Technical Report

April 24, 2014

Anders Strand
Emil Taylor Bye
Petter S. Storvik
Odd M. Trondrud
Ole Bauck

Eksperter i Team
Norges Teknisk-Naturvitenskapelige Universitet

1 Abstract

The abstract is a tl;dr-type thing that *very* briefly summarizes the introduction, method, result and conclusion.

1 Contents

1 Abstract **1**

2 Introduction **3**

3 Concept Description **4**
3.1 Terminology 4

4 SWOT **6**

5 Method **7**
5.1 Robot 7
5.1.1 The overall system 10
5.1.2 Wiggle It 10
5.1.3 Driving 11
5.1.4 Manipulator 11
5.1.5 Testing it all with a local interface 11
5.1.6 Sensor 11
5.1.7 Exception handling 11
5.1.8 Future problems/challenges 11
5.2 Agent-server communication 12
5.2.1 Sending and receiving commands 12
5.2.2 Sending and receiving sensor data 12

6 Results **14**

7 Conclusion **15**

2 Introduction

Our group was assigned to the EiT Instrumentering og Styring over nett village, which has a focus on remote control over the internet. We decided to build a platform to facilitate the communication between operators and the devices they wish to remote control over the internet. Devices connected to the platform should have their functionality, available commands and sensory readings made available to authorized users through the platform. The platform should be modular in order to more easily allow the adding of functionality to meet domain specific requirements. It should also meet typical security requirements: Eavesdropping on the communication should not be possible and some secure authorization scheme should be supported in order to restrict access to devices connected to the platform.

Various rescue services often encounter situations where assertaining the risk of entering a location is difficult. A burning house is an example of this, especially if the only information available is what the eye can see and one does not know if there are people trapped inside or not. Another example of such a situation is cave exploration, both above and under water. Rescuing divers is both difficult and time-consuming. A remote controlled device (e.g. such as a wheeled robot) could be sent instead of a live human, reducing the risk of loss of life. While various systems exist that are designed to serve the purpose of rescue operators' "eyes and ears", these are typically domain or application specific. If the same communication platform could be used for any given scenario, one forces the developers to modularize their systems, potentially resulting in an increase of re-usability.

Communication will be done over HTTP, with the possible messages specified by an API. The platform will function as a web-service with a RESTful API. This means users will be able to access the system through a regular web browser when connected to a WLAN or 3G network. In theory the platform will allow any internet capable device connected to the internet to be remote controlled by anyone with an internet connection anywhere in the world, given that the necessary software to communicate with the platform has been written.

We also showcase the feasibility of such a system by using it to remote control a robot with four wheels, a three-jointed grabber, sensors and a camera. The development of this robot and the software required to control it through the system is also detailed.

3 Concept Description

The project's goal is the construction of a platform that facilitates the communication and authorization aspects of remote controlling a device over the internet. Agents connect to the platform through which their sensory information, available commands and functions are made available to authorized users. Use of the platform increases the possible simultaneous audience of a connected agent, as users do not have to query the agent directly for the information. Agents actively query the platform to obtain commands from users. In the same manner, users must actively query the platform to obtain new information from or about the agents. See Figure 3.1 for an illustration of how the communication between the platform and connected entities.

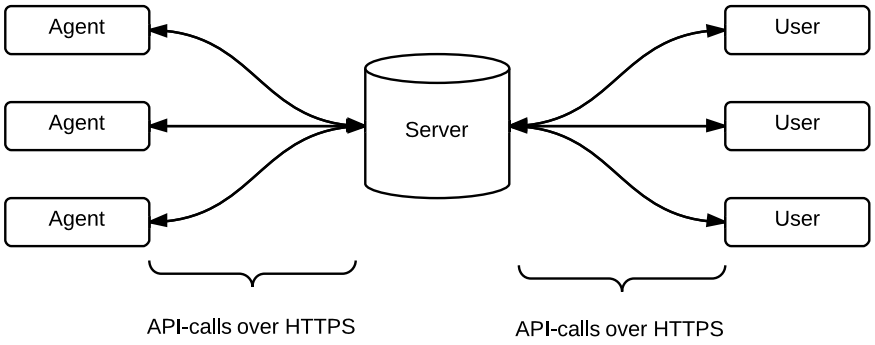

Figure 3.1: An overview of the concept showing three users and three agents communicating through the platform.

3.1 Terminology

Below follows an explanation of the various words that refer to different parts of the system.

Server the physical computer or cluster of computers on which the service is hosted.

Service the program that facilitates the communication between clients and manages user authentication.

