Høgskulen på Vestlandet

Bacheloroppgave - Data/Informasjonsteknologi

DAT190

Predefinert informasjon

Startdato:	25-05-2018 09:00	**Termin:**	2018 VÅR
Sluttdato:	04-06-2018 14:00	**Vurderingsform:**	Bestått/ikke bestått
Eksamensform:	Vurderingsmappe	**Studiepoeng:**	20
SIS-kode:	203 DAT190 1 MA-1 2018 VÅR		
Intern sensor:	Adrian Rutle		

Deltaker

Navn:	Kolbein Toreson Foldøy
Kandidatnr.:	2
HVL-id:	149909@hvl.no

Informasjon fra deltaker

Tro- og loverklæring *:	Ja
Jeg bekrefter at jeg har registrert oppgavetittelen på norsk og engelsk i StudentWeb og vet at denne vil stå på vitnemålet mitt *:	Ja

Gruppe

Gruppenavn:	D1
Gruppenummer:	9
Andre medlemmer i gruppen:	Jonas Zacharias Dørum Backer

Western Norway University of Applied Sciences

Common programming interface for robots with communication

Felles programmeringsgrensesnitt for roboter med kommunikasjon

Education Programme Bachelor Computer Engineering

Department of Computer science

Faculty of Engineering and Science

Submission date: 04.06.2018

Number of words: 10831

Jonas Zacharias Dørum Backer

Kolbein Toreson Foldøy

Faculty of Engineering and Science

BACHELOR PROJECT TITLE PAGE

Report title: Common programming interface for robots with communication	*Date:* 04.06.2018
Author(s): Jonas Zacharias Dørum Backer, Kolbein Toreson Foldøy	*Number of pages: 33* *Number of appendix pages: 5*
Field of study: Computer engineering	*Number of CDs/DVDs:* 0
Contact person: Adrian Rutle	*Classification:* Open
Remarks:	

Assigner: Western Norway University of Applied Sciences	*Assigner reference:*
Assignor's contact person: Adrian Rutle	*Phone:* +47 55 58 77 91

Summary:

An extension to a Domain Specific Language for robot script generation containing methods for communication between robots of different programming languages.

Keywords:

Programming	Robots	Communication

Høgskolen på Vestlandet, Avdeling for ingeniør- og økonomifag
Post address: Postbox 7030, 5020 BERGEN   Visit address: Inndalsveien 28, Bergen
Phone 55 58 75 00   Fax 55 58 77 90   E-mail: post@hvl.no   Homepage: http://www.hvl.no

PREFACE

This report is written by Jonas Zacharias Dørum Backer and Kolbein Toreson Foldøy, software engineering students at Western Norway University of Applied Sciences.

We would like to thank our HVL contact person, Adrian Rutle, for the help and feedback he has provided throughout the project.

TABLE OF CONTENT

1 INTRODUCTION

Robots in our society

Robots come in different sizes and shapes, with different purposes and abilities. Robots are machines programmed to do something, by control or by itself (Oxford dictionary, 2016).

In today's society robots have revolutionized the efficiency and productivity in many field. The use of robots in the field of manufacturing cars have increased the capacity and quality of the cars, in addition to protecting workers from performing extremely dangerous tasks. In agriculture drones are being used to analyze soil, plant seeds and trees as well as choosing the best moment to harvest. Whether robots are being used for bomb disposal or vacuum cleaning depend on both its hardware and the program they are running (Mazur, 2016).

1.1 Goal and motivation

The goal of this assignment is to research the possibility of establishing communications over bluetooth or WiFi between robots programmed with a common language. The already made common language called "Commonlang" will have to be modified to cater to this functionality. When this has been achieved, we want to be able to get the robots to cooperate in a sensible fashion.

At first our goal will be getting a communication link between the computer and each robot up and running, by hardcoding functionality for this in the robots' respective language. Methods and protocols for this will have to be decided. Then we can start working on a link between the robots via a computer. When this functionality has been achieved we must also integrate this into the "Commonlang" language, this will be the largest part of the assignment. This "Commonlang" must be translated into java code, and then the into the language each robot understands, as has already been done without the communication possibilities (Gya and Solhaug, 2017).

1.2 Context

The assignment is based on work done by students of Western Norway University of Applied Sciences from the spring of 2017. The previous work focused on code generation for multiple robots via a common language. Our project is to continue working on the earlier assignment to allow the robots to communicate between each other.

1.3 Limitations

There are multiple cases of limitations when working on a project. As we do not have a project manager, problems might occur regarding direct decisions and control. Communcation between the members of the project, as well as the supervisor, can also be a limiting factor. A big part of the assignment is working with the domain specific language made by the previous bachelor group. It may prove difficult to further implement communication methods for the language in its current state. The assignment should be done over the course of a few months, and with the lack of experience of wireless communication between robots, time will be one of the most considerable limitations.

1.4 Resources

Resources for the project comes in three variants, hardware, software and knowledge based.

1.4.1 Hardware:

Two robots will be used during the assignment, one EV3 lego robot running “EV3dev” and the “Brickman” software and an Arduino based robot we’ll flash ourselves.

Electronics like breadboard, wires and Arduino based shields.

A router to set up a local network, for ease of access and isolating our results making troubleshooting easier.

Personal computers will be used to write software and writing to / flashing the robots.

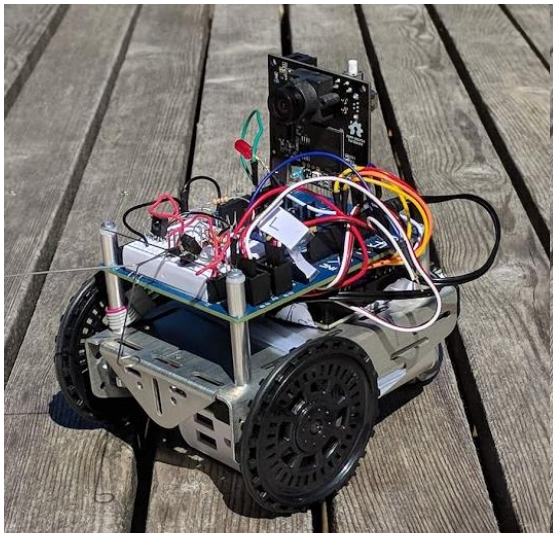


Fig. 1.1: Arduino robot

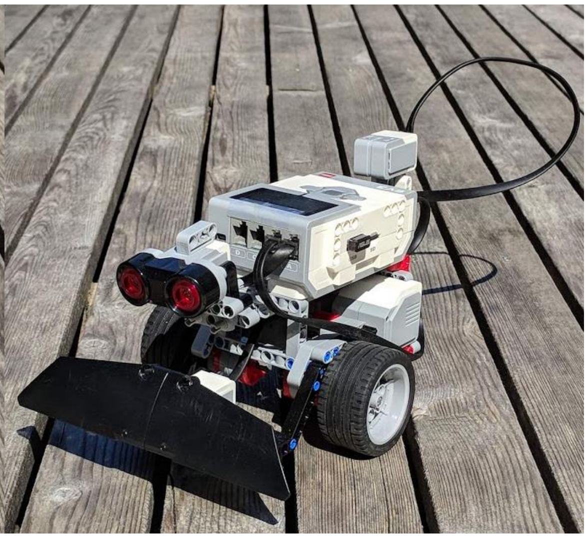


Fig. 1.2: Lego Mindstorm EV3 robot

1.4.2 Software:

Python IDE / APIs, Eclipse IDE, Xtext API, Arduino IDE / APIs, Putty, winSCP, PixyMon will be used.

1.4.3 Knowledge based resources:

Project supervisor, the previous assignment this one is based on and the students who did it, articles on the web describing any part of what we're trying to do and API documentation.

1.5 Organization of the report

The report is organized with a structure consisting of a series of main headings, beginning with an introduction to the project. It starts off by introducing the reader to what a robot is and its role in our society, followed by the motivation and goals set for the project and resources used. In the project description the reader will learn how the robots are programmed, the project members background, as well as initial requirements and solution idea. The next heading is project design which contains the methodology used throughout the assignment, possible approaches on how to achieve the goals set and tools used. Further, in detailed design, it is described in detail which design decision have been made and why. Thereafter, the evaluation method is explained, followed by the evaluation results. In the discussion section, the consequences of the approaches chosen is discussed, in addition to potential improvements that would have been done if we could start over again. Lastly, we conclude by summarizing the goals that were set from the start, confirming whether they have been reached and further work that can be done.

2 PROJECT DESCRIPTION

2.1 Programming of robots

Every robot requires some form of program running, whether it is being manually controlled or running autonomously. To perform the desired actions, the robot's processing unit will read machine code, a series of ones and zeros. The machine code is a result of a compiled set of instruction written in a specific programming language, which vary from robot to robot. (Oxford dictionary, 2018).

2.2 Practical background

The members of the project group are both software engineering students with a couple of years experience in programming with different tools and programming languages. A major part of the project is programming in C and Python. The project members have a limited experience in C programming for Arduino from the spare time, but next to no experience with Python.

2.2.1 Project owner

The owner of the project is Western Norway University of Applied Sciences, Institute of Computing, Mathematics and Physics, a university college institute in Bergen.

