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

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. 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.

The system will communicate over HTTP, with the possible messages specified by an 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.

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

3 Concept Description

Concept description goes here

3.1 Terminology

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 a live person that is interacting with or wants to interact with the system.

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

SWOT analysis of the project goes here. I guess one of our weaknesses is that everything is run on top of https but then again that might be one of our strengths depending on the area for which one wants to apply the system. Also what about monetization? I mean we've got like, no prospects for monetization. Is that bad? Should we write about that?

From Wikipedia:

> Setting the objective should be done after the SWOT analysis has been performed. This would allow achievable goals or objectives to be set for the organization.

Well we did not do that.

Strengths

- everything is built on already existing technologies
- mutability/modularity/expandability or whatever
- robots are pretty cool and I've heard the internet is hip these days

Weaknesses

- http is kinda slow I guess? for video at least. Something about how this limits the possible real-time usage of the system
-

Opportunities

- nothing like this exists and being first is always cool
-

Threats

- I think I speak for the entirety of the group when I say that we'd rather be on a beach somewhere sippin' piña coladas or wherever, like in an igloo or something, than doing *work*.

5.1 Vehcle

To demonstrate the platform it was decided to make a vehicle controlled remotely over internet. This vehicle had to be able to drive, turn, use a rabotic arm and send camera feed to the controller. 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
- Dynamixel servomotors
- USB2Dynamixel
- Dynamixel SDK

Rasppberry Pi (Pi) is a small single board computer. It has two USB inputs, ethernet, HDMI and gpio pins. This makes it the perfect prototyping computer for this kind of projects. The Pi is running its operating system from a SD card, and the OS is RaspBian which is a debian based linux distro. To controll the Pi a regular laptis is used. 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.

Dynamixel servomotors[1] are motors that allows for precise controll of angle and velocity. These motors are controlled by three wires, power, gnd and signal. WRITE SOME MORE HERE!

USB2Dynamixel is a USB device that allows easy controll of dynamixel servo motors. 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.

Dynamixel SDK[2] 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.

[1] `http://www.robotis.com/xe/dynamixel_en`

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

5.1.1 Wiggle It

The first step is to make a servo motor move. Since the USB device is linux compatible and the library also works in linux the Pi was used from the begining of the project.

Figure 5.1: fig:1servo

To make the servo motor move an example c-file form the library was used[3]. For this to work there are som 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.
- Port - Which port the USB2Dynamixel is connected to. This is used to set up the communication to the motors.
- Mode - The servo motors can run in two different modes. Endless turn (like a wheel) and start2stop (from start position to specified position).

The servo motors used for this test had IDs 7 and 12, and can be found on the side of the servo. In linux the USB device can be found under /dev/usb, and using the Pi this device was /dev/usb0. By replacing this parameters in the example file two servos was wiggled.

5.1.2 Driving

The next step is to make four wheels cooperate!

5.1.3 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.

[3] LOCATION TO EXAMPLE FILE

5.2 Vehicle-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 HTTP 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.

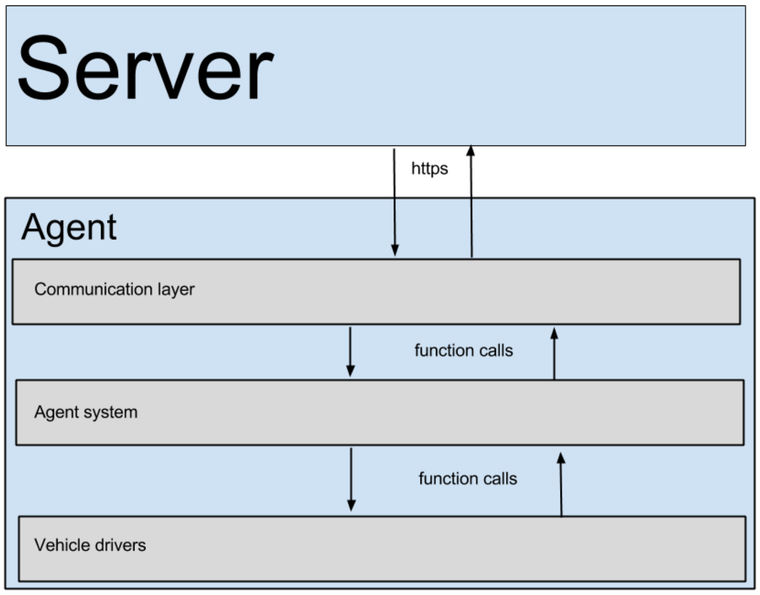

Figure 5.2: Structure of server-agent communication

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.

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.