Client any software that sends requests to the service and receives the responses.

Agent any entity that is connected to the service that can receive commands through the service. The term "agent" refers to the entire entity, which is considered to begin where the requests to the service are generated and end wherever the commands are acted out. A robot, its actuators and sensors, the software that communicates with the service, and the computer that is connected to the robot which the software runs on, and any other parts or components are considered part of the agent.

User any non-agent entity that interacts with the service. This includes actual people and automated services – anything that interacts with the service but cannot receive commands through it.

User client any client that allows users access to the service.

4 SWOT

SWOT analysis is one of the most commonly used marketing strategies. This analysis identifies both strengths and opportunities, and more important weaknesses and threats.[1] Its also important to use the information gathered in the SWOT analys to form project objectives and long term goals. The following SWOT analysis will be brief and will not go into detail.

/textwidth/textwidth

Figure 4.1: SWOT

Strengths

- Everything is built on already existing technologies
- Module based, which means easy to expand
- Relatively cheap harware and free software

Weaknesses

- Latency caused by http limitations
- Limited to the dynamixel servos (possible to support more manufacturers)
- No team to take the project further

Opportunities

- No universal platform (that we could find) exists
- Many different solutions have been tried, but no module based and easy to develope

Threats

- Existing technologies doesnt allow easy expansion, but there are many good soluitions for different specific tasks
- Marketing problems

[1] http://www.businessnewsdaily.com/4245-swot-analysis.html

5 Method

5.1 Robot

To demonstrate the platform it was decided to make a vehicle with a manipulator controlled remotely over internet. In addition the robot should have sensors that send data (if available) continuously to the server and be able to send camera feed to the operator. Due to the possibility to expand this later with more complex functionality, the program was written in C++. The following was used to make this vehicle:

- Raspberry Pi
- Raspberry Pi Camera
- Dynamixel AX-12 Servomotors
- Dynamixel AX-S1 Integrated Sensor
- USB2Dynamixel
- Dynamixel SDK

Rasppberry Pi (Pi) is a single board credit-card-sized computer. It runs on an 700 MHz ARM processor. It has two USB inputs, ethernet, HDMI and gpio (general purpose input output) headers. This makes it the perfect prototyping computer for this kind of project. The Pi is running its operating system from a SD card, and the OS is RaspBian which is a debian based linux distro.

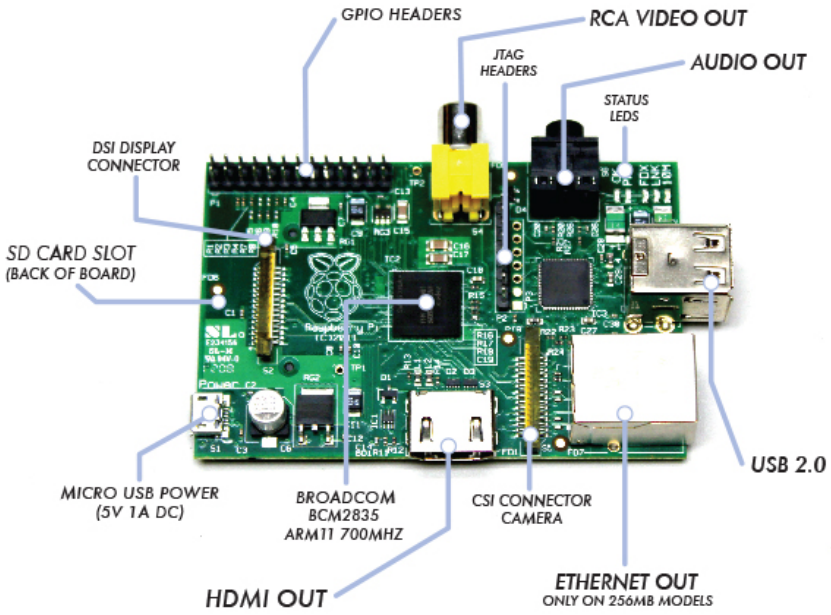

Figure 5.1: Raspberry Pi overview over peripherals

To control the Pi you can hook up a keyboard and a monitor directly to the board, or you can control it from another computer. This is achieved by using a SSH server on the Pi and a SSH client on the computer. SSH (Secure SHell) is a protocol that allows one computer to remotely controll another via command line. A widely known program for doing this is PuTTY. To login over SSH, you need to know the IP-address of the host. This can be done by setting the IP-address static or to scan the network to find out which IP the Pi would have. Since both of these approaches are difficult on a big network like NTNU, we had to find another approach. The solution was to connect the Pi directly to the computer via an ethernet cable. To do this we had to do some modifications, the full tutorial on how to do this see [1]. To get internet access we first shared the Wi-Fi connection on the computer described here [2]. Later we used a Wi-Fi usb dongle. The driver for this dongle installed automatically and all we had to do was to configure */etc/network/interfaces*, and add the following:

```
auto wlan0
iface wlan0 inet dhcp
wpa-ssid "<nameOfNetwork>"
```

