Remote Monitoring and Controlling of Robotic Systems with MissionControl
Daniel Swoboda*, Raphael Weinfurter, Markus Pinter, Florian Ungersböck, Christoph Käferle, Daniel Honies
Federal Technical Secondary College in Wiener Neustadt. Austria (*i12032@student.htlwrn.ac.at)

Remote Monitoring and Controlling of Robotic Systems with MissionControl

***Abstract*— This publication introduces MissionControl, an application suite created to simplify the workflow of getting remote data readout of robotic systems during their development on multiple, independent clients. This environment has been developed as a project in the 12th grade for HTL Wiener Neustadt. MissionControl and the underlaying MIDaC-Protocol (Modular Information Display and Control Protocol) were developed with competitive, educational and hobbyist robotics in mind. It's not intended for industrial applications other than prototyping. This publication focuses on the capabilities of MissionControl and MIDaC while an overview of the technical aspects is given. It also introduces use-cases and situations in which MissionControl can help developers.**

Keywords*— *remote monitoring; remote controlling; educational robotics; networking

I. Introduction

The remote monitoring of a robotics system is the key for fast error detection and correction during the development phase of a robot. While some systems like the KIPR Wallaby already offer a kind of remote data readout by default this hardly fulfills the necessities of multiple device and multiple developer workflows. Since there might be a variety of robotic systems in use as well as many different client devices such as phones and tablets there is the need for a flexible, standardized system that can run on a variety of robotic controllers and can be accessed with a variety of different client devices.

Another need of developers during the development phase is the ability to remotely control a bot in order to test configurations or problem solving approaches before implementing them.

One solution to these problems is MissionControl, a free and open source application suite designed to transmit data from a robotics controller in order to show it to a user on a client device. It makes use of the MIDaC Protocol, which was created for MissionControl, and is meant to be multi-platform and easy to port.

II. Implementation

MissionControl was designed to allow users to customize it as much as possible while keeping it simple to use. All Clients as well as the server are open-source and can be adapted to the specific users needs. The components were also designed to be easily interchangeable which was achieved by standardizing the communication protocol and locating most of the application logic to the server.

A. MIDaC Protocol

The MIDaC Protocol is an application layer (DoD model & ISO/OSI model) protocol [1, 2]. It was designed to allow easy data exchange between the server and the clients. The

protocol is also used for inter-process communication via Unix Domain Sockets which is necessary in the current implementation of the server.

Clients communicate with the MissionControl server over an existing network communication. MIDaC protocol was designed with communication over a standard (Wireless-) Local Area Network via Ethernet or Wi-Fi in mind but since it is an application layer protocol it can be used with most other intercommunication technologies that provide a method for establishing a continuous connection. However the current implementation only works with TCP sockets and technologies that allow TCP based communication like Wi-Fi and Ethernet.

The protocol makes use of the JSON data format for every part of the communication. JSON was selected for its focus on serialization, vast support across multiple languages and platforms, its readability, its better performance in network operations and it's lower overhead compared to XML [3].
Although existing protocols could have been used, the design of a custom protocol made it possible to tailor it to the needs of educational and hobbyist robotics like Botball. This approach also removed possible overhead of already existing protocols. [4]

A connection is established via a three-way-handshake in which information is exchanged for the server to know which data it can send and for the client to know which UI elements must be generated and what data to expect. A schematic representation of the handshake can be seen in "Fig. 1".

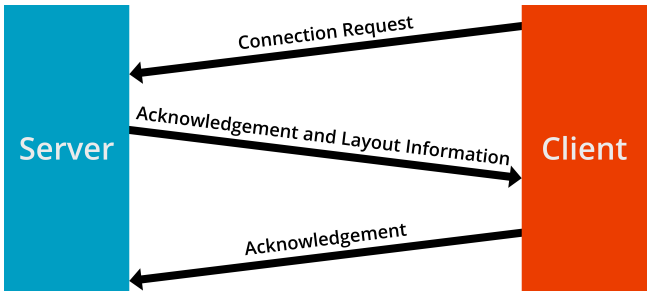

Fig 1. A schematic representation of the MIDaC protocol handshake.

