NTNU
Norwegian University of
Science and Technology

# The NTNU Cyborg v2.0: The Presentable Cyborg

Jørgen Waløen

Master of Science in Cybernetics and Robotics
Submission date: June 2017
Supervisor: Sverre Hendseth, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

# Task Description

The Cyborg project is currently undergoing a process of integrating its modules and we are trying to avoid adding new functions to the Cyborg, but rather integrating and improving the ones that we have. The student shall, in prioritized order:

- Work towards making the robot ready for presentations and demonstrations before the end of the semester.
- Be a resource for the Experts in Team groups, and guide them on the topics the student worked on last fall.
- Implement the controller node on the Cyborg.
- Write a guide for the new students entering the project next semester.

NORGES TEKNISK-NATURVITENSKAPELIGE UNIVERSITET

TTK4900 ENGINEERING CYBERNETICS, MASTER'S THESIS

# The NTNU Cyborg v2.0: The Presentable Cyborg

MASTER'S THESIS

*Author:*
Jørgen WALØEN

*Supervisor:*
Ass. Prof. Sverre HENDSETH
*Assisting Supervisor:*
PhD Cand. Martinius KNUDSEN

June 2017

NTNU
Norwegian University of
Science and Technology

# Sammendrag

Denne oppgaven fokuserer på å gjøre roboten NTNU Cyborg klar for presentasjoner. Jeg presenterer et begrep, **Den Presentable Cyborg**, som er tilstanden der Cyborgen er klar for presentasjoner. Jeg presenterer også en annen tilstand, **Den Presentable Robot**, som er den samme tilstanden bare uten signaler fra de biologiske cellene. For begge disse tilstandene bestemmer jeg hva kravene er, og prøver å nå dem. Sluttresultatet er at vi når Den Presentable Robot, men uten integrert ROS kommunikasjon mellom datamaskinene på Cyborgen. Den Presentable Roboten kan navigere i kjente omgivelser og bevege seg etter egen vilje, har en fungerende, implementert kontrollernode, leverer strøm til all *hardware* fra robot basens batteri, har en fungerende nettverksløsning og er enkel å starte på presentasjoner.

Jeg presenterer også en ny prosjektstruktur som jeg kaller "prosjektets kjerne". Jeg argumenterer for at vi burde fokusere på å ha en stabil og pålitelig kjerne i prisjekter, som vil gjøre det lettere for studenter å fokusere mer på deres individuelle oppgaver som hører til utenfor kjernen. Ved enden av semesteret er prosjektets kjerne stabilt.

For å gjøre roboten klar for presentasjoner så har jeg veiledet og assistert EiT gruppene, påvirket hardware designet på Cyborgen, lagd en start boks og start program, funnet og implementert en nettverksløsning for Cyborgen og dens datamaskiner og implementer controllernoden lagd av en tidligere student. Den ferdige løsningen er testet og har fungert bra.

Gjennom arbeidet mitt har jeg gjort valg som vil være til fortjeneste for kommende studenter ved å gjøre Cyborgen lettere å jobbe med. Disse valgene inkluderer å tillate lettere debugging og å skrive en guide til Cyborgen. Sistnevnte tror jeg vil gjøre stor forskjell for de kommende studentene til prosjektet.

# Abstract

The focus of this thesis is to make the NTNU Cyborg robot ready for presentations. I present a term, **The Presentable Cyborg**, which is the state in which the Cyborg is ready for presentations. I also present another state, **The Presentable Robot**, which is the same state but without the signals from biological neural cells. For both these states I decide what the requirements are, and attempt to reach them. The end result is that we reach The Presentable Robot, but without completed ROS communication between the computers on the Cyborg. The Presentable Robot is capable of navigating a known environment and move around of it's own free will, has a working, implemented controller node, powers all the hardware from the robot base' battery, has a working network solution and is simple to start for presentations.

I also present a new project structure, **the project's core**. I argue that we should have a stable and reliable core in the project to let the participating students focus more on their individual tasks that belong outside the core. By the end of the semester the project's core is in a stable state.

To make the robot ready for presentations I have guided and assisted the Experts in Team groups, influenced the hardware architecture of the Cyborg, created the Startup Box and Startup Script, found and implemented a network solution for the Cyborg and it's embedded computers and implemented the controller node made by an earlier student on the Cyborg. The complete solution has been tested and shown to work well.

In my work I have made choices that will benefit future students in making the Cyborg easier to work on, including making a network solution that allow easy debugging and writing a guide to the Cyborg. The latter of which is I believe will have a great impact in the upcoming semester.

# Preface

In the last year I have put a lot of time and effort into this project. I have loved working on it and I find the work I have done very satisfying. It has, however, taken a bit more of my spare time than I probably should have let it. I love finding good solutions to problems, and I love to tinker with Linux, computers and electronics, which makes this project perfect for me. It is my enjoyment of solving problems and reaching goals that has driven me in this project.

I am very grateful for the privilege of working with things I love to do, and be able to make a career out of it. My time at NTNU has surely been a roller coaster ride, and I believe many of my fellow students would say the same.

I want to thank everyone who has worked on the Cyborg project, you have all made this project more enjoyable. I also want to thank my supervisor Sverre Hendseth for his guidance in the last year, and Martinious Knudsen who where the assisting supervisor. Of course I cannot give thanks without including my family, who has supported me no matter what.

Now a new chapter in my life begins, but I will always remember my time at NTNU, it has truly been unforgettable.

Jørgen Waløen, June 2017

> “Live as if you were to die tomorrow.
> Learn as if you were to live forever.” - Mahatma Gandhi

# Contents

# List of Figures

# Chapter 1

# Introduction

This semester I have continued my work on the NTNU Cyborg. Just like my specialization project, this thesis has focused on making an integrated solution. I present my suggestion for a Cyborg ready for presentations, which I name **The Presentable Cyborg**, and introduce a new concept to the project structure I call **the project's core**. I have been a resource for the Experts in Team groups and guided them in their process. I have made sure the project's participants have been working towards the same goal.

In addition to this I have also worked on finding and implementing good solutions for a network solution and start-up procedures, and also implemented the controller node created by a previous student. Towards the end of the semester I decided to write a guide for new students to the project, the reasoning behind this is that I think it would be a waste of time for them to do all the research I have done all over. It also gave me a chance to demonstrate the extent of my work and research in a more structured way. I decided to include the guide in the back of the Appendix in my thesis, as I want it evaluated as part of my work. I have also included the code I have written in the attachment file. I deliberately chose not to include and discuss all of that code in my thesis, as I find it a bit redundant. The results are presented in my thesis, and the code is included for those who wants to know how the sausage is made.

The reason why the guide was needed, and the project's core needed to be stable, was clear to me as soon as I first started working on the Cyborg project almost a year ago. The Project was difficult to enter as there was so many things to read up on. Project structure was lacking, and it seemed to me that the earlier students had worked very much individually and not found good, lasting solutions that integrated well with each other. This is why a lot of the hardware was changed in the last year and why the controller node was created. I believed then, and still believe, that the biggest challenge the project faced at that point was the lack of a good, reliable foundation that all smaller parts of the project could be built around. That is why I set out to solve this issue. This foundation is what I have named the project's core. Thus the project's core is a central part of The Presentable Cyborg.

Unfortunately the API for the MEA signals was not completed this semester, and we did not reach the goal of making a Cyborg. I discuss this in more detail in Chapter 4

The reason why I chose the name **The Cyborg v2.0** was because Nævra[14] in his

report used the term **The Cyborg 1.0**. Since then the Cyborg has had major changes to it's architecture, and I decided it was time to name the new and better Cyborg with an appropriate name.

## 1.1 Work Distribution

Because I have worked on so many different things this semester and because it has been a lot of practical work, there is no clear boundary between the different stages, and there is no reason for me to present a Gantt chart. I will, however, say that there is a lot of work and evaluation behind my solutions. I believe I have, with the help of other students of course, solved one major challenge that has haunted the project. As I hope the reader will recognize, the complete solution has come from many deliberate, and though through, decisions. I hope that my guide will help give a better perspective of the work and research behind my decisions, and help show just how many nuts and bolts I have been working with in this project. I want to save students in this project from many of these nuts and bolts by having them working more outside the core, so they have less to worry about. What I can say about my work distribution is that research, practical work and writing has all stretched through the whole semester. I have always had The Presentable Cyborg as the goal to reach, my vision for this semester.

## 1.2 Work not Mentioned Further

Some work did not make the cut when I wrote this thesis, so I will list them here:

- I contacted a group of students that where in the same Experts in Team village as me a year ago. They made a program that would create music by attempting to evaluate the mood of images. I though the results where very interesting and though it could be an interesting way to use the MEA signals. I looked through their code and report, but there where two things that kept me from pursuing this further: The code was written in Java and the MEA signal API was never finished.
- I have written about my problems with the MobileEyes software to some degree in my thesis. I spent days trying to make it work because I had to try different operating systems, and that required me to install them all. If it were not for Andersen[3] having used MobileEyes on Linux and made it work, I probably would have tried the Windows option sooner. There were other issues through the project which made me have to install operating systems, including the fact that I suddenly needed a Windows laptop to debug with (which I did not have) and at least one issue on the robot base. Point being: Time was lost on installing operating systems.
- I made a fully functional prototype of the Start Box.
- I attended the Experts in Team presentation.
- I attended the Cyborg Project's meetings throughout the semester.
- I spent quite some time looking at the controller node and ending up not changing it, and I also spent a significant time simulating it to learn about it and test

it. After all I had to know it would satisfy my requirements for The Presentable Cyborg.

- A lot of time went into researching and evaluation network options. I discussed several in my specialization project, but I did even more work on it this semester. The reason why I did not include them was that it I could not find any interesting ways to present them. It would just be a lot of rambling and diagrams. They did not satisfy my requirements anyway, and will not be of any use to anyone in the future, so I decided to not include them.

# Chapter 2

# Background

## 2.1 The NTNU Cyborg

The NTNU cyborg consists of two main projects that will become one in the future:

1. The social robot.
2. The biological nerve cells.

My thesis focuses on the social robot part. The Cyborg project is an ambitious attempt and engages many students from many disciplines[1].

### 2.1.1 What is a Cyborg

The term cyborg was coined by Manfred E. Clynes and Nathan S. Kline in 1960[8]. The term is short for **cyb**ernetic **org**anism, and is used for beings with at least an organic part and a robotic part.

### 2.1.2 Our Goal

The goal for the *robot* side of the project, as stated on the website[19][15]:

> "Goal: Develop an social interactive robot to wander the campus hallways. The robot will eventually become a cyborg through interaction with biological nerve cells."

For a more detailed list of goals for the robot see Chapter 4.
The goal for the *cyborg* project as a whole, as stated on the website[1]:

> "In this project, the aim is to enable communication between living nerve tissue and a robot. The social and interactive Cyborg serves as a platform for studying neural signaling properties, robotics and hybrid bio-robotic machines. The project aims to bring NTNU to the forefront of international research within these areas, while creating a platform for interdisciplinary collaborations and teaching."

My goal this semester will be to make sure the *social robot* is ready for presentations.

The Cyborg project wants to connect the Cyborg robot to biological neural cells which resides at St. Olavs Hospital. To read their signals and to give them stimuli they use MEAs (see Section 2.3). We want to use these signals to affect the decisions of the robot, thus creating a cyborg.

## 2.2 Introducing Software and Tools

### 2.2.1 Pioneer LX Robot Base: Hardware

The Pioneer LX[16] have some useful sensors, including the SICK S-300 laser range sensor, bumper sensors in front and front and rear sonar sensors. Its embedded computer is capable of running Linux, ROS and all basic modules. Its computational power in the area of parallel processing (GPU) is low. The included SDK is capable of using the laser range sensor to map some of the obstacles and rooms, but because its mounted low it will not be able to map obstacles above a certain height. We also have a mounted skeleton on top of the Pioneer, and this makes navigating obstacles above ground even more challenging.

Figure 2.1: The Pioneer LX robot base.

#### SICK S-300

SICK S-300 is the laser rangefinder module on the Pioneer LX. It has 240° field of view, 30 meter maximum range and ½° angular resolution. This may appear as a low resolution for things further away, but when you move the robot to map the surroundings it works well. When creating a map of an area the Mapper3 software will smooth out the area and create a high resolution map for you. One major issue with the SICK S-300 is that it cannot see obstacles above the robot base. Therefor we need to be careful with tables and other similar obstacles.

#### The Pioneer's Kill Switch

The **kill switch** is a big red button that has to be raised before the motors will receive any power. If the robot ever acts unexpectedly, or is a danger for people or itself, we can hit this big red button to stop it. If needed, hardware can be connected to a special power connection on the battery which will stop delivering power when the switch is down.

### 2.2.2 Pioneer LX Robot Base: Software

The Pioneer LX and it's software is a reliable and advanced solution to a very complicated problem, which allows us to focus on the other things we want to develop.

**Pioneer LX Software Development Kit**

The Pioneer LX Software Development Kit (SDK) consists of several useful tools. The most relevant tools for my project are ARIA, ARNL, Mapper3 and MobileEyes. Some are free, ARIA is open source, and some must be purchased. They are available at the Mobile Robots website[2] along with information about them.

**ARIA**

This Open Source C++ library is designed to easily give access to networking, accessories and the controls of the robot. Also usable from Python, Java and Matlab. A ROS interface, **ROSARIA**, is available for ARIA. This is the key software for accessing the robot base's internal peripherals.

**ARNL**

ARNL is the intelligent localization and navigation software for the Pioneer LX. Utilizing the laser rangefinder it can be used to navigate and position the robot in a known map, and includes path-finding. A ROS interface, **ros-arnl**, is available for ARNL. ARNL makes the complicated task of navigation and positioning into a simple task.

**Mapper3**

Mapper3 is a software used for generating maps from laser scans, and adding obstacles and defining areas on the map. Mapper3 will take a laser scan log file (a .2d file) from the **sickLogger** program and create a .map file. It will also remove moving obstacles like peoples feet. It will leave you with a nice continuous outline of the area it maps. You can even modify the map on a robot over a network connection.

**sickLogger**

sickLogger is a program included with ARNL. It will log the data from the laser range finder, and store it in a `.2d` file.

**MobileEyes**

MobileEyes makes the Pioneer's sensor inputs and motion available, visualized in real-time. This program can be run from a host computer by connecting to the robot over a network. The programs main purpose is to visualize the robots vision, in our case this means the position in a map and unknown obstacles like feet. The program can also be used to create maps and to control the robot.

When you run the software you are asked to log in with the user name and password of the robot's OS and to give the IP of the robot.

**MobileSim**

MobileSim is used to simulate the sensor information given in a known environment. When you start MobileSim you select what robot model you are using and what map you want to simulate in. Once MobileSim is running you can start your Arnl server which will read the sensors as if you where in the actual real environment.

### 2.2.3 ROS: Robot Operating System

The following summary from Wikipedia[17] says it better than I could:

> "Robot Operating System (ROS) is a collection of software frameworks for robot software development, providing operating system-like functionality on a heterogeneous computer cluster. ROS provides standard operating system services such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management."

#### Modular Design

One of the main focus points of ROS is to make your robot modular. You begin your project with a core, aptly named the **ROS core**. Then you add the packages you need, either by designing them yourself or by using some of the more than 3000 packages designed by others and shared with the public[10]. The modular nature of ROS allows you to choose what modules gets to run, and when.

ROS has a significant community behind it, and the users are coming both from the public and, more and more in later years, the commercial sector.

#### Communications Infrastructure

Because of ROS' cluster support, communication between hardware is imperative. On the Cyborg this communication happens on the Cyborgs subnet, deployed on the Cyborg's router.

### 2.2.4 Arduino

Arduino[5] is a company that produces open source hardware and software. I used a cheap Arduino board to create the start box, but wrote the code in C in stead of using the Arduino IDE. The EiT group used a NodeMCU board, which has an ESP8266 chip, to control the LED cube. This was programmed with the Arduino IDE.

### 2.2.5 Avrdude

To upload the code to my start box I used Avrdude[7]. Avrdude is available as a package for Ubuntu, and I recommend using **Synaptic Package Manager** to install it. Part of the idea is to be able to update and flash the start box from the robot base using only a terminal.

### 2.2.6 Serial Communication and SPI

To communicate between the start box and the robot base I used the serial communication UART over USB on the Arduino. To communicate with the OLED screen I used SPI. Both of these protocols are implemented on the start box. The UART communication can be accessed from one of the /dev/ttyUSB connections. To see all serial USB connection use the command `ls /dev/ttyUSB*`.

The Raspberry Pi also uses serial communication to communicate with the ESP8266 based NodeMCU.

### 2.2.7 Linksys WRT54GL

The Linksys WRT54GL[12] router is a small and cheap router. It lacks many of the advanced and often expensive functions of new routers. It is a updated model of a 14 year old router. The included firmware lacks some of the configuration options we want.

### 2.2.8 OpenWRT

From the OpenWRT website[21]:

> "OpenWrt is described as a Linux distribution for embedded devices.
>
> Instead of trying to create a single, static firmware, OpenWrt provides a fully writable filesystem with package management. This frees you from the application selection and configuration provided by the vendor and allows you to customize the device through the use of packages to suit any application. For developer, OpenWrt is the framework to build an application without having to build a complete firmware around it; for users this means the ability for full customization, to use the device in ways never envisioned."

An OpenWRT router can be accessed over SSH or through the web interface LuCI. Not all routers can run OpenWRT, see the hardware guide on their website for specific hardware.

### 2.2.9 Ubuntu and Xubuntu

Ubuntu is a popular Debian based open source Linux operating system, and is widely used in development applications. We are aiming at making all modules Linux compatible because Linux is reliable, efficient and most importantly open to be customized. Xubuntu is Ubuntu with the desktop environment XFCE which has a smaller footprint than the default Unity. In my specialization project I explained why I use Xubuntu.

### 2.2.10 Htop

Htop is an advanced process viewer for Linux. You can monitor and terminate processes from a terminal, which makes it very useful for debugging over ssh.

### 2.2.11 Slack, Box and Github

To communicate and share files between project participants we use the services Slack, Github and Box. Slack is aimed at discussion and communication, you can discuss in groups or send direct messages and share documents and images. Box is a file sharing cloud service where participants can synchronize with the shared folders and files. All files relevant to the project can be found in the projects Box folder or on Github.