2.2.2 Previous work

The bachelor assignment builds on previous work by other students. The main project has gone over multiple iterations. Some students made a drag and drop interface for programming robots easily. Then another group of students made a language known as "Commonlang" which supports simple text-based programming of robots that are programmed with different languages by parsing the original code to both C and Python. This project was originally meant to be a continuation of the previous project, but they went their own way using xtext (a framework for making domain specific languages) instead of a graphic interface. Our assignment will further develop the "Commonlang" language allowing for communication possibilities (Gya and Solhaug, 2017).

2.2.3 Initial requirements specification

Initial requirements of the assignment is to further implement the already existing common language with a set of methods. These methods will be generated to the robots and their respective programming language which will allow them to interact with each other. The robots should process the data that are wirelessly received and respond to it in accordance to the program that was written to the robots. When the requirement of sending data is met, the robots can potentially send their positions, the path they have driven or simple boolean values to be used in a meaningful way.

2.2.4 Initial solution idea

First, communications between the two robots and a computer must be established. This will be done by setting up the robots as HTTP servers and the computer as a client making requests to the HTTP servers. To make this process easier the robots would be coded directly, at this point there is currently no connection to the common language. Once this has been achieved as a proof of concept we used the computer to serve the connection between the two robots. The next step will be the integration of this functionality into the "Commonlang" language and make it useable by the simple programming interface. We will then commence the final stages of testing which also took place during all other stages of development (Belche, Peon and Thomson, 2018).

2.3 Literature background

The literature used range from peer reviewed research papers to articles on webpages. Research into specific details on how protocols, scripts, software and sensors work specifically is usually based on API documentation, guides, previous bachelors and articles. More theoretical knowledge like design science method, DSLs in general and theory behind robot programming on a higher level is mostly gathered from theses and published research papers.

3 PROJECT DESIGN

3.1 Model / abstraction level

There are a lot of different ways to program robots, from writing the robot specific scripts directly to using different frameworks with varying functionality. Commonlang is one of many frameworks with tools meant to aid the process of programming robots. The version developed on in this assignment is a text-based framework with metamethods, function abstractions providing a higher-level language to make the language easier to master.

There is also a work in progress visual interface with drag and drop functionality being developed alongside the version being worked on for this assignment.

Commonlang is not the only framework available for programming of robots, ROS and papyrus-rt are other examples. ROS (The Robot Operating System) is an open source project existing of a collection of tools, libraries and conventions, its aim is to simplify programming of robots of all shapes and sizes offering complex behaviour (ROS, 2018).

Papyrus real time is another visual framework for programming robots. It generates C++ through the UML-RT modeling language. The language is based on capsules and protocols, capsules define a concurrent behaviour specified by a state machine. Capsules receive messages defined by a protocol, which is then processed by its state machine (Redding, 2018).

One of the unique aspects of commonlang is its focus on outputting multiple scripts with different languages. The output scripts layout is dependant on the XML configuration files that contain what language it should be parsed to (C or Python) and the implementation of metamethods.

3.2 Required Capabilities

Commonlang is the domain specific language where you can easily write scripts through abstracted metamethods defined in the XML configuration files that must be provided for each robot. The goal, to implement communication functionality, whilst still keeping the usability and ease of access to an acceptable level, has to be implemented through modifying DSL itself or through the metamethods (Gya and Solheim, 2017).

3.3 Design Science methodology

When conducting design science research, a certain set of analytical techniques and perspectives are used. The goal is improving and understanding the behaviour of aspects of Information Systems (IS), this is done in two ways (Vaishnavi, 2017).

“(1) the creation of new knowledge through design of novel or
innovative artifacts (things or processes) and (2) the analysis of the artifact’s use and/or performance with reflection and abstraction.” (Vaishnavi, 2017).

These artifacts include algorithms, human to computer interfaces, system designs or languages. The research is the activity that allow understanding of a certain phenomenon. In this assignment the design science methodology will be used both in research and evaluation of the results later (Vaishnavi, 2017).

3.4 Possible approaches

3.4.1 Communication protocol MQTT

A possible approach is setting up a publish-subscribe-based system with the communication protocol MQTT (Message Queuing Telemetry Transport). A computer will be set up to work as a server, forwarding messages to units subscribed to the specific topic that another unit has published a message to (MQTT.org, 2018).

3.4.2 Communication protocol HTTP (Robot side)

The approach will be to implement a web server on each of the robots. Both robots will be able to receive and send data to each other using their respective IP addresses with HTTP requests. They will initially communicate indirectly via a computer. The computer will be connected to each of the robots’ servers as a client to which it will forward data. There will also be options to manually send commands to either or both robots (Belche, Peon and Thomson, 2018).

3.4.3 Communication protocol HTTP (PC side)

Another approach will to be to set up the computer as an HTTP server and robots as clients connected to the server. In this case the clients should either continuously ask the server for new data or if it has been modified since last time it was checked. It is uncertain how efficient this approach will work for the robots. The clients will send data as parameters appended to the server IP address (Belche, Peon and Thomson, 2018).

3.4.4 Communication protocol Telnet

Should using the HTTP or MQTT protocol prove inconvenient, the last approach would be checking out the possibilities of using a Telnet based system. This would be a simpler implementation, but not quite as versatile as using HTTP. The reason HTTP is a good solution is in part to its ease of access to other devices connected to the same network. The Telnet solution would be less flexible and would require a more predefined set of commands to be fired at specific times, instead of having the computer send and receive requests (Postel, 1980).

3.4.5 Arduino robot using external WiFi shield

The Arduino robot at disposal for the project is already equipped with an original Arduino board. The board itself does not have hardware that allows it to send and receive data wirelessly in any shape or form. An external shield/board is required to give the board the ability to connect wirelessly, as is desired. The approach will be to mount a board called Wemos D1, which is an Arduino board specifically made for wireless connection, on top of the currently installed Arduino board (Banzi, M., 2018).

As the Wemos is a microcontroller, a script needs to be running on it that will forward the data it receives to the main Arduino board through serial communication (SPI). Alternatively, a WiFi shield could be mounted on the Arduino board instead of using the Wemos, which would send the data directly to the board. However, as this shield is not at disposal and would cost time and money to acquire, the Wemos may be the better approach for this project (Grusin, M., 2017).

3.4.6 Arduino robot using Wemos D1 board

With this approach, the Arduino board already installed on the robot will be replaced with the Wemos D1 WiFi board. As they are both Arduino compatible boards, there would be little change in the code written for the robot. The downside of the approach is that the boards have different pin numbers, meaning it cannot be directly mounted to the robot shield (Parallax) that is installed onto the robot's frame. In order to connect the two, wires have to connect each of the boards' pins together. When the wiring is done, the pin numbers in the code needs to be adjusted and it is all set up. With this solution only one script is needed, handling all the communication as well as all the other sensors and motors (Stackexchange, 2018).

3.4.7 Commonlang grammar modification

When integrating the communication into the language, one of the approaches would be to modify the grammar, adding reserved keywords that the language would interpret as specific methods.

3.4.8 Metamethods in XML for communication

Another approach to how the communication could be implemented into the language is to create metamethods for it to use after importing them into the metamethod collection. These methods are implemented for each robot in their respective XML files (Gya and Solheim, 2017).

3.5 Specification

We are going to create extensions to an existing language to support communications between robots via WiFi. First, communication will have to be established between the robots through a computer, when this is done this functionality can be added to the language. The user will then have to use the Commonlang language to write easy commands for the robot, now with the added functionality of communication. This code will be compiled into java and then Python and C. The output files can then be written to the robots and a program on the computer will be running to manage the communication features if used (Gya and Solheim, 2017).

3.6 Selection of tools and programming languages

3.6.1 Software and documentation:

For this assignment multiple integrated development environments (IDE) will be used for different segments of the task and programming languages.

Python IDE will be used for developing and testing Python code for the Lego Mindstorm EV3. Later this code will automatically be generated by the common lang. The methods used for the EV3's Python code comes from the Python application programming interface (API).

Arduino IDE is the code editor used for writing code for the Arduino robot. The code will contain methods from the Arduino API which will allow us to set up a WiFi-server or client, among other things.

Xtext is a framework used for development of programming languages. It lets you define your language by describing grammatical rules as well as defining data types and grammatical structures. Based on these sets of rules, you are able to generate a code generator. Xtext will be used for further development of the common language. It comes as a plugin for Eclipse IDE (Xtext, 2017).

Eclipse IDE is the developing environment where, together with Xtext, the common language will be developed as well as writing the script that be generated to Java code.

Putty is an open-source terminal emulator that provides us with the ability to control the Lego Mindstorm EV3 with commands over Secure Shell (SSH) protocol. It comes in handy in the earlier stages for prototyping and running scripts directly on the robot.

WinSCP is a free program for transferring files using File Transfer Protocol (FTP). It is used for transferring the runnable code from the computer to the robots.

PixyMon is a program used to configure the Pixy camera mounted on the Arduino robots, setting it up to track objects of a desired colors, in addition to adjusting contrast, light exposure and more.

3.6.2 Programming languages:

Commonlang is the programming language created by the students of last year. This is the language one writes in to create the script and commands for the robots. It will be parsed into Java code. The Java code will be further be parsed into the EV3 and Arduino's respective languages, Python and C (Gya and Solheim, 2017).

3.7 Project development method

Because this is a two-man project, an agile development method seemed like the best way to go. Communication within the group is relatively easy to maintain because the both of us work together every day at the same place. The focus will be on developing the most critical functionality to begin with, as is typical for agile development (Cockburn and Highsmith, 2001).

