firstly set some start-up code to let it know the command has been set. Then, the programmer needs to set the iRobot's mode by mode commands. There are four operating modes supported by iRobot Create, which are Off, Passive, Safe and Full. Different sets of permitted operations are defines for these four modes.

After setting the operating mode, the user can do the actual work by sending particular opcode. IRobot has provided several built-in demo operations like Cover and Dock or Drive Figure Eight, so users can conveniently use them by using their opcodes. Yet in this project, more complicated operations are necessary. For example, the iRobot sometimes may need to follow a specific path defined by the centralized web server and read the sensors' data in the meanwhile, it is impossible for iRobot to achieve this kind of work by the default demo commands.
As a result, the programmer can use more precise command to do such jobs. Actuator Command is introduced as a way to control the iRobot Create's actuators: like wheels, speaks, LEDs and so on. Typical behaviors involving driving, playing songs, showing digital output can be done by this set of commands. Besides that, Input commands enable one to read the data retrieved from iRobot Create's built-in sensors.
Another different category of Open Interface Command is Script Command. The users can define a combination of above commands to manipulate the iRobot to do some operations in order automatically. In this script, wait command can be used to let the iRobot wait for a time period or a specific event, which afford more flexibility for programming iRobot Create.4.3 Necessary operations for robots using www.
Because of the limited hardware condition, there are no additional accessories like command module or wireless module in all the scenarios. The most fundamental behavior in this project is moving and capturing images. As mentioned in Chapter 3, one laptop is attached with the iRobot Create, which means the image capturing and processing are designed to be handled in the laptop-side. Thus, it is not iRobot's responsibility to collect and receive the image data. So the main task for an iRobot

Create is quite clear. It is mainly used for moving. Common behaviors, which related with moving are following a predefined path, staying and waiting for one particular event to continue. The future requirements for driving the iRobot mainly concentrate on the accuracy and precision.

However, if the project goes further by other group or organization, the full set of available accessories should be taken into account. With the assistance of those devices, iRobot Create can perform more powerful operations than present.

4.4 Robot module implementation

The robot module can read the command from an action script. The action script for iRobot Create defines the robot's operation accurately, so it can perform the corresponding actions step by step.

Before the module executes the specific command, it is tremendously crucial to find some c++ library to automatically convert the basic operation to the string which can be accepted by irobot open interface. Open Interface itself is quite hard to use and debug conveniently. It takes a long string consisting of integers, which represent the corresponding commands need to be executed.

IRobot Create communication Library is a third-party library which wraps all the low level programming stuff into a C++ based class. iRobot::create class has a unusually large list of member functions, which directs the iRobot to perform specific operations. All these member functions can be categorized into three groups:

1) Action functions
2) Sensor functions
3) Setting functions

Action functions normally send the action operations to the irobot create. When the robot receives this signal, it will automatically perform this operation until next action operations arrive or this action is completed.
Sensor functions typically tell robot create which sensors need to be taken care of. If one of them get something to read, the remote control terminal can receive the data returned from the irobot create. Be aware that this information is also caught by the sensor functions.
Setting functions take care of the general setting of irobot create. In the initialization process, it handles the set up task. In the middle, it can reset the irobot configuration and make it perform some unusual actions.4.4.1 Action Script for iRobot Create
Action Script plays a very important role in this system. It is a XML file which can be recognized by the robot client, so the client can translate it into a type of command that the robot can perform. A typical action script XML file can be shown below:

```
<?xml version="1.0" encoding="UTF-8" ?>

<Action num="4">
    <Command>
        <Type>MOVE</Type>
        <Arg>100</Arg>
        <Data>5</Data>
    </Command>
```

```
    <Command>
        <Type>TURN</Type>
        <Arg>90</Arg>
        <Data>5</Data>
    </Command>
    <Command>
        <Type>STOP</Type>
        <Arg>2</Arg>
        <Data>2</Data>
    </Command>
</Action>
```

Every XML file starts with a declaration tag, which sets xml version as 1.0 and encoding as UTF-8. After that, the next tag should have the name as same as file type name; otherwise the file is deemed to be invalid. For instance, for the action xml file, the first tag except declaration tag is Action tag. The data inside this tag varied depending on the file type. Tthat is also the reason why the system needs so many different specialized parser classes to extract the real data content.

An action file comprises a number of command tags, which defines a single action can be read by a human being. In the irobot scenario, Comand tag has three sub-tags, named Type, Arg and Data, which are interpreted and forwarded to the irobot.

The control module of irobot create normally follows the process below:
Create Robot;

```
Robot.initialize();
Robot.setSensorListener();
While(Robot does not complete the job)
    START LOOP:
    IF teaching process is running THEN
    getContentFromActionXMLfile();
    END IF
    execAction();
    ListenToSensorStatus();
    IF learning process is running THEN
    WriteActionScript();
    END IF;
    AnalyseData();
    END LOOP;
```

1) Initialize the create configuration. (like switching to the full mode)
2) Set the sensor listener
3) Enter into a while loop, stop until the job completes
4) Fetch the data content from XML file (if applicable)
5) Execute the command according to the action type and action args
6) Generate the new action script(if applicable)
7) Detect the sensor status
8) Make the analysis and start a new cycle.

It is noticeable in the current system, the image capturing and matching process is happened in the analyseData() function. However, its function is not restricted to the image processing; it can be customized to other type of task. Thus, this can be very flexible to achieve the robot-oriented job.

4.5 Conclusion

To sum up, iRobot Create underlies the whole system. However, the robot can be various as long as the programmer implements its interface and writes its own code. It is expectable that the more power the robot has, the stronger the system is.

Chapter 5 Internet Protocol and Communication Model

Internet Protocol and Communication Model are the most essential part in this system. It is the real infrastructure that enables the other party to utilize this system. In order to design a better open interface, the internet module is more likely to put more emphasis on flexibility and extendability.

This chapter firstly discusses the Internet Communication Model, which gives a rough introduction of the function of main components and how they communicate with each other. Then, three leading user cases in this system are presented to illustrate the basic internal protocol. After that, some implementations details are described to make the reader better understand this system.

5.1 Internet Communication Model

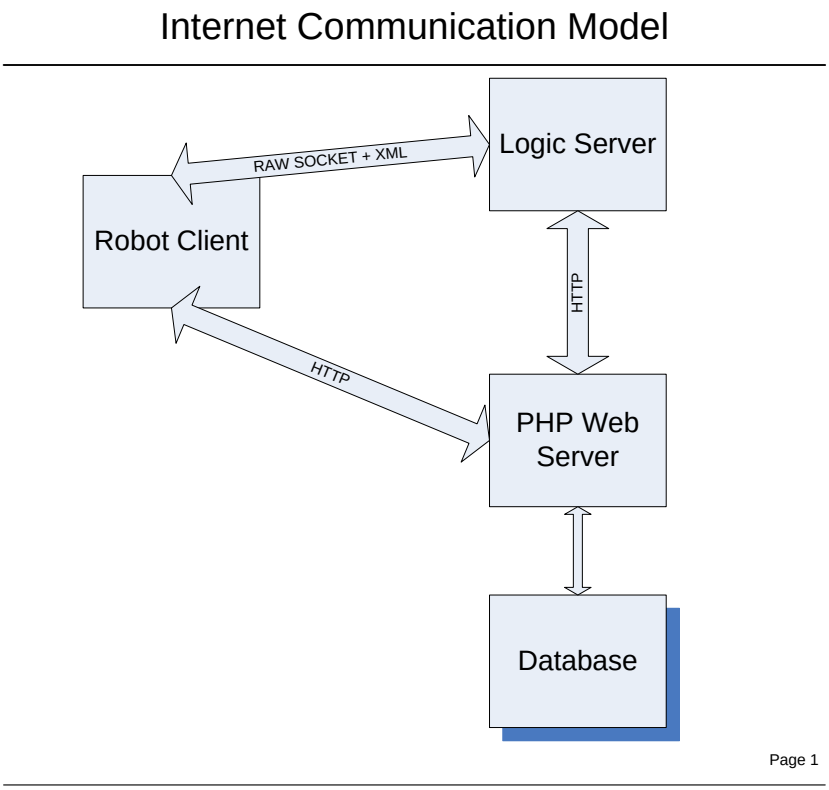

Figure 5.1 Internet Communication Model

Internet Communication Model defines the basic structure of the system network. In chapter 2, three components - Robot Client, Logic Server, PHP Web Server, are introduced concisely in terms of modules' interaction. However, the reason for having these three different hosts is from the consideration of internet communication.

5.1.1 Robot Client

The network architecture follows the classic Client/Server Model [2]. However, it is worth pointing out that the Robot Client here is not a traditional client in a network model. To increase the performance of the whole system, it is not wise to put too many functions on the client side. Any logical computing jobs should be placed in the logic server side instead of Robot Client. The power of the local computing device is

normally limited, while either the logic server or the web server can be ran on much stronger computers, even super computers. The system does not expect the robot client to have strong computing ability in the first place. Thus, that is why the system makes servers deal with the most time-consuming job.

However, it does not mean robot client should be as tiny as possible. In practice, it is not a simple output and/or input device. Robot Client's feature is heavily determined by the nature of the running robot, which means the module in this part is highly diverse. It is extremely hard to find a unique standard to meet all the robot's requirements without any preprocessing work. And enabling the internet standard to accept all types of data would be impractical and deteriorate the extendability greatly. To let Robot Client utilize the internal protocol, it is necessary to convert the robot-based data into a universe standard, which can be recognized by the server on the internet. As a result, Data Exchange Module and Script Recognition Module are implemented to do this kind of work.

Besides, some robots may need assistance from other devices, or perform some combined operations collectively. Theoretically, Robot Client should have the ability to coordinate the operations between different devices. Yet in practice, this module is quite hard to standardized, so the third-party user should write customized code for doing this if he wishes.