## 2.3 The MEA Signals

Multielectrode arrays are used to collect neural signals from the biological cells. The MEA signals are the gathered data which will be sent to the Cyborg from the lab.

## 2.4 Controller node

The controller node is a ROS module that acts as the state machine for ROS modules that can interfere with each other and that need concrete boundaries. These modules require control of the Cyborg in some way. The first version of the controller node was completed by Andersen[3] in February of 2017. See Chapter 9 for more information.

## 2.5 Our Long Term Vision

The long term vision for the cyborg is to create a social and interactive cyborg that uses biological nerves for some task. As of mid April, the list of critical projects for the Cyborg from the website[18] looks like this (has since been changed):

1. Main coordinator node
   (a) Develop a coordinator module that coordinates the activity of all other modules. E.g., in the form of a state machine.
   (b) Implement efficient booting of the whole system (hardware and software) using Linux scripts ROS launch mechanism
   (c) Coordinate/handle the robot navigation module
2. Robot vision
   (a) Integrate new ZED stereo 3D camera and Nvidia Jetson TX1 card
   (b) Follower functionality:
      i. Continue development of one of the robots core functions: enable the cyborg to decide upon who it should seek out, walk up to and start interaction with.
      ii. This module should work with map coordinates
   (c) Purchase/build mount and stabilizer for the camera (possible aid from EiT)
   (d) Other possible work:
      i. 3D modelling
      ii. Develop skeleton tracking for ZED
3. Integration of biological neurons
   (a) Work with PhD student at IDI who is setting up 2-way communication between biological neurons and robot.
   (b) Decide on what the neurons should control on the Cyborg (e.g. mood or LED head with stimulator box)
   (c) Make a ROS node for handling the neural data (receive and send)
4. Power box
   (a) Voltage regulator for regulating Pioneer LX battery unregulated power supply

(b) Easy connection to the Pioneer LX regulated power

(c) Install USB hub if necessary

5. Wifi control

   (a) Set up hardware and software so that the cyborg can switch between wifi zones without losing connection.

   (b) Solve dynamic IP problem

6. Server

   (a) Build and integrate a Linux/ROS server that can handle heavy processing.

   (b) Should handle: Face animation, control of Facebook node, neural network simulation and other processing that is beneficial to put on a server.

   (c) IP handling of Pioneer LX and Nvidia Jetson

7. Trollface

   (a) Continue integration of the Troll face with speech abilities Implement an embedded screen, microphone and speaker

8. Speech

   (a) Continue development on the cyborgs text-to-speech and speech-to-text abilities

9. LED head and stimulation panel

   (a) Design and implement a LED head which is to be driven by the activity of biological neurons

   (b) Make a button panel which enables the user to send electrical stimulation to the biological networks

10. Robot body design

    (a) Design and build an aesthetic body for the Cyborg

11. NFC and QR

    (a) Implement an NFC module and QR codes so that people can be directed to the Cyborg web page using their cell phones.

12. Screen and keyboard for debugging

    (a) Mount an embedded screen and keyboard behind the robots back-panel for easy controlling/debugging of the Pioneer LX Linux machine.

    (b) The screen should preferably also have a switch that enables controlling the Nvidia Jetson Linux as well

13. Data analysis on biological neural data

(a) Analyze the recordings we get from the cultured neural networks

(b) Possible methods:

   i. Multivariate data analysis: PCA, PLS, PLSR, ICA etc. (TTK19)

   ii. Graph theory, network theory, clustering analysis etc.

   iii. Adaptive data analysis (TTK7) etc.

(c) Possibility to write a scientific paper

These tasks include the things we look at this semester, but also future tasks. The list on our website also include some less critical tasks that I will not discuss in my paper. Because this is a long and difficult list, I will discuss my vision for the presentable Cyborg in Section 3.2.

# Chapter 3

# Specifications for the Presentable Cyborg

## 3.1 Introduction to the Specifications

The process of deciding the end goal for the semester was very much a collaboration between the participants in the project. Early in the semester the decision was made to change the direction of the project quite a bit. We would have a greater focus on making an actual cyborg, and not just a robot. Certain parts of the robot was therefore removed, but may be included again in the future. The most significant parts that we removed where:

- The arms
- The iris
- The gatekeeper

The hardware has been influenced by the work of many students by now. In my specialization project this fall I made several changes to the hardware, although not all of the planned changes was implemented. The EiT groups brought one very interesting idea to the table this semester; the idea to represent the MEA signals using LED-strips. This will be our first major step in making the robot into a cyborg.

In this chapter I define the terms The Presentable Cyborg and the project's core. Many of the goals for the Cyborg project as a whole was been established before I entered the project. We, the students, has had a big impact on the development of the robot part of the project. I cannot take credit for all the ideas in the specifications. The specifications for The Presentable Cyborg and the project's core are within the constraints of the project's goals presented in Section 2.5

## 3.2 Vision for the Semester: The Presentable Cyborg

I was hoping to have *a presentable cyborg by the end of the semester*. Think of **The Presentable Cyborg** as the Cyborg in the state where it can be presented on demonstra-

tions. It should make decisions independently of any human. To reach this goal it was first necessary for me to define what the *critical requirements* are.

1. The Cyborg needs a working network connection to its biological cells, using a working API. This is critical if we want to call it a cyborg.
2. The nerve signals must be used for some action or decision making, or else it is not a cyborg.
3. The controller node must be working and implemented on the Cyborg. This is necessary because two modules cannot control the Cyborg at the same time.
4. The Cyborg must be able to navigate in it's environment. We expect the Cyborg to move around of its own will.
5. The Cyborg's hardware must be able to use ROS to communicate over a network. The Cyborg will consist of several embedded computers. All of these will need to communicate with each other, using ROS, over a network.
6. The Cyborg's hardware must be powered by a working voltage regulator connected to the base's battery. These embedded computers and their peripherals need to be powered by the battery.
7. The Cyborg must be easy to bring to demonstrations and it must be easy to start the correct demonstration.

If the API is not ready for the demonstration, there will not be any biologic component of the Cyborg, and it is not a cyborg but a robot. Therefor I will also use the term **The Presentable Robot**. This is the same state as The Presentable Cyborg, but without critical requirement 1 and 2.

We have also identified the requirements we should aim for as **desired goals**:

- Be able to identify humans using the ZED stereo camera. Amund will be working on this task this semester.
- Implement a low level voice communication.
- Have LEDs that visualize the MEA signals. The EiT groups will be working on this task this semester.

Having LEDs to visualize the MEA signals was only one suggestion to use the signals, that is why it is not classified as critical. The Cyborg could use the signals to navigate or change emotional state, and still satisfy critical requirement number 2.

The task of making an API for the MEA signals has been assigned to a PhD candidate on IDI. However, as he has reminded us several times, he is very busy with other projects. He requested help on finishing the API in project meetings, but no one was able to spare time to help him. Mostly because it is a large and difficult task that could be a specialization project or even a thesis in of itself. When I learned with certainty that the API would not be finished in time, I adjusted my work accordingly. We could still have a Presentable Robot by the end of the semester.

## 3.3 The Project's Core

Another term I coined is **The Cyborg project's core**, or simple the project's core. I will use this term for the collection of tasks that require a high degree of integration with each other. If the project's core is finished, then students can work on their individual tasks without worrying about how it will integrate with other tasks. The prime example of a core task is of course the controller node. Once the controller node is implemented and working, the students can add states without having to worry about other states, other than when entering and exiting the state they are working on. Another example would be the Cyborg's subnet, which makes it trivial to communicate between the embedded computers over ROS. The current parts that I consider to belong in the core category is:

1. Controller node on the Cyborg (state machine).
2. Navigation capabilities.
3. The Cyborg subnet.
4. A working ROS environment on the Cyborg.
5. A working voltage regulator that can deliver power to all components.
6. Having the proper hardware on the Cyborg for solving the current tasks.

This list is very similar to my critical requirements for The Presentable Cyborg for obvious reasons; I want the core to be complete as soon as possible as it will simplify all further work on the Cyborg to have a stable core. The last point requires the core to evolve along with tasks outside the core, but the core will be a much more rigid structure than most tasks.

## 3.4 Conclusion

The process of reaching The Presentable Robot is discussed in more detail in Chapter 7, 8 and 9. At the end of the semester the project's core has reached a stable state. This, combined with the guide, should make further work on the Cyborg an easier task.

# Chapter 4

# A Project Overview

## 4.1 Dependencies between Parts of the Project

Since we had several parts of the project that was dependent on other parts, and sometimes other students, I thought it would be useful to make a *dependency diagram*. See Figure 4.1. Usually these diagrams are used in computer science to show dependencies between modules. I realize it may be uncommon to use in the way I have used it, but it fits perfectly for the task. This way everyone in the project knows which part to prioritize, and who is dependent on their work. It also demonstrates how important each part is. In the diagram an arrow from A to B means A is dependent on B. By A dependent on B I mean that A will not be functional unless B is completed to satisfaction.

The red boxes are critical components to the demonstration, while the blue are highly desired and the white are bonuses. I also instructed the other students to report to me if they could not finish any of the critical components in time, because then I would have to try to finish it for them. If we only finished the red tasks we could still have a Presentable Cyborg. I presented this diagram to the other students early in the semester, and the only change I made was to make navigation and movement red from blue.

As mentioned in Section 3.2 the MEA signals was the biggest uncertainty for us, but it was critical for us if we want a *cyborg*.

## 4.2 Conclusion

In Chapter 13 I will discuss the end results for the semester in more detail. It is important to know how critical each part of the project is if there is ever a need to prioritize certain parts.

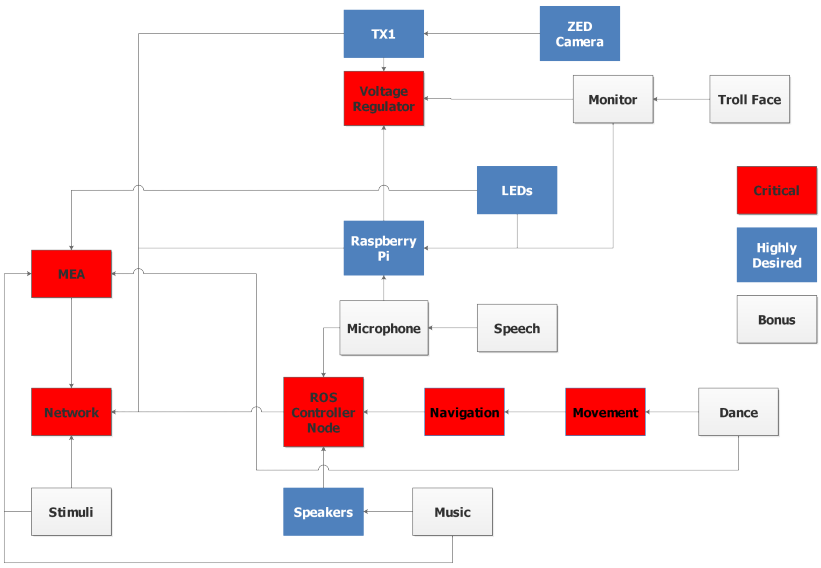


Figure 4.1: The project dependency diagram show what parts of the project are dependent on others.

# Chapter 5

# Assisting the Experts in Team Groups

## 5.1 Introduction

I find it difficult to write about my contributions to the EiT groups since they are often subtle. On some topics I may have contributed more than I think I have, but the opposite may also be true. I want to start off by assuring the reader that the majority of their work has been done independently. At the same time, I have made myself available both on Slack and in real life. I have visited the groups on their workdays, made sure the progress is on schedule and engaged in their issues. Some times they have had questions for me, other times I am the one with the questions.

Since my assignment is to make sure the cyborg is ready for its demonstration, it was necessary for me to keep track of the progress for the most important designs. In some cases this was simply to drop in on a group and have a talk about their progress and offer my help. Other times this would be to assist in more important choices and even to contribute in a more practical way.

The EiT groups has done a phenomenal contribution on the hardware side of the project, and without them we would not have gotten as far as we did. I believe the EiT groups can be a great resource, but can also cause the projects to diverge. In total there where 10 EiT students, which means a potentially large contribution. I personally believe, and in accordance with talking to other students who has worked on the project, that this contribution can potentially have some negative impact if it does not have a proper direction. I can also use earlier EiT work as an example. The arms, for example, where an interesting contribution, and the students worked hard on them, but I believe they where the wrong contribution at the time they were made, simply because there was so many important things missing at the core of the project. At the start of the semester I had a meeting with one of the EiT groups where they said they have been told by their EiT professor that they should work with what they wanted to work with, and not just be bossed around. At that point I had worked on the project long enough to know what would happen if the EiT groups didn't have constraints or a direction that aligned with the rest of the project. So in this meeting I had to convince them to work

within the constraints of the project and to include some of my plans in their work. The result were that they made the voltage regulator, bought a raspberry pi, a microphone, a case for the TX1 and attempted to implement a monitor, all of which was a continuation of my research. They also contributed in other areas, and also had their own idea implemented; the LEDs. The LEDs turned out great and was a very creative idea. The reason why this implementation of a new idea was a positive was because it aligned itself with the goal of making a presentable Cyborg by using the nerve signals in a very creative way. Thus, they were able to create things they wanted and also help with some already planned tasks. More on their contribution can be found in their EiT reports[4][13].

In this chapter I will try to explain some of my contributions and to give a realistic impression of my level of contribution.

## 5.2 Realizing my Plans from Last Semester

In my specialization project I attempted to tie together many loose ends of the project, but unfortunately not all my ideas where realized. There was simply too much to do. I did, however, do a lot of research and preparation for solutions. Some of these solutions has now been realized by the EiT groups. These are the following things they have been able to finish that I was not:

- Buy a case for the TX1 and mount it, and the TX1, on the Cyborg.
- Build a voltage regulator that can power all the different computers on the cyborg from its battery.
- Integrate a Raspberry pi for different tasks, possibly receiving the troll face stream.
- Buy and install a microphone.

In addition to this they also came up with the idea of using LEDs to visualize the nerve signals, and then implemented the LED solution, including an LED ROS module. In the following sections I will present some of my influences to the EiT groups' work.

## 5.3 The LED Controller Micro Controller

On Wednesday the 29th of March I was checking in on the group working on the LEDs, and they where having some issues. They where using the Arduino Nano to drive the 240 LED cube, and they where able to identify one major problem; a too small internal memory on the Arduino Nano. They where also having trouble with the speed of the serial communication. I was able to recommend they switch to a faster Arduino-compatible MCU with more RAM, thus effectively removing their memory problems. I knew it would be very simple to switch to another Arduino compatible board. All they would need to do is to update the Arduino IDE to be compatible with the new board, and define the correct pin in their code. No major changes is needed. They where unsure, however, how they would acquire new Arduino boards in time, as the shipping can be quite slow. Luckily I have used several of these boards for private projects, and had a few laying around at home. I gave them three options to choose between, and

on Wednesday the 5th of April they connected a NodeMCU board, which is powered by an ESP8266, to their LED design. Their code worked straight away thanks to Arduino's great compatibility between boards.

## 5.4 Voltage Regulator

The students working on the Voltage regulator has done so mostly without any help from me this semester. Their design is, however, a continuation of my design. The students did check with me what voltages was needed and what ampere requirements we where aiming to satisfy. They made a box for all the components and installed automobile fuses in an array, which are easily changeable. The automobile fuses was something I wrote about in my specialization project. They included variable switching voltage regulators also in case some unforeseen voltage would be required in the future.

## 5.5 My Role in the Project: The Person to Ask

I started out by saying my contributions where often subtle. Some of my contributions has simply been to give answers to "is this a good solution?" or "is this what you had in mind?". Some times there has been questions about things I have written about in my specialization project, like what type of case the TX1 should have. By themselves, these are minor contributions, but when you add them up they become something more. It is good to have someone with an overview in a project; someone to make sure everyone is moving towards the same goal.

As time passed I found that many parts of the project is dependent on me and the knowledge I have acquired. It turns out that things like starting the robot, running modules or configuring any of the computers are things that I am used to, but other students may not have learned. Often students are only familiar with the specific parts that they have been working on. Because of this I have been asked for information many times by others working on the project. I can say with confidence that I am currently the person with the best overview of the robot part of this project. I have attempted to live up to this responsibility by being available on Slack and for meetings at school. This changes as soon as I leave NTNU, and my priorities has shifted through the project to leave the project in the best possible state for new students. You can read more about this in Chapter 10.

## 5.6 Raspberry Pi Start Script

I saw, after they where finished working, that the EiT students had made some attempt to make a startup script without being successful. I understand why, because many guides online would have the script run as *root*. This does not work well with ROS and I did have a few failed attempts before I managed to get it working. The EiT groups had turned off the auto start of the desktop environment. I assume to save resources. However, it is much simpler to auto log in and set up start scripts with it. For more on this see the guide. I set up a start script that auto starts the random LED blinking when we start up the Cyborg.

## 5.7 Conclusion

The contributions from the EiT students brought us a big step forward. I have successfully made myself available to answer questions and contributed some to their solutions. They did the work, but I helped make sure their work was aligned with the project's goal. In their presentation one group said communication could sometimes be difficult, but the other group said it worked well. I have been on Slack and email almost every day, and on Wednesdays I was sometimes even there in person when they where working, so I am unsure why they felt communication could be difficult.

# Chapter 6

# The New Hardware Architecture

## 6.1 Introduction

When I was working on the hardware of the Cyborg last semester I was a bit overwhelmed by the task. During my last two semesters I have worked towards a more structured setup that will be easier to work with, and I personally believe I have achieved this.

Many of the ideas that shaped the new hardware architecture has come from me, but some ideas had been discussed even before I entered the project. When it comes to implementing the hardware other students have contributed more than me; I cannot take credit for the layout and attachment of most of the hardware, much of which has been done by the EiT groups.

## 6.2 Architecture Diagram with Protocols and Power Connections

Figure 6.1 show the current hardware with communication protocols and power connections. Power connections are connected to the voltage regulator using either mini-fit or USB connections. Peripherals without a visible power connection draw power from the USB connection.

## 6.3 Hardware's Location on the Cyborg

The following list corresponds to the numbers shown in Figure 6.2. The Start Box is located at the back and the NodeMCU is located inside the LED cube.