Less crucial functionality will be implemented thereafter, and time constraints will determine the quality of the final product.

3.8 Evaluation method

Testing will be performed both during and after completion of the project. Artifact testing will be carried out where possible, and functional testing otherwise. An evaluation of functionality and ease of access will also have to be done as this is being made to cater to users of less complex understanding of the concepts of programming robots.

As part of the functional testing a scenario was created during development of the MQTT communications functionality. The scenario called “Red light” is based on a children's game. The game consists of a leader and a group of players. The goal is to be the first player of the group to reach the leader. However, the group may only advance towards the leader when he/she is not looking. If the leader catches a player moving when facing them, the player in question loses and will have to start from the beginning again.

This scenario has been created using robots. The Arduino robot acts as the leader, whereas the EV3 robot is a player. The robots are positioned two meters apart facing each other and the game starts when the Arduino gets the go ahead from the PC. The Arduino robot then turns 180 degrees, which is noticed by the EV3. The EV3 starts moving slowly towards the Arduino robot, until it gets a “red light” message from the Arduino. The message is sent once the Arduino robot is midway turning back towards the EV3. The EV3 should now be at a standstill and the Pixy starts looking for significant movement of the reflector on the EV3. If the Pixy does not sense a movement above a threshold the Arduino robot will turn back again, and the EV3’s sonic sensor tells it to move. This continues until the EV3 has reached the Arduino robot with its touch sensor. If the Pixy camera notices the EV3 moving when it should not, it will send it a “loss” message and the EV3 will move back to start.

4 DETAILED DESIGN

The project is quite technical in nature and there are many decisions to be made regarding design. What communication protocol is the most suitable for a simple, yet stable connection between a limited number of robots? How can one utilize the sensors in a good manner? What should be the abstraction level of the code, that will let people easily program the robots, yet being able to do specific tasks?

As the commonlang will get the extension of being able to set up a wireless connection between the robots, a communication protocol needs to be chosen. At first it had to be determined what kinds of protocols the two robots is able to use. The EV3 supports both Bluetooth and WiFi, where as the Arduino board only supports WiFi. Naturally WiFi is the clear choice.

Further it had to be established how the robots were to communicate. There are multiple ways to set up a communication over WiFi. Using HTTP was under consideration for some time. The initial plan was to set up a web server on each of the robots, sending and receiving data over HTTP packages. This seemed to be cumbersome and complicated to some degree especially considering it is not a protocol purposely made for this kind of task. Another protocol, MQTT was presented at a meeting with the project supervisor. The research into MQTT proved fruitful, it fulfilled all current requirements of the project and a decision was made to use this as the communications protocol for the robots.

4.1 MQTT

MQTT (Message Queuing Telemetry Transport) is a more lightweight messaging protocol that allows the robots to subscribe to topics. They can also publish messages to the topics, so that units subscribed to the topic will receive the messages. A broker is required for this setup to work. It keeps track of available topics and units connected to the server, forwarding incoming messages to all the units subscribing the respective topics (MQTT.org, 2018).

For a client to subscribe on a topic it needs to know the IP address of the broker. It’s a static IP address, meaning it won’t change from time to time. This comes in handy as the IP address is hardcoded in the scripts running on the robots.

The work on this project mainly takes place on the school campus. Wireless connection of the robots is a central part of the project but can be hard to establish on the school’s own network. It requires not only the network’s SSID and password, but to log in with a personal student account, which is not a trivial task. Therefore, a router has been acquired to set up a simple local area network which is easy for both the PC and the robots to connect to.

4.2 PC

The PC is an essential part of the setup. It is being used for multiple purposes, such as writing program code, uploading scripts and sending and receiving messages wirelessly. As MQTT is the prefered communication protocol, a program called Mosquitto needs to be installed. It grants the ability to set up a broker and enter commands such as subscribing and publishing messages to desired topics into the command prompt window (Eclipse Mosquitto, 2018).

Depending on how the scripts are written for the robots, they can initiate and run the program themselves or they can start when receiving a specific message from the PC. The PC can also be used for listening to the commands sent between the robots to see if the scripts are running as intended.

The Arduino robot is using an object detecting camera which needs to be set up and calibrated via the PC. Using a software on the PC called Pixymon, one can define which objects to detect based on its color. Simply drag a selected area over the object in the visual tool using the live feed from the camera. This must be done almost every time the robots are in a new environment will different light and brightness because the camera is quite sensitive to light exposure (Rowe, 2018).

4.3 Arduino

4.3.1 Wemos D1 R2 Setup

As it would take some time and cost to get a WiFi module for the original Arduino UNO R3 that was present on the parallax robot, a decision to go with another Arduino UNO like board was made. The board called Wemos D1 R2, a popular board within the DIY electronics communities, is roughly based on a Arduino UNO R3 with similar hardware specifications and physical design only it supports wireless communication due to its built in WiFi chip. However, the IO pins are quite different supporting about half the input/output capability of the Arduino. Because of this it could not so easily be slotted into the parallax module made to forward each of the original Arduino UNO R3 pins (Stackexchange, 2018).

Whilst still using the HTTP approach a solution to this that was tested, was using both boards, plugging the Arduino into the parallax and using a serial connection between the

Arduino and the Wemos where the Wemos would forward incoming and outgoing messages to and from the Arduino and the Arduino would contain the logic to deal with these messages and make the parallax react accordingly. An SPI (Serial Peripheral Interface Bus) was set up between the rx/tx ports on the two boards and software to support the forwarding of messages was written with some success. Due to the inexperience of the group members it took some effort to read the data on the Arduino, sometimes the bytedata sent over the link would be out of order or not interpretable. Baud rates was matched, and messages sent by the HTTP server running on the Wemos was verified. In the end we managed to turn an LED on and off with a phone browser displaying the web server hosted by the Wemos sending bit-data over serial to the Arduino module setting the led pin high and low, however this seemed like a very complicated way to do one of the lesser tasks we had to do. Hosting a web server seemed to draw a lot of the Wemos' resources and physically having two boards on the parallax was not ideal. This topic was raised at a meeting with the project owner and supervisor and another approach was suggested (Grusin, M., 2017).

Fig. 4.1: Arduino UNO R3

Instead of using HTTP the communication would now be done over MQTT, more on how the Wemos runs MQTT will be explained later. With this new approach a decision was made to also go with the Wemos board alone, putting aside the Arduino UNO board. Seeing as the pins did not match up to the parallax board directly, wires were run from and to the appropriate pin slots. The Wemos would from now on run both the logic, telling sensors and motors what to do, whilst also keeping track of incoming and outgoing messages. Although this sounds like a lot more work the ease and lightweight of MQTT would allow this without exceeding the Wemos' capabilities. Charts were made to keep track of which connections went where and to keep track of how this lined up to the previously used Arduino board (Stackexchange, 2018).

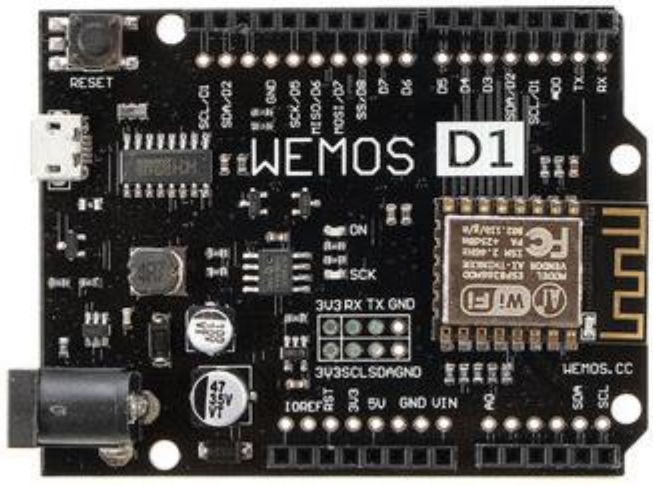


Fig. 4.2: Wemos D1 R2

4.3.2 Communication protocol (MQTT)

As stated previously the Wemos board runs MQTT as its communication protocol. It does this by importing and declaring a PubSubClient. This client then looks for specifically named variables and methods to be set and called at specific times. First off it needs to establish a connection to the broker through the router on the local network, this is done by instantiating three variables called ssid (identifier of the router), password (of the router) and mqtt_server (ip of the broker on the same network). The setup function in the sketch sets up the client with a server/broker through the client.setServer method which takes the ip of the broker and the default mqtt port of 1883. It also sets the clients callback method responsible for parsing incoming messages. The main "loop" function of the Arduino sketch (the one required for running Arduino code) calls for a reconnect if the client is not connected and also runs client.loop(), "This should be called regularly to allow the client to process incoming messages and maintain its connection to the server." (O'Leary, N. 2018).

The client can subscribe to topics it wants to receive messages on and publish on topics it wishes to publish on, these messages always go through the broker and is distributed to clients subscribed to the topic published on.

In the “red light” demo the Arduino robot has to send messages to the other robots playing about stopping and losing. It subscribes only to the message that starts the game sent from the computer.