[1] http://pihw.wordpress.com/guides/direct-network-connection/

[2] http://anwaarullah.wordpress.com/2013/08/12/sharing-wifi-internet-connection-with-raspberry-pi-through-lanethe

```
wpa-psk "<networkPassword>"
```

We had to set up a own network to do this, because the eduroam network uses different setup.

Raspberry Pi camera

Dynamixel AX-12 servomotors are motors that allows for precise controll of angle and velocity. These motors are controlled over a half duplex UART, which is a byte oriented asynchronous serial communication protocol. You can control the motors and receive feedback by sending commands to it corresponding to the control table in the datasheet [PUT REFZ H3R3]. The most important features is the control of position and velocity. The servos can work like normal servos where you can put in the desired position, and a controller inside the servo will make the servo go to that position. This position is limited to between 0-300 degrees (see datasheet). The servo can also work in so called ”Endless Turn” mode, where the servos can spin infinite. Here you can only control the velocity which the motors run. ”Endless Turn” mode is activated by setting the angle limits (CW Angle Limit and CCW Angle Limit) to zero.

Dynamixel AX-S1 Integrated Sensor is a sensor device capable of measuring sound, brightness, heat and distance to objects. It is also capable of making sound. The communication is the same as the servomotors and the sensor is connected to the same bus.

USB2Dynamixel [3] is a USB device that allows the computer to create a virtual serial port (UART) and with some other circuitry, communicate with the servos over the USB port. This USB requires no driver installation when running linux, and since RaspBian is a linux distro this simplifies things. The USB2Dynamixel is inserted into one of the USB ports on the Pi and the servor motors are connected to the device.

Since the Pi also has a hardware UART driver on two of its gpio headers, we thought about if we could communicate with the servos through these pins. To do this we had to make the circuitry described on page 8 in the datasheet [PUT REFZ H3R3]. We would also have to implement our own code for the lower part of the communication (where we used a library from the manufacturer, more on that later). UART is also widely supported by many lower end microcontrollers which don’t have the support for an OS. Therefore using UART would mean that the code would be even more platform independent.

Dynamixel SDK is a programming library for controlling dynamixel servo motors. This library is available for Windows, Mac and Linux and easy to run on the Pi. Since the library is written in C it is easy to use it in this C++ project. There is a great API reference [4] with the library. The functions treat the lower part of the communication over USB2Dynamixel. There are functions for initializing the communication, terminate the communication, sending byte and words (16 bit),

[3] http://support.robotis.com/en/product/auxdevice/interface/usb2dxl_manual.htm

[4] http://support.robotis.com/en/software/dynamixelsdk.htm

receiving byte and words, and for ping. Ping is for checking if a device is connected. There is also a page describing platform porting [5]

5.1.1 The overall system

nice drawing

communication (because of threads)

easy to develop further

5.1.2 Wiggle It

The first step is to make a servo motor move.

Figure 5.2: Servo motor test setup

To make the servo motor move the example-program for the motor class could be used [6]. Alternative the example-file readWrite form the dynamixel library could be used [7]. For this to work there are some important parameters which should be known:

- ID - Each motor has its own ID. This way its possible to control which motor that should get the command. The ID need to be change in the code to the ID of the motor. If the motor ID is unknown, you could use the *pingAll()* function. This will search for IDs on the bus, and print out those that are active
- Port - Which port the USB2Dynamixel is connected to. This is used to set up the communication to the motors. USB devices under linux can be found under */dev/usb*. When testing we used deviceIndex = 0 (*/dev/usb0*).
- Baudnumber - used to set the speed of the bus. This should always be one, which is 1Mbit/s, unless lower speeds are needed.

By replacing these parameters in the example file the servos should move back and forth when enter is pressed.

[5] `http://support.robotis.com/en/software/dynamixel_sdk/sourcestructure.htm`

[6] LOCATION TO MOTOR EXAMPLE PROGRAM

[7] LOCATION TO READWRITE EXAMPLE FILE

5.1.3 Driving

The next step is to make four wheels cooperate!