1. The Pioneer LX robot base.
2. The TX1 in its case.
3. The Cyborg Router.
4. The Raspberry Pi.
5. The voltage regulator.

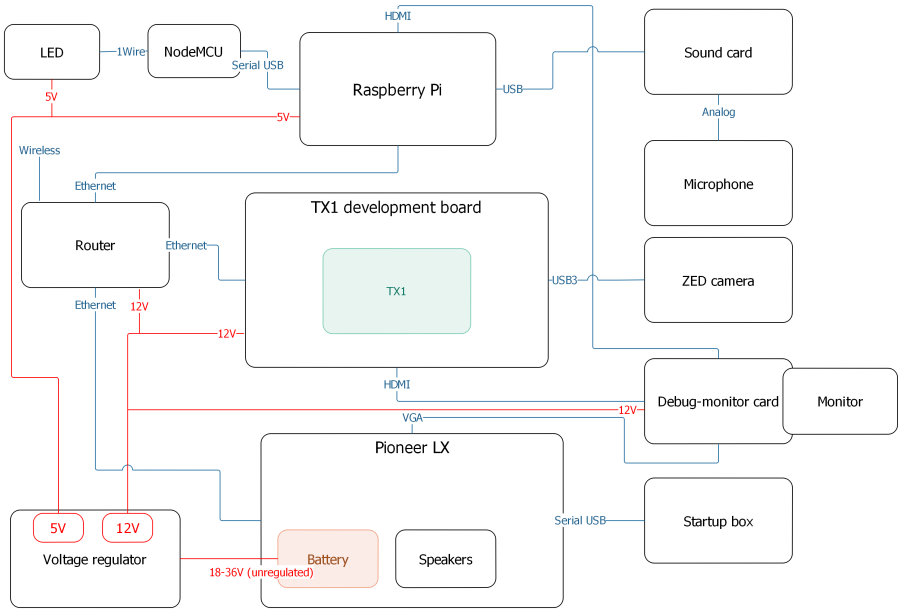


Figure 6.1: The architecture diagram with communication protocols and power connections. Unfortunately the monitor card and monitor where not fully implemented.

6. The ZED stereo camera.

7. The microphone.

## 6.4 Conclusion

The architecture design was heavily influenced by my work in the specialization project, but it has also been influenced by many other students. It was the EiT groups who attached most of the hardware to the Cyborg. I will discuss the Cyborg's network solution in more detail in Chapter 8. We have made the hardware Linux compatible and easier to work with. Since we finally have a voltage regulator, the Cyborg will no longer be limited by smaller batteries that need to be charged separately as was the case with the Cyborg laptop. New students to the project should find the system much easier to debug and configure due to the Cyborg subnet.

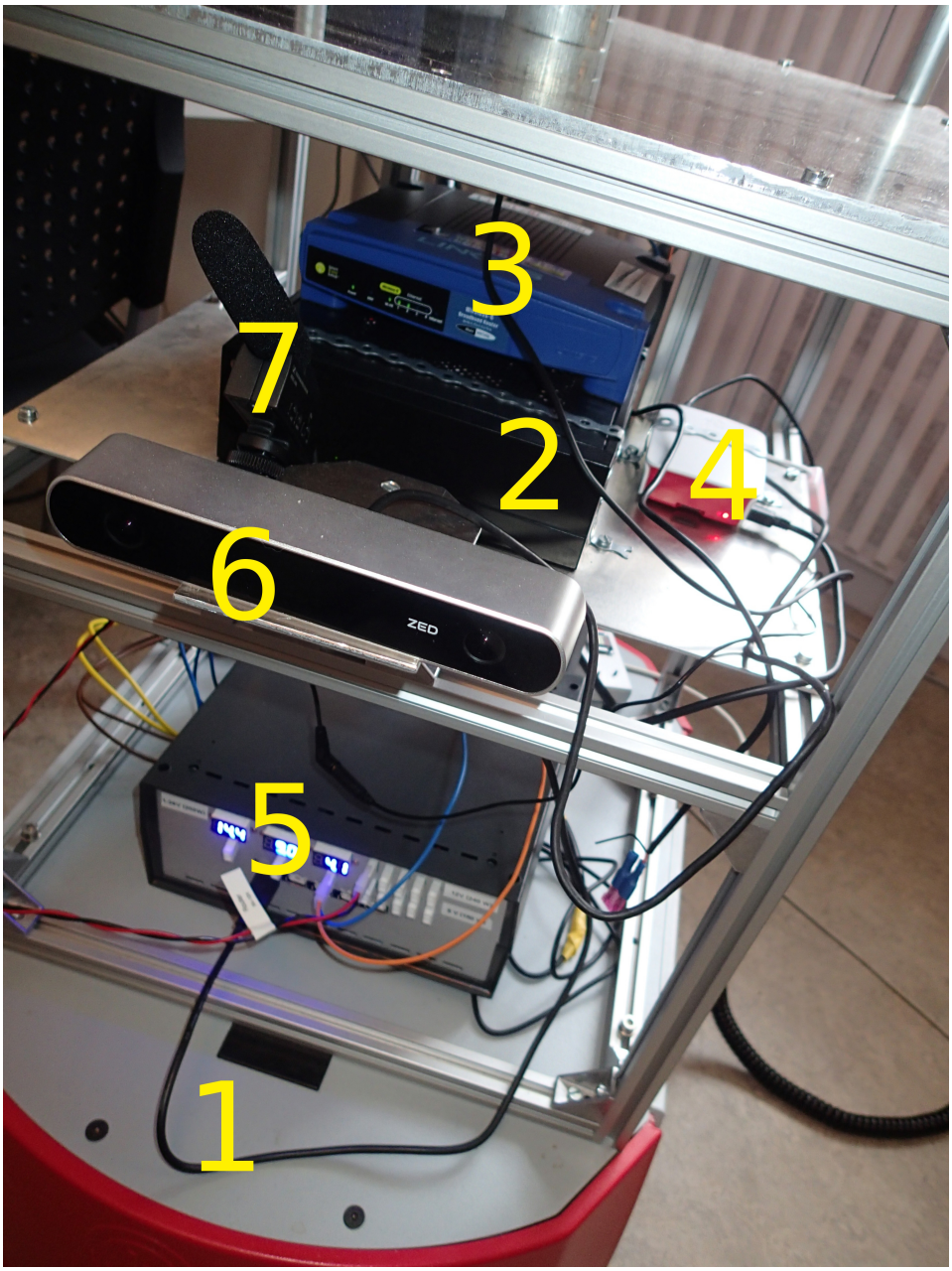


Figure 6.2: Important hardware components on the Cyborg.

# Chapter 7

# The Start-up Script and Box

## 7.1 Introduction

One of the challenges of demonstrating an independent robot like the Cyborg is how to start all the modules and make sure the robot starts the correct operations. The first step towards a solution was the startup script. This is a script that will run when the OS on the robot base is starting. The simplest solution here would be to simply load all modules necessary for a demonstration and let the Cyborg do it's thing. There are, however, times where we might not want it to go straight to demonstration. We may want to use the joystick to drive the Cyborg from the Cyborg office to Glassgården, or we may want go straight into scanning a map or changing location.

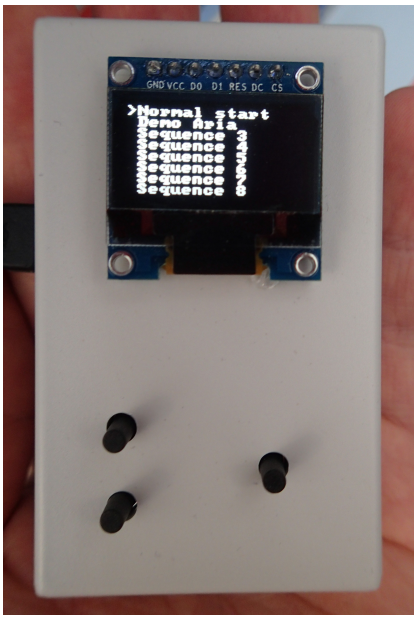


Figure 7.1: The Start Box.

To satisfy all these possibilities I decided to make a start-up box that we can use to choose what mode we want to start in. This gives us full freedom to implement and choose starting sequences, using only the box and no keyboard, mouse or monitor. The box is small enough that it can be hidden discretely, contrary to a debugging monitor with mouse and keyboard. Debugging can now also be done over the Cyborg's subnet.

One challenge that took some extra time was that Xubuntu 16.04 did not have the option to automatically log in to a preferred user when the computer is started. After searching online for some time I found a solution[9], but then later I re installed the OS and for some reason the option is available when installing. Now the OS will log in to

the correct user at start up.

It was my idea to make the **Startup Box**, but having a start script has been discussed for a while.

## 7.2 Communication during the Startup Sequence

To communicate between the robot base and the start-up box I decided it would be best to use serial communication over USB. Then I could make an Arduino device that could draw power over the USB connection and at the same time communicate over the same USB connection. I decided to use python for the start-up script because of its inherent simplicity and compatible libraries.

When I run a python script and use it to listen to a serial USB connection to an Arduino, it will reset the Arduino device. To work around this I had to make sure the communication protocol was designed correctly. Another challenge is that the user may choose the start-up sequence on the box before the robot base is ready to receive which sequence to start. The box will therefor wait for a ready signal from the robot base. To avoid having to receive a ready signal from the robot base while at the same time receiving user input, the box will not accept user input until the robot base is ready.

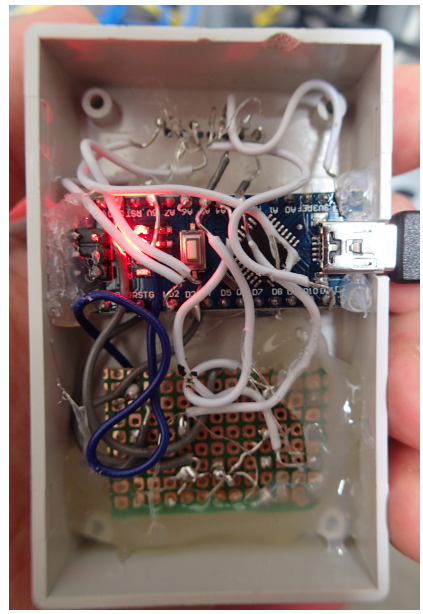

Figure 7.2: Inside the Start Box. A bit messy, but it works. Trust me, I am (soon) an engineer.

Once the user input has been received, the Startup Box will display the selected sequence and send the selected sequence to the robot base. The script running on the robot base will start ROS and the correct modules for the selected sequence. In the future the script can instruct the Raspberry Pi and the TX1 of what sequence to run.

## 7.3 Hardware Solution

The hardware solution was selected for simplicity and reliability. I chose an Arduino Nano with an Atmel XMega 328p and a small 128x64 pixels OLED screen using the SSD1306 driver. The Arduino is connected to the OLED screen through SPI, which is faster than I2C, but require more wires. I used a small box designed for electronics projects to hold the design, and included three tactile push buttons. The push buttons are soldered onto a small board to assure a visually pleasing layout and to make sure they stay in place. The parts where glued in place and should, theoretically, withstand a lot of use. Making a new box with my provided documentation should also be simple if it is ever necessary.

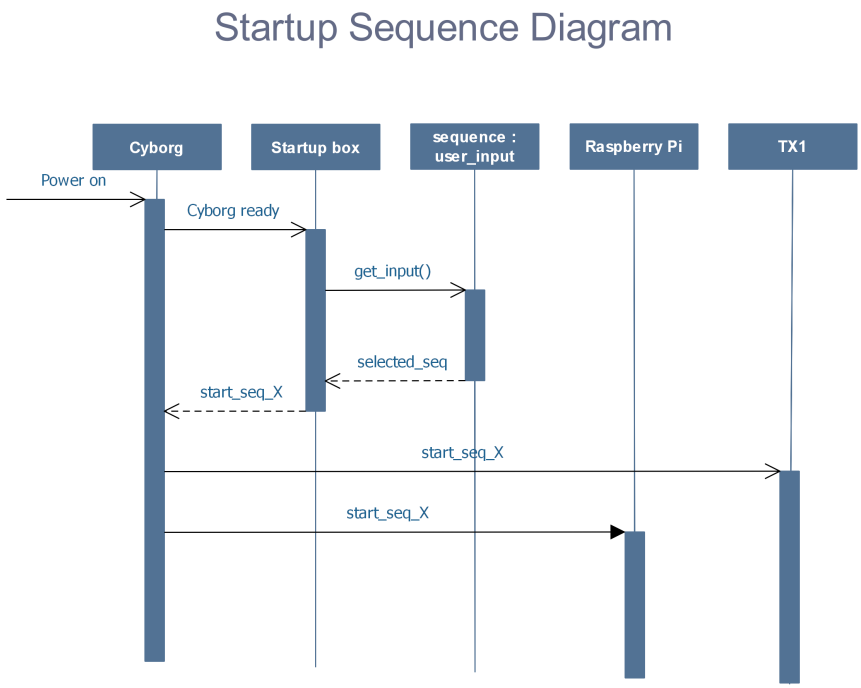


Figure 7.3: The communication sequence diagram for the startup box and the embedded computers.

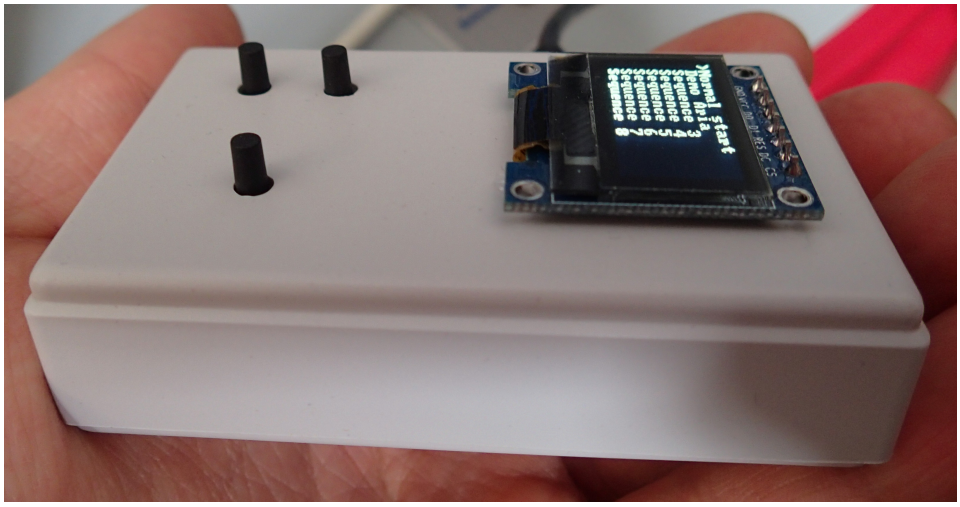


Figure 7.4: The Start Box from the side. I drilled holes for the OLED pins and the buttons.

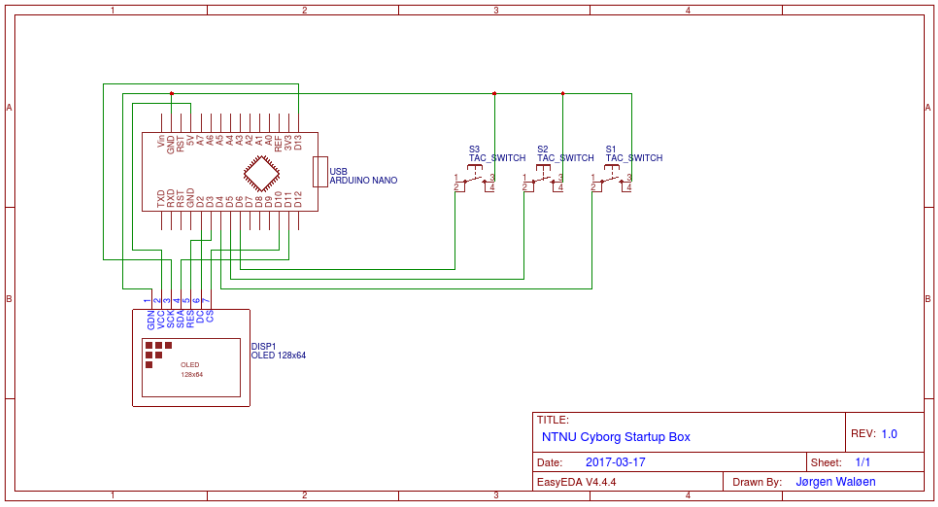


Figure 7.5: The schematics for my Startup Box. The Arduino Nano draws power from, and communicates over, the USB connection.

## 7.4 The Box' Software Solution

I had several choices for software design. I could use an Arduino library for the OLED screen and the Arduino IDE. This would be a very quick solution and offer a great versatility. Versatility was not, however, a priority. The screen needs to be able to output text, and be able to clear itself, and that is about it. Many of the Arduino libraries come with licensing demands, like Adafruit expecting you to include a header, copyright and

splash screen in your code. I was not too keen on that. Out of respect for the developers and because I could solve this task myself I decided to write my own library using only the initialization sequence for the OLED screen from Adafruit. I also decided to not use the Arduino IDE but rather write the code in C and compile using a Makefile and upload using avrdude. This may seem like a cumbersome solution but it was one I was comfortable with using and I know the code can be shared without issue. The Makefile is a slightly modified version of one shared by OmegaV[6]. By following my steps it should be easy for future students to update the Start Box.

The only thing that needs to be updated in the future is the actual menu text, and if necessary the menu needs to be expanded beyond its 8 entries. Since the Python script will parse the message from the Arduino and decides what start-up sequence to execute based on a number, I assure modular sequences through separating each sequence by number. The Python script is where most of the future updates will happen, but I designed the software so that updating the menu is very easy as well.

The pins D4, D5 and D6 are set to high with an internal pull-up, and when the button is pushed the pin is grounded, which the software will register as a button press. The software checks for button press every 150ms plus the overhead of updating the screen.

I wrote all the drivers for the Start Box myself. Writing drivers like these I learned in the *Byggern* course, which I am very familiar with.

### 7.4.1 Compile and Upload

I used a guide from OmegaV[6] to write a Makefile that would compile and upload to the Start Box. The only changes that may have to be done in the future is the USB port number in the Makefile, which I discuss in the guide. Dependencies can also be found in the guide. To change the menu on the Start Box, edit the `main.c` file and run the following command in the folder:

```
./flash
```

`flash` in a small shell script I wrote to ensure elevated privilege and the correct option for the Makefile. The correct avrdude command in the Makefile was quite difficult to make, and the one suggested by OmegaV would time out when attempting to connect to the Arduino.