4.3.3 Sensors

The Arduino is equipped with multiple sensors and two continuous servos. The servos take an input value from 1300 to 1700 where 1300 is 100% pwm duty cycle clockwise movement and 1700 is the same counterclockwise. To stop the motors 1500 is written to them, although this varies a bit from servo to servo dependant on how accurate they are. Our right servo requires a signal of 1495 to stand relatively still. Since the Arduino robot does not have a gyro getting it to turn an exact number of degrees turned out to be challenging. It is dependent on surface friction, battery voltage and accuracy of the servos. However, the colour tracking Pixy camera mounted on top of the robot solves most of these issues when it is most critical that the robot is facing a certain direction (Parallax, 2018).

The Pixy camera is a camera calibrated on a computer to track certain colours, it is used to turn to a specific player in the “red light” game and check for movement of that player. It measures the width of the tracked object and looks for a pixel change in this value beyond a certain threshold. If movement is confirmed it publishes a “loss” message to the one in question and this player must start over from the startline (Rowe, 2015).

The Arduino also has a couple of whiskers used to determine when it has bumped into something. This could be used if it is going to take the place of a player and needs to identify then it has won.

4.3.4 Battery

The Parallax Arduino robot contains a battery tray underneath that takes 5 AA batteries. The supplied voltage is then brought down to 5V by the Wemos board that takes 5~9 volts DC. These batteries, however, would not last very long so instead of using non-rechargeable batteries a decision was made to go with a standard USB powerbank. These typically supply 5 volts, ideal for our purpose, last longer and are rechargeable.

4.3.5 Code

The code for the Wemos board also called a “sketch” in Arduino lingo has to be flashed to the board beforehand and this is the code it will run when it is powered. Since it is not an actual Arduino running the sketch some changes has to be made both in the setup for writing the sketch and within the sketch itself. Changes in the setup includes downloading a package of modules for ESP8266 specific boards including this into the Arduino IDE and selecting the Wemos D1 R2 as the board to be written to as well as setting the upload speed to 921600. The change within the sketch itself is that an include of the ESP8266WiFi.h package must be made to make the sketch interpret the pin names that are specific for these types of boards, like "D0, D1" (Stackexchange, 2018).

Like the Arduino the Wemos will first look for a method called setup() and then a loop() method that will run forever. The setup is used to set pin modes, set up the motors, Pixy camera, MQTT client and serial (talk to PC when it is plugged in). The loop runs through the tasks the robot should perform over and over, like finding the players in the "red light" game, checking for movement of these and telling them to start over if necessary.

4.4 EV3

4.4.1 Communication protocol (MQTT)

The EV3 runs a distribution of Linux called debian, this allows us to use SSH to log into the machine and do tasks like installing software, running Python scripts and editing these. This method was used to install a MQTT library called paho. In the scripts containing communication functionality a paho MQTT client is imported and set up to connect to a broker, subscribe to topics and publish messages on specific topics.

In the "red light" demo the EV3 receives a message from the Arduino telling it to stop, the EV3 reacts to this message by stopping after a random "reaction time" set by a random number. It also publishes a message to the computer as a confirmation that the message has been received. If the EV3 wins, meaning it touches the Arduino, its pressure sensor triggers an MQTT message publish to signal the Arduino that it won the game.

4.4.2 Sensors

The EV3 is configured to use two sensors, an ultrasonic sensor measuring distance from the sensor to the nearest object in centimetres from 3-2550cm and a touch sensor letting it know when it has bumped into something. The ultrasonic sensor is used to determine when it is safe to start approaching the Arduino as it is turning around and is no longer looking at the EV3 gathering movement information. This is done by comparing the last ultrasonic reading to the next, if it exceeds a threshold the EV3 knows the Arduino is turning away. The pressure sensor is triggered when the front of the EV3 touches the Arduino and the game is won, as mentioned this publishes a message to the Arduino telling it the game is over (Burfoot, J. 2018).

4.4.3 Code

The EV3 runs Python scripts transferred to it beforehand. These scripts can be run directly from the EV3 block interface if they are marked as executables. You can also connect to the EV3 over SSH using some program like “Putty” to run the .py files with Python3.

The scripts use a common setup/loop setup where the main method calls the setup first and then the forever running loop. The setup is used to make variables and connect to sensors and motors using the ev3-lang-python library, and also to set up the paho MQTT client.

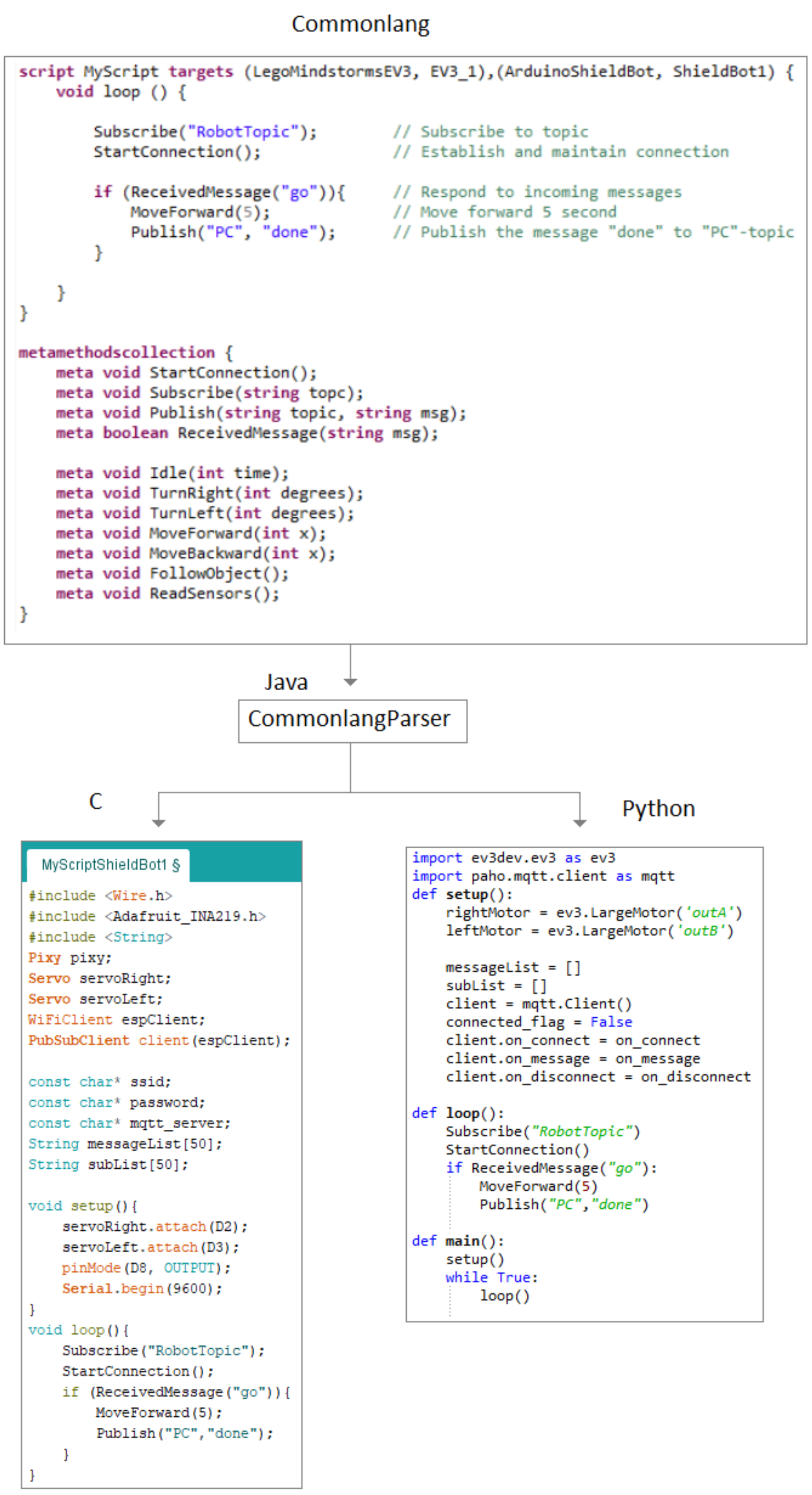
The loop continuously checks for incoming MQTT messages keeping connection to the broker and reacting to messages on a topic it is subscribed to. Depending on the nature of these messages it would run or stop the motors, read its sensors and go into predefined states.

4.5 Commonlang

Commonland is a domain specific programming language created specifically to write code for a set of various robots using different programming languages. It is an imperative and structured language, and in many ways similar to Java as the code will be further generated to. It consists of various control flow constructs to be found in other programming languages, like For, If and While statements (Gya and Solhaug, 2017).

The script is written in a file with the extension name “.commonlang” and consists of two main blocks. The first part contains the script definition, name of the script and a set of what robots the script will be generated for, in the form of configuration file names for each robot. The program code will be written inside a method called “loop” contained in the script block, and consists of user methods and metamethods. The output code, in C and Python, will continuously iterate this method, hence it does not allow for storing states, like integers and booleans.