B. Server

The server was implemented in the Python 2.7 and C languages to allow for cross platform usage. Python was used since it is supported on almost all Linux devices and higher-performance robotics controllers, like the ones used in Botball, and makes it more convenient to realize socket communication and multiple-client management. A platform specific program that creates an abstraction of the controller to unify the way the server accesses the actuators and sensors, which is called the MissionControl Robotics System Abstraction Layer (RSAL), was created. It was implemented using C since most controller APIs like the Link and Wallaby Controller API, ROS and Robotics Library are native to the C/C++ languages [5-8] and allows to use the server, which was implemented in Python, on any platform without making changes to it.

The interprocess communication between both parts is done via Unix Domain Sockets which are a standard method for interprocess-communication in Unix-like environments [9]. The server as well as the actual prompted data and the controlling routines are configurable to

allow for a maximum of customization. This is done via the MissionControl Controller Markup Language (MC2ML, see section D) and the config file.

When one or more clients are connected the server periodically checks for new data from the RSAL, which is collected by calling the functions specified in the used MC2ML file. These values are put into a JSON format, which structure is defined within the MIDaC protocol. After receiving the data from the RSAL it is sent to all connected clients. The server also periodically checks for new control commands from the clients. When control inputs have been sent to the server, the messages are forwarded to the RSAL which performs the corresponding actions specified in the MC2ML file. A schematic presentation of the data flow is depicted in "Fig. 2".

The MIDaC protocol specifically is used in the server for handshaking, providing a standard for data formatting and for communication between the server and the client as well as both parts of the server.

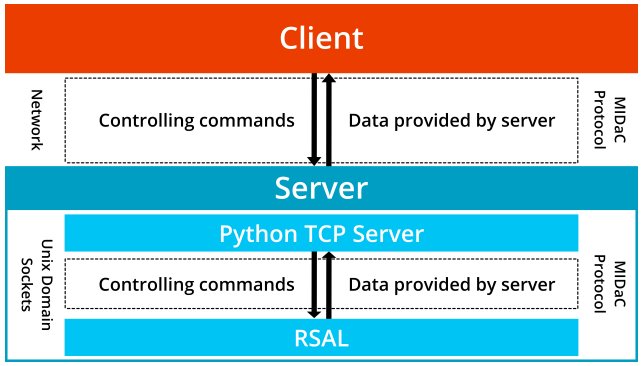

Fig 2. A schematic representation of the data flow in a MissionControl setup, the used mediums of transportation (left of dashed lines) and the application layer protocols (right of dashed lines).

C. Clients

Native clients for Android and iOS were developed to gain maximum performance and battery-efficiency. In addition to the mobile apps a WebSocket based Web-Client was developed which gives access to the controller data from desktop PCs or not natively supported Smartphones or Tablets. By supporting the most common devices it is easy to integrate MissionControl in existing environments without the need of adaption to the workflow. This approach also allowed to show-case the broad support of the technologies on multiple platforms that are the basis of MissionControl.

The clients were implemented using technologies, design (as it can be seen in "Fig. 3") and programming languages which are native to each platform. They make up the presentation layer of the MissionControl suite and only handle the data display and the transmission of control signals to the server, the control logic is implemented only on the server side in the RSAL.

All data between clients and servers is transmitted via sockets. Both, the iOS and Android client, use socket libraries in which the receive function is blocking until data is incoming. Because of that socket operations run in a dedicated thread. The JavaScript WebSocket API however makes use of events, so a function has to be specified which is called whenever data is incoming. Actuator controlling commands are sent when the UI elements assigned to them are used.

The UI update rates are independent of the server update rate and were selected based on the performance of the client devices. Hence, all values are received, but not all are shown if the server update rate is higher than the UI update rate. In the case of a server update rate less than the UI update rate, all values are shown for a longer period of time. During this reaction time there will be a loss of data, but with no consequence for the monitoring process.

