Projectteam Zebro onboard navigation system

Part 1

Thesis

Version 2.0

June 22, 2015

Delft University of Technology

Supervisor:
Chris Verhoeven

Abstract

This document describes the design process towards a functional Onboard Navigation System board, and the onboard communication with other modules in the Zebro Explorer. Furthermore the design process of the communication outside the Zebro consisting of a graphical user interface and the wireless communication is described.

Alex Janssen, 4231333
Peter Stijnman, 4215788

TU Delft Delft University of Technology

ZebRo

Challenge the future

Version	Date	Comments
0.1	1-05-2015	Backbone of thesis
0.2	4-06-2015	Start of actual writing
0.5	5-06-2015	Further writing plus intergrating parts
1.0	19-06-2015	Finished the first complete draft
1.2	20-06-2015	First spel check done by Peter Stijnman
1.4	21-06-2015	Added power and financial budget and Zebro logo
2.0	22-06-2015	Final draft

Contents

1 Introduction

The Zebro project is an initiative from the Technical University Delft in cooperation with the Rotterdam University and the Hague University. The goal of the project is to make a modular and robust robot which could be used as a mars rover. The idea for modularity is that single parts of the robot van be updated, hard- and software. The main noticable trait of the Zebro are its C-shaped legs instead of traditional wheels. These legs give it an advantage over other typical mars rovers that it does not get stuck and can climb over objects easily.
For extra encouragement the Zebro project is taking part in the European Rover Challenge (ERC) which is organized by the European Space Foundation and regionalne centrum naukowo-technologiczne. The challenge it's end goal for the team is to produce the new modules. New modules that need to be made are motor drivers, a robotic arm and a new communication system for example.
This thesis is about the latter, the communication system and on top of that autonomous walking. This includes the communication from the Zebro to a base station, which is a computer, and the communication between the different modules within the Zebro.
There is already a backbone PCB on which all the different modules can be realized. This means that our own PCB can be placed upon the backbone. Through this backbone it is possible to communicate with the different subsystems. On the backbone there are two microcontrollers, one for the power management (BPU) and one that links all the modules together (ZPU). This ZPU will be used by our module to extract from and send data to.
On the Zebro a new module has to be manufactured as well as an interface for on the computer from which the zebro is controlled. The simple overview of the system is seen in figure 1. Antennas need to be selected along with the peripherals needed to complete the ERC. All these peripherals will be discussed in the chapters specifications and design choices.

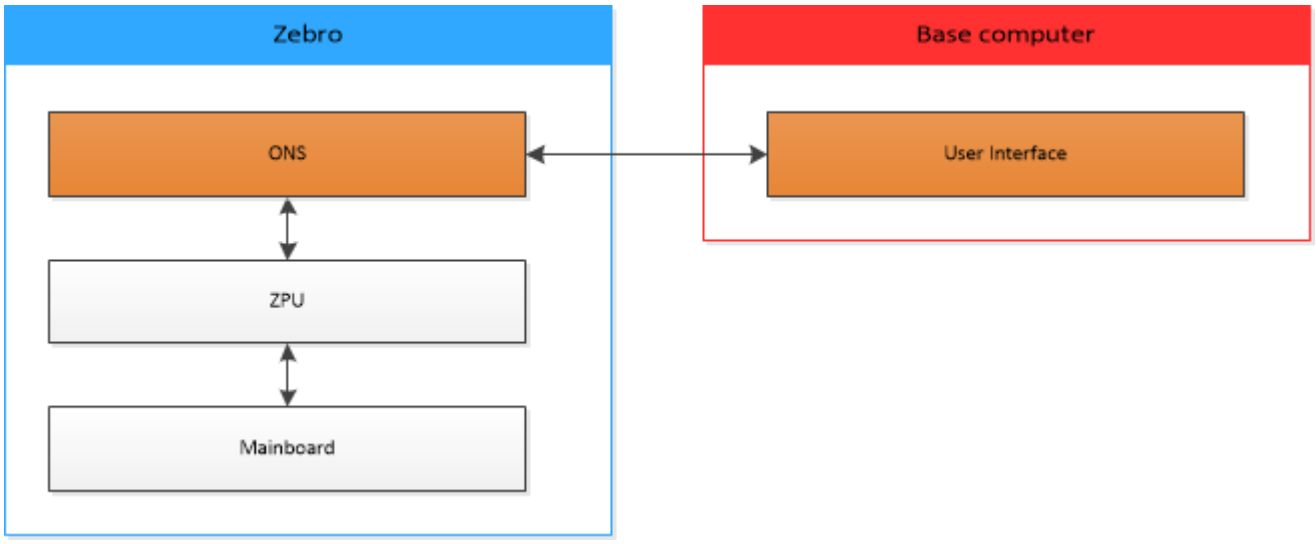


Figure 1: A simple overview of the total system.

2 Specifications

Since the Zebro explorer is competing in the ERC it will need to fufill certain tasks. The main task our group is focussing on is the so called 'blind traversal'. In this challenge the robot needs to navigate from a starting point to the next 3 locations, these are given in GPS coördinates. Between location 1 & 2 there is a 2m wide obstacle blocking the shortest path. If the robot reaches the last location within 10m, indicated by the green circle, the challenge is completed. A map depicting the challenge is seen in figure 2.

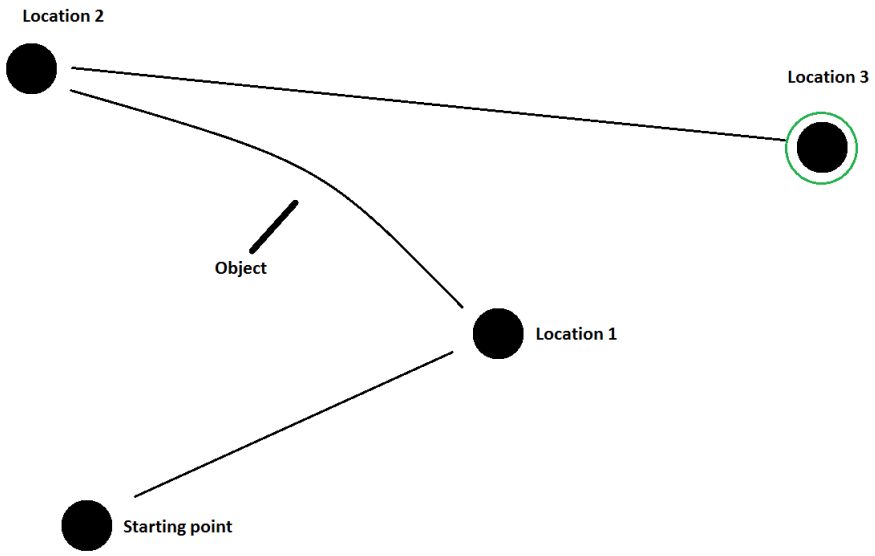


Figure 2: A map of the challenge called 'The blind traversal'.