The second block is called “metamethodcollection” which is a set of predefined methods that every robot with a different programming language is able to run and has its own implementation of. Each metamethod declaration consists of the reserved keyword “meta”, followed by a return type, name and its corresponding parameters. For example, the metamethod “MoveForward” will for some robots make two of the wheels spin forward, others four in a different speed. Some robots may even be flying, meaning they will have rotors spinning in different directions to be able to fly forward. The metamethods are defined and implemented in the robots two XML files.



```
script MyScript targets (LegoMindstormsEV3, EV3_1),(ArduinoShieldBot, ShieldBot1) {
    void loop () {

        Subscribe("RobotTopic");          // Subscribe to topic
        StartConnection();                // Establish and maintain connection

        if (ReceivedMessage("go")){       // Respond to incoming messages
            MoveForward(5);               // Move forward 5 second
            Publish("PC", "done");        // Publish the message "done" to "PC"-topic
        }

    }
}

metamethodscollection {
    meta void StartConnection();
    meta void Subscribe(string topc);
    meta void Publish(string topic, string msg);
    meta boolean ReceivedMessage(string msg);

    meta void Idle(int time);
    meta void TurnRight(int degrees);
    meta void TurnLeft(int degrees);
    meta void MoveForward(int x);
    meta void MoveBackward(int x);
    meta void FollowObject();
    meta void ReadSensors();
}
```

```
#include <Wire.h>
#include <Adafruit_INA219.h>
#include <String>
Pixy pixy;
Servo servoRight;
Servo servoLeft;
WiFiClient espClient;
PubSubClient client(espClient);

const char* ssid;
const char* password;
const char* mqtt_server;
String messageList[50];
String subList[50];

void setup(){
    servoRight.attach(D2);
    servoLeft.attach(D3);
    pinMode(D8, OUTPUT);
    Serial.begin(9600);
}
void loop(){
    Subscribe("RobotTopic");
    StartConnection();
    if (ReceivedMessage("go")){
        MoveForward(5);
        Publish("PC","done");
    }
}
```

```
import ev3dev.ev3 as ev3
import paho.mqtt.client as mqtt
def setup():
    rightMotor = ev3.LargeMotor('outA')
    leftMotor = ev3.LargeMotor('outB')

    messageList = []
    subList = []
    client = mqtt.Client()
    connected_flag = False
    client.on_connect = on_connect
    client.on_message = on_message
    client.on_disconnect = on_disconnect

def loop():
    Subscribe("RobotTopic")
    StartConnection()
    if ReceivedMessage("go"):
        MoveForward(5)
        Publish("PC","done")

def main():
    setup()
    while True:
        loop()
```

Fig. 4.3: Code generation from Commonlang to C/Python

4.6 XML

Each robot needs two XML files for the Commonlang to generate code for the respective languages. The first XML contains various type of data, such as the file format for the language files (py or C), global variables, declarations and method calls for the setup method of the final script. It also contains which metamethods the robot can execute. In this XML the metamethods have a simple code segment, either calling its corresponding method from other XML where it is implemented or returning a value from global variables or forwarding the return value from the method it is calling (Gya and Solhaug, 2017).

The other XML file contains the implementation of the methods in the robots' language. For instance, by setting a motor to run for the duration specified in the parameters or read the value of a digital pin and return it. The methods with its implementations are then placed directly into the final script as it is already being written in the correct language. Additionally, it contains a tag called "assignments" which is for setting up the sensors to the right pin values, as it will vary from robot to robot.

4.7 Commonlang grammar vs XML

An important decision had to be made regarding whether the communication should be integrated as part of the Commolang grammar or as additional metamethods through the XML files. Initially, the pros and cons had to be established for either case. By editing the grammar of the language, it would result in fewer unnecessary lines of code in the XML files that are never called in the program. It was discussed upon the idea of the Commonlang generating scripts that are clean and tailored for the program. Instead of pasting all the methods and variables from XML files, only the necessary lines of code get included. Minimizing the size of the program code becomes especially important considering some robots may have a very limited program memory onboard. As of now, the code written in Commonlang is all inside the loop method, meaning it will iterate through all the program's lines of code as long as it is powered. This results in multiple methods being called unnecessarily and is handled by adding an IF statement in each method checking global variables if it has already been run once. To exemplify this, the method "Subscribe", which places a topic from its parameter to a subscriber list, should only be run once in the beginning of the script.

Doing changes to the language has its benefits. However, it has its drawbacks too. The complexity of modifying Commonlang is greater than making metamethods for it to use. Another essential point is that the grammar is made for handling metamethods specified in the robots' XML files. The domain specific language as it stands creates an environment for general purpose robot programming which could be compromised by adding grammar specifically for MQTT communication. Although it is a domain specific language, keeping it as generic as possible is a goal in of itself. For further development of the language, another communication protocol may be desirable. By refraining from doing grammatical adjustments, it becomes quite manageable to add or edit already existing metamethods for communication.

Considering the drawbacks as well as the group’s limited experience working with domain specific languages and the time schedule, a decision was made to integrate the communication through metamethods.

4.8 Metamethods

A crucial part of integrating communication through metamethods is making the code intuitive and user friendly, and at the same time not limiting what the end user is able to do. Various approaches to how the syntax and methods should end up like have been considered. Keeping the code simple has been one of the most important aspects in the consideration, in addition to what the current version of the language can do.

One of the approaches was to let the end user create a user method surrounding a selection of metamethods. This user method would then be connected to a message string through a metamethod called “OnMessage”. This is how it would look like:

```
OnMessage("go", forwardAndLightsOn);


void forwardAndLightsOn() {
        MoveForward();
        LightsOn();
}
```

This approach would result in more code to write. It would also require a change in the language’s grammar, allowing it to take a method as a parameter. After considering variations of similar syntax, the group settled on another approach. It utilizes the simplicity of metamethods combined with IF statements, which is already a part of the Commonlang grammar. The metamethod is called “ReceiveMessage” and takes a single string parameter, returning a boolean value. Here is an example:

```
if (ReceivedMessage(“go”)) {
        MoveForward();
        LightsOn();
    }
```

The method is checking whether of not the robot has received a specific message. When the robots receive messages, it triggers a callback called “on_message” that will put the messages in a list. This is because the callback method is hardcoded in the XMLs and cannot be dynamically changed. Instead this ReceivedMessage method was created to scan through the message list to check for the string from its parameter. If it finds it, a single instance of the message will be removed from the list and it will return true, false if not.

In addition to the ReceivedMessage, there are three other important metamethods that is implemented as part of the communication in Commonlang. These are called “StartConnection”, “Publish” and “Subscribe”. StartConnection is an method taking no parameters and has no return value. It has two important features. The first is to establish and maintain a connection with the broker, and the second is running a MQTT specific method that is called “client.loop”. The client.loop method needs to be continuously run as it is allowing the client to process the incoming messages. It was discussed whether the method should have parameters allowing the end user to choose, for instance the brokers IP address. To keep the Commonlang code as clean and simple as possible, it was concluded that this can rather be changed in the XML. StartConnection should be in the top of the main loop method of the program as a connection is required for the rest of the connection methods to be functional.

To be able to publish messages to a topic of one’s choice, the Publish method will take two parameters: the topic name and the message content.

The Subscribe method is taking a string parameter with the name of the topic one want to subscribe to. However, once the Subscribe method has been run, the MQTT method client.subscribe has not been run yet. Initially, the callback “on_connect”, that is called when the connection is established, would run the client.subscribe for a predefined/hardcoded topic. To be able to dynamically choose the topic via Commonlang, a list of topics was created. The Subscribe method will put the topic name into this list, and the on_connect callback will run client.subscribe on every element of the list. As it can take second to establish a connection to the broker, it is not necessary to write the Subscribe methods in the Commonlang before StartConnection, although it is advised.

5 EVALUATION

5.1 Evaluation method

The evaluation method used is both based on case studies and ex post artifact evaluation.

Ex post evaluation is performed after the creation of an artifact and as described by Yand and Padmanabhan (2005) is divided into two dimensions, real and abstract. Within these dimensions there is a division between automatic and human subjects. The real situations illustrate quantifiable data that can be evaluated, collected and calculated by observing actual use. Real situations also include the subjective opinions of users of a system or technology. The abstract setting describes historic data experiments and opinion analysis of historical data (Pries-Heje et al, Jan 2008).

As for the case studies, two were performed. This is far from adequate if the goal was to use this data to evaluate the quality of the product, but at least it gave us some pointers into where the extension could improve for future work. As our goals expanded during development, from just extending the language to support communication into also making it as easy to use as possible, the latter was tested to assure that the extension was somewhat intuitive and useable. The subjects of the study were also not ideal candidates as they are both writing a bachelor’s in software engineering. Something intuitive to a software engineer might not be as easy to access for a lay person not acquainted with programming in general.

The subjects were asked to make a simple script containing all metamethods used for communication and a single metamethod from the core with some logic. The commonlang robotscript would make the robot start a connection to the broker, tell another robot to move forward and check for an incoming message to move forward. They were given a simplified explanation of what the metamethods did as well as how the MQTT communications protocol worked.

The case study falls under two categories of evaluation methods as they are both qualitative if you think of them as subjective interviews and design science oriented as a human to computer category artefact.

As for ex post artifact evaluation of the project there are several interpretations of how to evaluate such artifacts as our expansion package to commonlang.

March and Smith (1995) divides design science into two, building and evaluating. Focusing on the performance of the artifact compared to the criteria set beforehand, asking whether the artifact worked or not and how and why it works. (Pries-Heje et al, Jan 2008)

Hevner et al (2004) Focuses on observational, analytical, experimental, testing and descriptive evaluation in a strict and frequent manner. There is however little guidance on how to choose among the methods for different kinds of artifacts. (Pries-Heje et al, Jan 2008)

Venable (2006) divides evaluation into two, artificial and naturalistic. Artificial being empirical or not, includes the theoretical and simulation evaluation whilst the naturalistic is meant to measure the performance of the artifact in a real environment with real people or systems. (Pries-Heje et al, Jan 2008)

These are just some of the approaches used to evaluate artifacts using a design science method. The three methods will be referenced whilst reviewing the evaluation methods used.