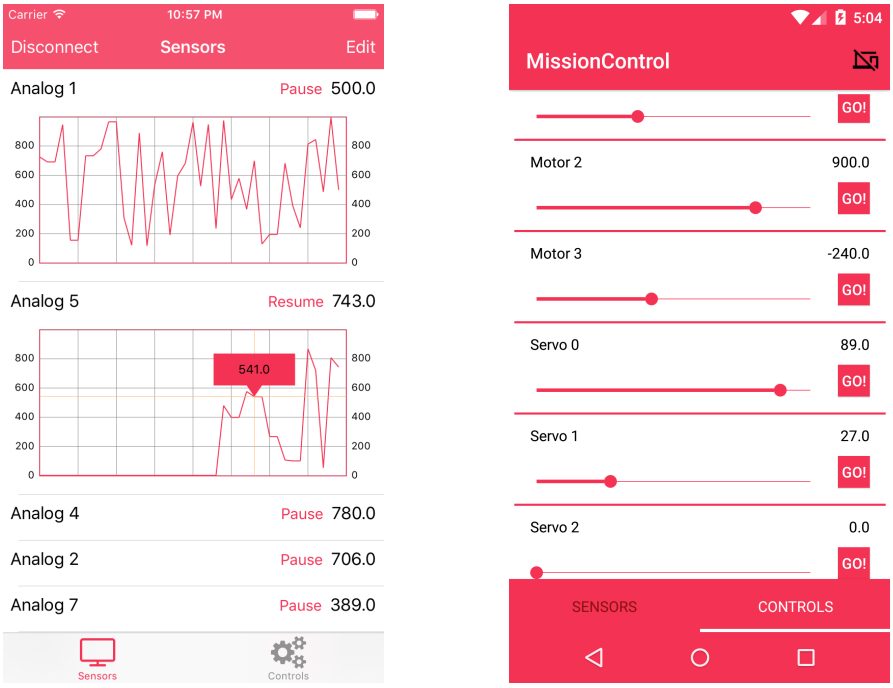

Fig 3. A screenshot of the iOS version of MissionControl (left) in the sensor readout view and of the Android version of MissionControl (right) in the actuator control view.

D. MissionControl Controller Markup Language (MC2ML)

To make it easier to implement support for additional controllers and to customize what data is shown on the clients as well as what actuators can be controlled by the user a markup language called MissionControl Controller Markup Language was developed. MC2ML is a XML based language in which the libraries that need to be included, the functions to access sensor data and the functions to control the actuators on the robot are defined. The language therefore also incorporates parts in which plain C source code is written. Providing a standardized markup language makes it easier to develop controller-specific MissionControl implementations without any knowledge of internal libraries used by the server. Hence none of these libraries have to be exposed and errors that can be made while adapting to a new controller or to a specific set-up are minimized. Through it, it was possible to simplify the process of bundling and deploying controller-specific support files.

A parser that is bundled with the server was written to create and compile the platform specific RSAL out of the MC2ML file.

III. Integration in a Botball Environment

Botball, being an excellent educational robotics program, is great for learning about mechanical design and programming of robotics systems. To enhance the teaching effects of Botball a prototyping tool with remote control and data readout capabilities like MissionControl can help by providing greater insight into what the bot actually detects while it’s running a program. MissionControl is compatible with both the KIPR Link and KIPR

Wallaby, the two newest robotics controllers that were developed for Botball. The design of MissionControl made it feasible to implement it on both controllers where it runs stable with more than 25 updates per second.

Once MissionControl is installed on the controller, it can be launched as a service in autostart mode. Since MissionControl hardly needs any resources if no client is connected (see "Server Performance Analysis"), it doesn't affect controller performance. This makes MissionControl available on the developers demand without the need of being activated before using. Since most Botball students use the comfortable Wi-Fi compilation feature, the Wi-Fi must be set up on the controller where, no additional set-up is required.

If a robot is finished, a MC2ML file can be created that is adapted to the final robot's design. Within the file the sensor ports used on the bot can be defined and named with the other ones being ignored whereupon additional data sources, i.e. blob tracking counts, can be added. Control groups can be defined in which i.e. collecting a game table item can be an action performed by the click of a button.

A. Comparison of different workflows

One common way to monitor sensor data, used by many Botball students, is to use the output on the built-in display of the controller. This approach makes it hard to monitor the robot outside the developer's visual field since users have to be able to watch the display at any time. Another common way is to launch the program via a remote Secure-Shell (SSH). While making remote monitoring possible, no real time graphs can be created and the output is dependent on the running program. The third way of monitoring sensor data is the new Web UI of the KIPR Wallaby. This makes it possible to see the display output from any connected browser. However it completely lacks any kind of control and the data displayed is dependent on the running program as with SSH and the standard display output. The different methods are compared to MissionControl and graded in "Table I".

TABLE I COMPARISON OF DIFFERENT TASKS AND ATTRIBUTES OF THE DATA READOUT AND CONTROL METHODS INTRODUCED ABOVE. "+" IS OPTIMAL, "~" IS ACCEPTABLE, "-" IS SUBOPTIMAL.