## 7.5 The Script

As shown in Figure 7.3 the script is designed to initiate communication with the start box. The reason for this design is that when the python scripts starts to listen on the USB connection, it will restart the Arduino. This is why I do not allow any input on the Start Box until the python script is listening on the USB connection.

Once the user has input the desired start sequence, the script will initiate the correct sequence and open the different terminals that will run the different programs. The reason why I chose to open all the programs in new terminals is so it will be easier to debug if needed. One unfortunate consequence is that the script cannot be run over ssh, as the new terminals will crash. Even when adding `-X` to the ssh command. I was not able to find a solution to this, and over ssh the start sequence must be run one command at a time. In the future, the only file that needs to be changed is the

`sequences.py` file that specify what each sequence should do. The exception is if the script cannot find the Arduino, to which a solution is given in the guide in the Appendix.

In the guide I have also included what needs to be done if the OS on the robot base is re-installed. There I describe the process of setting up Xubuntu to auto-login and how to give permission to communicate over serial connections. This is the same process I used. In addition to this the following dependencies must be installed for the script to work:

```
sudo apt-get install python-pip
sudo pip install pySerial
```

### 7.5.1 The Script Files

```
~/start_scripts/sequences.py
~/start_scripts/start.py
~/start_scripts/arnl_start.sh
~/start_scripts/controller_start.sh
~/start_scripts/roscore_start.sh
```

These files are used for starting the communication with the Startup Box and starting the correct sequence of modules based on the user input from the start box.

**start.py**

This file initiate the communication with the Startup Box and parse the response. This should not need to be modified unless the startup box' USB port number changes. This is the file to run to start the script.

**sequences.py**

This file contains the correct starting sequences for all the sequences. We can modify this to change a sequence or add a new one.

**The shell scripts**

The shell scripts have to be used to initiate the correct setup for ROS and catkin upon opening a new terminal to run the modules. If the correct `source` commands are not present in the shell scripts, the modules will fail to start:

```
source /opt/ros/kinetic.setup
source ~/catkin_ws/devel/setup.bash
```

If you follow the ROS tutorial these commands will have been added to run automatically upon opening a terminal, but for some reason it is not the case when a terminal is opened from a Python script. Hence these shell scripts are necessary.

## 7.6 The Start Sequences

I wrote two start sequences. **Normal Start** will start the Cyborg in demonstration mode. Currently this means only starting ROS and the controller node on the robot base, but in the future this should include launching the correct programs on the Raspberry Pi and the TX1. I did not include this because none of them has a start script or modules that communicate with the controller node. For a demonstration the LEDs can be set to blinking at random for fun, but they cannot read the MEA signals.

### 7.6.1 Normal Start

In **Normal Start** mode the Cyborg will run the controller node. This sequence will run these three commands in three different terminals and in the given sequence:

```
roscore
rosrun rosarnl rosarnl_node
roslaunch cyborg_controller controller.launch
```

This will start the ROS core, run the ros-arnl module needed to navigate in a known environment and launch the controller node and all related ROS nodes. I created none of these programs. Note that a map must be added to the Arnl server, for example using MobileEyes. After suggestions from me the Mobile Robots support decided to add the option to change a robot's map through ROS also, partly because of the issues with MobileEyes but also because it might come in handy in the future. Documentation on this can be found on the ros-arnl Github page.

### 7.6.2 Demo Aria

The **Aria Demo** mode will start the Aria demo located at

```
/usr/local/Aria/examples/demo
```

This demo allow you to control the robot base with the joystick. This demo was created by Mobile Robots to demonstrate many of their example files at once. Once you start the demo you can switch between modes by pressing different keys on the keyboard. The Aria demo will start by default in the `teleop` mode. This mode allow you to control the robot base with either a keyboard or the joystick. The problem with this mode is that it is a **safe mode**, meaning it will not allow you to drive into things. Unfortunately the distance the robot base will stop from any obstacle is too far, making it impossible to navigate in the school halls.

Because I did not want us to have to carry around a keyboard to change the mode I started looking at solutions. After reading through the documentation located in the Aria folder, I decided to compile an example file for controlling the robot base with a joystick, the `joydriveActionExample`. I compiled using the following command in the example folder:

```
make joydriveActionExample
```

Unfortunately this example file returned an error because it could not read the joystick, which was attached. Instead I modified the `demo.cpp` file and recompiled it. The old `demo` and `demo.cpp` I renamed to `demoSafe` and `demoSafe.cpp`. I added a comment in the file and the section I changed is shown below.

```
// activate the default mode
teleop.activate();
```

The new code looks like this:

```
// activate the default mode
// changed 2017 from teleop to unguardedTeleop - Waloen
// old file named demoSafe.cpp - Waloen
unguardedTeleop.activate();
```

The new demo program worked just as I wanted. One thing to note is that the kill switch must be disabled when you start straight into Aria demo with the Start Box. Optionally you can use a keyboard to press 'e', which will enable motors, once you have deactivated the kill switch after starting the Aria demo.

## 7.7 Conclusion

The start box works as I wanted, and has already been very useful for me when I need to move the Cyborg or do tests in Glassgården. Currently only two starting sequences is implemented. This amount can easily be extended up to 8. Configuring the start script should be simple for future students. Configuring the Start Box menu is also very easy. All you have to do is to edit a file and run a script. It is so much faster and simpler than using the Arduino IDE, and can also be done over SSH in a terminal.

I believe the startup box will prove itself very useful in the future.

# Chapter 8

# The Cyborg Network

## 8.1 Introduction

Last semester I discussed several options for a network solution. At the time it was unsure who would be implementing the solution and most of the work I did was background research. Early this semester I qualified the network solution as a critical requirement for the demonstration. As the other students selected their tasks for the semester, this was one of the left over tasks, and thus it became my responsibility by default. Attempts have been made to get a Cyborg user for accessing **Eduroam** or some other school network, but the progress on this has been too slow. Hence I had to be creative and designed a solution that could be changed to adapt most future changes, for example if we get access to any of the school's wireless network's. The current solution can access any wifi network, and thus be used with both mobile networks and school networks. Students working on the project can even use their mobile phones to give Internet access to the cyborg. The router can be accessed via Ethernet to make any changes.

## 8.2 What a Solution Must Satisfy

1. The Raspberry Pi, TX1 and the Pioneer base all need to be connected on the same sub-network for easy communication.

2. The embedded computers should all be able to access the Internet.

3. The router should allow a high degree of customization.

4. The router should be able to connect to the Internet over wireless.

## 8.3 Evaluating Options

I will not include my complete evaluation process, as it is long and tedious. I have included a summary here. For a discussion on optional network architectures see my specialization project from last semester.

The last requirement I presented the last section presents a challenge; what is the best way to connect the cyborg to the Internet. Is it over a 4G mobile connection or through NTNU's wireless routers. We have had problems with NTNU's wireless routers in the past. Specifically we lose the connection for a few seconds when the cyborg moves between the wireless stations placed throughout Gløshaugen. We may want to connect to the school network in the future, but as of now we have not been granted a user account to use with the cyborg. Thus we have to use a mobile network for now, while still having the possibility to use the school network in the future. There is also the possibility that we cannot overcome the problem i describes when moving between the wireless stations.

Further, of the four needs presented above, the first two is covered by most wireless routers available to consumers. The third is less common and the fourth is very uncommon since most routers are locked to be wireless servers and not wireless clients. Professional routers with a high degree of customization are expensive and often focused on security, the last of which will cause overhead in the configuration process.

Many cheap routers will satisfy our needs if only we could configure them as we want. Thankfully we have several alternative firmware options available for simple consumer routers, among them DD-WRT, Tomato and OpenWrt. These are aimed at unlocking the full potential of your router. After researching some options I decided to use OpenWrt[21]. OpenWrt fulfills our needs, has a nice web interface and is known for allowing a high degree of customization, which I verified through reading through their documentation[20].

## 8.4 Selecting a Router

My main focus was to find a router that would work with OpenWrt, so I started with that in mind. Linksys has several OpenWrt compatible routers, some of which are powerful, high-end, multi-connection routers meant for large family homes and offices. They have a higher power requirement and larger size, so not ideal for us. I chose a newer model of an over 14 year old model, a series of routers that has had a central position in the open source community for router firmware. The Wrt in OpenWrt is from the name of this series of routers. Linksys embraces the open source community, but clearly states that installing OpenWrt will void your warranty.

The choice fell on the Linksys WRT54GL[12], with four Ethernet connections and a maximum link rate of 54 Mbps. Installing OpenWrt was be done from the included firmware by simply uploading the correct OpenWrt firmware[11] file on the **Update Firmware** page of the original WRT54GL firmware. Users report[11] online that the router can run on anything between 5 and 12 Volts, and under 1 ampere. The downside with this router is that I had to use an older version of OpenWrt because of hardware limitations. The version I used was Backfire 10.03.1. This downside is not a deal breaker for us as this version is good enough.

## 8.5 The Network Architecture

The architecture is shown in Figure 8.2, with two uncertainties:

- It is uncertain whether the Cyborg server will communicate with the Cyborg over the school network or over the Internet. This depends on the next uncertainty:
- We still do not know for a certainty whether we can use the school network to access the internet, or whether we need to use a 4G mobile network.

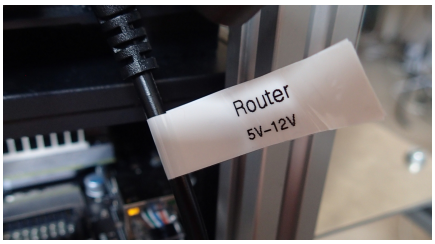


Figure 8.1: The cable tie I added to the router's power cable.

I designed the architecture to work around both uncertainties. This will also allow us to test whether the problems with the school network is persistent, while still having the option to use mobile networks with very small changes to the settings on the router.

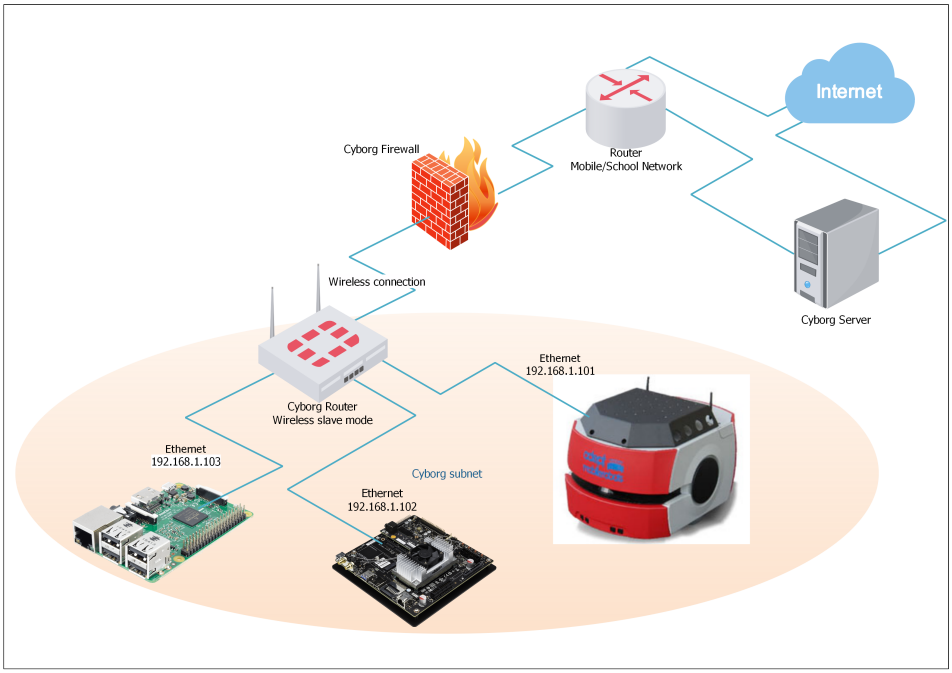


Figure 8.2: The network architecture. The Cyborg will either communicate with the Cyborg Server through the school network or over the Internet. MEA signals will be accessed over the internet.

## 8.6 Hardware Adjustments

I soldered and connected a detachable wire between the voltage regulator and the router. The router has been used as a normal router by me for weeks now, and there has been no issue with the power supply to the router.


Figure 8.3: I added a label with the router's IP.

## 8.7 Configuring the OpenWRT Firmware

To access the router I recommend using the web based software by typing the routers IP in the search bar of a web browser once you are connected to the Cyborg's subnet. The router has the standard router IP adress: `192.168.1.1`

I gave the Cyborg's embedded computers static IP's based on Figure 8.2. The DHCP lease configuration is shown in Figure 8.4.

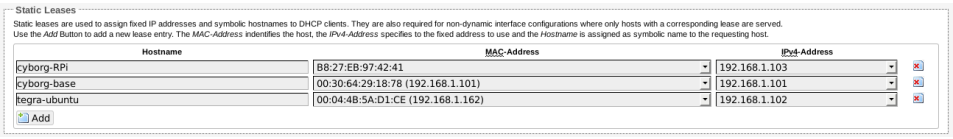


Figure 8.4: The DHCP lease configuration used to set static IPs.

To configure the DHCP leases you need to navigate to the sub-menu shown in Figure 8.5.

### 8.7.1 Connecting the Router to the Internet

**Wireless**

Follow this step by step guide to connect the router to a wireless network. The procedure is the same whether it is a school network or a mobile network.

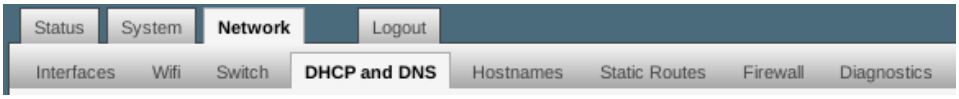


Figure 8.5: The sub-menu for configuring the DHCP leases.

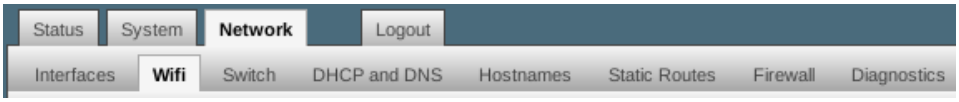


Figure 8.6: The sub-menu for configuring a wireless connection.

1. Navigate to the Wifi Sub-menu. Figure 8.6 show the sub-menu for configuring wireless connection, both for server and client modes.

2. Scan for all available wireless networks by clicking the scan button for the **wl0** controller on this page. Figure 8.7 shows the correct button.

3. A list of available networks will appear. Click **Join Network** for the network you want to join.

4. You can make changes to the setting if needed, but entering a password should suffice.


Figure 8.7: To connect to a wireless network, click the scan button for the **Broadcom Wireless Controller**.

**Wired**

To give the Cyborg Internet access over a wired connection you can connect the router to the switch on the Cyborg office. This is very useful for large updates and downloads to save mobile bandwidth.

## 8.8 Navigating the Cyborg's Subnet

I wanted to implement a design that would allow easy access to all the Cyborg's computers. By using a subnet on the Cyborg I did not have to set up any port forwarding or DNS subscription; we can simply access the Cyborg with an Ethernet cable or possibly over wireless. To access the cyborg base over ssh you only need to connect to the subnet with an Ethernet cable and type:

```
ssh cyborg@192.168.1.101 -X
```


Figure 8.8: I added a label to the router with the current static IPs.

The same command works for the other embedded computers and their IP's, where `cyborg` must be changed with the appropriate user for that computer. The `-X` parameter allows opening graphical interfaces over the ssh connection, for example a text editor or Mapper3.

For accessing and editing files you can also use a file manager in Linux. Look for a option similar to "Connect to a server", and use the ssh option.

### 8.8.1 Wireless Debug Connection

It might be possible to access the Cyborg's subnet over wireless while at the same time connecting to the internet over wireless. The WRT54GL has two antennas, but whether we can use client and server mode over wireless at once is still unsure. The reason why this would be desired is that we could debug and connect with MobileEyes without using an Ethernet cable. The solution to achieve this may be as simple as adding a wireless USB dongle to the USB port on the USB port on the router and possible install a package for it. For now we can use an Ethernet cable to connect to the subnet, but a wireless option should be explored in the future.

## 8.9 Conclusion

I installed and configured OpenWrt successfully. I also connected the router to the voltage regulator and made sure it works as intended. In Section 9.6 I present the results of two tests I did on the implemented controller node. In both these tests I used the Cyborg network to debug. Throughout the project the subnet has proven itself very useful by giving me easy access to the embedded computers on the cyborg. In the future we can set up ROS to communicate between the embedded computers, something ROS is built to do seamlessly.

# Chapter 9

# Implementing the Controller Node and Modules

## 9.1 Introduction

Andersen[3] had a complete and working state machine and the simulations were very promising. I had to overcome the following challenges when I was implementing the controller node on the Cyborg's base:

- The MobileEyes software does not accept certain symbols when you attempt to log on to a robot.
- The MobileEyes software will not run properly in Linux.
- Andersen made a installation shell script for dependencies, but it does not work properly.

The first of which cost me a lot of time since when I installed Xubuntu last semester I created a user and password identical to the previously used OS. Both the password and the user name included the dash symbol -, which is not accepted by MobileEyes. I did attempt to make changes to this in Xubuntu, but it turned out that changing the user name in Ubuntu is not a guaranteed success and may cause some annoying bugs. I created a new user also but for some reason this user had trouble running ROS. I did not bother to try and fix these issues and all that may have followed. I re-installed Xubuntu on the robot base and made sure similar issues would not occur again, hence no dashes.

The reason I needed MobileEyes was so I could change the map on the Arnl server on the robot first and foremost, but it is a very useful program for debugging. Without giving the Arnl server a map, the base would refuse to move.

## 9.2 The Controller Module by Thomas Rostrup Andersen

The controller node presented by Andersen plays an important role in the outcome of my work. Andersen delivered a fully functioning controller node and state-machine

in February. Although the controller node was designed and realized, it still needed to be implemented on the Cyborg and run on the actual robot hardware. Andersen's design included only the most basic modules, and further implementation of ROS modules will be required.