To evaluate the artefact / artefacts created in this project it would at first seem like a typical March and Smith approach was used. Some progress would be made and then evaluated through testing to see if it fulfilled the required specification. More specifically a metamethod would be created, a block of functionality bound to a domain specific language creating some code in a script running on a robot platform. Evaluation of this block would take place after creation, in steps working its way outwards until it ran successfully on the robot. The steps would be:

1. Checking whether a hardcoded version of the final step would run on the robot.
2. Writing a commonlang script including the metamethod made.
3. Making sure that the domain specific language would interpret the block written in the XML configuration file.
4. The intermediate generated java file could now be analytically inspected to make sure it contained the expected code.
5. Making sure that the java file would be interpreted and create the language specific final script.
6. Inspect the script and successfully be able to write it to the robot.
7. Testing the functionality of the metamethod on the actual running robots.
8. Reviewing the performance of the metamethod and possibly trying a different approach.

The evaluation as described above seems to also include parts of the Hevner et al ways of interpreting evaluation in design research. Through observation and analytical evaluation, we can determine whether or not criteria have been met for the different steps, like analysing code and observing the robots behaviour. There's also the programming aspect which in nature is experimental and is tested in various ways including performance testing and validating that the robot does what's expected.

In the above description an artifact could be interpreted as a metamethod, however an artifact also covers the whole of the language or project for that matter. Dividing the project into smaller parts will let us evaluate parts at a time, the project will also have to be evaluated.

Ex post evaluation of the project includes evaluating whether or not our criteria were fulfilled and that the performance of the product is decent, as well as determining how and why it works, similar to the March and Smith (1995) approach to evaluation. In addition, Venable (2006) describes how it's important to also test the performance in a real environment with humans or systems, where we will discuss the system part in more detail seeing that the human aspect is covered somewhat in the case study. (Pries-Heje et al, Jan 2008)

To determine whether the project fulfills the criteria set, testing of the DSL with metamethods generating scripts that run and successfully with communication validates that at least it works. Then adding to the commonlang script different ways of communicating over MQTT and giving it a bigger workload would test performance to a degree. This is where having a simulator to run extensive testing would be an invaluable tool. Simulation testing as described by Sun and Kantor (2006) can be an efficient way of testing a lot of variables quickly and would allow us to test behaviour without the restrictions of having to write the script to the robots everytime. This is not something we have been able to develop in the timespan of working with this bachelor. (Pries-Heje et al, Jan 2008)

Testing the system in a real environment is a requirement as described by Venable (2006), this has been done of both the core language and the communications extension of it by writing different kinds of scripts and running these on the robots to see how they perform. (Pries-Heje et al, Jan 2008)

As for how and why our software works there are multiple ways to go about evaluating it. March and Smith (1995) argues that "using natural science methods for theorizing about IT" is a way of evaluating how and why an artifact might work. This could be interpreted as going deep into the science of programming, trying to prove how the program does what it’s expected to do. Due to limited time and understanding in this area, another way of evaluating how and why our artifact works is through understanding of the mechanisms embedded in the domain specific language, configuration files, scripts and robots. This knowledge will help with understanding how and why activities contribute to the intended outcome of the artifact. (Pries-Heje et al, Jan 2008)

5.2 Evaluation results

Through the case studies we found that the test subjects could write the commonlang language with the communication extension scripts with relative ease. The subjects found the setup and syntax of the language familiar, and the use of metamethods similar to using a package. The scripts they made would run fine on the robots and fulfilled the requirements requested. They also found most of the naming of metamethods made sense. Only one problem arose, the metamethod “boolean ReceivedMessage(string msg)” returns true if the robot has received the message parameter. The subjects expected it to also contain a topic parameter as to know where the message came from. Although it might be a bit confusing when first starting off this is a conscious design choice and as so it’s been decided to keep it this way. If you would like to identify where the message came from you would have to use different messages depending on who sent it. The reason it’s been designed this way is to be able to react to a message independent of where it came from without having to check if you received that message from each of the robots.

Through design science ex post artifact evaluation, we continually found ways to fix and improve the individual metamethods through the methods mentioned under evaluation method. Through real world system testing we were able to confirm functionality and performance to a degree. With this testing improvements could be made until requirements were met. MQTT communication issues were usually quick to resolve as they usually did or did not work, however the previous metamethods made originally for the language were not as simple to troubleshoot. The reason for this was often that something would work to a degree but needed calibrating and adjusting of thresholds to be predictable. Things like turning a certain amount of degrees and calibrating the pixy camera so it would detect movement, but not when there were none proved to be issues we had to solve in various ways, both with hardware and software.

Lastly through building experience with how all parts of the domain specific language worked in conjunction with the robots, conclusions could be made to where we could make improvements in the configuration files and where improvements could be made in the future. Part of these discoveries were how semicolons are added to every line when generating a script for C code. This means all metamethods must adhere to these rules, which means bunching up code into one line because a semicolon is not allowed on the opening of any loop or if statement. Things like ending a metamethod with an if statement means you'll have a closing squirly bracket ending the method, in C semicolons after this bracket will result in the code not compiling. The previous bachelor group the makers of the language apparently noticed this and decided to instantiate an "int i" in this location to circumvent the problem. This is probably one of the things we would have liked to fix but seeing as we went with making a package for the language instead of changing the DSL to add communication we ended up doing the same botch. This is very bad practice, not recommended at all, and may have further complications further for anyone wishing to add functionality to the language.

After evaluation the result fulfills the requirements of the assignment, however there are still improvement potential both when talking about the DSL, metamethod performance and ease of access. The extension could also have been made as a module package making it even more convenient if a change in communication protocol is needed.

6 DISCUSSION

6.1 Consequences of chosen approaches

Two main approaches were chosen through the development of the project, which communication protocol to use and whether the DSL should be changed to support communication or make changes in the configuration files of the robots to make a package like addition to be used by the language.

The use of MQTT as the communication protocol brought with it both challenges and solutions. None of the group members had used MQTT before, so after finding out that HTTP would not be the optimal choice for the assignment some time was used to get up to speed with how to use MQTT. The protocol proved to be far less resource intensive which suited the project well as the robots provide limited performance. It was also relatively simple in design making it easy to learn. As MQTT uses a broker to convey messages the simplest solution was setting up a computer to do this task, this might not be the perfect solution for everyone as one might want the robots to talk directly to each other. Talking directly could be achieved by having one of the robots be the broker, but this is not something we developed functionality for as it could depend on the MQTT libraries used whether this is an option. MQTT worked well for the application as it satisfied the requirements of the project, however it could have been useful to check out more alternatives and done research into each alternative. As there was a deadline for the project and uncertainties about how long development in the further stages would take, a decision was made to go with MQTT in this instance.

Another important decision was whether to change the DSL itself, create a new DSL or make changes in the configuration files of the robots to develop support for communication. Some testing and creation of DSLs were done as research into how much work it would take to make our own or change the existing one. Making of metamethods was also performed and proved to be a lot faster and easier to implement than doing changes to the language. Another reason to make metamethods was not making the DSL to specific. The DSL should be domain specific for programming of robots, but not necessarily specific to which communications protocol is used. It cannot be expected that MQTT will be ideal for all communications cases and implementing it through metamethods will allow other developers to substitute the MQTT specific artifacts with their own protocol. This makes it more modular in nature which was a big decision factor. The trade-off to using this approach is that the final scripts written to the robots contain a lot of unused code specific to communications regardless of whether it's used or not. Making the communication specific to the DSL could solve this, as it would check if communication was used and included the necessary code. One could then potentially make larger scripts without reaching the memory limit of the robot, but the communications code does not take up a large enough percentage of this memory to make a sufficient difference. The larger problem is linked to readability of the scripts as the code clutters it up a bit.

6.2 Potential improvements if starting over

If we had started over, we would have chosen a similar approach. For instance, the time used testing HTTP as a possible protocol was valuable even though we decided not to use it because it pointed us in the direction of a more lightweight solution. The time researching DSLs gave us valuable insight into how the language works giving grounds for evaluation of the final product and a potential solution to the communication implementation. There is however always something to improve, starting over or provided more time we would like to have the required code for MQTT communication in a separate XML included by the DSL when generating the script. This would require changes to both the way the XML is set up and to the DSL itself. As it stands the DSL combines two XMLs to generate the script, the third one would be included if communication was used, and the content would be specific to the protocol used. This would further increase modularity and make it easier to add different kinds of protocols to be used.

7 CONCLUSIONS

7.1 Summary of goals

The main goal of the project was to further develop the existing domain specific language Commolang in such a way it can produce scripts for robots giving them the ability to communicate wirelessly. To reach the final goal, multiple intermediate goals had to be accomplished. After setting up an MQTT connection between PCs, the next goal of significance was to establish a connection between the PC and each of the robots, followed by communication between the two robots. When starting work on how to integrate the communication part into the Commonlang language, an additional goal became apparent: making the code user friendly and intuitive.

7.2 Confirmation of reached goals

As the robots are now sending and receiving data between each other and PCs when running scripts generated by commonlang, one can conclude that the main goal of the project has been reached.