Task	**Methods for remote controlling and monitoring**			
	Display Output	***SSH***	***Wallaby Web Output***	***Mission Control***
Set up per use	None	SSH Connection	Controller: None Client: Connection via Web App	Controller: None Client: Connection via App or Web App
	+	-	~	~
Simultaneous connected devices with output	None	1	1	>1
	-	~	~	+
Distance	Normal Viewing distance (~50cm)	Same network	Same network	Same network
	-	+	+	+
Custom Control abilities	Hardware button and virtual buttons	Parsing console text input (depends on running program)	None	Fully customizable UI-elements. Always available
	~	~	-	+

Task	Methods for remote controlling and monitoring			
	Display Output	***SSH***	***Wallaby Web Output***	***Mission Control***
Client platforms	None	Multi platform (SSH application required)	Multi platform (Web browser required)	Multi platform (Web browser or native Apps required)
	-	+	+	+
Controller platforms	Controllers with built-in display (e.g. Wallaby, Link)	Multi platform (Wallaby, Link, most other platforms)	Wallaby	Multi platform
	~	+	-	+

IV. Server Performance Analysis

With most robotics controllers in educational robotics being sophisticated Linux based computers, it is possible to make use of many of the technologies that can be used on a regular computer. However processing power of most controllers is way below modern PCs with single-core processors and clock speeds below 1GHz on the KIPR Link and KIPR Wallaby (the main targets for MissionControl). Because of that a system like MissionControl has to be written with performance in mind in order to not worsen the performance of other programs running on it. The runtime usage of resources of the MissionControl server was tested by letting a robot drive straight while running the server at an update rate of 20 updates per second and periodically connecting and disconnecting up to two clients at the same time. First one client was connected for 20 seconds and then disconnected, after that 2 clients, with 10 seconds pause, where connected simultaneously. The average of the CPU usage in idle and under load where calculated. To monitor CPU and memory usage during the tests the command line task manager *top* was used.

A. Performance on the KIPR Wallaby

The KIPR Wallaby is the newest and current controller used in the Botball competitions, it features built-in Wi-Fi and can also be connected to a PC with USB using Ethernet via USB. It runs a custom Linux OS on an ARMv7 processor and natively supports Python 2.7, so MissionControl can run out of the box.

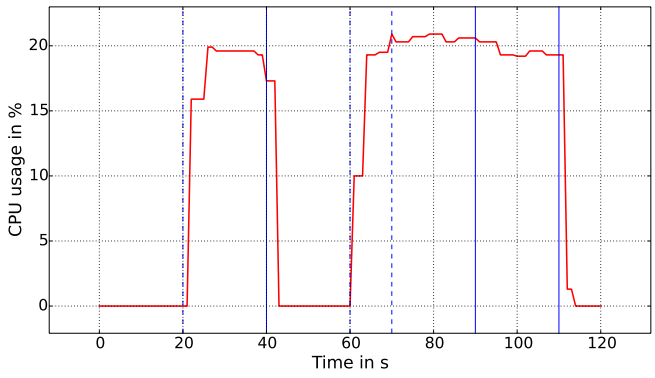

Fig 4. CPU usage of the MissionControl server on the KIPR Wallaby. Solid, vertical, blue lines mark a disconnecting client; dashed, vertical, blue lines mark a connecting client.

As shown in “Fig. 4” the average CPU usage of the server is 21.35% when clients are connected. An additionally connected client adds about 1% of CPU usage. If no client is

connected average CPU usage drops to 0% in idle. Neither robot speed nor execution time were affected by MissionControl, although more than 20% CPU usage is considered high.

B. Performance on the KIPR Link

The KIPR Link is the previous generation Botball controller which was used from 2013-2015. It also features built-in Wi-Fi and runs on Ångström-Linux which is a Linux distribution created for scalability and embedded devices [5, 10]. Its CPU is an ARMv5 processor and it also natively supports Python 2.7, although some parts of the Python standard lib are missing, which have to be added manually [5].