The controller module decides which module gets to be active at any given time. This is necessary to avoid conflicts between the modules. To solve this, Andersen chose to use the ROS actionlib protocol. Andersen also implemented a state machine. When the robot enters the idle state, it will use a PAD emotional state model to decide what action to take next. Such models use the three dimensions pleasure, arousal and dominance to represent all emotional states. Further implementations included a basic navigation system.

### 9.2.1 How to Simulate the Controller Node

In the Cyborg guide, which is included in the appendix, I have describes the process of simulating the Controller Node.

### 9.2.2 Challenges Encountered when Simulating Andersen's Design

When Andersen finished his work in late February, I started to implement his design on my personal hardware. I had kept up to date on his project to some degree before this. He had made a shell script for installation I used as a guide for installing dependencies. He also included most of the necessary information on the GitHub pages. Unfortunately I encountered several very time consuming bugs when I tried to Simulate his design, but it was not his fault. After installing all the dependencies and I tried to run the simulation, the Arnl server would not load a map to navigate in. The documentation on this subject, both in READMEs and wikis, where very limited. After some time I found out that I needed to use MobileEyes to connect to the arnl server on the robot, and then enter "Robot Configuration" to select a map. Whenever I tried this, MobileEyes would crash, caused by a segmentation fault. I asked Andersen if he had experienced this, and he had not. He had run Ubuntu 16.04 64bit on a virtual machine. Thus I assumed the fault was not caused by using Ubuntu, but rather by using some mix of 32 bit and 64 bit programs and operating systems. As soon as I received the first segmentation fault, I contacted the MobileEyes support, but did not receive an answer for weeks. The assumption that this was an OS architectural issue led me to spent a lot of time trying to find a solution. I was also afraid the fault may have been from using Xubuntu. So I reinstalled several different Ubuntu versions and installed any kind of dependency that may be needed to run MobileEyes. I searched their wiki, mailing lists, the Internet in general and contacted the support again. After trying different solutions I attempted to connect to the robot using MobileEyes from a Windows computer. This means i run the robot simulation on one laptop with Ubuntu 16.04 and MobileEyes on another Windows laptop. This worked. Soon after I solved this issue, I received answers from the support. One of the answers where not helpful at all and suggested rebuilding the arnl server, the other answer I received was from someone who had gotten the same error as me in Ubuntu. He suggested connecting from Windows, which I already knew.

### 9.2.3 Evaluating Andersen's Design

**Lack of Network Support**

Andersen simulated on one computer and the simulated robot could only be in one state at any given time. This works for the most basic tasks. It is also paramount that the Cyborg has such a state machine, yet we must not forget to exploit one of ROS' most important strengths; it can be distributed. Since we have three powerful embedded computers on the cyborg running ROS, and a server hidden away, we have many possibilities for parallelism.

**Lack of Functioning Modules**

Since Andersen focused on the controller node, he implemented some modules that are no longer being used. A good example is the **Simon Says** module. Since the cyborg does not have a working camera vision, it is not possible to play Simon Says. The same can be said for the modules that require a microphone.

**Never Implemented on the Cyborg**

Andersen never implemented the design on the actual Cyborg, and thus I had no idea whether it would work as intended.

**Full Autonomy**

To work around the issue of having states which would not work properly in his system, Andersen ensured that the Cyborg will eventually leave the state and continue with another task. His design makes the Cyborg autonomous, which is exactly what I required for the Presentable Cyborg. This saved me a lot of work.

## 9.3 Installing the Requirements

The installation script Andersen made failed because of an error returned from one of the commands. It can still be used as an installation guide. I see no reason to create a new script, future students can just follow it as a recipe and solve any errors by following the Ubuntu recommendations when they appear during installation.

## 9.4 Downloading and Compiling the Modules

To download all the current modules the following commands can be used in the catkin work space source folder (`/catkin_ws/src/`):

```
git clone https://github.com/thentnucyborg/cyborg_command
git clone https://github.com/thentnucyborg/cyborg_ros_controller
git clone https://github.com/thentnucyborg/cyborg_ros_conversation
git clone https://github.com/thentnucyborg/cyborg_ros_idle
git clone https://github.com/thentnucyborg/cyborg_ros_music
git clone https://github.com/thentnucyborg/cyborg_ros_navigation
git clone https://github.com/thentnucyborg/cyborg_ros_text_to_speech
git clone https://github.com/thentnucyborg/executive_smach_visualization
git clone https://github.com/MobileRobots/ros-arnl
```

Once the updated modules are present in the catkin work space source folder, you can run `catkin_make` in a terminal in the catkin folder to compile all modules.

### 9.4.1 Localization and Navigation

The ros-arnl module is made by Mobile Robots. This module integrates the Arnl server with ROS, and makes it possible for us to access the Arnl server over ROS. After talking to the Mobile Robots support they included the functionality to change the map of the Arnl server in the ROS module.

## 9.5 Simulating the Controller and Using MobileEyes

First you need to start the MobileSim software, or else the Arnl server will fail. This software imitates the sensors on the robot base, and allow the Arnl server to believe it is in a real environment. MobileSim will ask for which robot type you are using and what map you want to simulate. In MobileSim you can choose what actual position to simulate.

To run the ROS modules I open three different terminals and run these three commands, one in each of them:

```
roscore
rosrun rosarnl rosarnl_node
roslaunch cyborg_controller controller.launch
```

I use the same process as for the "Normal Start" in the start script. When simulating, however, I can run MobileEyes from the same computer that I simulate the controller node on. If so I enter `localhost` for the robot's IP address and do not need to enter username or password. It was when I did this that I experienced many issues with MobileEyes. I also added a generic, royalty free music in my home folder and named it `music.mp3` which is the file played in the Cyborg's idle state. The simulation will continuously update the `graph.png` image in your home folder as it changes states. This graph is shown in Figure 9.1.

The where only one or two more things I had to change in his design. In the `statemachinemonitor.py` file, he has used his home folder on his computer, and not the generic home folder address which is the tilde symbol. I am unsure if I also had to create an empty file before running the controller node the first time or it would fail, but that may have been the music file. If you look at the error message it should be obvious.

```
/cyborg_ros_controller/src/statemachinemonitor.py
```

## 9.6 Testing the Implementation in Glassgården

### 9.6.1 Wednesday the 3rd of May

The first test in Glassgården was on Wednesday the 3rd of May. This was not intended as a demonstration but rather a test to find out what is missing. The first challenge to appear was that the Aria demo which is used to drive the Cyborg around with the joystick started in **safe mode**. This mode will stop the robot base whenever there is any obstacles that can be dangerous. The problem is that it is very sensitive. It did not even want to move through the passageways in Gamle Elektro because it was too narrow. It is useless for us in this state as we cannot even drive the Cyborg to Glassgården. To Exit the safe mode, you need a keyboard.

The second challenge was that it did not want to move. I assumed it was because the Arnl server did not have a map.

The positive of the test was that the controller node worked, and I was able to change the controller states by using the command line tool Andersen wrote. The voice module was working and we could hear it talking, even if we could not talk to it through the microphone since it is not implemented in ROS. All in all a good first run in Glassgården. Accessing the embedded computers over the network for debugging worked well, and the modules were running as expected.

### 9.6.2 Wednesday the 7th of June

I had a good idea about what went wrong in the last test and how to solve those issues. How I solved the Aria demo challenge from the first test I explained in Section 7.6.2. To solve the second challenge I had to connect to the cyborg with MobileEyes while the implemented controller node was running. When I did that I could quickly see that my first assumption was not far off. The real issue was that the Cyborg though it started out in the Cyborg office, causing it to be lost. The cyborg need to know approximately where it starts, or it cannot find its position in the map. With Mapper3 I implemented a new starting point where the home location was implemented by Andersen and the issue was solved.

With the controller node working I ran several tests and the Cyborg acted as expected, which is just like it did in the simulation I did. See the video files in the delivery attachment and my comments on them in Appendix A.

## 9.7 Conclusion

I successfully simulated Andersen's design and then successfully implemented it on the Cyborg. I tested both his command script and his state machine. I successfully used MobileEyes and an ssh connection to debug while testing. The Cyborg changed states and both roamed freely, and navigated to known places, without any user input. The Cyborg made comments and played music as expected. The safety mode on the Arnl server seem to work and the robot base avoided obstacles.

> "Everything is proceeding as i have foreseen"
> - Emperor Palpatine, Return of the Jedi

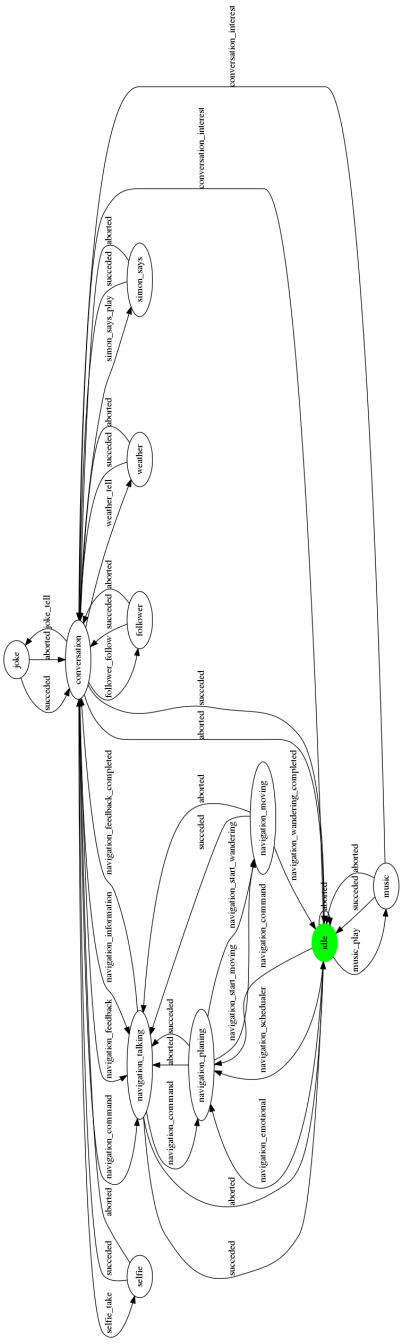


Figure 9.1: The state machine graph produced by Andersen's controller node. This was generated while I was simulating his design.

# Chapter 10

# The Student's Guide to the NTNU Cyborg

## 10.1 Introduction

In late April I came to a realization: I set out last semester to glue together the project's many pieces, and continued on that this spring. When I started this semester I had my mind set on finishing the practical work and prepare an impressive demonstration. I had to change my goal when I heard that it would not be possible to finish the nerve signal API after talking with the PhD candidate, who works on it, in April. This meant there would be no Presentable Cyborg by the end of this semester. There could, however, be a presentable robot. During the semester looked at the controller node and ROS modules, but there was not enough time to make any major changes to them once they where implemented on the cyborg. They already satisfied the requirements for The Presentable Cyborg, and I felt the project's core was nearing a stable state. That is why I decided to focus on creating the guide for the next students who will work on the project. Andersen's controller node will suffice for a robot presentation.

This thesis has required me to learn almost everything that others know about the hardware and software of the Cyborg, and a lot of things other do not know about. I have kept up to date on almost all work on the robot side of the project. From personal experience I know that entering this project can be a bit daunting and confusing. I also spoke with Kaja, who did her specialization project on the Cyborg, and she had similar experiences going into the project. Not only is the individual components of the project complex in of them self, making them all cooperate correctly adds even more complexity.

I decided that since I am leaving the project and will not be available for future students, I should put some extra effort into making an introduction to the project for the new students. I decided to make a guide containing the most important parts of my master thesis and specialization project and some new guides based on tasks I have had to learn. First I made a list of what I though should be included in the guide, which you can find in Section 10.2.

I also hope the guide will emphasize how much work and research has gone into all

of the smaller changes I have made. When I write in my thesis that I re-installed the OS on the robot base, it may seem like a trivial task, but as the guide will show; there are many underlying tasks that has to be completed. Behind every “how-to” in the guide there are many hours of research and testing, and sometimes days. This is also why I think there should be an updated guide for the Cyborg completed every semester. It will drastically reduce the research time and frustration for new student. Some of the software for the robot base is not documented well enough in my opinion (Arnl) and others have had bugs that have been hard to solve (MobileEyes). There is absolutely no reason why new students to the project should go through all the research I have when I can present them with easy step by step guides that any student should be able to follow.

## 10.2 The list of what I think would help new students

The idea is that any student with the guide at hand should be able to do the most common tasks on the Cyborg and have information on the basic parts of the project available.

1. Introduction to Hardware and Software
2. How-to guides for common tasks.
3. For each task, a list of links/useful information if relevant.

## 10.3 Conclusion

The resulting guide can be found at the very back of the Appendix. I felt so strongly it was necessary to finish this guide, even if it was not part of my first task description. A lot of the background information is from my thesis or specialization project.

The two main reasons I had for making this guide was:

- To help the future students of the project.
- To give a better perspective of the scope of my work.

I believe my guide will be of good use to the new students and writing the guide has allowed me to demonstrate work that I did not feel belonged in my thesis, but was part of my workload none the less.

# Chapter 11

# Suggested Future Work

Many future tasks have been suggested already, here I will present some tasks that are closely related to my work, that may or may not have been discussed before.

## 11.1 Cyborg Internet Connection

Either the Cyborg should have access to one of the school's networks, or a 4G router could be purchased to be turned on while having demonstrations. As long as the Cyborg server and the MEA signals' server has a static IP or DNS subscription, the Cyborg should not need it.

## 11.2 Debug Wireless Connection to the Cyborg's Subnet

As i proposed in my thesis, there is a good chance that the Cyborg router can run both as client and server, and thus allow wireless debugging and wireless connection to the Internet at the same time. One possible solution to this could be to add a wireless dongle to the router's USB port.

## 11.3 ROS Communication Between Computers

I did set up the code for this on the Raspberry Pi, but did not have time to test it. Sinde the Raspberry Pi only have one ROS mode anyway, and did not need communication with the base, I just made the LED demo start on boot.

## 11.4 Nerve Cells Stimuli

We have discussed implementing buttons that can send stimuli to the neural cells for some time. This require the robot to be able to access the internet and the MEA API must be completed.

## 11.5 Dialogue

We have implemented a microphone, but the ROS module need to be set up to work with it.

## 11.6 Updating the Guide every Year

If, as I hope, the guide is useful in the next semester, it should be updated regularly to make sure the knowledge of the Cyborg does not leave with the students. It is time consuming and difficult to dig through all the old reports. An option could be to use the Cyborg wiki for this in the future.

## 11.7 Clothes for the Cyborg

The Cyborg looks messy, even if it work well. I had an idea for making it more presentable and pleasing to the eye. Why not but some large secondhand clothes and dress up the Cyborg. Cut and old sheet for the front to look like a t-shirt, and hang a large hooded sweater on the back. The clothes can be hung like drapes to make it easier to access the hardware inside.

## 11.8 Integrating ROS Modules

This has been discussed to great lengths already. There have been created many ROS modules for this project, and there is quite a bit of work to make them working with the new Cyborg. Decisions have to be made for which to keep in the projects.

# Chapter 12

# Discussion

Even if I am pleased with the result of my work, there is always room for improvements. I intentionally wrote this thesis and the included guide in a straight to-the-point manner. It is often difficult to predict if the reader wants a short and concise description of my work, or as much details and background information as possible. I have tried to land somewhere in between. Writing the guide allowed me to present a large part of what I have learned and created in a very short and to-the-point manner, while making the information accessible to other students.

Writing a report on a large amount of practical work is surprisingly challenging. I have tried my best not to write my thesis like a work-journal, but I probably failed to avoid that at times. I have to get the information of my work in here somehow.

Another point I could have focused more on is the background information. Then again, this is not a scientific paper. I felt at the time of writing it that the correct thing to focus on was making the information accessible and to the point. The word I am looking for is that I believe the background information is *sufficient*, and that the strength of my work probably lie more in the results of my work.

I am pleased with how far I got, but at the same time I wish I had more time so I could help truly bring the Cyborg to life with even more functionality.

# Chapter 13

# Conclusion

The first and most important of my tasks was to attempt to get the Cyborg to a state where it could be presented in demonstrations. I have successfully gotten the Cyborg to a state where it can move around autonomously in Glassgården. I achieved this by implementing the controller node made by Andersen, which again is based on earlier student work. Since the API for the MEA signals was never finished, it was not possible to make the Cyborg into an actual cyborg. The signals was supposed to light up the LEDs or maybe affect some decision. The lack of a working API was outside of my control.

The second task was to guide the EiT groups. There was a small complaint on the communication, but the overall results speak for themselves. There has been a clear direction and goal in the project this semester, and I like to think I helped with that. I am also very pleased to see many of the ideas from my specialization project implemented. I have made myself available and answered questions throughout the semester.

The third task was to implement the controller node on the Cyborg. This was done successfully, and thanks to this we now have an autonomous, moving robot. We could still have had presentations with only the LEDs, if they used the MEA signals, and call it a cyborg, and that is why this third task was a separate task from the first. I decided, however, to make it part of my critical requirements for the presentable cyborg.

The fourth task was to write a Cyborg guide. I have written a guide that contain the the information i believe will be of great use for the upcoming students.

If we look at the critical requirements for The Presentable Cyborg we can see that requirement 1 and 2 was not reached, but as I have explained this was outside my control. We should have had more students on the project. Requirement 3 and 5 was reached and requirement 4 very close. The reason why 4 is not completely satisfied is because the embedded computers currently has nothing to communicate to each other. Setting up the communication with the current network architecture is very simple. Because it makes no difference to the Cyborg in it's current state whether ROS i configured to communicate over the network I will argue that we reached the goal of The Presentable Robot.

All of the current parts of The Cyborg project's core is in a stable and reliable state, which should make it much easier for future students to focus on their specific tasks. In

the future I recommend updating the guide and making sure the project's core is well documented and cared for, since it will be the foundation for the Cyborg project.

# Bibliography


[1] *About the Project*. URL: https://www.ntnu.edu/cyborg/about.

[2] *All Software*. URL: http://robots.mobilerobots.com/wiki/All_Software.

[3] Thomas Rostrup Andersen. *Controller Module for the NTNU Cyborg*. NTNU, 2017.

[4] Roy Angelsen et al. *Group 1, A Cyborgs life: Lighting up the neural activity*. EiT NTNU, 2017.

[5] *Arduino Webpage*. URL: https://www.arduino.cc/.

[6] *AVR on linux*. URL: http://omegav.no/wiki/index.php/AVR_on_linux.