This will result in 100 points for the competition. Bonus points are acquired by walking from the second location to the third autonomously. Only the position of the robot can be sent to the Zebro explorer, the rest of the computing has te be done onboard. This will net the team an extra 25 points. When 2 navigation techniques are used that complement each other agian an extra 25 points will be rewarded to the team. And the last 10 bonus points will be awarded when the Zebro explorer can return back to the starting point after finishing the challenge.
The first thing that is absolutely necessary is being able to send and receive data to and from the Zebro explorer. This communication system will be tested by the judges. The robot will be set to 1000m from the base station and there should still be communication possible. The second feature the robot should posses is being able to receive GPS coördinates about its position. For this a GPS module is needed ofcourse, and more specifically one that has a better accuracy than 10m. Further the robot should be able to be manually controlled by the user, it is preferable to have a video feed to see where the robot is heading. The video feed should be of an okay quality, not full HD since the signal needs to be transmitted over 1km and the amount of frames is important as well. The video feed should not feel laggy thus the amount of frames per second needs to be around 20 at least.
To score the bonus points we need an extra technique for navigation and since the robot needs to be able to dodge objects a system that can detect these objects would fit perfectly. Together with the GPS module the robot has two navigation techniques which complement each other.
In order to walk autonomously from the second location to the third a code needs to be written on the microcontroller of the Zebro explorer itself. The last 10 points however are just a matter of time, being quick enough to get back to the starting point.
The most important specification is that the design needs to be as modular as possible. This is

because when there is a better solution to a problem that it can be easily incorporated in the rest of the design. Plus when in Poland something goes bust it can be easily swapped out with spare parts.
To summarize the specifications of our part, ZP_EX_ONS_XX:

1. A communication system that is able to send and receive over 1000m.
2. A GPS module that gives the location of the Zebro explorer within 10m.
3. A system that can detect objects in the surrounding at a distance of 10m.
4. A camera with 240p quality and around 20 fps.
5. A user should be able to control the Zebro explorer manually.
6. The Zebro explorer should be able to walk autonmously from one point to another.
7. An user interface containing all the necessary information to control the Zebro explorer.
8. A system where peripherals can be switched out, with either new ones or improved versions.

So specification number 1 of our module would be called ZP_EX_ONS_01 and so forth.
These subsytems are seen in the function tree in blue in figure 3. The orange blocks are possible solutions to implement these subsystems.

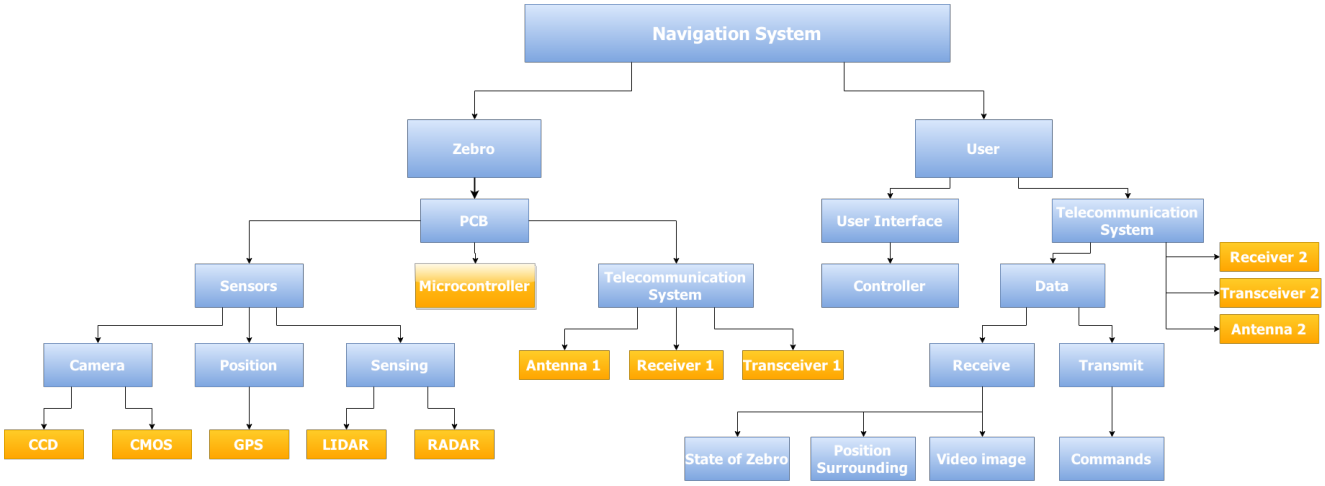


Figure 3: Function tree. The blue blocks indicate the different subsystems and the orange indicate the different possible solutions.

The subsystems that are part of this thesis are the complete user side, the telecommunications system on the Zebro as well as the PCB and the cameras. The other parts plus the autonomous walking algorithm is explained in the thesis onboard navigation system: part 2. This means that specification number 2, 3 & 6 is not discussed in this thesis.

3 Design Choices

3.1 Communication system

For the ERC the Rover will need to communicate to mars, but this will be, of course, scaled down to reasonable distances on earth for the challenge, namely 1000 meters. In order theory in the form of a link budget analysis and the hardware part which is mainly our two transceivers. One on the mars rover itself, and one on the user side, and from that it is possible to decide which module will be best for this task.

3.1.1 Link budget analysis

For communications from point to point a link budget analysis needs to be calculated.

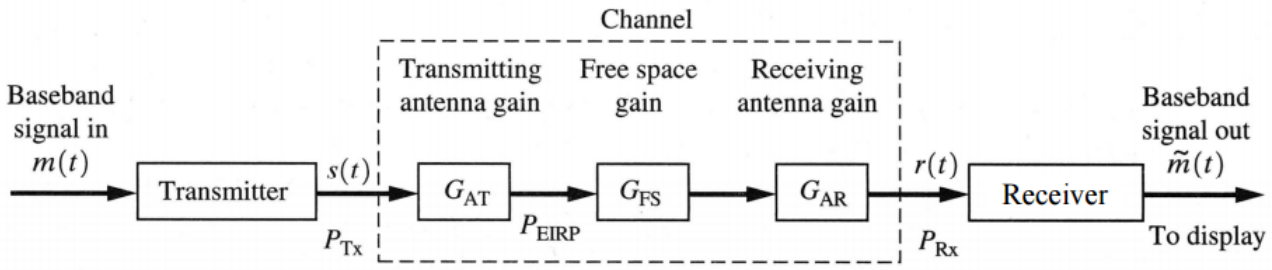


Figure 4: Link Budget Analysis [9]

Here the Transmitter will send a modulated signal, which at the receiver side will be influenced by a couple of factors. The P_{tx} is the output power by the sender, and P_{EIRP} is the total power including the antenna gain transmitted by the sender. The power loss after transmitting is called free space gain G_{FS}. This loss can calculated at the output of the receiving antenna P_{rx} according to Equation 1. Note that all the variables are in dB in this equation.

$$(P_{Rx})_{dB} = G_{AT} - G_{FS} + G_{AR} + P_{Tx} \tag{1}$$

3.1.2 Free space gain

The free space gain is the gain that happens when the wave travels from point to point. The free space gain can be calculated G_{FS} using equation 2.

$$G_{FS} = (\frac{\lambda}{4\pi d})^2 \tag{2}$$

Where $\lambda = c/f$ with c the speed of light which is approximately $3 \cdot 10^8 m/s$. Since it is more natural to see this as loss instead of a gain, and to work in dB rather than a factor. The equation for the free space loss, $(L_{FS})_{dB}$ becomes:

$$(L_{FS})_{dB} = 20\log(\frac{4\pi d}{\lambda})dB \tag{3}$$

3.1.3 Antenna gain

The antenna gain is approximated by Equation 4.

$$G_A = \frac{4\pi A_e}{\lambda^2} \tag{4}$$

With this approximation different antenna options can be viewed with the gains in the table below.