In the iRobot Create scenario, a camera should be attached with the robot to capture the image. Unfortunately, the system does not have an independent camera device for experiment. Instead, as illustrated in chapter 2, a local laptop with an integrated camera is connected with iRobot Create. In other words, the laptop is responsible for reading the image from the camera and doing the analysis. There are two choices to do this stuff.

1) The first one is sending all captured images to the logic server. So the server can make the analysis and generate the corresponding action script afterwards. This approach is consistent with the system designing criterion, which considers the heavy computing job should be placed in the logic server. However, the system cannot ignore the side-effect of keeping sending images over the internet. The local network can be tied down if there are too many robots running together. And the logic server is required to have a extraordinarily strong network condition.

2) The second one is sending the selected images or even raw data for logic server to utilize. One normal way is to pick the meaningful image and then do some preprocessing job. This method can increase the usage of the network and reduce the transfer delay, therefore improving the overall efficiency. The drawback is also obvious. Sometimes the server just cannot get enough information about the robot client's current status. The accuracy would be compromised.

In the current stage, the system takes the second approach after weigh the pros and cons of each solution. But for the future revision, maybe the first one is more appropriate, because the network problem may not be a big problem in the future.

5.1.2 Logic Server

Logic Server is the analysis component in this system. It is good at handling the reasoning problem. The computing power in one Logic Server is supposed to be very strong. So it can deal with some computing-consuming problem concurrently.

Logic server is a multi-threaded server written by c. It has the ability to maintain a large amount of active connection in the same time. As a result, every time the server establishes a new session with Robot Client, it should launch a new thread and let it handle the subsequent communication. This implementation is done by using POSIX pthread under linux platform.

The communication between the logic server and robot client is based on raw socket programming and XML tech. XML file is used to define the internal protocol, so it can be transferred between two end hosts. The reason why the system takes advantage of socket programming is that the efficiency and performance are the most essential factor in this communication process. As mentioned before, various data contents are supposed to be transferred under this protocol. It is predictable that the data package can probably be quite large. Robot Client may not have a strong network condition, which will affects the performance of the whole system.

The most notable feature of Logic Server is it is scalable and extendable. Because the problem sets for different robots are various, the logical reasoning module highly depends on the robot's feature and its corresponding problem sets. It is vital that one third-party users need to write their own logical reasoning module to perform the specific learning task. For example, if one user wants his robot to learn how to take one book to the shelf, he needs to define the reasoning module first, which describes how to move the book accurately to the shelf based on the experience. Because this case can hardly be copied to other robots' scenarios, for another user, it is difficult to reuse this existing module with his running robot and problem sets.

However, due to the standardized Internet Open Interface, Model Recognition Module and Script Generation Module are relatively fixed. So the third party user can establish their own servers by rewriting the Logical Reasoning Module and reusing the rest.

5.1.3 Web Server

Web Server is a website application built in a centralized computer. It is based on the classic LAMP (Linux+Apache+Mysql+PHP) [8] platform and can be utilized for both system administrator and third-party programmers.

Apache Web Server has a web interface for the system administrator to maintain the data storage system or change system configuration. The web pages are written by basic HTML and PHP. It would be quite convenient for the user to handle the daily maintenance through a website interface.

More importantly, Web Server is the only component which has permission to access the database. It is also noticeable that any communication involving web server uses HTTP. For users who want to make changes in the Database, the solution is to use the

api provided by a web server.

The web server interface is mainly concerning about two aspects. Managing the system resource, including database and file systems, is done by using some php files. Moreover, web service api guarantees a certain level of security.

As described before, either logic server or robot client cannot access the database directly. Instead, they need to use the web interface to complete the necessary operations. During the analysis process, some checking operation will be placed in order to prove the validity of the received data. Only a limited set of operations can be carried out by the other host, which can preserve the consistency and integrity of the stored data.

Besides, the web server can also secure the session established between Robot Client and Logic Server by using the session key. On top of that, the web server records all the details about valid accounts. All users are categorized into different groups and given different level of privilege. By using this information, system can implement access control based on the privilege for the data storage system.

5.1.4 Interaction between components

As shown in figure 5.1, the communication methods between the various components are different. For Logic Server and Robot Client, the communication is made by raw socket programming, whereas it is through http for those connections established between the web server and other components.

HTTP [14] (HyperText Transfer Protol) is the most widespread network protocol for the internet application. It is initially used to define a method to publish and receive HTML pages.

It is obviously that, the web server can communicate with other type of server via HTTP conveniently. It consists of dynamically generated webpage (written by PHP in this case), so the other hosts can make the necessary operations by accessing these pages.

For a server written by c/c++, it needs some third-party library to implement HTTP functions, like parsing the header, getting the content and sending the data. Writing from scratch is a pain for the individual programmer. Because the programme robustness highly depends on the code itself, some small mistake in the code may lead to a risk for crashing the whole system. Thus, it is wise to utilize some well-designed HTTP library.

Libcurl[9] is a multiprotocol file transfer library, supporting all the famous internet protocol, including HTTP, HTTPS and FTP. It is highly portable, which is proved to be perfectly compatible with most UNIX systems. libcurl is free, thread-safe, IPv6 compatible, feature rich, well supported, fast, thoroughly documented and is already used by many known, influential and successful companies and numerous applications.

In addition, this system wraps the libcurl up to perform its own operations.

Httpmodule in the source is responsible for implementing the necessary operations. And then it is compiled to be a dynamic linking library for other modules to use.For the interaction between Robot Client and Logic Server, the situation is much simpler. They use low level TCP protocol to make the communication. The data content is defined by the XML file.

5.2 Internet Protocol

From figure 2.2, it is clear that there is an intermediate layer between servers and Robot Client. Its function is to provide the local software with a standard interface, which enables the user to take advantage of the service supported by the server.

However, there is another issue to care about. What standard should the interface follow? In other words, what is the programming convention in the communication to process? In response to this question, the system self-designs a simple internet protocol to do the communication job.