[7] *AVRDUDE - AVR Downloader/UploaDEr*. URL: http://www.nongnu.org/avrdude/.

[8] Manfred E. Clynes and Nathan S. Kline. *Cyborgs and space*. Sept. 1960. URL: http://web.mit.edu/digitalapollo/Documents/Chapter1/cyborgs.pdf.

[9] *How to Enable Auto-Login in Xubuntu*. URL: https://appuals.com/how-to-enable-auto-login-in-xubuntu/.

[10] *Is ROS For Me?* URL: http://www.ros.org/is-ros-for-me/.

[11] *Linksys WRT54G, WRT54GL and WRT54GS*. URL: http://wiki.openwrt.org/toh/linksys/wrt54g?s[]=wrt54gl.

[12] *Linksys WRT54GL Wireless-G Wireless Router*. URL: http://www.linksys.com/my/p/P-WRT54GL/.

[13] Espen Moen et al. *Gruppe 2, LED-hode og Spenningsforsyning*. EiT NTNU, 2017.

[14] Jonas Fyrileiv Nævra. *The NTNU Cyborg 1.0*. NTNU, 2015.

[15] *NTNU Cyborg*. URL: https://www.ntnu.edu/cyborg.

[16] *Pioneer LX Research Platform*. URL: http://www.mobilerobots.com/ResearchRobots/PioneerLX.aspx.

[17] *Robot Operating System*. URL: https://en.wikipedia.org/wiki/Robot_Operating_System.

[18] *Suggestions for student projects (title and location of website was changed since i cited it).* URL: https://www.ntnu.edu/web/cyborg/student-projects.

[19] *(The page has since been changed, and I was not able to find the teplacement. See the other reference for the cyborg wesite).* URL: https://www.ntnu.edu/cyborg/robot.

[20] *Welcome to OpenWrt.* URL: http://wiki.openwrt.org/doc/start.

[21] *What is OpenWrt?* URL: https://openwrt.org/.

# Appendices

# Appendix A

# Comments on Video Attachments

I have included two video files from my test on the 7th of June. Here I will comment on them and what they show. I must apologize for the quality, I was trying to walk behind it at all times so i could reach the kill switch quickly if I ever had to. There where tables and display boards all over Glassgården and I did not know if the robot base would try to walk under any of them. It is also worth mentioning that the controller node would continue to select new things to do, but I did not want to make the videos too long.

## A.1 demovid_001.MOV

- I begin with the normal starting procedure and include MobileEyes from a debugging laptop placed on top of the Cyborg.
- MobileEyes ask me if I want to enable motors, but this is not necessary to do, I just do it to remove the small window.
- At 00:42 the Cyborg start moving and it has selected a destination. Hence it is in the `navigation_moving` state.
- Sometimes it will see obstacles that are not there and try to move around them, and it stopped just before its goal and failed to reach destination. I did not figure out what caused this.

## A.2 demovid_002.MOV

- This video starts before the Cyborg is awake, and we have to wait before we enter the sequence we want.
- This time the Cyborg wanders aimlessly and has not selected a destination. It is the same state in the state diagram but different options. The filming is terrible here because I was afraid the robot base would run into, or under, something.
- At 03:10 it selects a destination and starts moving.
- At 05:00 it reaches its destination and says “I am happy about elevator” and soon after starts playing music.

# Appendix B

# The Cyborg Guide

The rest of this thesis is dedicated to the guide. Pay attention to page numbers, which is separate from the rest of the thesis. The guide can also be found in the attachments for the delivery.

# Table of Contents

# List of Tables

# List of Figures

Chapter 1

# Introduction

When I was writing my master thesis in the spring of 2017 it was my second semester working on the Cyborg project. I had been working on many different parts of the project in an attempt to make the project move towards a collective goal. Part of the difficulty with the project thus far had been that the different pars of the project was diverging and not always moving in the same direction.

The second challenge I had become aware of was the difficulty of entering the project as a new participant. Information was scattered over many long project reports and master theses. Just getting the Cyborg up and running was a challenge, not to mention the frustration of trying to understanding what every student before you had produced and how to continue working on their work. Code had been written, hardware assembled, and trying to get a good overview of the situation was very time consuming.

Out of this frustration, which was also experienced by other students, I made this guide. The intention of this guide is simple: to help new students find information, get an overview and help navigate the current state of the project without having to dig through too many old reports.

The structure is simple:

- Chapter 2 collects information about the hardware components of the Cyborg.
- Every chapter beginning with "How to" is a how to guide to solve a certain task.

If the new students find this guide useful I recommend making it a tradition to collaborate and include their most important practical information in this report, and update it every year.

Chapter 2

# Introducing Hardware and Software

## 2.1 Hardware Overview

The current hardware architecture can be seen in Figure ... . The monitor and its controller board where never finished due to and issue with the ordered controller board.

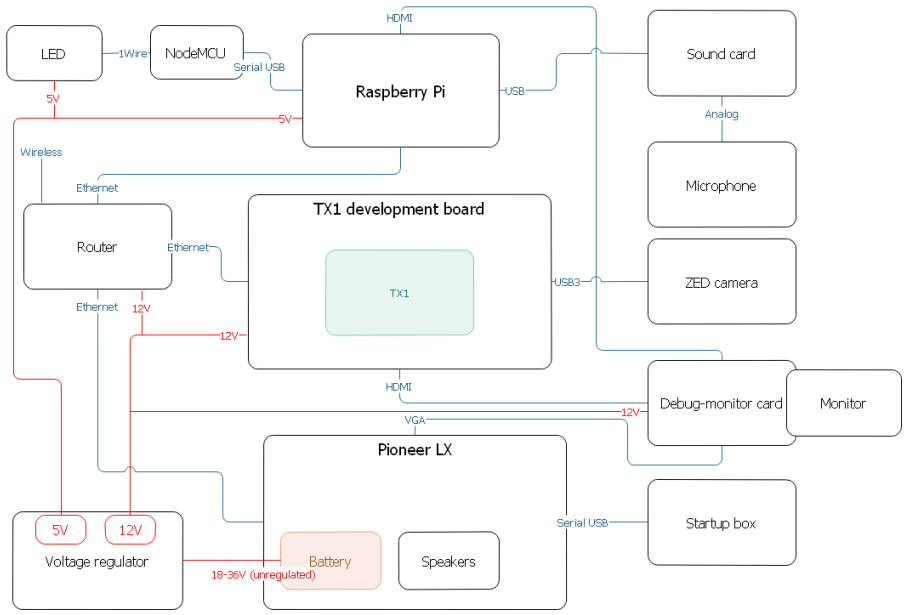


**Figure 2.1:** The architecture diagram with communication protocols and power connections. Unfortunately the monitor card and monitor where not fully implemented.

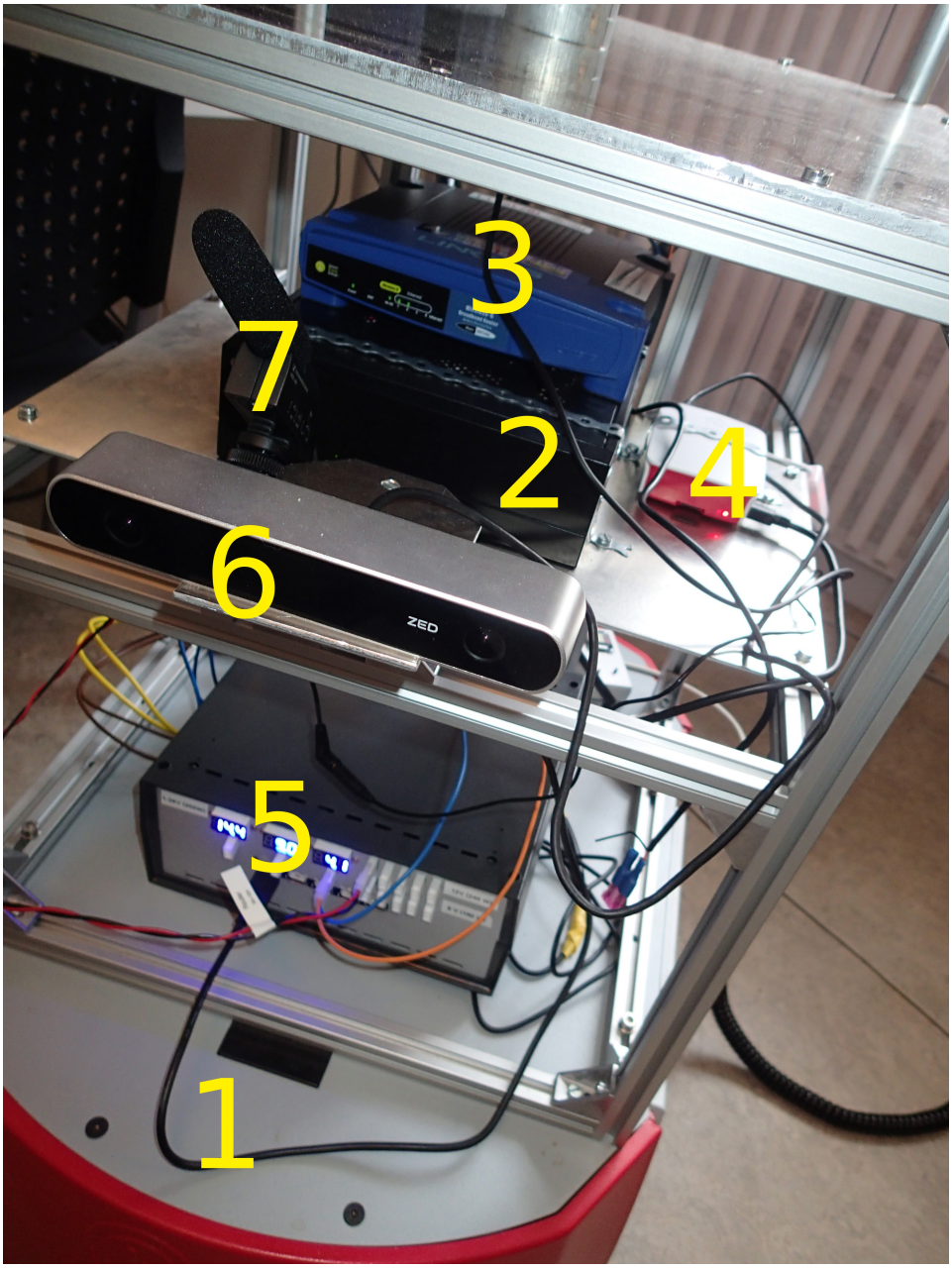


**Figure 2.2:** Important hardware components on the Cyborg.

The following list corresponds to the numbers shown in Figure 2.2. The Start Box is located at the back and the NodeMCU is located inside the LED cube.

1. The Pioneer LX robot base.
2. The TX1 in its case.
3. The Cyborg Router.
4. The Raspberry Pi.
5. The voltage regulator.
6. The ZED stereo camera.
7. The microphone.

## 2.2 Pioneer LX Robot Base: Hardware

The Pioneer LX have some useful sensors, including the SICK S-300 laser range sensor, bumper sensors in front and front and rear sonar sensors. Its embedded computer is capable of running Linux, ROS and all basic modules. Its computational power in the area of parallel processing (GPU) is low. The included SDK is capable of using the laser range sensor to map some of the obstacles and rooms, but because its mounted low it will not be able to map obstacles above a certain height. We also have a mounted skeleton on top of the Pioneer, and this makes navigating obstacles above ground even more challenging.

**Figure 2.3:** The Pioneer LX robot base.

`http://www.mobilerobots.com/ResearchRobots/PioneerLX.aspx`

### 2.2.1 SICK S-300

SICK S-300 is the laser rangefinder module on the Pioneer LX. It has 240 field of view, 30 meter maximum range and  angular resolution. This may appear as a low resolution for things further away, but when you move the robot to map the surroundings it works well. When creating a map of an area the Mapper3 software will smooth out the area and create a high resolution map for you. One major issue with the SICK S-300 is that it cannot see obstacles above the robot base. Therefor we need to be careful with tables and other similar obstacles.

### 2.2.2 The Pioneer's Kill Switch

The **kill switch** is a big red button that has to be raised before the motors will receive any power. If the robot ever acts unexpectedly, or is a danger for people or itself, we can hit this big red button to stop it. If needed, hardware can be connected to a special power connection on the battery which will stop delivering power when the switch is down.

**Table 2.1:** The Pioneer LX Embedded Computer

| | |
|---|---|
| CPU | Intel Atom D525 (”Pineview”) 64-bit Dual-Core 1.8GHz |
| RAM | 2 GB DDR3-1066 RAM |
| Storage | 16 GB solid state (SSD) |
| Graphics | Integrated Intel GMA 3150, (Gen3.5+), DirectX 9 support, OpenGL support, Pixel Shader v2.0.(No OpenCL support.) |
| USB | 3 external general purpose USB 2.0 ports |
| WIFI | Intel Centrino Advanced-N 6235ANHMW (dual band) 802.11 a/b/g/n |
| Bluetooth | Yes |
| Expansions | The integrated computen can not be modified. Devices may be added via ethernet, RS-232 serial, or USB. |

**Table 2.2:** The Pioneer LX Platform

| | |
|---|---|
| Operating system | Linux Ubuntu 12.04 |
| Size | 50 cm width, 70 cm length, 45 cm height, (37 cm height without equipment deck) |
| Weight | 60 kg |
| Maximum Speed | 1.8 m/s |
| Continuous Run Time | 13 hours |
| Recharge Time | 3.5 hours |
| Payload | 60 kg |
| Power Outputs | 5, 12, 20 VDC, switchable |
| Laser Range Sensor | Sick S300 laser rangefinder, 240 field of view, 30 meter maximum range, angular resolution |
| Other Sensors | Front and rear ultrasonic (sonar) range sensors.<br>Front bumper panel. |

### 2.2.3 Pioneer LX Robot Base: Software

At the start of this project the robot base was purchased with some useful tools to make robot navigation and movement easier. The robot base is a Pioneer LX from Mobile Robots which is capable of positioning itself in rooms and navigating around in rooms. It can also see obstacles it needs to avoid. It is a reliable and advanced solution to a very complicated problem, which allows us to focus on the other things we want to develop.

### 2.2.4 Pioneer LX Software Development Kit

The Pioneer LX Software Development Kit (SDK) consists of several useful tools. The most relevant tools for my project are ARIA, ARNL, Mapper3 and MobileEyes. Some are free, ARIA is open source, and some must be purchased. They are available at the Mobile Robots website:

`http://robots.mobilerobots.com/wiki/All_Software`

### 2.2.5 ARIA

This Open Source C++ library is designed to easily give access to networking, accessories and the controls of the robot. Also usable from Python, Java and Matlab. A ROS interface, **ROSARIA**, is available for ARIA. This is the key software for accessing the robot base's internal peripherals.

#### ARNL

ARNL is the intelligent localization and navigation software for the Pioneer LX. Utilizing the laser rangefinder it can be used to navigate and position the robot in a known map, and includes path-finding. A ROS interface, *ros-arnl*, is available for ARNL. ARNL makes the complicated task of navigation and positioning into a simple task.

#### Mapper3

Mapper3 is a software used for generating maps from laser scans, and adding obstacles and defining areas on the map. Mapper3 will take a laser scan log file (a .2d file) from the **sickLogger** program and create a .map file. It will also remove moving obstacles like peoples feet. It will leave you with a nice continuous outline of the area it maps. You can even modify the map on a robot over a network connection.

#### sickLogger

sickLogger is a program included with ARNL. It will log the data from the laser range finder, and store it in a `.2d` file.

#### MobileEyes

MobileEyes makes the Pioneer's sensor inputs and motion available, visualized in real-time. This program can be run from a host computer by connecting to the robot over the network. The programs main purpose is to visualize the robots vision, in our case this means the position in a map and unknown obstacles like feet. The program can also be used to create maps and to control the robot.

When you run the software you are asked to log in with the user name and password of the robot's OS and to give the IP of the robot.

#### MobileSim

MobileSim is used to simulate the sensor information given in a known environment. When you start MobileSim you select what robot model you are using and what map you want to simulate in. Once MobileSim is running you can start your Arnl server which will read the sensors as if you where in the actual real environment.

## 2.3 The Voltage Regulator

The voltage regulator delivers the correct voltage levels from the robot base' battery to the different embedded computers on the Cyborg. It is a vital task as we do not want to charge several batteries independently.

### 2.3.1 Mini-fit Connection Plugs

We got these from the Cybernetics Workshop. We do not have the correct clamping tool for them, so I used a pair of pliers and some solder to attach them to my wires.

### 2.3.2 ROS: Robot Operating System

The following summary from Wikipedia says it better than I could:

> "Robot Operating System (ROS) is a collection of software frameworks for robot software development, providing operating system-like functionality on a heterogeneous computer cluster. ROS provides standard operating system services such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management."

`https://en.wikipedia.org/wiki/Robot_Operating_System`

#### Modular Design

One of the main focus of ROS is to make your robot modular. You begin your project with a core, aptly named the **ROS core**. Then you add the packages you need, either by designing them yourself or by using some of the more than 3000 packages designed by

others and shared with the public. The modular nature of ROS allows you to choose what modules gets to run, and when.

ROS has a significant community behind it, and the users are coming both from the public and, more and more in later years, the commercial sector.

`http://www.ros.org/is-ros-for-me/`

#### Communications Infrastructure

Because of ROS' modular design, communication between modules is imperative. On the Cyborg this communication happens on the Cyborgs sub net, deployed on the Cyborg's router.

## 2.4 The Debug Monitor

The debug monitor is an unfinished task that I wrote about in my specialization project, and which one of the EiT groups tried to implement this semester. See their report for details. We may want to use it for the troll face at some point.

## 2.5 Raspberry Pi

The Raspberry Pi is currently being used to run the LED ROS module, but future tasks may include listening with a microphone and streaming the troll face.

## 2.6 Arduino

Arduino is a company that produces open source hardware and software. I used a cheap Arduino board to create the start box, but wrote the code in C in stead of using the Arduino IDE. The EiT group used a NodeMCU board, which has a ESP8266 chip, to control the LED cube. This was programmed with the Arduino IDE.

## 2.7 Avrdude

To upload the code to my start box I used Avrdude. Avrdude is available as a package for Ubuntu, and I recommend using **Synaptic Package Manager** to install it. Part of the idea is to be able to update and flash the start box from the robot base using only a terminal.