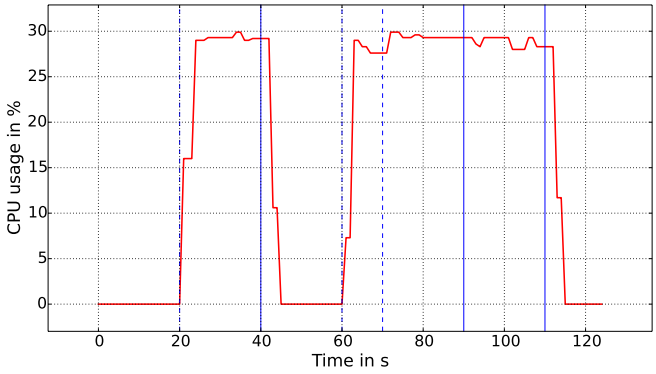

Fig 5. CPU usage of the MissionControl server on the KIPR Link. Solid, vertical, blue lines mark a disconnecting client; dashed, vertical, blue lines mark a connecting client.

When a client is connected CPU usage averages at 27.8% as it is seen in “Fig. 5”. As on the KIPR Wallaby CPU usage increases by about 1% for any additional client connected, idle performance drops to 0% usage and the performance of other running programs doesn’t seem to be affected by MissionControl, although CPU usage is even higher.

CONCLUSIONS

Having an open protocol and a set of tools for remote monitoring of robotic systems mostly enhance the workflow of developing robots. It gives the developers greater insight in what their robots are actually measuring in almost real time. But for many developers it’s also important to have a modular system since the used robotics controller might change over time. MIDaC and MissionControl are an approach to deliver an open application stack while still being modular enough to be used on a multitude of clients. It makes use of standard and well-proven technologies like TCP/IP sockets and JSON which are integrated in most modern programming languages.

However, one of its major problems is its incapability to run on low-end controllers that are not running full Unix systems with networking support. This excludes most Arduino and Arduino-like controllers. Another problem is the need of preexisting infrastructure like networks and the need to install the server on the controllers, which requires knowledge of the Unix shell.

In the future MissionControl has to be tested in a great variety of scenarios while support for more controllers has to be added by the project team as well as third-parties. Future co-operations between teachers and academics should improve MissionControl and help with integrating it more in educational events like Botball.

APPENDIX

MissionControl is an open-source project developed by Team items for HTL Wiener Neustadt and F-WuTS. Source code and documentation is available on GitHub (http://github.com/team-items/) and on the MissionControl website.

ACKNOWLEDGMENT

The authors would like to thank Dr. Michael Stifter for his support during the work on this publication; DI Harald Haberstroh for his support and sharing his knowledge during the development of MissionControl and MIDaC and DI Harald Breidler for his support as the project advisor.

REFERENCES

[1] V. G. Cerf, E. Cain "The DoD Internet Architecture Model", http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.88.7505&rep=rep1&type=pdf, Publication, 1983, accessed 2016-02-20

[2] C. Facchi, "Methodik zur formalen Spezifikation des ISO/OSI Schichtenmodells“, Chapter 1, https://www4.in.tum.de/publ/papers/Diss_Facchi.pdf, publication, 1995, accessed 2016-02-20

[3] N. Nurseitov, M. Paulson, R. Reynolds, C. Izurieta, "Comparison of JSON and XML Data Interchange Formats: A Case Study“, http://www.cs.montana.edu/izurieta/pubs/caine2009.pdf, publication, 2009, accessed, 2016-02-20

[4] N. Yocom, "The Definitive Guide to Linux Network Programming", Chapter 7, ISBN 1590593227, textbook, 2004

[5] B. McDorman, J. Southerland, "A Look Inside the KIPR Link", http://files.kipr.org/gcer/2013/proceedings/McDorman_A_Look_Inside_the_KIPR_Link.pdf, publication, 2013, accessed 2016-02-20

[6] S. Zeltner, D. P. Miller, "Kiss Your Old KISS Goodbye", http://www.gcer.net/scoring/papers/KISS_Miller_KissYourOldKISSGoodbye.pdf, publication, 2015, accessed 2016-02-20

[7] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, A Ng, "ROS: an open-source Robot Operating System", http://pub1.willowgarage.com/~konolige/cs225B/docs/quigley-icra2009-ros.pdf, publication, 2009, accessed 2016-02-20

[8] http://www.roboticslibrary.org, website, accessed 2016-02-20

[9] J. Wolf, "Linux-UNIX-Programmierung", 2nd Edition, Chapter 11, ISBN 3-89842-749-8, textbook, 2006

[10] http://www.angstrom-distribution.org, website, accessed 2016-03-16