Be aware that the internet protocol is defined by the XML file. There is a specialized XML parser class cmdParser for reading and writing this XML file. A typical command XML file is shown below.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Protocol type="2" method="1">
    <data>
        <Username>irobot_server</Username>
        <SessionKey>rgftoewq3</SessionKey>
        <RobotID>1</RobotID>
        <ProblemID>1</ProblemID>
        <Comment>OK</Comment>
        <Objects>1</Objects>
    </data>
</Protocol>
```

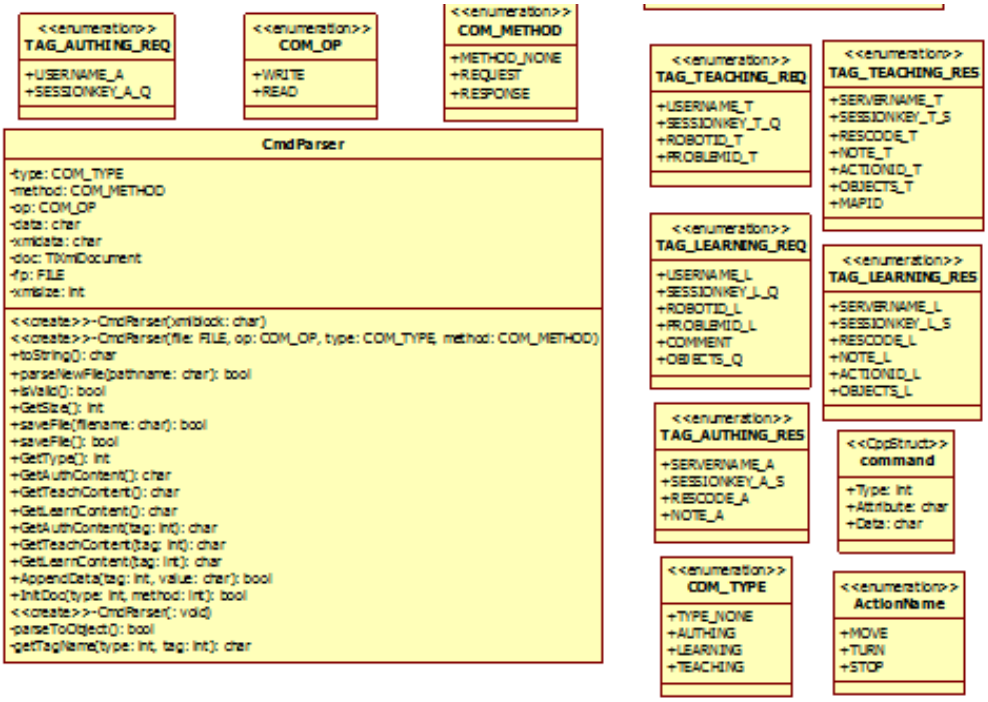

Figure 5.2 Command Parser Class

The command starts with a xml declaration tag, which defines the xml version and encoding method. After that, the Protocol tag is showing up. It is the identity tag to specify which type it is. Inside Protocol tag, there should be a data tag containing the real data content. The element embedded in the data tag differs accordingly to various type of the xml file. In the current stage, all the possible values are listed in figure 5.2.

Protocol tag has two attributes – type and method. Type is used to identify whether the command needs to be read or written. Method tells the system what kind of operation this file contains. Currently, there are three methods: authentication, teaching and learning. Next section will discuss more about the details.

According to the specified method, the permitted tag set inside the data tag is different. Normally, it contains SessionKey, Username, RobotID and Comment. Yet the tag set may be extended in the future for fitting more other notable robots. There are three classic communication scenarios in this system, they are:

1) Authentication:Every established session should be authenticated first by the PHP server. This process takes a random session key to avoid data leakage and malicious attack.

2) Collecting Experience:The remote logic server can start a collecting session to make the local suitable robots grabbing data from their own environment. The incoming data are analyzed first, and then integrated into an action script by PHP server for future reuse.

3) Learning Knowledge:The robot can issue learning request for some known problem to the logic server. If found, the robot can get an url resource to access, which is administrated under a PHP server. These three common scenarios

comprise the basic Internet Infrastructure for the system. Every other user who wants to take advantage of this system should at least follow the above standard.

5.2.1 Authentication Process

Before any communications happens between any hosts, it is mandatory to authenticate all the end-users by the web server. The web server will also assign a temporary session key to the end-users to guarantee the communication security. As described in Figure 5.2, the typical authentication process is:

1) In order to start a new session with a logic server, Robot client will issue a request for ordering a temporary session randomly generated by the web server. Once the web server receives this message, it firstly identifies the id of the sender and checks its privilege via database. If the user account is valid, the web server will return with a session key, which is recorded in the database for future usage as well. Be aware that this session key will be expired by a certain period. In other words, the validity of this session key can only last for a limited time length.

2) After Robot Client receives the session key, it can use it to make the connection with the logic server. In the connection phase, Robot Client would send the session key as well. Once the logic server knows the session key, it would check its validity by using web server's web interface. Then the web server would look up the database and return the result back to the logic server.

3) If the result is positive, the logic Server just confirms the connection and starts the communication from then on. Otherwise, it will decline this connection and ask the Robot Client to try to reconnect if applicable.

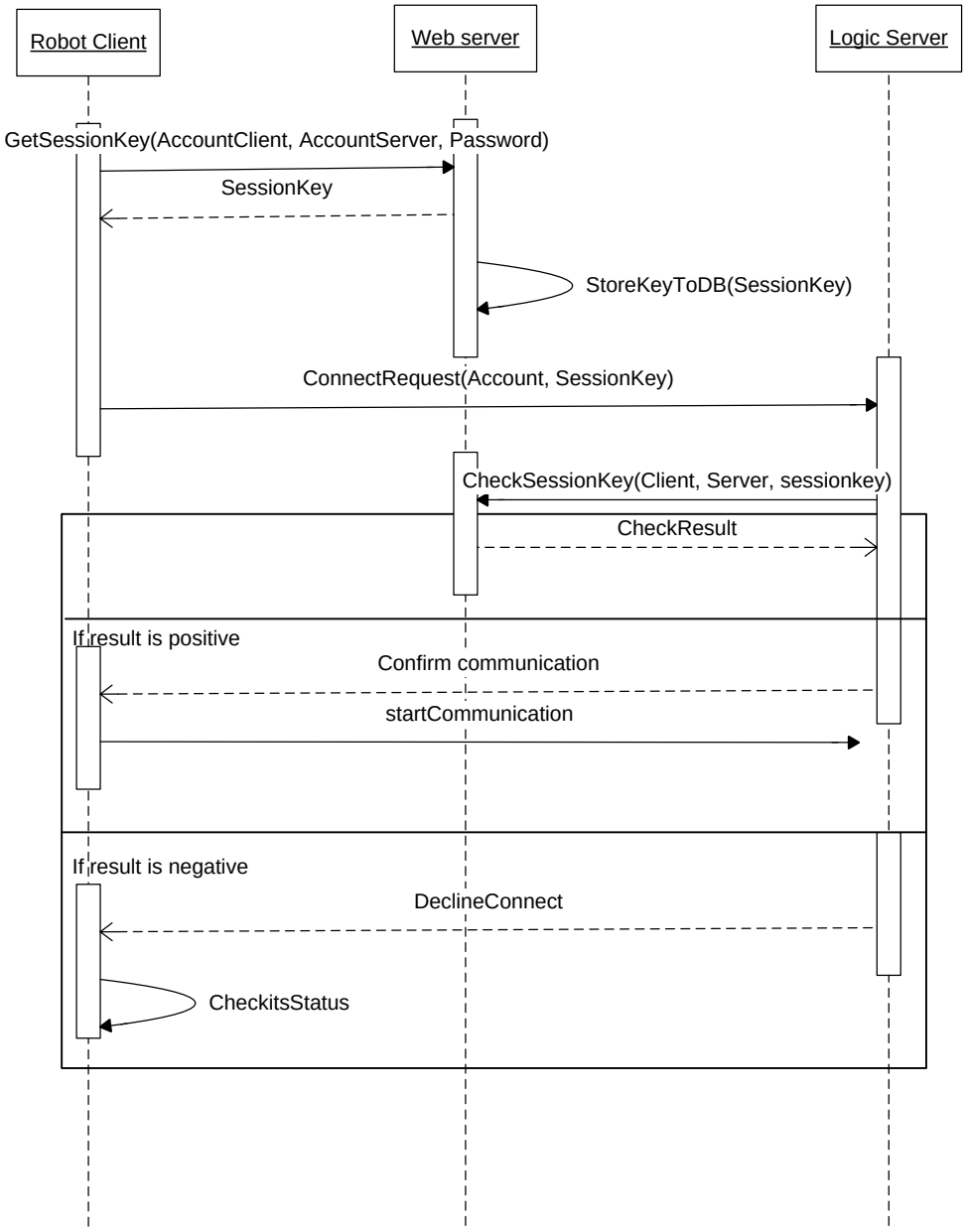

Figure 5.2 Authentication Process

The meaning of introducing the authentication process is to guarantee a higher level of system security. The web server has the responsibility to secure every component in this system. Otherwise, any third-party attackers can easily disguise as some legal users or super users, which may damage the security of the whole system.

Therefore, authentication process is mandatory to all the connection established by any component in the system. Any subsequent communication must specify the corresponding session key in the first place. After verifying the validity of the communication entity, the normal communication can then be started.

5.2.2 Collecting Experience Process

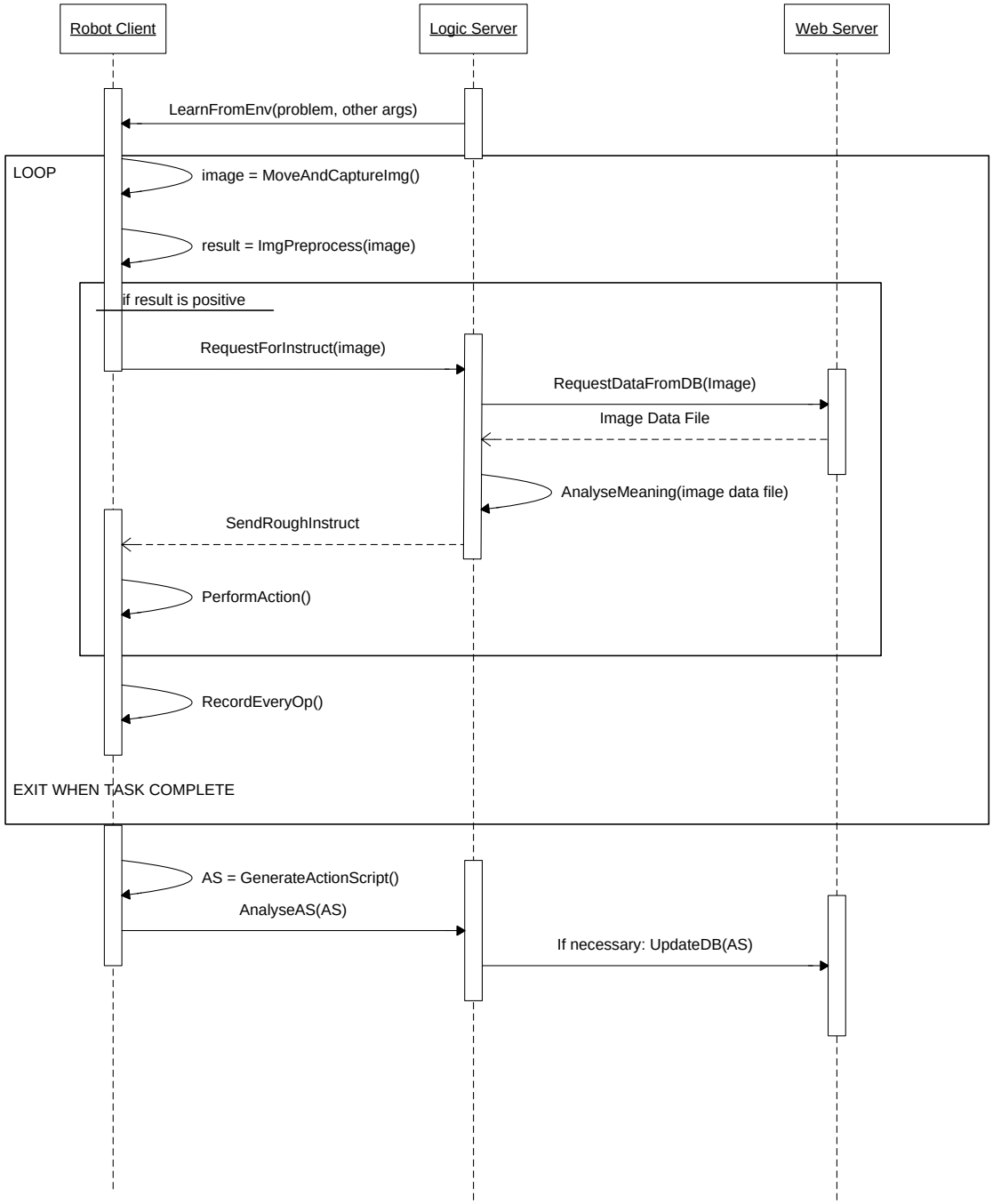

Figure 5.3 Collecting Experience

After doing the authentication process, the system can choose to either collect Experience or Learning knowledge. The collecting process is shown in figure 5.3. The general procedure involves all the three components. It is noticeable that the scenario here is partly based on the irobot create. Some steps may not be carried out for other types of robots. For example, the image capture function here only works in the camera-equipped robots. For other types of robots which do not need the image

processing job, it is not necessary to take advantage of this function.

As mentioned before, the communication between Logic Server and Robot Client depends on the specific type of the underlying robot. The sequence diagram above is based on iRobot Create. However for other robots, it still defines the general idea about how to make the communication.

The typical communication process can be described as follows:

1) When the logic server wants to get the learning data from iRobot Create, it will issue a learning request with the target problem and other learning parameters (such as objects and maps).

2) After Robot Client receives the learning request, it will start to move randomly and capture all the images it encounters. Then the captured image will be compared with several predefined images. If get matched, the corresponding image will be sent to the logic server. This process is used to ease the network pressure of the system. Sending every captured image is more likely to put more stress on the data transmission performance. Thus, the system just picks the image which contains some predefined label and then sends their data structure to the logic server.

3) After Robot Client selects the meaningful image, it will send it with an instruction about requesting the further information. In the iRobot Create case, this image should contain some objects which gives a suggestion about what to do in the next step.

4) Then, the logic server compares the received image with the image set, which is obtained by getting all the image resource from the web server. Once it is matched with some images, the logic server would forward this image id to the web server to get the relevant information.

5) After that, Robot Client analyses the data file retrieved from the web server and sends the action script back to Robot Client.

6) So Robot Client can perform this series of operations. After finishing one loop, the learning process will move on but every operation will be recorded for future usage. The whole loop will not be ended until the task truly completes.

7) After the task is finished, an initial action script would be generated by Robot Client and then sent to the logic server for further analysis. The logic server will slightly modify the action file and then update it to the database via web server interface.

Collecting Experience Process underlies the whole learning process. The logic server must have some action scripts first before Learning Knowledge Process.

5.2.3 Learning Knowledge Process

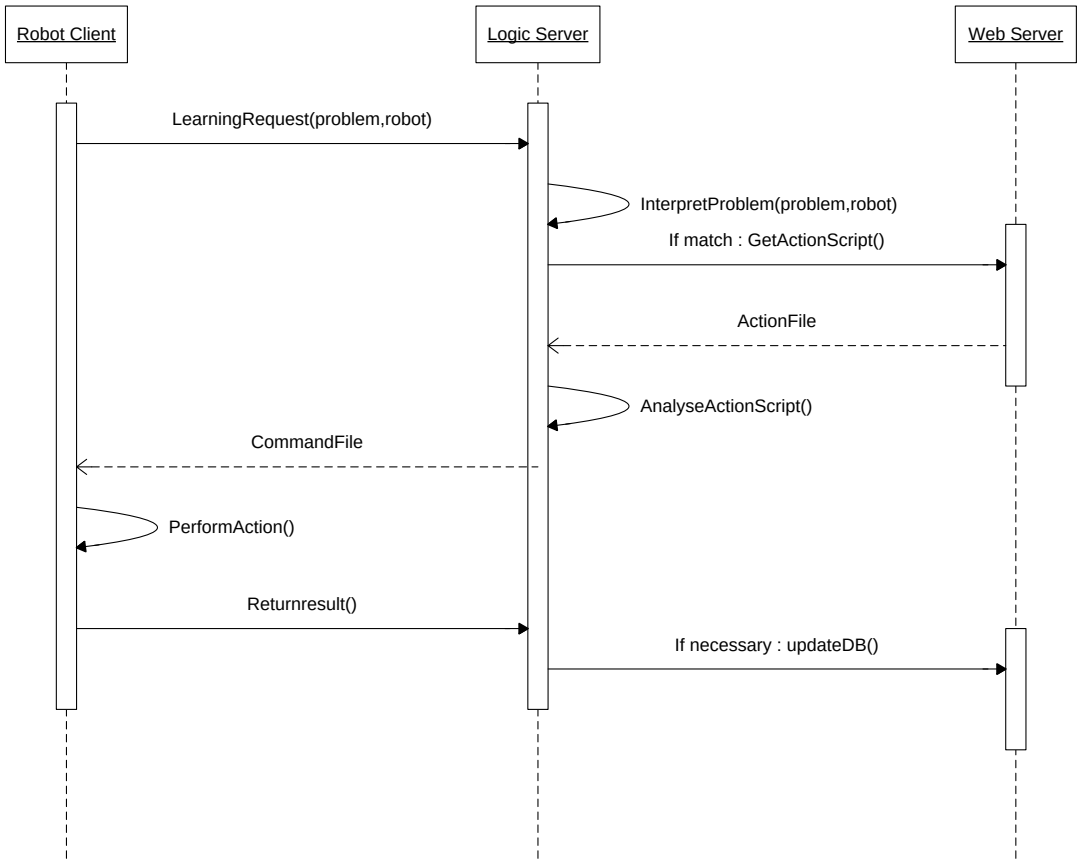

Figure 5.4 Learning Knowledge

Learning Knowledge Process is another scenario which enables Robot Client to get the action script regarding the provided problem. It is very essential for the logic server to teach robots how to work. Normally, it happens after the collecting experience process. Otherwise, the system will get nothing to teach. Figure 5.4 illustrates the general procedure of this function.

1) If Robot Client gets some problems to be taught, it sends a learning request to the logic server with the problem and robot information.

2) Logic Server will then interpret this problem and try to match this problem with any known problem. The matching process can be done either by comparing the problem id explicitly or by comparing the problem data xml file implicitly.

3) If Logic Server finds the matching problem, it will send a request for the action script on the specified problem. Then the web server returns the corresponding action script file.

4) After the logic server received action script file, it will make clear analysis on the file and convert it to a robot-understood file. Thus, a command file will be sent back to Robot Client.

5) Finally, Robot Client then can perform the operations defined by a command file. The client will monitor the result of executing this action script. If the result is negative, the web server will be updated by the logic server, which makes analysis on the result before in order to increase the accuracy.

5.3 Conclusion

This chapter describes the internet protocol and communication model. The network system consists of Robot Client, Logic Server and Web server three main components.

An internal internet protocol is defined by xml files. All communications between different hosts should follow the internet open interface.

There are also three principal user scenarios - authentication process, collecting experience and learning knowledge. Authentication is the prerequisite of the other two scenarios. Collecting experience process is used to obtain information from users. Learning knowledge is the procedure which the system utilizes to teach the individual robot to know how to address the specified problem.

Chapter 6 Experiment and Future work

This system is designed to suit the demand for most robots. However, due to limited physical conditions, the experiment can only be carried out for iRobot Create. It is expectable that more types of robots can be tested under this platform in the future. So the system can have the ability to truly accept this specific type of robots. This chapter describes the experiment procedure and its result in details. Section 6.1 describes the testing cases which are needed to be tested. It generally talks about the common scenarios which occur in the experiment. Section 6.2 shows the real experimental process. The final result and its analysis are given in Section 6.3. On top of that, future work will be discussed in Section 6.4.

6.1 Testing cases for iRobot Create