Table 1: Antenna gains and effective area's

Antenna Type	G_A	A_e
Isotropic	1	$\lambda^2/4\pi$
Half-wave dipole	1.64	$1.64\lambda^2/4\pi$
Horn, mouth area A	$10\text{A}/\lambda^2$	0.81A
Parabola, face area A	$7\text{A}/\lambda^2$	0.56A

As can be seen the Isotropic antenna has a gain of 1, this is because it sends an equal amount of power, P_{Tx}, in all directions thus the isotropic antenna is used as a reference for the antenna gains. This gain means that the antenna does not send power in all direction. Resulting in more power sent in a specific direction rather than less power in all directions.

Isotropic antenna

An isotropic antenna has as a disadvantage a low gain, but it has some advantages as well. One of the advantages is that it radiates in all directions with the same power. All sides will be covered with the sent signal. The far-field pattern showing this can be seen in figure 5.

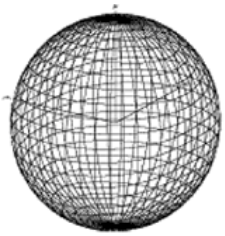


Figure 5: Isotropic far-field

Another advantage of the isotropic antenna is that it is smaller than other antennas. Together with the advantage of its far-field pattern it is an easy to implement versatile antenna. The length of this antenna is dependend upon the wave length of the signal. The length can be calculated using equation 5. Here L_ant is the antenna length in meters, and f the frequency of the signal in MHz.

$$L_{ant} = \frac{300}{f} \tag{5}$$

Half-wave dipole antenna

A Half-wave dipole antenna has an antenna gain G_A of 1.64. This is higher than that of the isotropic antenna, but this goes at the cost of the coverage area as seen in figure 6.
Also the antenna is a lot bigger than an isotropic antenna. It takes up much more horizontal space instead of just vertical. An example can be seen in figure 7. Horizontal length of the antenna can calculated by forumla 6.
The lenght of the antenna, as can be noticed from the name, is calculated using equation 6.

$$L_{ant} = \frac{\lambda}{2} \tag{6}$$

Parabolic antenna

Another much used option is a parabolic antenna. This antenna has a high gain G_A as can be calculated with equation 7. Here r is the radius of the antenna in meters.

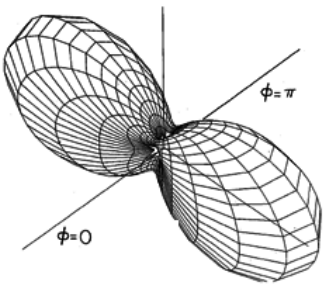


Figure 6: Half-wave dipole far-field

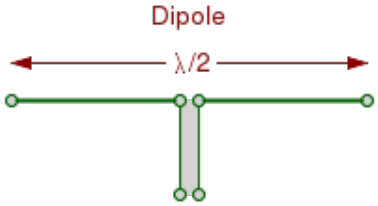


Figure 7: Half-wave dipole form

$$L_{ant} = \frac{7\pi r^2}{\lambda^2} \tag{7}$$

It can be easily conducted that the gain of a parabolic antenna is dependend on its size, and thus has it has the potential to have a much larger gain than other antenna's. On the other hand, this type of antenna needs to be directed towards the signal because of its coverage area as seen in figure 8.

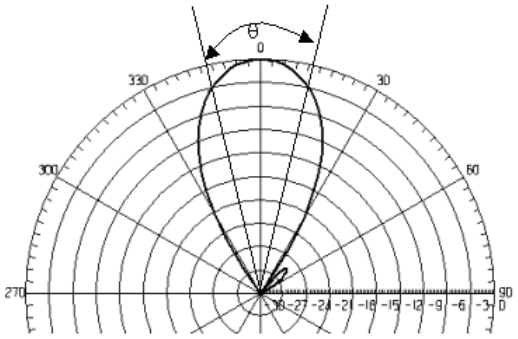


Figure 8: Parabolic antenna beam width

When not in need to have your signal emitted all around the sender, or you know which direction the signal is coming from this is the best option. With the ability to have a high gain, the signal strength will be much stronger. But it has the downside that is it big and not very suited for mobility.

Antenna choice

The rover will be walking around, and it is not always possible to precisely know how it is oriented towards the base station. There is a GPS and compass inside so it could be possible to mount a small parabolic antenna but this has some big disadvantages. First of all it needs to be focussed always to the right direction, which requires a motor and power. Second

the antenna is big, and the robotic arm on the top can be in the way sometimes which will decrease the signal power or even lose signal when it is in the way. This is why a parabolic antenna is not an option. The Half-wave dipole antenna looks more like a Isotropic than a parabolic does, but this one shares the same disadvantages of being very big, and not covering all sides around it. Even if it does, the robot could be climbing a hill, which would mean the signal needs to come from the top of the antenna, which is not possible with a parabolic or half-wave dipole antenna. This is why a Isotropic antenna is needed on the Zebro explorer itself.

This has as a consequence that there is low power gain at the Zebro side, and since the zebro needs to keep low powered, this cannot be compensated on the Zebro side. At the user side however there is room and possibility for a larger parabolic antenna and high sending power. Since it is known where the Zebro is at all times using GPS it is possible to aim the parabolic antenna at it. By using a high power sender and parabolic antenna at the user side and a low power isotropic antenna at the Zebro side, it is possible to compensate for the low gain of the isotropic antenna and have a strong connection. When taking the size of the parabolic antenna at the user side to be 0,5 meters in diameter, resulting in a gain of XXdB.

3.1.4 Protocol and frequency

The choice was made to use 2 communication protocols, Wifi and Zigbee. All the essential data, like commands and responses from different modules will be sent using zigbee protocol. The wifi protocol will strictly be used for transferring camera images. This is done because of serveral reasons. The first reason to do this in 2 seperate systems has to do with the fact that both protocols work at different frequencies. Zigbee is able to work at 868 MHz which is an open band in Europe, and Wifi works at 2.4 GHz. An other reason has to do with the free space loss as shown below.

Free space loss

When using wifi the free space loss can be calculated using equation 3, with the wavelength $\lambda = c/f$ where $f = 2.4GHz$ and $c = 3,0 \cdot 10^8 m/s$.

$$\lambda = \frac{3,0 \cdot 10^8}{2.4 \cdot 10^9} = 0.125m \tag{8}$$

Using equation 3 the free space loss using wifi can be calculated, transmitting over at least 1000 meters. Since this is free space loss, and there is influence of the weather and other objects. A factor 2 is chosen to be sure that there is a connection at that distance. So the distance used in the calculations is 2000 meters.

$$(L_{FS})_{dB} = 20\log(\frac{4\pi d}{\lambda})dB = 20\log(\frac{8000\pi}{0.125})dB = 106dB \tag{9}$$

When using Zigbee which has a frequency at 868 MHz the free space loss, according to equation 11, would be:

$$\lambda = \frac{3,0 \cdot 10^8}{0.868 \cdot 10^9} = 0.346m \tag{10}$$

$$(L_{FS})_{dB} = 20\log(\frac{4\pi d}{\lambda})dB = 20\log(\frac{8000\pi}{0.346})dB = 97dB \tag{11}$$

As is seen there is a factor 9 dB less loss using zigbee which is the same as a factor 8 more power. This makes it possible to reach much greater distances with the zigbee module, and also making the base communication which will run over this protocol more reliable and less prone to disconnecting.

Bit rate

The downside of the Zigbee protocol is the low bit rate of 20 kb/s this is why it is not possible to use just the Zigbee protocol. As can be seen in chapter 3.2.4 the camera has a bit rate of 1 Mb/s, which can not be achieved by the Zigbee protocol. The Wifi protocol 802.11g can work, according to the IEEE802.11 standard, at an average rate of 22 Mbit/s, which is fast enough for this application. For this reason the communication goes over the more stable and reliable Zigbee protocol and the camera images over the faster Wifi protocol.