## 2.8 Serial Communication and SPI

To communicate between the start box and the robot base I used the serial communication UART over USB on the Arduino. To communicate with the OLED screen I used SPI. Both of these protocols are implemented in the start box. The UART communication can

be accessed from one of the /dev/ttyUSB connections. To see all serial USB connection use the command `ls /dev/ttyUSB*`.

The Raspberry Pi also uses serial communication to communicate with the ESP8266 based NodeMCU.

### 2.8.1 Linksys WRT54GL

The Linksys WRT54GL router is a small and cheap router. It lacks many of the advanced and often expensive functions of new routers. It is a updated model of a 14 year old router. The included firmware lacks some of the configuration options we want.

## 2.9 OpenWRT

> "OpenWrt is described as a Linux distribution for embedded devices.
>
> Instead of trying to create a single, static firmware, OpenWrt provides a fully writable filesystem with package management. This frees you from the application selection and configuration provided by the vendor and allows you to customize the device through the use of packages to suit any application. For developer, OpenWrt is the framework to build an application without having to build a complete firmware around it; for users this means the ability for full customization, to use the device in ways never envisioned."

Source:

`https://openwrt.org/`

An OpenWRT router can be accessed over SSH or through the web interface LuCI. Not all routers can run OpenWRT, see the hardware guide on their website for specific hardware.

## 2.10 NVIDIA Tegra X1

Tegra is NVIDIA's mobile processor family. The NVIDIA Tegra X1 (TX1) is the newest and most powerful addition to the family. The Tegra module integrates an ARM CPU, a memory controller and the GPU into one System on a Chip (SoC) module. The ARM CPU has four ARM Cortex-A57 cores and four ARM Cortex-A53 cores in a big.LITTLE architecture, which provide significant power-saving. Documentation on all NVIDIA related subjects have been gathered from NVIDIA's web page unless otherwise stated. You need to sign up to be a member to get access to much of the NVIDIA documentation.

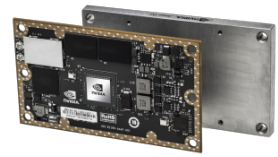

**Figure 2.4:** The NVIDIA Jetson TX1 module with protective case. From the NVIDIA Blog, link below.

## 2.11 NVIDIA Jetson TX1

The Jetson TX1 is the module sold by NVIDIA with the Tegra X1 integrated. The Jetson module can be mounted on several boards sold by separate producers, and is also sold separately.

> "Jetson TX1 exceeds the performance of Intels high-end Core i7-6700K Skylake in deep learning classification with Caffe, and while drawing only a fraction of the power, achieves more than ten times the perf-per-watt."

Citation from:

`https://devblogs.nvidia.com/parallelforall/nvidia-jetson-tx1-supercomputer-on-module-drives-next-wave-of-autonomous-machines/`

The TX1 has 256 cores and can deliver 1 TFLOPs (FP16), which means that it can compute $10^{12}$ Floating-point Operations Per Second of the *half-precision* number format, and 0.5 TFLOPs of *single-precision*. For comparison the GeForce 980 can deliver 4.6 TFLOPs on the *single-precision* number format, which is significantly more. In other words, the X1 can't compete with modern desktop NVIDIA cards, but it is the power consumption of the X1 which is its major advantage. The 0.5 TFLOPs can be delivered while consuming 10 to 15 Watts for the whole Jetson module, while the 980 alone can consume up to 165 Watts. The X1 is designed for running on batteries and still delivering cutting edge parallel processing. See link below for sources.

`https://devblogs.nvidia.com/parallelforall/nvidia-jetson-tx1-supercomputer-on-module-drives-next-wave-of-autonomous-machines/`

I may refer to this module as the *TX1* or the *TX1 module* throughout this report.

**Table 2.3:** Selected JETSON TX1 Features. From the documentation found in the Developer's Program

| Category | Feature |
|---|---|
| CPU | Quad-core Cortex-A57 complex |
| GPU | 256 core NVIDIA Maxwell architecture GPU |
| Memory | 4GB LPDDR4-3200<br>16GB eMMC 5.1 |
| Connectivity | BCM4354 w/ dual U.FL RF connectors:<br>Connects to 802.11ac Wi-Fi and Bluetooth enabled devices. |
| Advanced power management | Dynamic voltage and frequency scaling<br>Multiple clock and power domains |

## 2.12 NVIDIA Jetson TX1 Development Board

The Jetson TX1 can be bought already mounted on a development board. This development board includes most of the modules and I/O a developer would need, including power

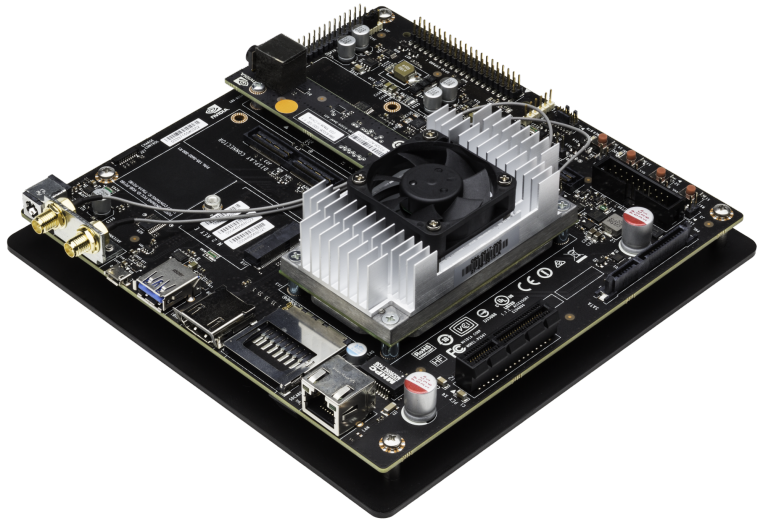

**Figure 2.5:** The NVIDIA Jetson TX1 development board. From the NVIDIA Blog, link above.

management, USB ports, Ethernet port, Wi-Fi, SD card reader, General IO pins, and much more. I may refer to this board throughout the report as the *TX1 board* or the *development board*.

## 2.13 ZED Stereo Camera

The ZED Stereo Camera uses two 4 mega pixel cameras to capture 3D video at high frame-rates and in high-resolution. The camera supports up to 2K video and depth perception from 0.7 to 20 meters. Because of its high frame-rate and depth perception it can also be used for positional tracking with six degrees of freedom. The camera requires 380 mA at 5 V, which means it must be connected to a USB 3.0 port on the Jetson board. Several useful tools have been developed, and others are in development.

Source:

`https://www.stereolabs.com`


**Figure 2.6:** The ZED stereo camera. From https://www.stereolabs.com.

## 2.14 ZED SDK

The *ZED Software Development Kit* consists of some very useful tools and several 3rd party integration option, most importantly ROS integration. The *ZED Depth Viewer* shows

you a 3D model you can turn around and a black and white image where the distance to every pixel is represented by the black to white scale. Darker pixels are further away. The *ZED Explorer* shows the two videos captured by the cameras side-by-side. Both tools lets you record video and images. Two other tools exist for diagnostics and file manipulation. The SDK is available from their website.

Source:

`https://www.stereolabs.com`

## 2.15 JetPack

JetPack stands for *Jetson Development Pack* and is the Jetson tool for flashing and installing operating systems and all the software tools required for development on the Jetson Embedded Platform. JetPack is installed and launched from a host computer and connects to the Jetson platform over USB and Ethernet.

## 2.16 Linux for Tegra: L4T

L4T stands for *Linux for Tegra* and is a driver package for the TX1 including the Ubuntu OS compiled for the development board. You use JetPack to install the L4T components you need.

### 2.16.1 Ubuntu and Xubuntu

Ubuntu is a popular Debian based open source Linux operating system, and is widely used in development applications. We are aiming at making all modules Linux compatible because Ubuntu is reliable, efficient and most importantly open to be customized. Xubuntu is Ubuntu with the desktop environment XFCE which has a smaller footprint than the default Unity. In my specialization project I explained why I use Xubuntu.

### 2.16.2 Htop

Htop is an advanced process viewer for Linux. You can monitor and terminate processes from a terminal, which makes it very useful for debugging over ssh.

## 2.17 The Cyborg Server

A Cyborg server is being set up and will be user for heavier computational tasks, like generating the troll face.

### 2.17.1 Slack, Box and Github

To communicate and share files between project participants we use the services Slack, Github and Box. Slack is aimed at discussion and communication, you can discuss in groups or send direct messages and share documents and images. Box is a file sharing cloud service where participants can synchronize with the shared folders and files. All files relevant to the project can be found in the projects Box folder or on Github.

## 2.18 The MEA Signals

Multielectrode arrays are used to collect neural signals from the biological cells. The MEA signals are the gathered data which will be sent to the Cyborg from the lab.

## 2.19 Controller node

The controller node is a ROS module that acts as the state machine for ROS modules that can interfere with each other and that need concrete boundaries. These modules require control of the Cyborg in some way. The first version of the controller node was completed by Thomas in February of 2017.

Chapter 3

# How to .. Network on the Cyborg

## 3.1 The Network Architecture

The architecture is shown in Figure 3.1, with two uncertainties:

- It is uncertain whether the Cyborg server will communicate with the Cyborg over the school network or over the Internet. This depends on the next uncertainty:
- We still do not know for a certainty whether we can use the school network to access the internet, or whether we need to use a 4G mobile network.

I designed the architecture to work around both uncertainties. This will also allow us to test whether the problems with the school network is persistent, while still having the option to use mobile networks with very small changes on the router.

## 3.2 Configuring the OpenWRT Firmware

To access the router i recommend using the web based software by typing the routers IP in the search bar of a web browser once you are connected to the Cyborg's subnet. The router has the standard router IP adress: `192.168.1.1`

I gave the Cyborg's embedded computers static IP's based on Figure 3.1. The DHCP lease configuration is shown in Figure 3.2.

To configure the DHCP leases you need to navigate to the sub-menu shown in Figure 3.3.

### 3.2.1 Connecting the Router to the Internet

**Wireless**

Follow this step by step guide to connect the router to a wireless network. The procedure is the same whether it is a school network or a mobile network.

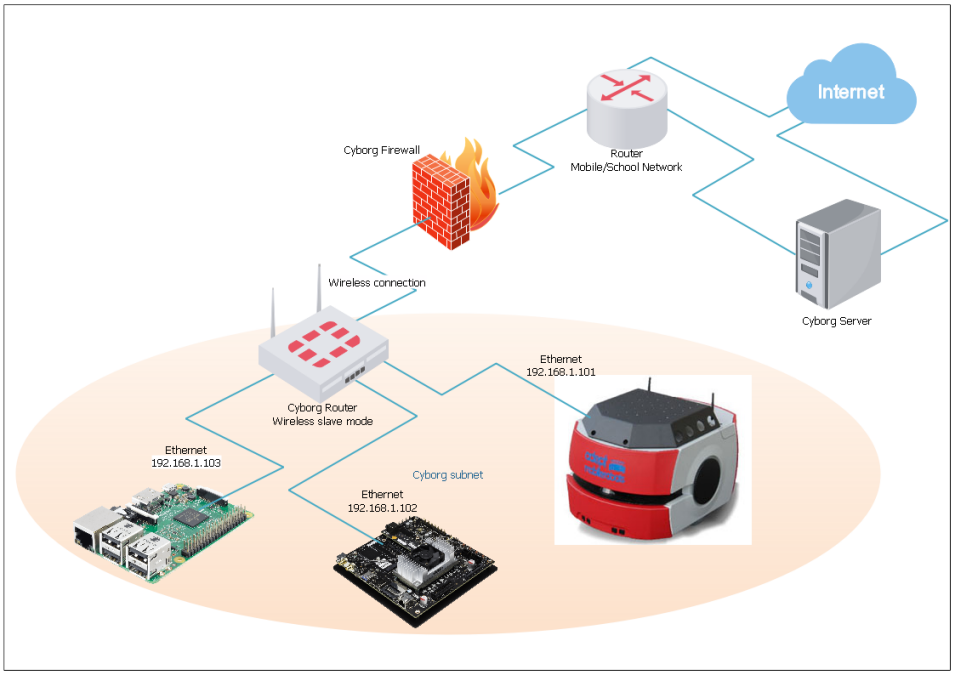


**Figure 3.1:** The network architecture. The Cyborg will either communicate with the Cyborg Server through the school network or over the Internet. MEA signals will be accessed over the internet.

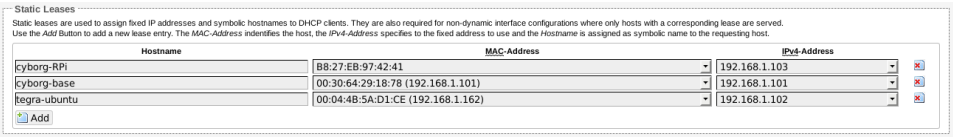


**Figure 3.2:** The DHCP lease configuration used to set static IPs.

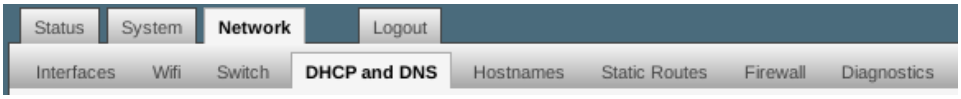


**Figure 3.3:** The sub-menu for configuring the DHCP leases.

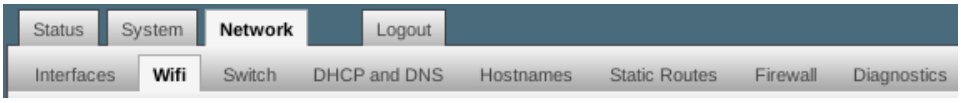


**Figure 3.4:** The sub-menu for configuring a wireless connection.

1. Navigate to the Wifi Sub-menu. Figure 3.4 show the sub-menu for configuring wireless connection, both for server and client modes.

2. Scan for all available wireless networks by clicking the scan button for the **wl0** controller on this page. Figure 3.5 shows the correct button.

3. A list of available networks will appear. Click **Join Network** for the network you want to join.

4. You can make changes to the setting if needed, but entering a password should suffice.


**Figure 3.5:** To connect to a wireless network, click the scan button for the **Broadcom Wireless Controller**.

#### Wired

To give the Cyborg Internet access over a wired connection you can connect the router to the switch on the Cyborg office. This is very useful for large updates and downloads to save mobile bandwidth.

## 3.3 Navigating the Cyborg's Subnet

I wanted to implement a design that would allow easy access to all the Cyborg's computers. By using a subnet on the Cyborg I did not have to set up any port forwarding or DNS subscription; we can simply access the Cyborg with an Ethernet cable or possibly over wireless. To access the cyborg base over ssh you only need to connect to the subnet with an Ethernet cable and type:

```
ssh cyborg@192.168.1.101 -X
```

The same command works for the other embedded computers and their IP's, where `cyborg` must be changed with the appropriate user for that computer. The `-X` parameter allows opening graphical interfaces over the ssh connection, for example a text editor or Mapper3.

For accessing and editing files you can also use a file manages in Linux. Look for a option similar to "Connect to a server", and use the ssh option.

### 3.3.1 Wireless Debug Connection

It might be possible to access the Cyborg's subnet over wireless while at the same time connecting to the internet over wireless. The WRT54GL has two antennas, but whether we can use client and server mode over wireless at once is still unsure. The reason why this would be desired is that we could debug and connect with MobileEyes without using a Ethernet cable. The solution to achieve this may be as simple as adding a wireless USB


**Figure 3.6:** I added a label to the router with the current static IPs.

dongle to the USB port on the USB port on the router and possible install a package for it. For now we can use an Ethernet cable to connect to the subnet, but a wireless option should be explored in the future.

Chapter 4

# How to Find Files on the Pioneer LX and the Raspberry Pi

## 4.1 Files on the Pioneer LX

### 4.1.1 Aria Files

Some important files and folders:

```
/usr/local/Aria/
/usr/local/Aria/examples/
/usr/local/Aria/examples/demo
```

The Aria files are located in the `/usr/local/Aria/` folder. The Aria demo file, with which you can control the Cyborg with the joystick, is located at

```
/usr/local/Aria/examples/demo
```

This is a modified file, see my thesis for details.

### 4.1.2 ARNL Files

Some important files and folders:

```
/usr/local/Arnl/
/usr/local/Arnl/examples/
/usr/local/Arnl/examples/arnlServer
/usr/local/Arnl/examples/sickLogger
```

The Arnl files is located in the `/usr/local/Arnl/` folder. The Arnl server can be launched separately if you want to control the Cyborg with MobileEyes.

The `sickLogger` program can be used to log the data from the laser range finder, and creates a `.2d` file that you can use to create a `.map` file with Mapper3. We used this program to make a map when MobileEyes was not working. The created `.2d` and `.map` files, whether created with sickLogger or MobileEyes, is saved in the folder

```
/usr/local/Arnl/examples/
```

### 4.1.3 Catkin Files

Some important folders:

```
~/catkin_ws/
~/catkin_ws/src/
```

When compiling with catkin, your source files should be in `~/catkin_ws/src/` and you run `catkin_make` in the `~/catkin_ws/` folder.

### 4.1.4 Startup-Script Files

The startup-script files:

```
~/start_scripts/sequences.py
~/start_scripts/start.py
~/start_scripts/arnl_start.sh
~/start_scripts/controller_start.sh
~/start_scripts/roscore_start.sh
```

These files are used for starting the communication with the Startup Box and starting the correct sequence of modules based on the user input from the Startup Box.

#### start.py

This file initiate the communication with the startup box and parse the response. This should not need to be modified unless the startup box' USB port number changes.

#### sequences.py

This file contains the correct starting sequences for all the sequences. Modify this to change a sequence or add a new one.

#### The shell scripts

The shell scripts have to be used to initiate the correct setup for ROS and catkin upon opening a new terminal to run the modules. If the correct `source` commands are not present in the shell scripts, the modules will fail to start:

```
source /opt/ros/kinetic.setup
source ~/catkin_ws/devel/setup.bash
```

If you follow the ROS tutorial these commands will have been added to run automatically upon opening a terminal, but for some reason it is not the case when a terminal is opened from a Python script. Hence these shell scripts are necessary.