The real experiment environment is in closed room. The main task for iRobot is to find one efficient path, moving from one specific starting point to another unknown fixed exit point in a closed room. During the whole experiment, iRobot Create is told to collect the experience first and then learn knowledge from the remote server.

6.1.1 Initial Configuration

Before testing the above two phases, it is necessary to configure the system to accept robots and user accounts. To make the initialization, the system's database must be changed to adapt to iRobot Create, and its user. The web interface provides the api to register new robot and new user in this system. Thus, the administration can make the modification easily through the website, instead of manipulating the database directly.

The relevant information about this robot needs to be recorded in the database as well. For example robot_type details should be supplied by the robot user. The table Robot_Action_Type_Relation should also be filled, because this table defines which action is applicable for this type of Robot. By doing that, the web server can check the validity of a new action script when it is generated.

In this case, an iRobot Create account is established for Robot Client, while an iRobot Server account is set up for the logic server. Four basic action types (move, turn, stop and read sensor) are defined in the action_type table, which are linked with iRobot robot type.

On top of that, a set of objects and a map are inserted into the database to indicate the robot's environment. Robot can access this content to get the surrounding information. Also, the problem should be defined by logic server and registered in the database system.

6.1.2 Collecting experience phase

In the collecting experience phase, iRobot moves randomly and tries to encounter some predefined objects which have special meanings. The first task for testing is to distinguish the meaningful captured image with other meaningless image. The solution is to tag all the necessary objects with a label shape. This shape is shown in the figure 6.1.

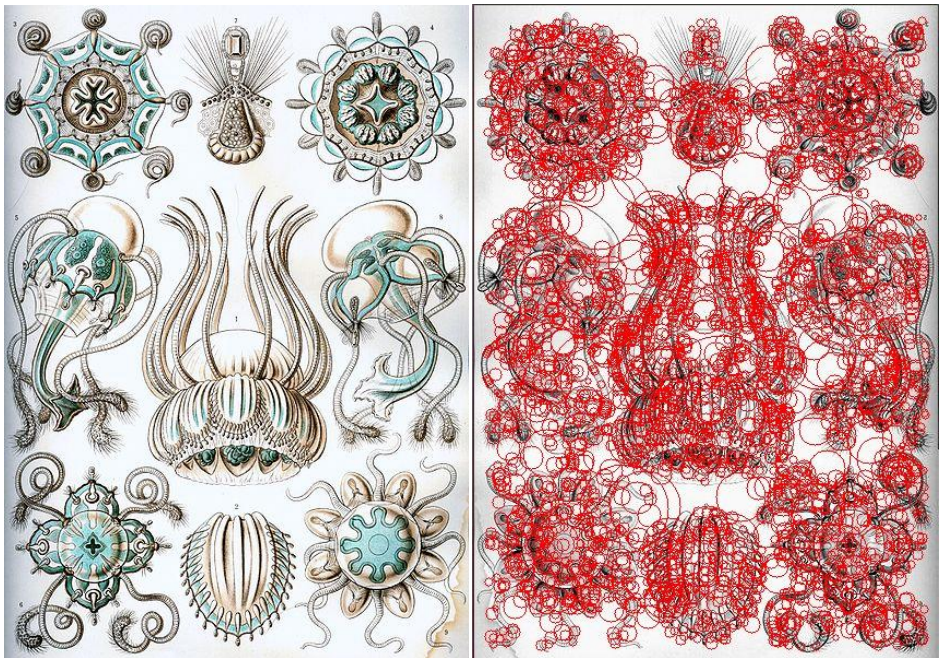

Figure 6.1 tagged image

The system has tested several label candidates to do this job. The reason for choosing this shape is that, this label is technically easy to be recognized. The right shape in this figure is generated by the system testing programme, which describes the interest points for the target image. Every circle here represents a feature area which can be identified by the system. Be aware that the system utilizes this kind of feature area to compare it with all the regions in the captured image. The more circle in the picture, the more accurate the matching process would be. Obviously, this image has a large size of image descriptor, so the system can hardly mismatch this label image with other objects' images in the real world.

After defining the label object, the target image would be selected to be sent to the logic server. The system already has a set of meaningful object. In this scenario, different images are corresponding to different distance from their locations to the exit point.

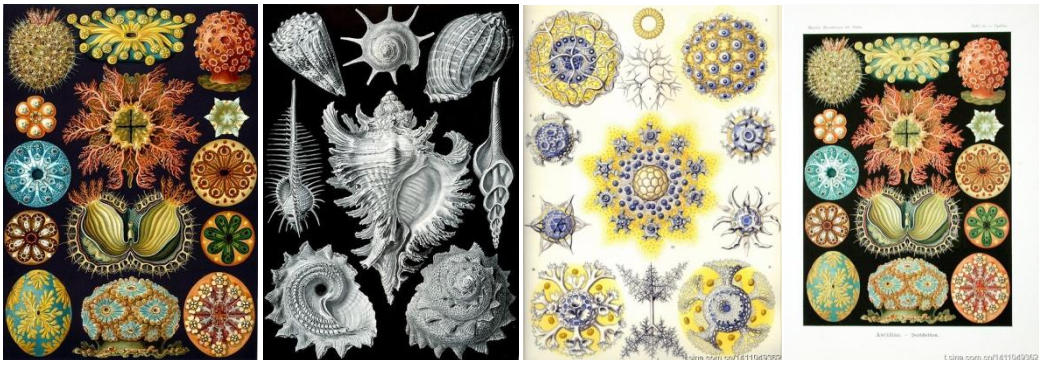

Figure 6.2 Meaningful Objects

In figure 6.2, each image represents the distance between its location and the exit point. The value of the distance for the above four objects decreases from left to right. When the robot encounter one of them, it will send back a signal to the server and the server will record all the encountered objects accordingly. If the moving trend is to

move away from the exit point, it will withdraw the recent actions and get back to the previous object. At that point, the robot will execute a different random action script to keep exploring.

It is noticeable that the reason why utilizing the above objects is similar as the reason for choosing the label image. They all have rich set of interest points, which means they are quite easy to be recognized.

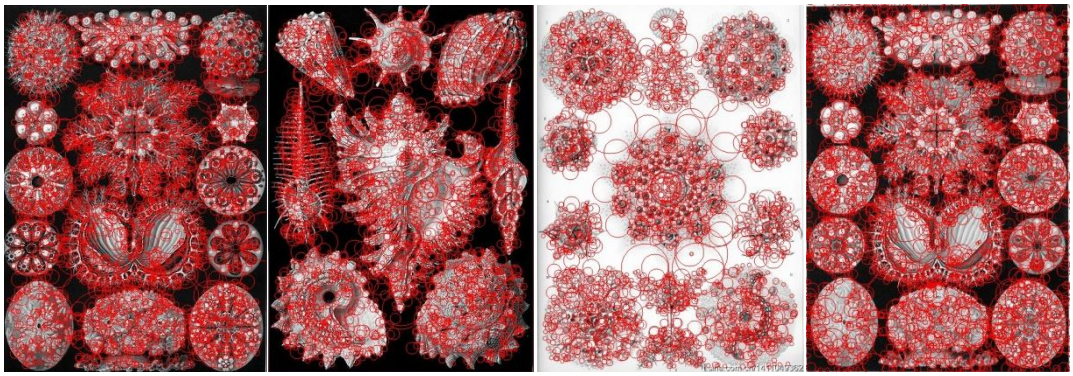

Figure 6.3 analysis versions of meaningful objects

The collecting experience phase needs to be run for several times. So the logic server can make the analysis on all the action scripts. Because the irobot's action script only accepts four basic operations (move, stop, turn and read sensor), the calculation is quite simple. It only needs to take all the moving operations together and calculate the overall distance. The final action script should be the one with the minimum distance.

One essential part of the collecting experience phase is how to define the random movement. The solution is to use multiple predefined action scripts. Robot client randomly choose one of them to make the move until the robot's action is interrupted (like encounter a meaningful object or some obstacle) or the action script is completely executed. In this case, the testing process utilizes four different xml files to control the random movement. They are described as follows:

1) MoveForward
   Move straightly for a certain distance. Stop when completes.
2) Turn
   Turn for a certain angle. Stop when completes.
3) MoveSemiCircle
   Move along the semi circle. Stop when completes.
4) MoveAndTurn
   Move a certain distance and turn gradually. Stop when completes.

The possibility of taking which xml files is not equal to 25%. In the testing process, MoveForward and Turn account for a larger percentage.

There is another thing need to be aware of. When iRobot Create reaches the edge of the room like a wall or any other fixed obstacles, it should have the ability to make a turn and move. iRobot Create has the ability to detect the static obstacles by sensors. Robot Client is designed to have that function to read this kind of sensor information. Thus, when such an interruption is occurred, it will stop the current operation and

tries to run the turn action file followed by MoveForward action file recursively.

6.1.3 Learning knowledge phase

The testing of learning phase is much simpler than that of Collecting experience phase. The system only needs one iRobot Create to be run in the room with the same starting point as the previous scenario. So the action script generated before can be effortlessly reused without any modifications.

There is no need to utilize the object in this scenario, because the action script only contains the necessary operations. Moreover, the action script is retrieved from the web server. The logic server does not have a lot of works to do, but only translate the action script into the command file.

Therefore, Robot Client can read the command file from the internet and execute the operations written inside. The image matching process only activated to detect whether iRobot Create has reached the destination or not after all the actions are completed.

6.2 Experiment Process

The experiment is taken place in a closed room. One iRobot Create attached with a local computer is shown in figure 6.4. This can be treated as the testing entity in the experiment process. The object images are attached with the target objects as shown in figure 6.5.

Figure 6.4 the robot which is attached with local computer

Figure 6.5 objects in the experiment

The first three objects on the upper column indicate the relative distance from their location to the exit point. From left to right, the distance is decreased gradually. The last one object in the lower column is the exit point, which means iRobot Create will automatically stopped when it comes across this object.

For the collecting experience phase, experiment has been recorded for twice. So the system can get two action scripts generated by the logic server. After that, the logic server can make a simple logic reasoning on which action script is better, namely which contains the shorter path. As a result, in the learning knowledge phase, iRobot Create could be told to perform the better action script.

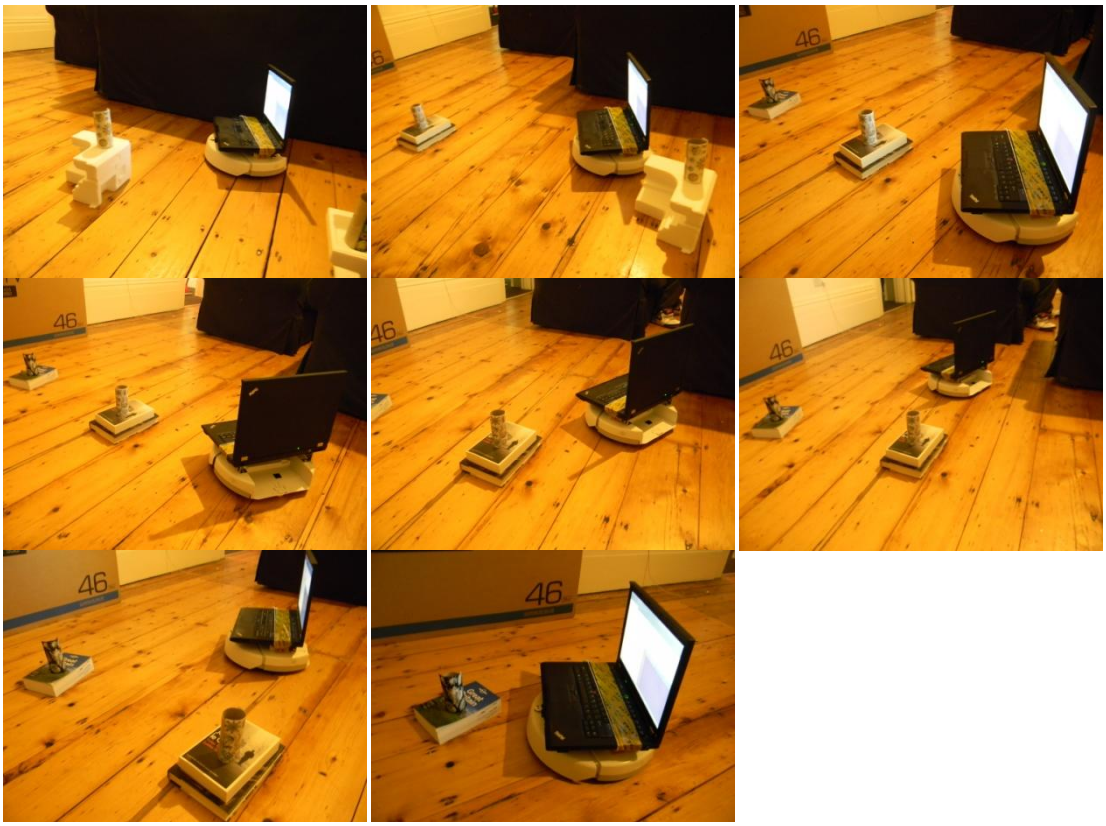

Figure 6.6 first experiment

Figure 6.6 shows the procedure of the first experiment. In the beginning, it moved straightly towards the first object. Remind that different objects represent different level of distance between this object and the final exit point. After it encounters the first object, it picked up a predefined moving script and as a result, it turned right and went straightforward. However, after travelling a short distance, one fixed obstacle prevented it from moving continuously. At that point, it picked a predefined script again and decided to turn left a little bit. After that operation, the robot kept moving in a line until it came across the third object. Because the third object represents the shorter distance to the exit, comparing with the first object the robot encountered before, the system would like to consider the robot is moving in the right direction. Thus, the robot turned a little bit and then moved along a semi-circle. Eventually it arrived at the exit point.

Figure 6.7 second experiment

Figure 6.7 illustrates the second experiment. In the beginning, the movement of the robot was quite similar as that in the first experiment. However, after the robot encountered the third object, the randomized action script just made it move back by a movesemi-circle script. So in that case, the robot came across the second object after

the third object. The system would believe that the robot was on a wrong direction and then issued a withdraw command to reverse the operations executed after it met the third object. Finally, the robot picked up a different choice when it encountered the third object again and got through the destination after that.

For the Learning Knowledge phase, the logic server successfully chose the first experiment as the more efficient one, so it sent the action script generated by the previous one to Robot Client. As a result, iRobot Create moved very similar like the last experiment.

Be aware that it is not guaranteed every experiment can be completed successfully under any conditions. Because of the limitation on the image matching ability and robot behavior capacity, sometimes the robot just cannot recognize the meaningful object or even the exit point. In such situation, the robot will not get the correct information about the current running status, therefore causing irrecoverable error in the run time. In particular, the image processing module in this system is still unstable and vulnerable. If the camera only captured a limited area of object image, it is unlikely to execute the image matching correctly.

The experiment has been taken for several times to get these two desirable results, because of the above reasons. Thus, this system is not perfect and it needs to be improved by using more effective image matching strategy and developing more accurate robot manipulation library.

6.3 Result Analysis and evaluation

This first objective of this system is to develop an internet platform to enable the various components to communicate with others. This is completed by implementing php web server, logic server and Robot Client. Also, the HTTP and TCP based communication model is set up for the communication work. In the practical testing, it is proved that data can be transmitted successfully between the different components.