Zigbee module

For the actual module the choice was made to use the Digi Xbee-Pro 868. This module has a maximal transmitting power of 315 mW or 25 dBm. The reason for using this module is its high power, reliability and the modularity of the module. The 868 MHz band is only allowed to use in Europe. When the Zebro would be used in other world parts, it is possible to simply switch out the module for another Xbee Zigbee module on another frequency. The receiver sensitivity is -112 dBm. This is also really powerfull compared to other modules, but using a safety margin a sensitivity margin of of -100 dBm is assumed where it will work guaranteed. With all these design choices the eventual sending power can be estimated on the rover side when assuming a satalite dish of 0,5 meters diameter on the user side. The gain when using a satalite dish is calculated using the following equation.

$$L_{ant} = \frac{7\pi r^2}{\lambda^2} = \frac{7\pi 0.25^2}{0.346^2} = 11.48 = 10.6dB \tag{12}$$

Since the choice has fallen on a module which can vary in its sending power, the Zebro is highly versitile and able to communicate over a longer distance than just the 1 km depending on its sending power, however this would ask more from the battery. Before choosing a definitif transmitting power, tests need to be done, which are described in the Future Plans chapter.

3.2 Camera

For the camera there was the choice between a CMOS and CCD camera. So first some literature about these two types of cameras.

3.2.1 CCD

When a CCD, short for charge-coupled device, is exposed to light the small photosites on the surface of the CCD capture the light and store it in the form of a charge, as seen in figure 9 .

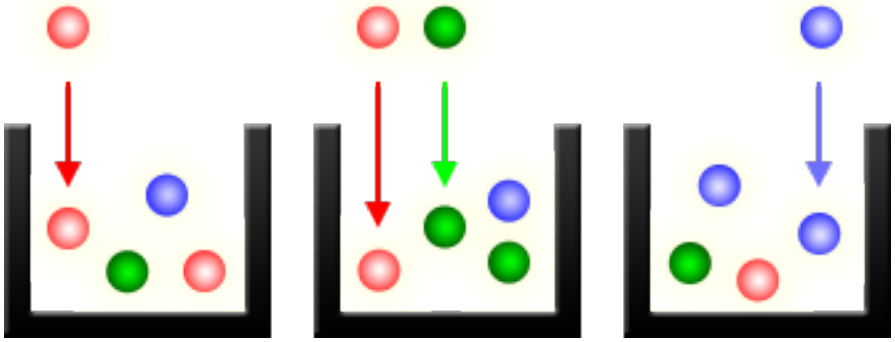


Figure 9: Representation of photosites, charges are collected at the electrodes.

To read out the charge of each photosite a complete row is put into an amplifier one photosite at a time and after that the amplified signal is then passed on to an ADC, analog to digital converter. When the complete row has been read out the row above it moves one down, as do all the rows above, and then that row goes through the amplifier and ADC. This process is done until all rows are read and the image is now digital. This is also where the name comes from , charge-coupled device, because the rows of charges are coupled to the row above them. When one moves down all do. A visual representation is seen in figure 10.

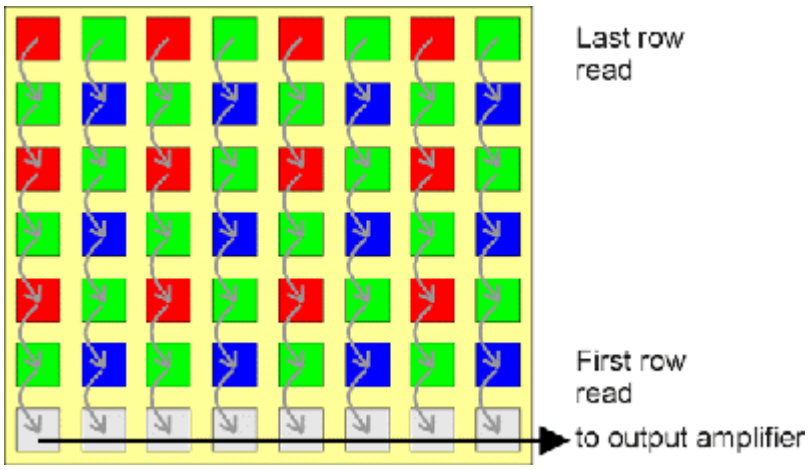


Figure 10: Visual image of how the CCD reads out the array of photosites. This happens one row at a time.

The polysilicon electrodes on the surface of the chip are so small and close together that the charge is kept intact when physically moving from the place where the light was actually captured to the place where the signal is amplified. To achieve this a clock is needed to move all the charges at the same time, this needs to be done by an off-chip (or secondary chip). This is in order to not interfere with the closely packed polysilicon electrodes on the chip.

Because the CCD needs extra circuitry which needs to be accurate, it can result in a very specified circuit with multiple power sources that need to operate at different, critical values which might not be regular. It is not rare for CCDs to have 5 or 6 of these power sources, this means that a lot of power is needed to operate the chip as intended.

3.2.2 CMOS image sensor

A newer technology is the CMOS image sensor, it can be easily incorporated with chips and other circuitry made on the same CMOS wafers. There are two basic types of CMOS image sensors, one being passive and the other active. The passive-pixel sensor works along the same principles as the CCD however now the circuitry is on the same chip as the sensors/photosites. This causes noise which can be seen on the image produced. The active-pixel sensors have extra circuitry at each pixel to cancel out the noise, this increases the quality of the image and makes it possible to go for higher resolutions. Here the performance can be equal to that of the CCDs. The negative side here is that this takes up extra space within the area of the photosites/pixels, see figure 11. This results in the photosites not being as close to each other as they are at the CCDs for example (the fillfactor is lower compared to CCDs). Because there is less area capturing the light. This results in a lower charge thus a lower amplitude signal running through the ADC. For the image itself this results in a lower light intensity. This makes the image appear darker than it actually is and therefor makes the CMOS image sensor less qualified to take images of dark environments.

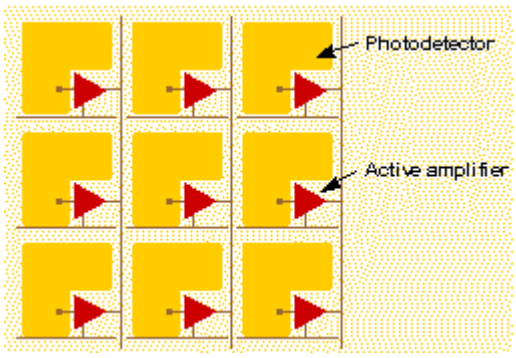


Figure 11: An abstraction of the circuitry around the photosites.

3.2.3 Comparison

CMOS imagers have a better intergration in circuits compared to the CCD cameras. Plus they use less power and are smaller. The negative here is that CMOS imagers have a lower image quality. Making them they less useful in high end image applications. Another negative for CMOS imagers compared to CCD cameras is that they are less flexible. This due to the circuitry already around the sensor, this is needed for it to work and cannot be changed while a CCD can be implemented in more systems. So the overall trend is that CMOS imagers are used for lower end imaging applications and in mass production while their counterpart the CCD is more suitable for the high end imaging applications.