- endless turn
- setting speed
- turning

5.1.4 Manipulator

inverse kinematic (drawing)

- gripper

5.1.5 Testing it all with a local interface

mouse and keyboard input

- x11/xming

5.1.6 Sensor

make sound

- read ir

5.1.7 Exception handling

failsafe mode

- ping
- threads
- tried some wheels

5.1.8 Future problems/challenges

What happens if some of the wheels is disconnected? The answer for this question will not be covered in this report because of the timeframe, but it is an important issue that has to be resolved. The optimal solution is to find an algorithm that calculates the speed that the remaining wheels, so that the vehicle can obtain normal function even in failsafe mode. This will most likely depend on the surface under the vehicle.

5.2 Agent-server communication

The vehicle needs a solid way of communicating with the server and other devices. We chose to use a server which hosts a rest API to achieve this. A rest API uses HTTPS requests to communicate text strings in JSON format. All actions are initiated by the clients, and the server is ”resting” otherwise. A JSON object is a normal string, formatted in a certain way known by both end points. This API is described in greater detail in the server section of this report. The vehicle part of the communication was written in C++ to ensure compability with the other modules. The following libraries were used

- Jansson - A library for processing JSON objects in C
- Curl - A library for making HTTP calls in C

5.2.1 Sending and receiving commands

Any client using this C++ framework for server communication can both receive commands from the server, and send commands to other client. Commands are sent using the *json_send_command* function. It takes a command string as input, as well as the receiving client id for the command. The *json_get_commands* function provides a vector of command strings. It contains all the commands that were sent to your client id since the last time this function was called. It is up to the user to iterate trough this vector and extract the command strings, as well as perform action based on the commands received. We use commands like ”forward”,”backward”,”turn_left” and ”stop”. It is up to the user what the various commands are, but both endpoints of communication must agree upon which commands are used. The server will be ”stupid” in this regard, as it only stores and send information when requested. All agents and users in the system can send commands to each other, unless acess restrictions are implemented.

5.2.2 Sending and receiving sensor data

Sensor data is sent and received in the very similar fashion as commands. The vehicle code must construct a table of sensors and their associated sensor values. In practice this table is a c++ map. The vehicle passes this map into the *json_send_data* function, which then constructs a json object containing all the data arranged in a predermined structure. The json object is sent to the server using its REST API. When a client wishes to get the latest sensor data from the vehicle, it simply calls the *json_get_data* function. It must specify which vehicle it want to receive data from. The function return a map of sensors to the client, identical to the one the vehicle uploaded.

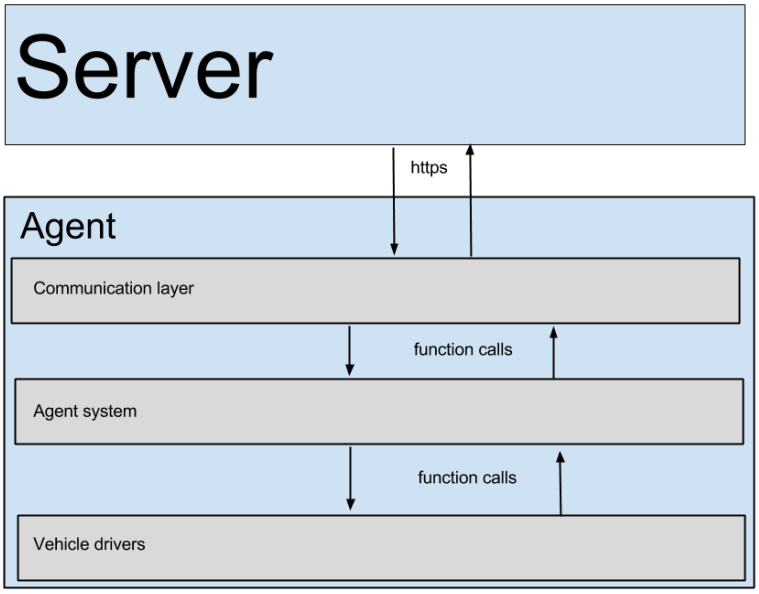

Figure 5.3: Structure of server-agent communication

6 Results

We should have like, a graph or drawing of the finished system here shouldn't with the involved components and actors and what gets transferred between them. Like those drawings Anders have been keeping safe in his lockerbox or whatever.

7 Conclusion

There should be some, like, text here. Probably. Summarizing what we got done and what kind of future work exists. Like "recommended avenues for future research." Oooh we should mention that the system can be used as the basis for a web service (like github) or on a local network to allow the remote control of devices on it (but not from the outside, like idk maybe a company would want to do this). The great thing is that once you've written the agent-client and user-client you can re-use these on other platforms.