The second objective of this system is based on iRobot Create and Image processing work. It requires the moving laptop can capture the image and perform the actions according to object recognition technology. However, for the real application, image matching is not always accurate enough. For example, if the capture angle is not good enough, probably the camera can only catch an image with partial object image inside. As a result, some of the objects may not be recognized properly. Besides, the robot may not move strictly under the instruction defined by the action script. So for the second objective, it is not perfect at this point.

6.4 Future work

It is expected that the system will be extended to accept more types of robots. Moreover, the system still has several points which can be improved in the future version.

1) Php web server can be extended to provide more functions than present to either the system administrator or third-party programmer. Currently, the web site and

the web interface are not very strong. The system only supports a limited number of functionalities. The web server has the potential to do some more intelligent work, and some more advanced features like eliminating the unnecessary data and analyzing the data content is not implemented. The improvement can be done in this side.

2) The further release should place more functions from Robot Client to Logic Server to increase the performance. During the experiment process, it is evident that in most cases network delay is no longer a very serious problem for the system. And in the reality, it is impractical to attach each robot with a local laptop to communicate with the logic server, especially when the number of the running robots is quite large. In the contemporary distributed system [13], the server tends to be super computer which is capable to deal with thousands of connections concurrently, which means performance will not be compromised by this strategy. On the contrary, the server's maintenance would be easier and the subsequent performance optimization could be done more efficiently. Thus, it is suggested to move the functionality residing in Robot Client to Logic Server side. For this kind of consideration, it is necessary to make this improvement in the next step development.

3) The local image matching and robot moving modules should be upgraded to meet the higher requirement for some more accurate work. The current image matching process is not accurate enough. The future development can either choose to utilize a new object recognition technology or do more research on SURF, in order to discover a more strong matching strategy.

4) Probably, the communication protocol needs to be extended to accept more types of commands. Thus, the command format may have some subtle changes then.

5) Ideally, the robot control module should be developed by the user itself instead of using existing third-party robot library. Normally, the system needs to manipulate the robot in a more accurate way. However, most of the third-party libraries are more likely to focus on providing a higher level interface for convenience, but sacrifice the accuracy of the operation. In order to control the robot more precisely, the system suggests taking advantage of the raw interface provided by the robot itself, or even sending the electric signal if applicable to fulfill the assignment. The clients can wrap this set of functions into a self defined lightweight class library. For this system, developing such an library of iRobot Create is quite complicated. Probably, this library can be implemented in the next step work.

Therefore, it is obvious that the system is far from fully completion. The other system developer can upgrade the project by improving the points described above.

6.5 Conclusion

The experiment is specialized to meet the specific requirement of iRobot Create. The testing cases are different from user cases in the other chapter. The normal problem case is to find the shortest path between one starting point to another ending point. The experiment takes advantage of several highly characterized objects to indicate the distance information, and make the iRobot Create run in this scenario. Collecting

Experience Process and Learning Knowledge Process are testing under this circumstance. The result is shown in terms of diagrams. But there are still several points that the system is weak in, especially for the testing on image matching and robot controlling parts. Most of the system's function is successful except these two parts mentioned above. And there are a lot of improvement can be made in the future version.

Chapter 7 Related work

This project can be treated as a simplified version of Roboearth Platform. And Roboearth [15] is a distributed and scalable platform for all kinds of robots to share their experience and knowledge online. It stores a wide variety of information, ranging from object recognition, navigation to tasks and hosts intelligent service. And it has the ability to manipulate or improve the data, which means a bunch of artificial intelligent algorithms are utilized to do the logical reasoning job. The whole project is based on Cloud Computing [16], because Cloud Computing has been proved to be a robust and flexible framework for distributed system. Thus, robots can share their knowledge no matter where they are.

The whole project is built on top of the Apache Hadoop framework [17], which is administrated and maintained by users. The reason for choosing such a typical private cloud is the framework is running in the local server, so the user has more flexibility to design their own APIs or interfaces and there is no bottleneck on resources, like the limitations of data storage quota and bandwidth. In order to do the logical reasoning, all the data are designed to contain semantic information. As a result, they are typically encoded with OWL [18]. And there is a centralized reasoning server running in the system, which is used to analyze these encoded data. Furthermore, the Database used in Roboearth is not the traditional relational database. It utilizes a special database component HBase provided by Hadoop to provide a Bigtablelike data store [19]. This type of database is quite suitable for operating on large amount of data set.

From the above description, we can see the main difference between Roboearth project and this project is that Roboearth strictly follows the design model of distributed system and is more likely to take advantage of current well-designed framework, while the project “a www for robots” is aiming at designing the architecture from scratch and prioritize system performance over interface standardization. Obviously, Roboearth is more powerful in large scale learning application, but I believe this project is more suitable for the small scale robot learning process, especially for those robot applications which based on the limited hardware or network condition.

Chapter 8 Conclusion

This dissertation describes the main content of the project “Robot using www”. The whole system is based on the network system. There are three major components in this system, which are Robot Client, Logic Server and Web Server. They can communicate with other parts to perform the action collectively. Besides, object recognition also underlies the whole system in the testing scenario, because the majority of system functions for iRobot Create rely on the matching between two images which contains the same object. iRobot Create itself can also be controlled by Robot Client to do the specific operations as needed.

An Internet Protocol and a set of open interface are designed to grant other software to utilize the server’s service. The server is mainly responsible for assigning the command script and summarizing the experience returned from the different clients. It is also coupled with a Database, which can make the maintenance more scientific and organized.

However, the project can still be extended further which means the actual work is likely to be changed in the future. But all the necessary abilities are mentioned in this report and the document itself can be viewed as a specification of the existing robot system.

Biliography

[1]iRobot company, iRobot Create Open Interface Specification. Http://www.irobot.com/hrd_right_rail/create_rr/create_fam/createFam_rr_manuals

[2] **Distributed Application Architecture**[online] *access: http://java.sun.com/developer/Books/jdbc/ch07.pdf*

[3] Murata, M., Kohn, D. and Lilley, C. (2009). **Internet Drafts: XML Media Types**. *IETF. Retrieved 2010*.

[4] http://www.grinninglizard.com/tinyxml/

[5] Yarger, R.J., Reese,G. and King,T.(1999).**MySQL and mSQL.** *O'Reilly & Associates, Inc. Sebastopol, CA, USA*

[6] http://www.irobot.com/uk/

[7] Thomas Moulard, **iRobot Create library communication library [Online]** access in:

http://www.nongnu.org/libirobot-create/doc/userguide.pdf

[8] Dougherty, D. (2001). **LAMP: The Open Source Web Platform**. *ONLamp*.

[9] http://curl.haxx.se/libcurl/

[10] Bay, H., Tuytelaars,T. and Van Gool,L.(2006) **Surf: Speeded-Up Robust Features.** E*uropean Conference on Computer Vision, 1:404-417*.

[11] Lindeberg,T.(1998) **Feature detection with automatic scale selection.** *IJCV, 30(2):79-116*.
[12] Evans,C.(2009) **Notes on the OpenSURF Library.** Tech. Rep. CSTR-09-001

[13] Lamport,L. (1978) **Time, clocks, and the ordering of events in a distributed system.** *Communications of the ACM CACM Homepage archive Volume 21 Issue 7*

[14] Berners-Lee, T.(2010) **HyperText Transfer Protocol**. *World Wide Web Consortium*.

[15] Schieble, B., Haussermann, K. and Zweigle, O.(2010) **ICT Call 4, RoboEarth Project, 2010-248942.** *The Complete specification of the RoboEarth platform*

[16]Armbrust,M. Fox,A., Griffith,R.,Joseph,A.D., Katz,R., Konwinski,A., Lee,G., Patterson, D., Rabkin,A., Stoica,I. and Zaharia,M.(2010) A view of cloud computing. *Communications of the ACM CACM Homepage archive Volume 53 Issue 4,*

[17] Hadoop,A., **a framework for reliable, scalable and distributed computing**. [Online] Access: http: //hadoop.apache.org.

[18] OWL 2 web ontology language document overview.(2007)[Online] Access: http://www.w3.org/TR/owl2-overview/

[19] Chang,F., Dean,J., Ghemawat,S., Hsieh,W.C., Wallach,D.A., MikeBurrows, Chandra,T., Fikes,A. and Gruber,R.E.(2006) **Bigtable: A distributed storage system for structured data.** *OSDI'06: Seventh Symposium on Operating System Design and Implementation*.

[20] Lowe,D.G.(1999). **Object recognition from local scale-invariant features** *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*

Appendix

From the code view, there are seven modules defining different functionality. Because the overall system is quite large, I only put some code files here. For more details, please see the data folder.

Phpmodule.

Createkey.php:

```
<?php
require_once("autoAuth.php");
$result = mysql_query("SELECT * FROM Account where Username =
'".$_POST['server']."'");
$row = mysql_fetch_array($result);

if($row['Privilege']=='N' || $row['Privilege']=='R'){
    exit("not connect to a server");
}else{
    //$flag = false;

    //while(!$flag){
    $key = getRandomString();
    $result2 = mysql_query("SELECT * FROM Session where Robot_Name =
'".$_COOKIE['Username']."' AND Server_Name = '".$_POST['server']."'");
    if(isset($result2)){
        mysql_query("DELETE FROM Session WHERE Robot_Name =
'".$_COOKIE['Username']."' AND Server_Name = '".$_POST['server']."'");
    }
//  echo "INSERT INTO Session
VALUES('".$_COOKIE['Username']."','".$_POST['server']."','".$key."',TIMESTA
MPADD(MINUTE,5,NOW())";
//  echo "INSERT INTO Session
VALUES('".$_COOKIE['Username']."','".$_POST['server']."','".$key."',NOW()+3
600)";
    $flag= mysql_query("INSERT INTO Session
VALUES('".$_COOKIE['Username']."','".$_POST['server']."','".$key."',TIMESTA
MPADD(MINUTE,5,NOW()))");
    echo $key;
    //}
}
require_once("footer.php");
?>
```

Cmdparser.h:

```
class CmdParser{
    public:
        CmdParser(const char* xmlblock);

        CmdParser(FILE* file, COM_OP op, COM_TYPE type = TYPE_NONE, COM_METHOD
method = METHOD_NONE);

        const char* toString() const;

        bool parseNewFile(const char* pathname);

        bool isValid();

        int GetSize() const;

        inline bool saveFile(char* filename){
            return doc->SaveFile(filename);
        };

        inline bool saveFile(){
            return doc->SaveFile(fp);
        };

        int GetType() const;

        inline const char** GetAuthContent(){
            if(GetType()!=AUTHING){
                return NULL;
            }else{
                return const_cast<const char**>(data);
            }
        }

        inline const char** GetTeachContent(){
            if(GetType()!=TEACHING){
                return NULL;
            }else{
                return const_cast<const char**>(data);
            }
        }

        inline const char** GetLearnContent(){
            if(GetType()!=LEARNING){
```

```
            return NULL;
        }else{
            return const_cast<const char**>(data);
        }
    }

    inline const char* GetAuthContent(int tag){
        if(GetType()!=AUTHING){
            return NULL;
        }else{
            return const_cast<const char*>(*(data+tag));
        }
    }

    inline const char* GetTeachContent(int tag){
        if(GetType()!=TEACHING){
            return NULL;
        }else{
            return const_cast<const char*>(*(data+tag));
        }
    }
    inline const char* GetLearnContent(int tag){
        if(GetType()!=LEARNING){
            return NULL;
        }else{
            return const_cast<const char*>(*(data+tag));
        }
    }

    bool AppendData(int tag, const char* value);
    bool InitDoc(int type, int method);

private:
    COM_TYPE type;
    COM_METHOD method;
    COM_OP op;
    char** data;
    char* xmldata;
    TiXmlDocument* doc;
    FILE* fp;
    int xmlsize;
    CmdParser(void){};
    bool parseToObject();
    const char* getTagName(int type, int tag);
```

```
};

#endif
```

Imgmatch.cpp:

```
int MoveAndCapture_test(int objnum, char** obj, int* objid ){
    CvCapture* capture = cvCreateCameraCapture(0);
    if ( !capture ) {
        fprintf( stderr, "ERROR: capture is NULL \n" );
        getchar();
        return -1;
    }
    //cvNamedWindow( "mywindow", CV_WINDOW_AUTOSIZE );
    /*IplImage* img[10];
    for(int i=0;i<objnum;i++)
        img[i] = cvLoadImage( obj[i], CV_LOAD_IMAGE_GRAYSCALE );
    */
    int i=0;
    bool flag = true;
    int status=10;
    while ( flag ) {
        // Get one frame
        printf("loop start\n");
        IplImage* frame = cvQueryFrame( capture );
        if ( !frame ) {
            fprintf( stderr, "ERROR: frame is null...\n" );
            getchar();
            break;

        }

        cvSaveImage("temp.jpg",frame);
        //printf("test\n");
        for(int j=0;j<objnum;j++){
        printf("loop %d, object %d %s\n", i,j, obj[j]);
        //printf("%d\n",justify(obj[j],"temp.jpg"));
        if(justify(obj[j], "temp.jpg")){
            printf("match ok!\n");
            if((status=processObject(obj[j]))==10)
                flag = false;
        }
```

```
        else
            printf("no match\n");

        }
        //cvShowImage( "mywindow", frame2 );
        //if ( (cvWaitKey(10) & 255) == 27 ) break;
        printf("loop end\n");
    }
    cvReleaseCapture( &capture );
    //cvDestroyWindow( "mywindow" );
    return status;

}
```

Auth.c:

```
CURLcode SetAuth(CURL *curl, char** content, char** header, const char* username,
const char* password)
{
    CURLcode res;
    FILE* fp;

    struct curl_httppost *formpost=NULL;
    struct curl_httppost *lastptr=NULL;
    struct curl_slist *headerlist=NULL;
    static const char buf[] = "Expect:";
    if(content!=0){
    *content = (char*)malloc(1);
    memset(*content,0,1);
    }
    if(header!=0){
    *header = (char*)malloc(1);
    memset(*header,0,1);
    }
    fp = fopen("result.txt","w");

    curl_global_init(CURL_GLOBAL_ALL);

    /* Fill in the file upload field */
    curl_formadd(&formpost,
            &lastptr,
            CURLFORM_COPYNAME, "Username",
            CURLFORM_COPYCONTENTS, username,
            CURLFORM_END);
```

```
    /* Fill in the filename field */
    curl_formadd(&formpost,
            &lastptr,
            CURLFORM_COPYNAME, "Password",
            CURLFORM_COPYCONTENTS, password,
            CURLFORM_END);


    /* Fill in the submit field too, even if this is rarely needed */
    curl_formadd(&formpost,
            &lastptr,
            CURLFORM_COPYNAME, "submit",
            CURLFORM_COPYCONTENTS, "send",
            CURLFORM_END);

    /* initalize custom header list (stating that Expect: 100-continue is not
     *      wanted */
    headerlist = curl_slist_append(headerlist, buf);
    if(curl) {
        /* what URL that receives this POST */
        curl_easy_setopt(curl, CURLOPT_URL,
"http://localhost/Robot/userAuth.php");
        /* only disable 100-continue header if explicitly requested */
        curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headerlist);
        if(content!=0){
        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION,write_data);
        curl_easy_setopt(curl, CURLOPT_WRITEDATA, content);
        }
        if(header!=0){
        curl_easy_setopt(curl, CURLOPT_HEADERFUNCTION, write_header);
        curl_easy_setopt(curl, CURLOPT_WRITEHEADER,header);
        }
        curl_easy_setopt(curl, CURLOPT_HTTPPOST, formpost);
        curl_easy_setopt(curl, CURLOPT_COOKIEFILE, "");
        res = curl_easy_perform(curl);
        //printf("%s%s",*header,*content);
        //printf("hahahahaah");
        curl_formfree(formpost);
/*
    char* test2 = (char*)malloc(1);
    char* test3 =(char*)malloc(1);
    memset(test2,0,1);
    memset(test3,0,1);
```