Some case studies were done to verify success of the secondary goal as well. The case studies were biased by the fact that the test subjects were software engineering students and that a very limited sample size was used. The subjects found the programming intuitive and user friendly, but this does not conclude that it would be perceived this way by someone who is not familiar with programming in general. The metamethods used are however as descriptive as possible and explained clearly in the documentation increasing the likelihood of a secondary goal success.

7.3 Target users

Programming of robots with similar functionality can be time consuming even if the robots share traits. With commonlang one could do the time-consuming work of making the functions and setup once, then use these functions or “metamethods” to make the scripts. Making these general abstract functions will allow faster and easier programming later. As one could make any kind of metamethods the functionality can vary greatly from a simpler move forward type function to highly abstract functions like water field or defuse bomb. The defined functionality is contained within XML configuration files, these are made and included by commonlang. This means anyone could make these robot specific files beforehand and make documentation for them so that the commonlang scripts can be made by less technical people. After the conversion from commonlang to java, the java files are interpreted by parsers to a specific language to be run on the robot, current languages include Python and C.

The target users of the application are broad, the most prominent of which might be laypersons wanting to program robots without needing to know the inner workings of a specific language script. The language also caters well to professionals or hobbyists wanting to program similar robots that interpret different languages.

7.4 Further work

7.4.1 Simulation

Testing and calibrating the robots was time consuming, connecting via usb or ssh, transferring the script then running the script when the robot was in a semi suitable location was not ideal. A simulator could mitigate a lot of these problems by being able to quickly run scripts in an isolated environment without variables like battery voltage and colour interferences in the environment. There was not enough time to develop a simulator, but having this could cut down on testing time, as well as being a solid tool for evaluation.

“Simulation serves as an important tool for the development, programming, and testing of robots and their control software. We anticipate that in the coming years the significance of simulation will grow as it becomes an essential part of robotic design, debugging and evaluation processes.” (Drumwright E. 2010).

7.4.2 Domain specific language

The work done to add communications functionality to “commonlang” was implemented through configuration files and metamethods, the DSL itself has remained untouched. This proves that adding functionality to the language with a package like approach is possible, however it’s not necessarily always the best way to go about it. It would be a good idea to implement communication through the DSL itself to test different approaches and determine which one best fits the purpose. The DSL should also be fixed going further by setting up a logic-based semicolon system instead of just adding a semicolon to each line when parsing to “C” code.

7.4.3 Additional metamethods

The metamethods limit what the user of “commonlang” can do with the scripts. Lower level metamethods give great customizability, but also increases the overall difficulty.

Higher level metamethods are specific and somewhat limiting but offers a great amount of functionality compared to how much work and understanding is required by the “commonlang” user (Gya and Solheim, 2017).

Further work would include expanding upon the library available to offer greater flexibility in how the user would like to use the DSL, possibly dividing the metamethods into three different collections with solid documentation. The user could then choose the level of customizability depending on his/her competence.

7.4.4 Additional connection protocols

MQTT is a solid communications protocol, however it’s not ideal in every situation. Being able to choose the communications protocol depending on the project would further increase the usability of the language. Further work could focus on developing support for HTTP or Telnet. Some projects might want to use direct communication between the robots as well, this could be developed for all the different protocols.

8 LITERATURE/REFERENCES

Banzi, M., Cuartielles, D., Igoe, T., Martino, G. and Mellis, D. (2018) *Language Reference* [online]. Publisher: Arduino. Available at: <https://www.arduino.cc/reference/en/> [Accessed 15 February 2018].

Belshe, M., Peon, R. and Thomson, M. (2018). *Hypertext Transfer Protocol Version 2 (HTTP/2)*. [online] Http2.github.io. Available at: <https://http2.github.io/http2-spec/#TLSUsage> [Accessed 14 April 2018].

Burfoot. J. (2018). *EV3 Sensors* [Online]. Publisher: LegoEngineering. Available at: <http://www.legoengineering.com/ev3-sensors/> [Accessed 4 April 2018].

Cockburn, A. and Highsmith, J. (2001). *Agile software development, the people factor*. [Online], 133 pages. Available from: <https://ieeexplore.ieee.org/abstract/document/963450/> [Read 3.4.2018].

Ddemidov. (2017) *Pure python bindings for ev3dev* [online]. Publisher: ddemidov. Available at: <https://github.com/ev3dev/ev3dev-lang-python> [Accessed 20 April 2018].

Drumwright, E., Hsu, J., Koenig, N. and Shell, D. (2010) *Extending Open Dynamics Engine for Robotics Simulation*. In: Ando N., Balakirsky S., Hemker T., Reggiani M., von Stryk O. (eds) Simulation, Modeling, and Programming for Autonomous Robots. SIMPAR 2010. Lecture Notes in Computer Science, vol 6472. Springer, Berlin, Heidelberg

Eclipse Mosquitto. (2018). *Eclipse Mosquitto™ An open source MQTT broker* [online]. Publisher: Eclipse. Available at: <https://mosquitto.org/> [Accessed 16 April 2018].

Ev3dev. (2015) *ev3dev is your EV3 re-imagined* [online]. Publisher: ev3dev. Available at: <http://www.ev3dev.org/> [Accessed 14 April 2018].

Ev3dev. (2015) *Sending and Receiving Messages with MQTT* [online]. Publisher: ev3dev. Available at: <http://www.ev3dev.org/docs/tutorials/sending-and-receiving-messages-with-mqtt> [Accessed 11 April 2018].

Grusin, M. (2017) *Serial Peripheral Interface (SPI)* [online]. Publisher: Sparkfun. Available at: <https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi> [Accessed 16 March 2018].

Gya, M. and Solhaug, T. (2017) *Common programming interface for multiple types of robots* [Bachelor]. Bergen: Western Norway University of Applied Sciences.

Hempel, R. (2015) *Python language bindings for ev3dev* [online]. Publisher: ev3dev. Available at: <http://ev3dev-lang.readthedocs.io/projects/python-ev3dev/en/stable/index.html> [Accessed 15 March 2018].

LogMaker360. (2016) *MQTT tutorial on Raspberry pi, Arduino and Python* [online]. Publisher: logMaker360. Available at: <https://www.youtube.com/watch?v=nAUUdbUkJEI> [Accessed 6 April 2018].

Mazur, M. and Wiśniewski, A. (2016) *Clarity from above* [online]. PwC. Available at: <https://www.pwc.pl/pl/pdf/clarity-from-above-pwc.pdf> [Accessed 24 May 2018].

MQTT.org. (2014) *MQ Telemetry Transport* [online]. Publisher: MQTT.org. Available at: <http://www.mqtt.org/faq> [Accessed 11 April 2018].

O'Leary, N. (2018) *PubSubClient API Documentation* [online]. Publisher: Knolleary. Available at: <https://pubsubclient.knolleary.net/api.html> [Accessed 11 April 2018].e

Parallax (2018) *Testing the Left and Right Wheels* [Online]. Publisher: Parallax. Available at: <http://learn.parallax.com/tutorials/robot/shield-bot/robotics-board-education-shield-arduino/chapter-3-assemble-and-test-11> [Accessed 2 April 2018].

Postel, J. (1980) *TELNET PROTOCOL SPECIFICATION* [Online], 15 Pages. Available from <10.17487/RFC0854> [Read 16 March 2018].

Pries-Heje, J., Baskerville, R. and Venable, J. R. (2008), *Strategies for Design Science Research Evaluation* (2008). ECIS 2008 Proceedings. Paper 87. <http://aisel.aisnet.org/ecis2008/87>

Redding, S. (2018) papyrus-rt | The Eclipse Foundation. [online] Eclipse.org. Available at: <https://www.eclipse.org/papyrus-rt> [Accessed 27 May 2018].

ROS.org (2018) *About ROS* [online]. Publisher: ROS.org. Available at: <http://www.ros.org/about-ros/> [Accessed 27 May 2018].

Rowe, A., LeGrand, R. and Robinson, S. (2015) *CMUcam5 Pixy* [online]. Publisher: Cmucam. Available at: <http://www.cmucam.org/projects/cmucam5> [Accessed 6 April 2018].

Seesiva (2015) *Step by step installing and configuring Mosquitto with Windows 7* [online]. Publisher: Sivatech Wordpress. Available at: <https://sivatechworld.wordpress.com/2015/06/11/step-by-step-installing-and-configuring-mosquitto-with-windows-7/> [Accessed 6 April 2018].

Stackexchange (2018) *Arduino Uno R3 to Wemos D1 R2 project migration - Pinout problems* [online]. Publisher: Stackexchange. Available at: <https://arduino.stackexchange.com/questions/49119/arduino-uno-r3-to-wemos-d1-r2-project-migration-pinout-problems?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa> [Accessed 13 April 2018].

Vaishnavi, V., Kuechler, W., and Petter, S. (Eds.) (2004/17). “Design Science Research in Information Systems” January 20, 2004 (created in 2004 and updated until 2015 by Vaishnavi, V. and Kuechler, W.); last updated (by Vaishnavi, V. and Petter, S.), December 20, 2017. URL: http://www.desrist.org/design-research-in-information-systems/.

Xtext (2017) *Getting started with xtext* [online]. Publisher: Xtext. Available at: <https://www.eclipse.org/Xtext/documentation/102_domainmodelwalkthrough.html> [Accessed 16 April 2018].