3.2.4 Choice

For our application it is not desirable to have a too high resolution for the camera since the bandwith of our WiFi module is limited and the amount of FPS is important as well. Therefor the choice was made to take a CMOS camera and benefit from the lower power consumption, the small size and the easy integration. The only negative side to this choice is that with low light intensity the camera will underperform. However the challenges will take place during the

day so that should not be a problem. The chosen model is seen in figure 12.

Figure 12: The chosen camera model.

The camera has the abillity to stream video, with JPEG compression, at the following quality and rates:

Table 2: Quality vs FPS

Quality	FPS
160*120	60
320*240	60
640*480	30

For the video feed it is preferable to have 480p, but the option to stream at a lower quality makes it possible to have a video feed even when the connection with the WiFi is not optimal. Further it has an UART interface which is also on the WiFi module. Both UART connections can handle the same maximum speed of 1mbit/s. Thus this camera fits perfectly in the design. It satisfies the specifications and more.

3.3 User Interface

To be able to control the Zebro an control interface is needed. There are different elements that need to be taken into account, which platform will be used, is it a possibility to use a controller and if so what type, what is the most logical way of using the interface and how is it possible to keep the software modular aswell?

3.3.1 Control from PC

For controlling the Zebro we have serveral options, a tablet, just a controller or an interface on a PC. We have decided the latter, because of serveral reasons. Using just a controller like a remote car is not possible since we will not be conrolling the robot in a direct line of sight. It must be able to walk at least a distance of 1000 meters. On top of that there are serveral modules which need to be controlled, like in this case we have an arm and motor drive which need to be controlled. Because of this all visual feedback is a minimal requirement. A tablet was also not compliant to our requirements. We wanted to be able to control the Zebro, and especially the arm with great accuracy. The tilt sensor cannot be operated accurate enough, and using on screen buttons would also decrease the accuracy, and the amount of space that can be used on the screen. Because of these reasons the choice was made to implement the user interface (UI) on the PC.
The implementation of the UI will happen in C# and XAML, because of the ease of use of the language in combination with creating user interfaces and the experience we have with the language. Matlab could also have been used for this cause but not all computers run matlab, and since we want to make this interface compatable with more Zebro robots next to the Zebro Explorer and be able to run on multiple computers this is not an option. On top of that making an interface using XAML will have better looking graphics as a result of acces to specific interface libraries.

3.3.2 Hardware controller

When controlling the Zebro from a PC there are multiple options on how to send commands to the robot. In this case it is possible to make a choice between keyboard and using a controller. The controller has as an advantage that it makes the interface easy to use, and can make controlling the arm and robot very easy and intuitive. The choice was made to use an Xbox controller, since there are good software library's available for it and the controller sticks are very precise, with a sensitivity in the X and Y axis running from -32.768 to 32.767. Also the availability of analog buttons which can be used for switching tabs, grabbing and speed control for example make it easier to control the Zebro.
The reason to not make our own controller is because the xbox controller is already a finished product which costed multiple tens of millions dollar to develop. It is ergonomic and ready for use. An image of the controller layout can be cound in figure 13.

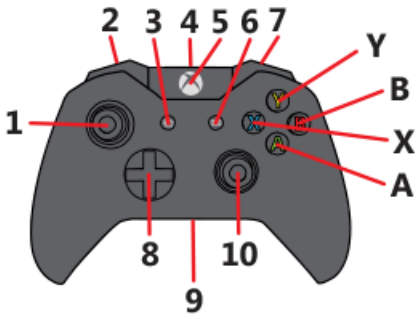


Figure 13: Xbox controller layout

3.3.3 Design

Since the option for a seperate controller is chosen a display is needed to show the user interface. The display allows for much information to be present for the user at one time and is just overall easy to use. Before making the user interface a sketch was made to have a layout of the design. This layout is seen in figures 14 and 15.

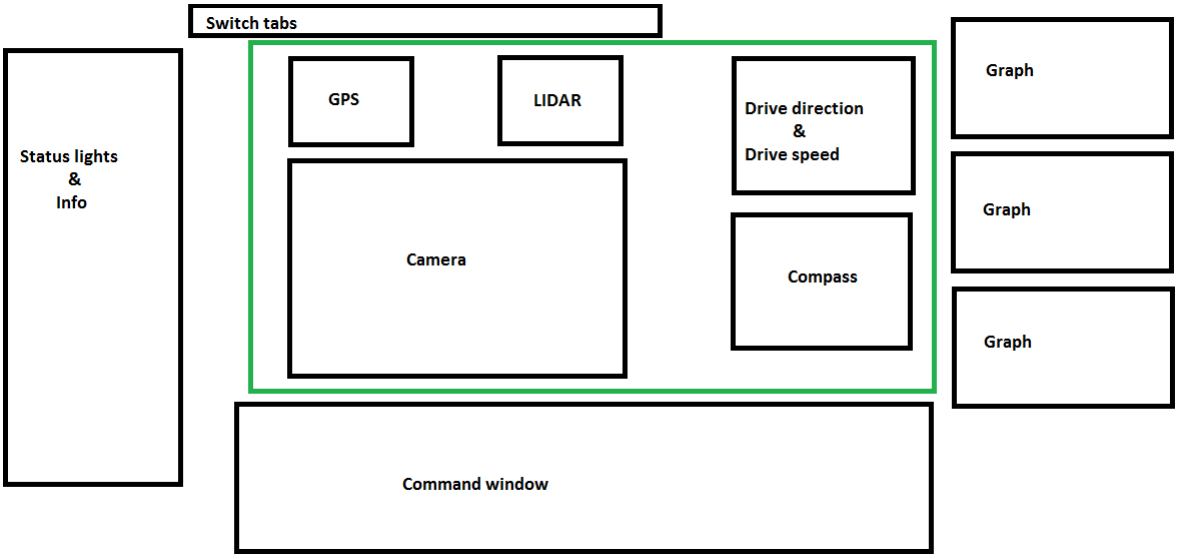


Figure 14: Layout of the driving and navigation window

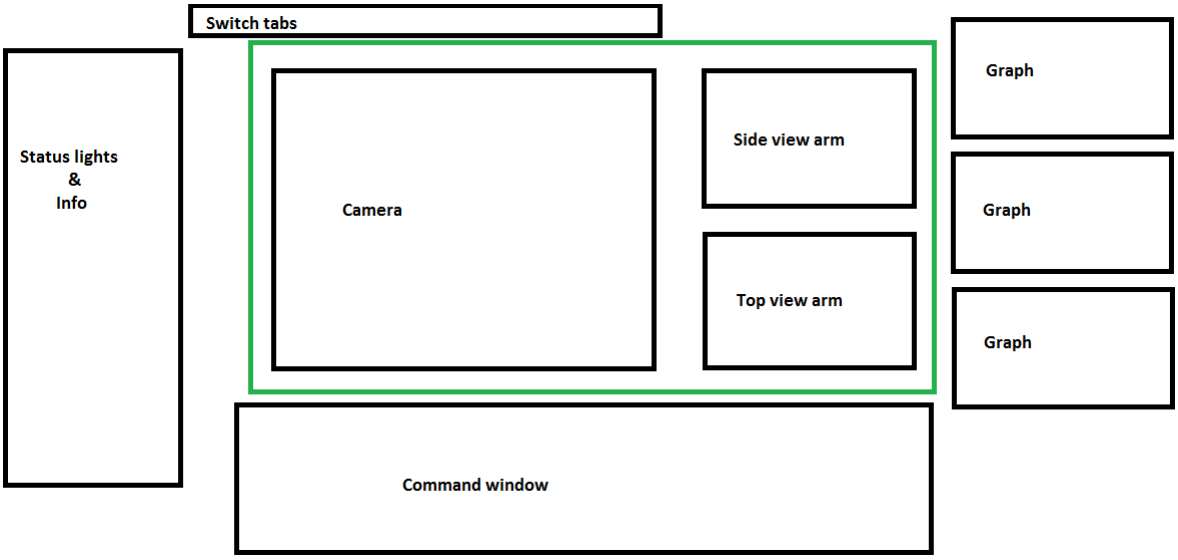


Figure 15: Layout of the Arm control window

Status lights & info

In this section of the user interface all the connections lights will be shown. This will include the lights for the controller and the XBee for example. If they are connected correctly the lights become green, if there is an error they will turn orange and red when they are switched off. Further status lights for the motors will be implemented, these will show how the motors are doing and if there are any problems. Since the Zebro is modular it is wise to make the interface

modular as well. Therefor more space is reserved for extra status lights when for example a new module is implemented.