### 4.1.5 Startup Box Files

All files related to the Startup box can be found in the folder below along with a list of some important files:

```
~/startbox/
~/startbox/main.c
~/startbox/flash
~/startbox/README
```

## 4.2 Files on the Raspberry Pi

### 4.2.1 Catkin Files

The location for the Catkin files on the Raspberry Pi is the same as on the robot base. In the source folder you will find the files for the LED demo.

### 4.2.2 Startup Script

I added a startup script to the Raspberry Pi in the same fashion I did on the robot base. The script is located at:

```
~/startup.sh
```

This script is run by the desktop environment when you log in. You have to make sure that the desktop environment will run at startup, which can be done with:

```
sudo raspi-config
```

You can edit the following file to enable auto log in for the desktop environment, followed by the next command which will update the configuration:

```
sudo vim /etc/lightdm/lightdm.conf
sudo dpkg-reconfigure lightdm
```

There may be other files on the Raspberry Pi I don't know about, so check the EiT reports.

Chapter 5

# How to Re-install an OS on the Pioneer LX

## 5.1 The Checklist

There is a good chance that someone will need to re-install the OS of the robot base in the future. If you for some reason need to re-install the OS on the Pioneer LX, there is a few critical things to remember:

- DO NOT use and dashes (-) in the username or password. If you do this you will not be able to use MobileEyes with the Cyborg. There might be other symbols MobileEyes will not accept, so be smart when choosing username and password.
- During the installation process you should select automatic login without password. If you do not do this you will need to use a keyboard whenever you start the Cyborg or write a configuration file in the OS for this.
- The startup script for the startup box must be added to the list of programs that start automatically on startup.
- For Aria to be able to connect to the peripherals you must make one specific Aria file and give your Ubuntu user access to the **dialout**.
- Dependencies and programs must be installed.
- To state the obvious: make backups of all modified files, scripts, code and maps and place them in the same location.

You can install a new OS from a USB like you would on any regular personal computer.

## 5.2 How to make the startup script auto-run at login

Small changes to the procedure may be needed for other Ubuntu versions. For Xubuntu 16.04 the following proceedure works:

1. Open the **Settings Manager**.
2. Go to the **Session and Startup** configuration.
3. Under the sub-menu **Application Autostart** you can add and configure what applications and scripts will run at start.
4. The startup script I created is added under the name **start_script**. The command I used can be found below.

#### The Correct Command Entry

I wanted the script to run in a new terminal window so it would be easier to terminate it. If it is not launched in a terminal window, we would have to use a process manager like Htop to terminate the script in stead of just closing the terminal. To run the script in a new terminal I enter the following command in the command entry for the **start_script** discussed above:

```
xfce4-terminal --command "python /home/cyborg/start_scripts/start.py"
```

## 5.3 How to add User to Dialout

Use the following command, but change `MyUser` to the user name of the Cyborg's Ubuntu user:

```
sudo adduser MyUser dialout
```

## 5.4 Installing Software

The Software checklist is the following:

- Aria
- Arnl
- Controller node installation script by Thomas.

Aria and Arnl can be downloaded from the Mobile Robots web page, and the installation script is available on the NTNU Cyborg Github page. Do not bother running the script, but rather run each individual command one by one.

Chapter 6

# How to Start the Cyborg and use the Start Box

## 6.1 Step by Step

1. Turn on the Pioneer LX robot base. Ignore the small screen on the robot base.
2. Wait for the menu to appear on the start box.
3. Use the two buttons on the left to select the correct start sequence.
4. Press the right button to select.

The startup box must be connected to one of the Pioneer's USB ports. The startup box gets both power and communication over the USB cable. In the Normal Start mode, the Cyborg expects to start on the footprints in Glassgrden, shown in Figure 6.1, facing down the hall.

## 6.2 Possible Issues

The startbox could have been assigned a different serial usb port than the startup script expects. If this is the case use

```
ls /dev/ttyUSB*
```

before and after unplugging the startbox to find which `/dev/ttyUSB` port it has been assigned. `/dev/ttyUSB0` to `/dev/ttyUSB11` should be used by the Pioneer's internal peripherals, and the startup box will usually be assigned `/dev/ttyUSB12`. Update the startup script with the correct port. This has never been necessary to do so far.

There should be a way to identify the Arduino in the Startup Box with its ID number, thus always finding the correct port, but since it so far has always been assigned the same number I did not look into this.

**Figure 6.1:** The correct start position for Normal Start. You can use *localize to point* from MobileEyes to start from another position.

## 6.3 How to Update the Startup Box' Software using the Makefile and Avrdude

To be able to update, compile and upload the Start Box' software, you need the following packages that can be downloaded using the Synaptic Package Manager in both Ubuntu and Debian:

```
avrdude
avr-libc
binutils-avr
gcc-avr
```

Their names are slightly different in arch linux:

```
% http://omegav.no/wiki/index.php/AVR_on_linux
```

The list of sequence names can be found in `main.c`. This is the only file you need to edit to update the menu text, unless you need to change the usb port in the Makefile. Remember, you can only fit 15 characters per line. To compile and flash the Start Box you only need to run the following command in the folder:

```
./flash
```

I add the avrdude flashing command here in case it would ever be needed (it should not):

```
avrdude -v -p atmega328p -c arduino -P /dev/ttyUSB12 -b 57600 -D -U flash:
    w:output.hex
```

Chapter 7

# How to Connect to the Embedded Computers on the Cyborg

You can use ssh to access all the Cyborg's computers once connected to the Cyborg's subnet. If for some reason this does not work, check that the SSH server software is installed. Usernames and passwords are available in the Box folder.

## 7.1 Step by Step

For Linux:

1. Use an Ethernet cable to connect to the Cyborgs router (NOT the port that says internet..)
2. Open a terminal and run the command: `ssh cyborg@192.168.1.101 -X`
3. (optional) Change the user and IP in the command above to the appropriate embedded computer you want to access.
4. The `-X` in the command is optional, with this you can open graphical programs over your ssh connection. Try for example to run `libreoffice`. You will then open Libre Office on the target computer, but it will appear on your computer as if you opened it on your computer.
5. Note: The startup script for the Startup Box cannot be run over ssh.

Windows users should be able to connect if they use Putty or something similar, but I have not tested this.

Chapter 8

# How to use the Cyborg Subnet to Debug

## 8.1 SSH and Htop

See Chapter 7 for how to connect with SSH. Over an SSH connection you can for example run scripts and start ROS modules. Once you have connected to the Pioneer with ssh, you can run Htop with the command:

```
htop
```

In Htop you can view and terminate processes.

## 8.2 MobileEyes

You can run MobileEyes in Linux, but as of June 2017 it does have several bugs I discovered. Mobile Robots have been notified, but they never said whether they where going to fix them. I therefore recommend running MobileEyes from a Windows laptop. Connect the laptop to the Cyborg's subnet. When you log in with MobileEyes an Arnl server must be running on the base for you to connect. Starting the Normal Start sequence will for example start the ros-arnl module, but you can also just run the Arnl server program.

Things to know about:

- That the Cyborg is in the correct position in the map. If not, you can use the **localize to point** tool to select where the Cyborg is.
- That the robot is in **Safe Mode**. This safe mode is different than the one for the Aria demo, and the robot base may drive within centimeters of obstacles.
- You can drive the Cyborg from MobileEyes, even when the controller node is running.

- You can both disable the motors and stop the Cyborg from MobileEyes.

Chapter 9

# How to Simulate the Controller Node on your own Hardware

## 9.1 Install Dependencies

Thomas, who made the controller node, also created an installation script for all the dependencies. If you try to run the script you will most likely get an error, causing the script to exit before it finishes. I recommend copy-pasting the commands one by one and running them in your terminal. You should use Ubuntu 16.04. Thomas used a Virtual Machine to simulate, so that is an option if you do not want to install an extra Ubuntu partition.

The installation script can be found at:

```
https://github.com/thentnucyborg/cyborg_setup
```

## 9.2 Download and Compile the ROS Modules

If you followed Thomas' guide you should now have installed ROS and created a catkin work space in your home folder. First, navigate to the catkin source folder:

```
cd ~/catkin_ws/src
```

Once in this folder run each of the following commands one by one to clone the Github repositories for all the modules you need.

```
git clone https://github.com/thentnucyborg/cyborg_command
git clone https://github.com/thentnucyborg/cyborg_ros_controller
git clone https://github.com/thentnucyborg/cyborg_ros_conversation
git clone https://github.com/thentnucyborg/cyborg_ros_idle
git clone https://github.com/thentnucyborg/cyborg_ros_music
git clone https://github.com/thentnucyborg/cyborg_ros_navigation
git clone https://github.com/thentnucyborg/cyborg_ros_text_to_speech
git clone https://github.com/thentnucyborg/executive_smach_visualization
git clone https://github.com/MobileRobots/ros-arnl
```

Now you can compile the sources using catkin:

```
cd ~/catkin_ws
catkin_make
```

## 9.3 Simulating

I assume that you by now have done the ROS tutorial. If not you might want to go through it before continuing. The following is an extract from my thesis.

First you need to start the MobileSim software, or else the Arnl server will fail. This software imitates the sensors on the robot base, and allow the Arnl server to believe it is in a real environment. MobileSim will ask for which robot type you are using and what map you want to simulate. In MobileSim you can choose what actual position to simulate.

To run the ROS modules I open three different terminals and run these three commands, one in each of them:

```
roscore
rosrun rosarnl rosarnl_node
roslaunch cyborg_controller controller.launch
```

I use the same process as for the "Normal Start" in the start script. When simulating, however, I can run MobileEyes from the same computer that I simulate the controller node on. If so I enter `localhost` for the robot's IP address and do not need to enter username or password. It was when I did this that I experienced many issues with MobileEyes. I also added a generic, royalty free music in my home folder and named it `music.mp3` which is the file played in the Cyborg's idle state. The simulation will continuously update the `graph.png` image in your home folder as it changes states.

Royalty free music:

```
www.bensound.com/royalty-free-music/track/ukulele
```

The where only one or two more things I had to change in his design. In the `statemachinemonitor.py` file, he has used his home folder on his computer, and not the generic home folder address which is the tilde symbol. I am unsure if I also had to create an empty file before running the controller node the first time or it would fail, but that may have been the music file. If you look at the error message it should be obvious.

```
/cyborg_ros_controller/src/statemachinemonitor.py
```

## 9.4 Using the Debug Script for the Controller Node

I recommend reading about this in Thomas' thesis, he refers to the debug script as the command line tool. You can find his thesis in the Box folder.

# Chapter 10

# How to Create and Configure Maps

## 10.1 Create a New Map

You have two options for creating a map. Either use MobileEyes or the sickLogger program.

### 10.1.1 MobileEyes

In MobileEyes you can use the **Tools ¿ Map Creation ¿ Start Scan** tool. As of June 2017 this does not work on Linux.

### 10.1.2 sickLogger

An option that does not require any other computers is the sickLogger program. You can start any setup that allow you to control the robot base, for example the Aria demo, and then start the sickLogger program. Move around the area you want to scan and then sickLogger will save the scan file (a `.2d` file). See Chapter 4 on file locations.

## 10.2 Configure a Map with Mapper3

Both scan files and map files can be opened in Mapper3 to be configured. With Mapper3 you can also configure a map currently on a robot by connecting over a network.

Chapter 11

# How to use the TX1 and its Tools

## 11.1 Introduction

I wrote this guide for my specialization project in the fall of 2017. If you intend to work on the TX1 I recommend signing up for the NVIDIA Development Program, it will give you access to all their documentation and tools.

`https://developer.nvidia.com/developer-program`

The following is an extract from my specialization project and explains how I installed the tools on the TX1. Make sure you have all the hardware you need for the process. More details can be found once you sign up to the NVIDIA Developer Program and check if there has been updates to this process, including newer sofware versions.

## 11.2 Installing L4T on the TX1

Jetpack 2.3.1 is only supported by Ubuntu 14.04 x64, so I made a fresh installation of Ubuntu 14.04 and proceeded to download Jetpack. Before I could flash L4T to the TX1 I had to install JetPack and make the necessary installations. I followed the guide presented by NVIDIA on their developer's site.

Installation guide:

`http://docs.nvidia.com/jetpack-l4t/#developertools/mobile/jetpack/l4t/2.3.1/jetpack_l4t_install.htm`

The installation guide is precise and sufficient as presented on their website, but I want to list the requirements given in the guide here along with a picture of the network setup. I do this to give a better understanding of what I did and for any later students that might find it useful.

Host Platform Requirements:

- Ubuntu Linux x64 (v14.04)

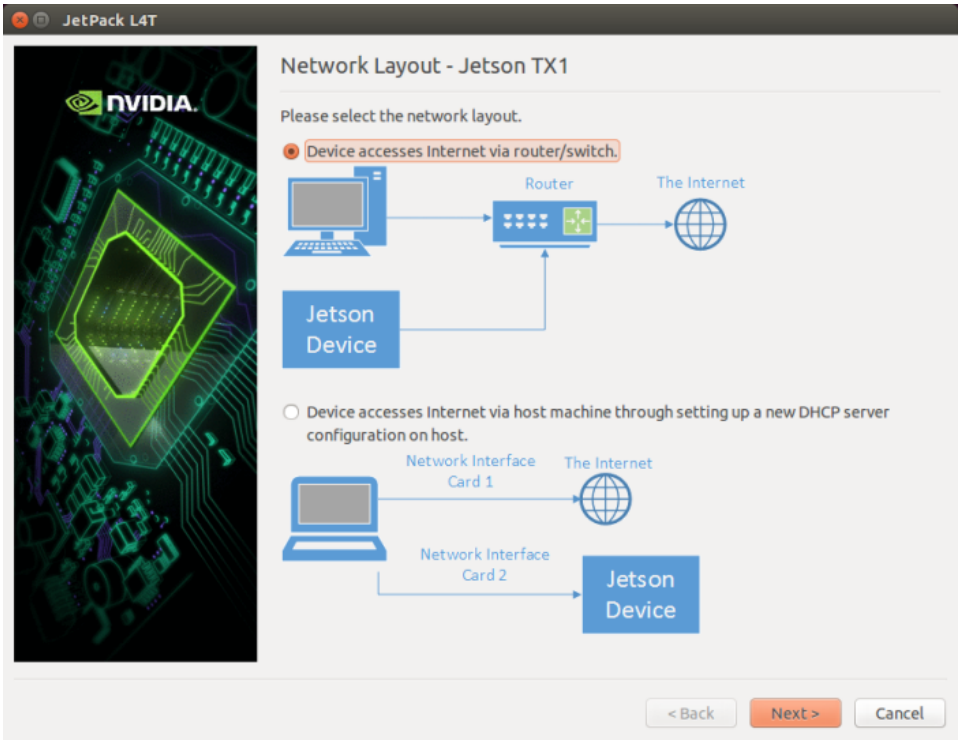


**Figure 11.1:** The possible network setups for using JetPack with a host machine and the TX1 board. From the installation guide.

- Valid Internet connection and at least 10GB of disk space is needed for the complete installation of JetPack.

Target Platform Requirements (Jetson Developer Kit TX1):

- USB Micro-B cable connecting Jetson to your Linux host for flashing.
- An HDMI cable plugged into the HDMI port on Jetson Developer Kit, which is connected to an external HDMI display.
- An Ethernet cable plugged into the on-board Ethernet port, which is connected to either a secondary network card on your Linux host or the same network router providing internet access for the Linux host.
- (Optional) To connect USB peripherals such as keyboard, mouse, and [optional] USB/Ethernet adapter (for network connection), a USB hub could be connected to the USB port on the Jetson system.

These two lists where copied from the installation guide. Since the TX1 board only has two USB ports I used the USB hub integrated in the computer screen when it was necessary. The network setup I used is the top alternative in figure 11.1.

Chapter 12

# How to use the ZED SDK

## 12.1 Introduction

I wrote this guide for my specialization project in the fall of 2017. This is the step by step process presented on the ZED homepage's get started page, but I have including some helpful steps. If you want to use the ZED camera on your own computer, or you for some reason need to re-install the OS on the TX1, you should find this guide useful. The guide is meant for Ubuntu 16.04 on regular computers. The TX1 will have OpenCV and Cuda installed if the TX1's OS was installed correctly using JetPack, and the only necessary step will be to install the ZED SDK for the TX1. **Some small changes may be needed for file names, versions, etc.**

Links and sources for this chapter:

`https://www.stereolabs.com/getstarted/`
`https://developer.nvidia.com/cuda-downloads`
`https://github.com/NVIDIA/DIGITS/blob/master/docs/InstallCuda.md`
`http://docs.opencv.org/3.1.0/d7/d9f/tutorial_linux_install.html`

## 12.2 Installing the ZED SDK and Dependencies

**Step 1:** Connect the camera to a USB 3.0 port.

**Step 2:** Make sure the lates USB 3.0 drivers are installed. Updating your linux system will suffice.

**Step 3:** Install CUDA and the latest NVIDIA drivers. I recommend using the `Additional drivers` in Ubuntu and select the newest drivers since this is the easiest way, but for the newest drivers see the NVIDIA home page. Check the ZED homepage for which CUDA is currently recommended. To install CUDA download the correct .deb file from NVIDIA and open a terminal in the .deb file's folder and execute the following commands:

```
sudo dpkg -i cuda-repo-ubuntu1604-8-0-local_8.0.44-1_amd64.deb
apt-get update
apt-get install cuda
```

**Step 4:** Compile and install OpenCV. Install dependencies by running:

```
sudo apt-get install build-essential
sudo apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-
    dev libavformat-dev libswscale-dev
```

Navigate to an appropriate installation folder in the terminal and clone the newest OpenCV repository:

```
git clone http://github.com/opencv/opencv
```

Navigate to the `opencv` folder you just cloned and make the build directory:

```
cd opencv
mkdir build
cd build
```

Use cmake to configure (if you mess something up in the next steps, you can use `make clean` to start over):

```
cmake -D CMAKE_BUILD_TYPE=Release -D CMAKE_INSTALL_PREFIX=/usr/local
    ..
```

Build with make:

```
make -j7
```

Install OpenCV:

```
sudo make install
```

**Step 5:** Run the ZED SDK setup tool to install the ZED driver, tools and samples.

## 12.3 Further Reading

See Amund's thesis for more on the ZED Stereo Camera. You might also find some useful information in my specialization project.