Definition of ‘robot’. Oxford English Dictionary. Retrieved November 27, 2016. <https://en.oxforddictionaries.com/definition/robot>

Definition of ‘machine code’. Oxford English Dictionary. Retrieved May 31, 2018. <https://en.oxforddictionaries.com/definition/us/machine_code>

9 APPENDIX

9.1 Risk list

https://docs.google.com/spreadsheets/d/1YT-cN_Cw4uKlQOe8w3aba_ZZVFeMindY6wZHttiHZ18/edit?usp=sharing

9.2 GANTT diagram

https://docs.google.com/spreadsheets/d/15f0rXMSo2YiFxVAuAkELRsNFyAYDB7dcG8ZHUvQdinw/edit?usp=sharing

9.3 User manual

- Setting up eclipse with commonlang
- Setting up mqtt on the computer
- Setting up mqtt on the ev3
- Setting up mqtt on th arduino
- Setting up ssh / winSCP
- Setting up ArduinoIDE
- Keywords added for communication
- For other keywords see previous bachelor appendix

Setting up Eclipse with Commonlang

1. Download and install **Eclipse** (Standard edition **OXYGEN** was used).
2. Open Eclipse, go to **Help** > **Eclipse Marketplace** and search for **Xtext**, install both **Xtext** and **Xtend**, **Xtext** might also install **Xtend** (Eclipse **Xtext 2.11.0** and Eclipse **Xtend 2.11.0** was used).
3. Download the **Plugin**, **Library** and **Sample Config** from this link: **https://drive.google.com/open?id=18JqyPNs1jT2XoJKwgpU6v4hU-1XkgXqC** it contains a **plugins folder**, add these two files to your **Eclipse installation plugins folder**.
   Could look something like this: **C:\Users\user\eclipse\java-oxygen\eclipse\plugins**.
4. Restart **eclipse** to let the **plugins** take effect.
5. Go to **File** > **Import** > **Git** > **Projects from Git** > **Next** > **Clone URI** > **Next**. Paste in the **URI** of the commonlang project: **https://github.com/MagnarGya/CommonLanguageForRobots.git**

6. Finish the import wizard using standard settings.
7. With the code successfully imported (some errors will initially show, but wait for these to clear up), right click the **org.xtext.commonlang** folder **Run As** > **Eclipse Application**.
8. Eclipse will then open a new eclipse window. Create a new **java project**, right click on the project folder and add an additional **source folder** called **src-gen**.
9. From the OLTRTA zip downloaded earlier add OLTRTA.jar library to the projects build path by: **Right click** project folder > **Build Path** > **Add External Archives**.
10. Each robot will need its own package within the src folder: **Right click** src folder > **New** > **Package**. Place the provided sample configuration XML files into the packages with the same name as the folder they are provided in. There are two XML files for each robot.
11. You can now create a new commonlang file by: **Right clicking** the src folder > **New** > **File**, and adding the suffix **.commonlang** to the file.
12. For documentation on writing the core commonlang language see the user documentation in the D23 report of the 2017 bachelor called “Common programming interface for multiple types of robots”. For documentation on the communication addition to this language see the documentation provided below.

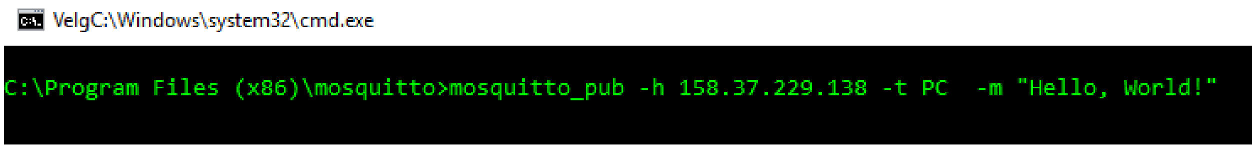
Setting up MQTT on a computer

1. Download mosquitto from the **Mosquitto.org** download section.
2. Some dependencies will have to be provided, **pThreads** and **OpenSSL**.
   ftp://sources.redhat.com/pub/pthreads-win32/dll-latest/dll/x86/
   http://slproweb.com/products/Win32OpenSSL.html
3. Download the **Win32 OpenSSL v1.0.2c Light** setup file and install it.
4. Get the **Pthreadvc2.dll**
5. Run the downloaded mosquitto file and go through the wizard. When prompted tick off services as well as files. Don’t delete the setup file.
6. When the install is done find the **libeay32.dll** and **ssleay32.dll** in the **OpenSSL-Win32** or **OpenSSL-Win32\Bin** folder. You’ll also need the **pthreadVC2.dll**. Put all these into the same folder as your mosquitto executable.
7. Reinstall mosquitto.
8. Run a **cmd**, type in **netstat -an** to make sure **TCP 0.0.0.0:1883 is LISTENING**
9. Check out: https://sivatechworld.wordpress.com/2015/06/11/step-by-step-installing-and-configuring-mosquitto-with-windows-7/ if it’s still up at the time of trying if you’re having issues.

10. If you want to start a mosquitto broker use the cmd, cd to the dir you installed mosquitto in and run the command **mosquitto**.
11. To subscribe do the same as over, but run the command **mosquitto_sub -h** <the ip of the broker> **-t** <the topic you wish to subscribe to>.
12. To publish do the same as over, but run the command **mosquitto_pub -h** <the ip of the broker> **-t** <the topic you wish to publish on> **-m** "<the message you wish to send>".

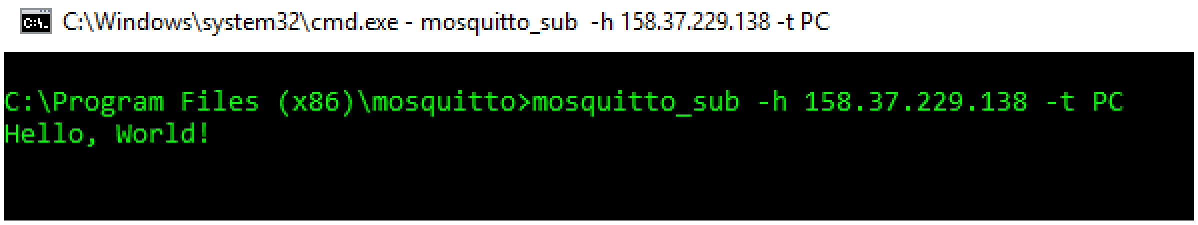
-h is a parameter for host followed by the host's IP address. **-t** stands for topic followed by the name of the topic, and lastly **-m** is for message followed by the message to be sent surrounded by quotation marks.

Example of entering the command for publishing a message:



```
C:\Program Files (x86)\mosquitto>mosquitto_pub -h 158.37.229.138 -t PC  -m "Hello, World!"
```

Example of entering the command for subscribing to a topic and receiving a message:


C:\Windows\system32\cmd.exe - mosquitto_sub -h 158.37.229.138 -t PC

```
C:\Program Files (x86)\mosquitto>mosquitto_sub -h 158.37.229.138 -t PC
Hello, World!
```

Setting up mqtt on the EV3

1. Get a robot running EV3dev, but does not yet have MQTT installed.
2. Run the command `sudo apt-get install mosquitto`
3. Run the command `sudo apt-get install python3-pip`
4. Run the command `sudo pip3 install paho-mqtt`
5. Run scripts with **python3**
6. The EV3 connects to a network before the script is run using a WiFi dongle.
7. Remember to check the setup XML to change the ip of the broker you wish to connect to.

8. For more information see: **http://www.ev3dev.org/docs/tutorials/sending-and-receiving-messages-with-mqtt/**

Setting up WinSCP with the EV3

1. Download and install **WinSCP**, get it at **winscp.net**
2. Run **WinSCP**
3. Be connected to the **same network** as the EV3
4. To connect with **default** setup: Host: **ev3dev** User: **robot** Password: **maker** Port: **22**
5. You can now drag and drop files from and to the EV3

Setting up SSH with the EV3 to run a script

1. Download and install **Putty**, get it at **www.putty.org**
2. Run **Putty**
3. Be connected to the **same network** as the EV3
4. To connect with **default** setup: Host: **ev3dev** Port: **22** click **Open**
5. A window will open: User: **robot** Password: **maker**
6. You can now use standard linux commands to navigate and run scripts
7. To run a script, make sure you are in the directory of the script, then **python3 <scriptname>.py**

Setting up mqtt on the Arduino/Wemos

1. The configuration for connecting to a network and mqtt broker is baked into the **XML** setup file of the Arduino.
2. In this file called **ArduinoShieldBot** you'll find the code that represents the setup in the final sketch, set the **ssid**, **password** and **mqtt_server** (ip of broker on the same network) to be something appropriate.

Reserved keywords for the communication part of Commonlang

1. **"Subscribe(string topic)"** typically called early in the loop to subscribe to topics. If you are also successfully connected to a broker you can now receive messages on this topic.
2. **"Publish(string topic, string message)"** called when you want to publish via the connected broker a message on a specific topic.

3. **“ReceivedMessage(string message)”** Checks if the parameter message has been received, independent of which topic it’s been received on.
4. **“StartConnection()”** does multiple things, it has to be run within the loop to initialize the connection to the broker, keep the connection open and reconnect if the connection is lost.
5. For documentation on writing the core commonlang language see the user documentation in the D23 report of the 2017 bachelor called “Common programming interface for multiple types of robots”.