Switch tabs

These will be used to switch the green box as seen in figures 14 and 15. The tabs that will be implemented from the start are one for controlling and navigating the Zebro and one for controlling the arm. When extra modules are added to the design they can be added to the interface too with their own tab. This keeps the user inteface clear and manageable and it allows for easy understanding.

Graphs

On the right side of the layout some standard graphs will be displayed. Things that the user always wants to monitor are the temperature of the Zebro, the battery power and the power usage. If one of these parameters are in a danger zone, for exmaple the battery power is only at 5%, it will affect the whole Zebro and thus it needs to be monitored always.
When other sensor data needs to be checked it can be added to the user interface or it can be plotted while the Zebro is in operation. The graphs are updated real time with the data supplied from the Zebro, information can also be manually requested. This keeps the XBee and WiFi lines as empty as possible rather than constantly pulling and pushing data.

Command window

The command window already says it, is used to manually send commands to the Zebro and to see what is sent back by the Zebro. In appendices C through E the commands that are possible to send are shown. The reason for this is that when something goes wrong or an error occurs in on the zebro, the user can manually fix the Zebro.
Plus for the challenge the jury wants to be able press a button and have the robot make an emergency stop. From the command window it is then possible to restart the Zebro and continue the challenge.

Driving and Navigation

When the tab for driving and navigation is selected a new screen appears within the green box as shown in firgure 14. In this screen a camera feed will be visible, together with the a GPS and a LIDAR map. These three will be used to see what is in front of the Zebro as well as scouting the surroundings to calculate a viable route to the end location.
Next to this a compass will be displayed to show the orientation of the robot as well as the direction and speed. These can be manually controlled and this part will give visual feedback of what the user is doing with the Zebro.

Arm control

The other tab is for the arm control as shown in firgure 15. Here it is important to have a large camera feed and some feedback on the position of the arm. For this we discussed with the arm team on the Zebro project. Together it was determined that a top view of the arm in respect to the Zebro itself as well as a side view to see how all the joints are oriented and how far the telescopic part is extended. When the arm changes from position the top and side view will update accordingly. With this and the cameras complete control over the arm is possible.

3.4 Complete design

In figure 3 the specifications for the system are shown. In this chapter all the design choices are made and explained. The resulting design tree is shown in figure 16. Again all the subsystems are shown in blue and now the design choices are shown in green.

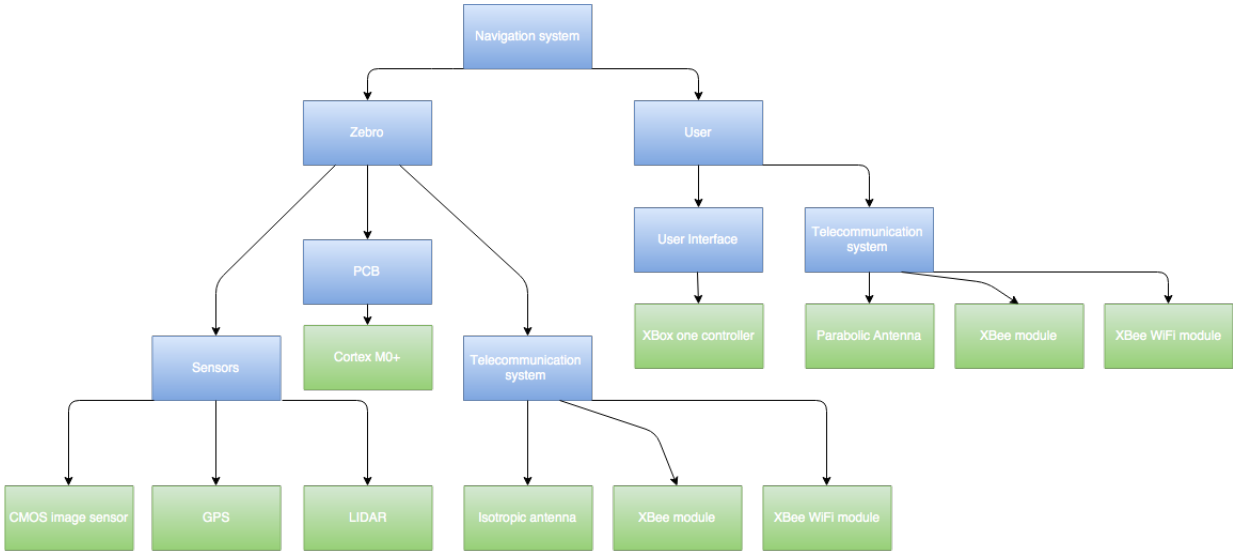


Figure 16: Design tree. The blue blocks indicate the different subsystems and the green indicate the choices made.

3.4.1 Power budget

Now that all the components have been picked out it is possible to make a power budget of the complete system. So in the table 3 the power consumption of all the different components are listed.

Component	Power consumption
Camera	0.5W
XBee zigbee	1.65W
XBee WiFi	1.65W
GPS	0.1W
LIDAR	0.5W
Cortex M0+	7.12e-3W
Servo	1W
Total	6W

Table 3: The power budget

Note that the power consumption taken here is the absolute worst case. Meaning that the XBee modules are sending continuously. So in total if all the peripherals are on the ONS system consumes 6W of power. However this is not the case, all the components can either be switched off or put into a sleeping mode where they consume a few μW.

3.4.2 Financial budget

Next to the power budget it is also possible to make a financial budget. This can be seen in table 4.

Component	Financial costs in euros
Camera	39.60
XBee zigbee	80.77
XBee WiFi	41.77
GPS	58.38
LIDAR	94.90
Cortex M0+	1.67
Servo	4.99
PCB	109
Total	695.42

Table 4: The financial budget

These cover all the big expenses, there are also a few connecters and sockets. However these cost less than the microcontroller so are not interesting to list here. The total cost of the prototype is 695,42 EU. Note that these costs can be cut when for example the PCB is not ordered with urgency or if the parts are ordered in large amounts to get a discount.
In the next chapter the implementation of these subsystems is discussed.

4 Implementation

4.1 PCB design

With the zebro being modular, it was neccessary to make our own PCB for on the backbone. This means that there is limited space and that there is already an existing connection between all the different modules. This constrains the freemdom of designing the PCB. The board needs to act as an central point for the zebro through which all the data flows from and to the user side. Next to this there are extra peripherals like the cameras and LIDAR. These need a connection to the microcontroller on the board. First of the board itself needed to be made. Next was the layout of the components and the last thing on the list is the routing of all the components. The final version of the board is visible in appendix A.