```
        curl_easy_reset(curl);
        curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headerlist);
        curl_easy_setopt(curl, CURLOPT_URL,
"http://localhost/Robot/autoAuth.php");
        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION,write_data);
        curl_easy_setopt(curl, CURLOPT_WRITEDATA, &test2);
        curl_easy_setopt(curl, CURLOPT_HEADERFUNCTION, write_header);
        curl_easy_setopt(curl, CURLOPT_WRITEHEADER,&test3);
        res = curl_easy_perform(curl);
        printf("%s%s",test3,test2);
        //if(test2!=NULL)
        free(test2);
        free(test3);
        curl_easy_cleanup(curl);
*/
        curl_slist_free_all (headerlist);
        fclose(fp);
//      printf("%s\n",content);
    }
    return 0;
}
```

Server.cpp:

```
void* connHandler(void* arg){
    int sockfd =*(int *)arg;
    char buff[255];
    char** sessionkey=NULL;
    ActParser* account = new ActParser("Account2.xml");

    sessionkey = (char**)malloc(sizeof(char**));
    memset(sessionkey,0,sizeof(char**));

    memset(buff, 0, 255);
    int recfile = open("backup.xml", O_WRONLY | O_TRUNC);
    int n, m;
    //fcntl(sockfd,F_SETFD, O_NONBLOCK);
    while( (n = recv(sockfd, buff, 255, MSG_DONTWAIT )) > 0 ){
        m = write(recfile, buff,strlen(buff));
        if (m < 0) error("ERROR writing to socket");
        if (m == 0) break;
    //  printf("%s",buff);
        memset(buff, 0, 255);
    }
```

```
    close(recfile);
    FILE* fp = fopen("backup.xml","r");
    CmdParser* command = new CmdParser(fp,READ, AUTHING, REQUEST);
    const char* clientname = command->GetAuthContent(USERNAME);
    GetSessionKey(account->getAccount(),account->getPassword(),clientname,s
essionkey);
    const char* key = command->GetAuthContent(SESSIONKEY);
    //printf("%s\n%s\n", key, *sessionkey);
    delete command;
    fclose(fp);

    FILE* fp2 = fopen("res.xml","w+");
    CmdParser* command2 = new CmdParser(fp2, WRITE, AUTHING, RESPONSE);
    char* temp=NULL;
    int size=0;
    if(strcmp(*sessionkey,key)==0){
        command2->AppendData(SERVERNAME, account->getAccount());
        command2->AppendData(SESSIONKEY,key);
        command2->AppendData(RESCODE,"100");
        command2->AppendData(NOTE,"Auth successfully");
        command2->saveFile();
//      temp = command2->toString();
        size = command2->GetSize();
        temp = (char*)malloc(size*sizeof(char*)+1);
        strcpy(temp, command2->toString());
        fclose(fp2);
    }else{
        command2->AppendData(SERVERNAME, account->getAccount());
        command2->AppendData(RESCODE, "101");
        command2->AppendData(NOTE, "Auth unsuccessfully");
        command2->saveFile();
        fclose(fp2);
    }
//  printf("test: %s\n%d\n", temp, size);
    //n = write(sockfd,command2->toString(),command2->GetSize());
    n = write(sockfd,temp,size);
    if(n<=0)
        printf("writing authing response error");
    /* ^^ AUTHING COMPLETE ^^ */
    /* Teaching Process start */
/*  recfile = open("backup.xml", O_WRONLY | O_TRUNC);
    //fcntl(sockfd,F_SETFD, O_NONBLOCK);
    printf("start reading!.\n");
```

```
    char buf[1000];
    while( (n = read(sockfd, buf, 255)) > 0 ){
        m = write(recfile, buf,strlen(buf));
        if (m < 0) error("ERROR writing to socket");
        if (m == 0) break;
        printf("%s",buf);
        memset(buf, 0, 255);
        break;
    }
    //printf("reading complete.\n");

    close(recfile);
    fp = fopen("backup.xml","r");
    command = new CmdParser(fp,READ, TEACHING, REQUEST);
    const char* username = command->GetTeachContent(USERNAME);
    key = command->GetTeachContent(SESSIONKEY);
    const char* robotid = command->GetTeachContent(ROBOTID);
    const char* problemid = command->GetTeachContent(PROBLEMID);
    //printf("%s\n%s\n%s\n%s\n", username, key, robotid, problemid);
    delete command;
    fclose(fp);
    printf("receive complete.\n");*/
    if(flag==0){
    fp = fopen("backup.xml","w+");
    SendLearningRequest(sockfd, fp, account->getAccount(), key,
"1","1","OK","1");
    fclose(fp);
    int actionid = ParseLearningResponse("backup.xml", sockfd);
    GetActionFile(actionid,account->getAccount(),account->getPassword(),"te
st2.xml");
    }
    If(flag == 1){
    ParseTeachingRequest("backup.xml",sockfd);
    fp = fopen("backup.xml","w+");
    SendTeachingResponse(sockfd, fp, account->getAccount(), key,
"26","1","1","100","OK");
    fclose(fp);
    }
    /* ^^ Teaching Process complete ^^ */
    close(sockfd);
    return NULL;
}
```

Create.sql:

```
CREATE TABLE `Robot_type` (
  `Robot_Type_ID` int(11) NOT NULL DEFAULT '0',
  `ROBOT_NAME` varchar(20) NOT NULL,
  `Description` varchar(200) DEFAULT NULL,
  `Robot_Type_Data_Location` varchar(30) DEFAULT NULL,
  PRIMARY KEY (`Robot_Type_ID`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 ;

create table Map( Map_ID Integer Auto_increment not null, Map_data_file
Varchar(30), Description VARCHAR(50), PRIMARY KEY(Map_ID) ) ENGINE=MyISAM

create table Location (Location_ID INTEGER AUTO_INCREMENT, POSITION_X INTEGER
NOT NULL DEFAULT 0, POSITION_Y INTEGER NOT NULL DEFAULT 0, POSITION_Z INTEGER
NOT NULL DEFAULT 0, Map_ID INTEGER, PRIMARY KEY(Location_ID), FOREIGN KEY
( Map_ID ) references Map ( Map_ID ) ON DELETE SET NULL ON UPDATE CASCADE );

create table Object(Object_ID INTEGER AUTO_INCREMENT, Object_type VARCHAR(20)
NOT NULL, Object_Status VARCHAR(5) DEFAULT 'NONE' CHECK ( Object_type in
('N','M','S','MS')), Location_ID INTEGER, PRIMARY KEY (Object_ID), CONSTRAINT
pk FOREIGN KEY(Location_ID) references Location(Location_ID) ON DELETE SET NULL
ON UPDATE CASCADE );

Create table Robot ( Robot_ID INTEGER AUTO_INCREMENT, Robot_Type_ID INTEGER,
Location_ID INTEGER, Robot_Password VARCHAR(20) not null, Primary key
(Robot_ID), Constraint pkT FOREIGN KEY ( Robot_Type_ID ) REFERENCES Robot_type
( Robot_Type_ID ) ON DELETE SET NULL ON UPDATE CASCADE, Constraint pkL FOREIGN
KEY ( Location_ID ) REFERENCES Location ( Location_ID ) ON DELETE SET NULL ON
UPDATE CASCADE);
Create table Action_Type ( Action_Type_ID INTEGER AUTO_INCREMENT,
Action_Type_Name VARCHAR(20) NOT NULL, Action_Type_Data_Location VARCHAR(30),
Description VARCHAR(50), Primary key (Action_Type_ID));
Create table Action ( Action_ID INTEGER AUTO_INCREMENT, Action_Type_ID INTEGER,
Action_Data_Location VARCHAR(30), Description VARCHAR(50), Primary key
(Action_ID), CONSTRAINT FK FOREIGN KEY(Action_Type_ID) REFERENCES
Action_Type(Action_Type_ID) ON DELETE SET NULL ON UPDATE CASCADE);
create Table Robot_Action_Type_Relation( Robot_Type_ID INTEGER, Action_Type_ID
INTEGER, PRIMARY KEY(Robot_Type_ID, Action_Type_ID) ,CONSTRAINT FK FOREIGN KEY
(Robot_Type_ID) REFERENCES Robot_type(Robot_Type_ID) ON DELETE SET NULL ON
UPDATE CASCADE, CONSTRAINT FK2 FOREIGN KEY (Action_Type_ID) REFERENCES
Action_type(Action_Type_ID) ON DELETE SET NULL ON UPDATE CASCADE  )
create Table Action_Object_Relation( Object_ID INTEGER, Action_ID INTEGER,
PRIMARY KEY(Object_ID, Action_ID) , CONSTRAINT FK FOREIGN KEY (Object_ID)
REFERENCES Object(Object_ID) ON DELETE SET NULL ON UPDATE CASCADE, CONSTRAINT
FK2 FOREIGN KEY (Action_ID) REFERENCES Action(Action_ID) ON DELETE SET NULL ON
```

```
UPDATE CASCADE  )
create table Account( Username Varchar(20), Password VARCHAR(20) NOT NULL,
Privilege Varchar(2) NOT NULL check( Privilege in ('R','S','N','B','SP')),
PRIMARY KEY (Username));
alter table Account add column Register_Time TIMESTAMP default
CURRENT_TIMESTAMP;
alter table Account add column Last_Login_Time TIMESTAMP;
insert into Account(Username, Password, Privilege, Last_Login_Time)
values('zhang123cnn','21122112', 'S', Now());
create table Session( Robot_Name VARCHAR(20), Server_Name VARCHAR(20),
Session_Key VARCHAR(20) NOT NULL UNIQUE, Validity_Time TIMESTAMP DEFAULT 0,
PRIMARY KEY(Robot_Name,Server_Name), CONSTRAINT FK1 FOREIGN KEY (Robot_name)
REFERENCES Account(Username) ON DELETE SET NULL ON UPDATE CASCADE,  CONSTRAINT
FK2 FOREIGN KEY (Server_name) REFERENCES Account(Username) ON DELETE SET NULL
ON UPDATE CASCADE);
alter table Object add column Object_Data_Location VARCHAR(30);
```