4.1.1 Board design

On the backbone we got the space to fit onto the ZPU board. This board had the dimensions of 75cm length (y-axis) and 80cm width (x-axis). During the layout of the components it became clear that using these dimension would result in 0.3mm gaps between the Xbee and WiFi modules and the connectors. This was a risk we were not willing to take because if the all components came out slightly bigger as mentioned in the datasheets it would not fit anymore. The decision was made to increase the length of the board an extra 31 mm and the width with and extra 2 mm.
This resulted in a board with the following dimensions: 106 mm length and 82 mm width.

4.1.2 Resistor values

The LEDs on the PCB are to show if the XBee and WiFi module are working or if there are any errors with starting up. These LEDs have a voltage drop of 2V and a forward current of 20mA. The supplied voltage from the microcontroller is 3V3, meaning that 1V3 is supplied to the resistor. Since the LEDS are inside the Zebro itself it is not necessary to let them shine at full brightness. Going for a 3mA current is bright enough to tell if it is on or not. In figure 17 the circuit is seen.

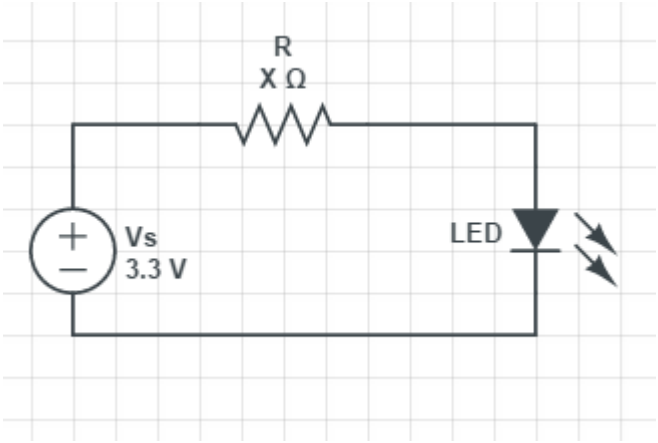


Figure 17: Circuit with a resistor R and a LED

Following ohms law:

$$R = \frac{U}{I} = \frac{1.3V}{3 \cdot 10^{-3}A} = 433\Omega \tag{13}$$

The resistor with the value closest to 433Ω is 430Ω. which will result in a current of 3.02 mA.

4.1.3 Layout

When the board design was finished the task was to make a layout of the board. There were a lot of constrains in the space on the board as mentioned earlier. This was due to the connectors of the ZPU board and the fact that the board had extra peripherals. These peripherals needed to be on the sides of the board to connect them easliy with wires and antennas. Otherwise these would be over the board itself. The other parts needed to be on the PCB were the microcontroller with external crystal, the multiplexer , a few LEDs and the DC/DC converters. It was chosen to have the DC/DC converters near the power supply itself to keep the tracks as small as possible. For the microcontroller it was convienient to have it in the middle of the PCB. The WiFi module and cameras were placed near each other so it was an easy choice to place the multiplexer in between these components. This all allowed for shorter tracks and easier routing.
An important decision that was made concerning the layout was to get rid of the connectors on the sides of the board. These are needed to place the PCB on the ZPU board. However none of the connections are actually needed for the design. Therefor the one 36 pin connector was replaced with two 4 pin connectors. This means there is more space for routing and the board is still stable on the ZPU board.

4.1.4 Routing

Because of the choices made with the layout, the routing was fairly straightforward. It all fitted on the top and bottom layer of the PCB. Concering the thickness of the tracks, we took 30 mil for the 24V and 2A power tracks. The maximum amount of current we will need is around 400mA though, this is because not all the peripherals are switched on at the same time. Another thing to notice is that the chosen DC/DC converters work to a maximum of 1A current, which is more than enough. The 5V tracks are made 20 mil thick and the 3V3 lines are 15 mil thick. These lines do not neccesarliy need to be this thick, but the room is there to do it and we do not want to take risk when for example the lines came out too small. The thickness of the tracks were calculated using an online tool [1] and letting Altium designer check them.
In the Appendix B there are two tables with the connections for the microcontroller and the multiplexer.

4.2 Camera subcircuit

The camera feed will be sent via the WiFi, as mentioned in the design choices. The Camera modules have their own PCB which saves the compressed frame in the microcontroller. Therefor there is no reason to connect the cameras to the microcontroller on our own board, it can be directly connected to the WiFi module.
To switch between the different cameras a multiplexer is used, it switches the UART lines so that either camera 1 or camera 2 is connected to the WiFi. The multiplexer is conencted to the microcontroller, this way it is possible to switch the camera over the XBee channel. In figure 18 the schematic of the multiplexer is shown.

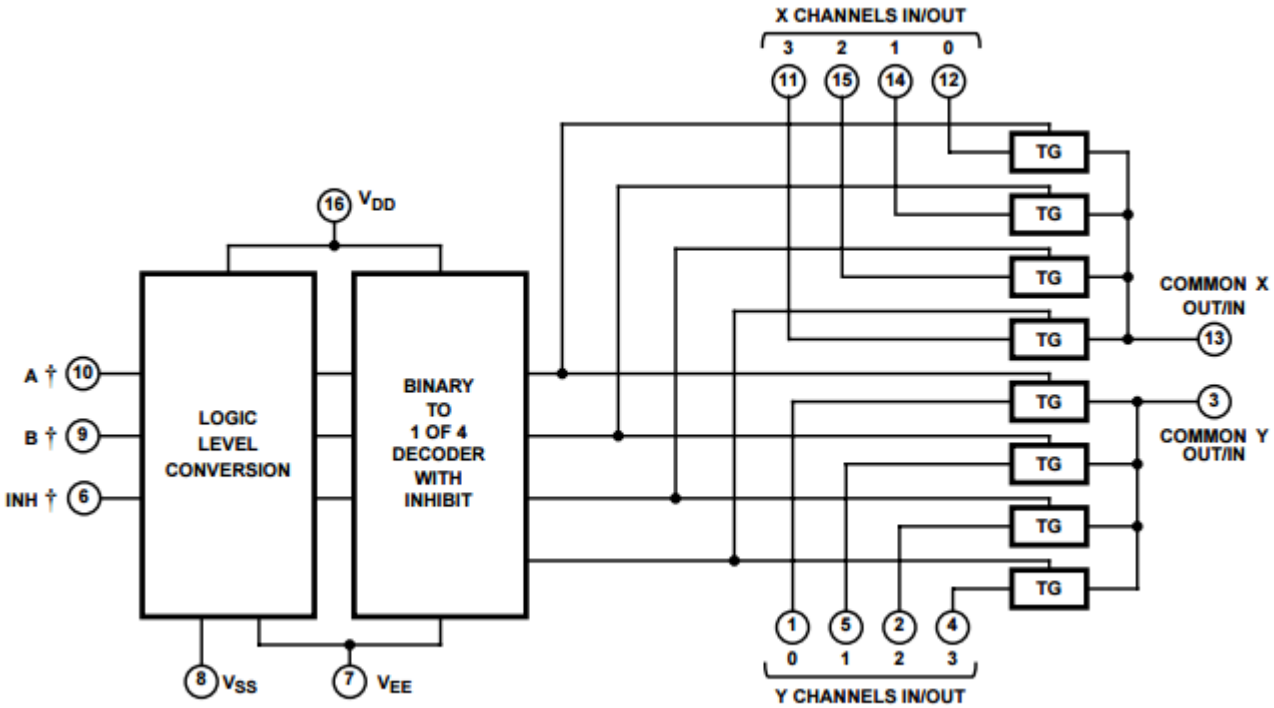


Figure 18: The schematic of the multiplexer.

The outputs, pin 3 & 13, are connected to thw WiFi module. The UART lines of camera are connected to pins 2,4 and 12,13. The selector A is connected to our microcontroller, input B and INH are both linked to the ground in order to have a ”0” input. Through this configuration it is possible to select the camera that is connected to the WiFi.
The bitrate of the UART interface of the camera is 921.6*kbps* as well as the bitrate of the WiFi UART interface. This makes it possible to have them both work at their maximum bitrates and not have to introduce stalls in one of the modules.

4.3 Zigbee

As stated in previous chapters the Digi Xbee-Pro 868MHz has been chosen to use. All Digi Xbee modules share the same footprint which can be seen in figure 19. Since it was not possible yet to use the zigbee module chosen, the whole system was implemented with a simpler Xbee 2.4GHz module.

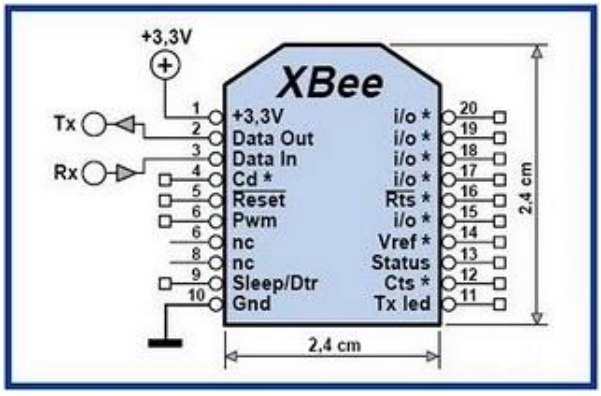


Figure 19: xbee footprint

The module can be used by attaching the 3V3 pin to a 3V3 source, and the ground pin to the ground of the arduino. An arduino was used to simulate the ONS board on the rover to programm and configure all the peripherals. The Tx is the data that the module received from the user side and the Rx the data that needs to be send to the user side. It has its own buffer onboard, which can be read through the Tx pin. The other module connected to the PC is connected by using the Sparkfun Xbee Explorer, shown in figure 20. This makes it easy to connect, and it allows for simple use by reading out the USB connected to the PC and sending data through it.

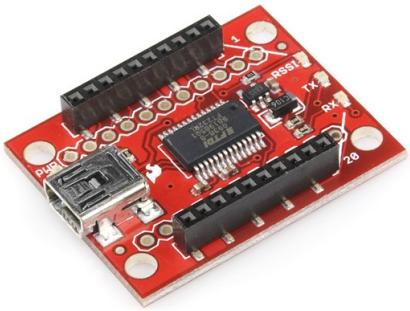


Figure 20: Sparkfun Xbee Explorer

In order to configure the module, code has been written on the arduino using C++ and behind the interface using C#. By having written a library in C# which has a send(string) function and a read() function it is possible to send commands to the other module attached to the arduino. The read() function keeps emptying the buffer asynchronous from the rest of the C# code. The same for the onboard code running for the arduino, the buffer keeps getting emptied in case a command arrives, and this happens asynchronous aswell from the rest of its tasks.

4.4 User interface

In the chapter design choices the layout for the user interface was made. The idea is to keep the design as clear and simple as possible. This way it is easy for new users to understand and find all the possibilities and features. Further one of the main tasks was to keep the user interface modular as well.

4.4.1 Navigation and Drive window

In figure 14 the layout is visible, in figure 21 the implementation of the design is seen.

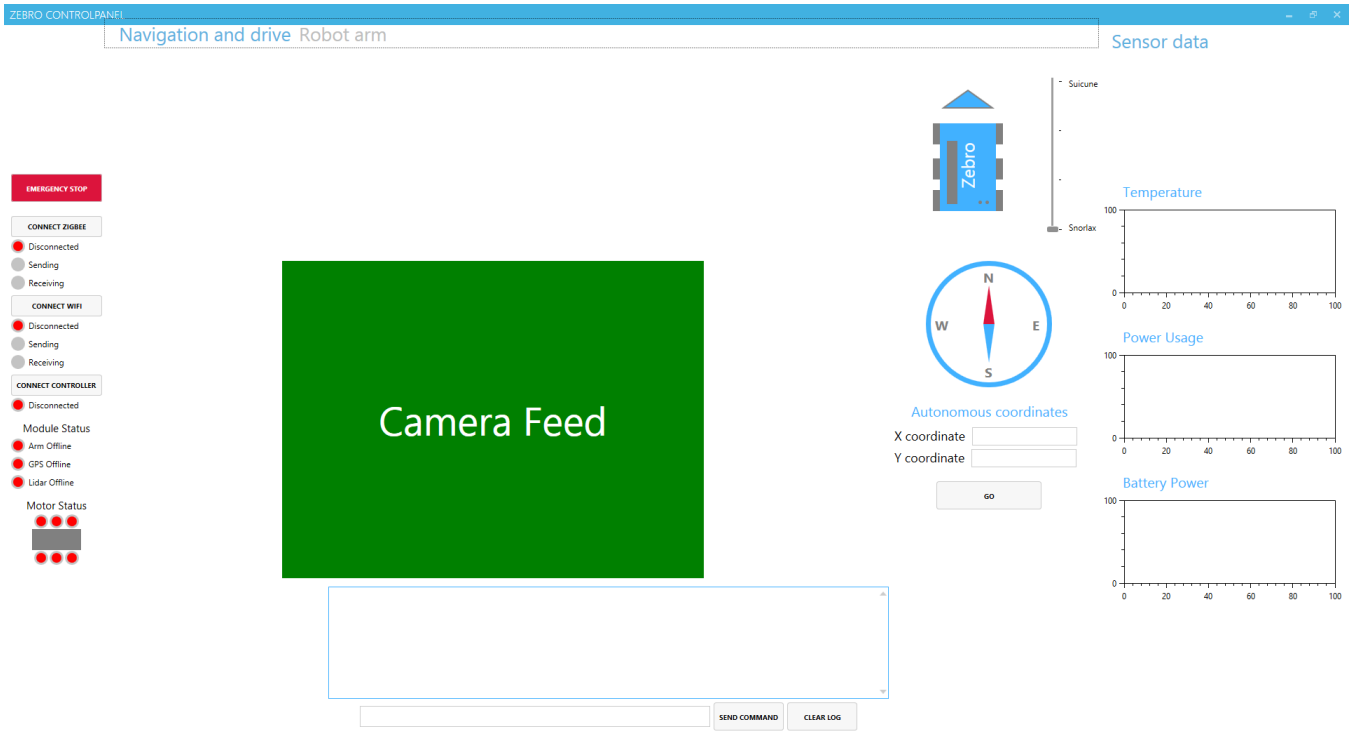


Figure 21: The implementation of the design for the Navigation and drive window.

It is noticable that initial layout looks like the interface that is designed. The interface was made in C# and xaml using visual studios. The reason for this is that we had experience in this language and the programming environment is one of the best out there.
The implementation of the compass and the mini Zebro can both rotate accordingly. The compass will get feed back from the sensors on the zebro and point in the right direction. And the mini Zebro has an arrow that rotates around it which indicates the direction the user is driving in. The sliderbar can be set to four different values to change the speed, ranking from standstill to top speed.

4.4.2 Arm control window

When the tab is changed to ”Robot arm” the window changes as seen in figure 22.

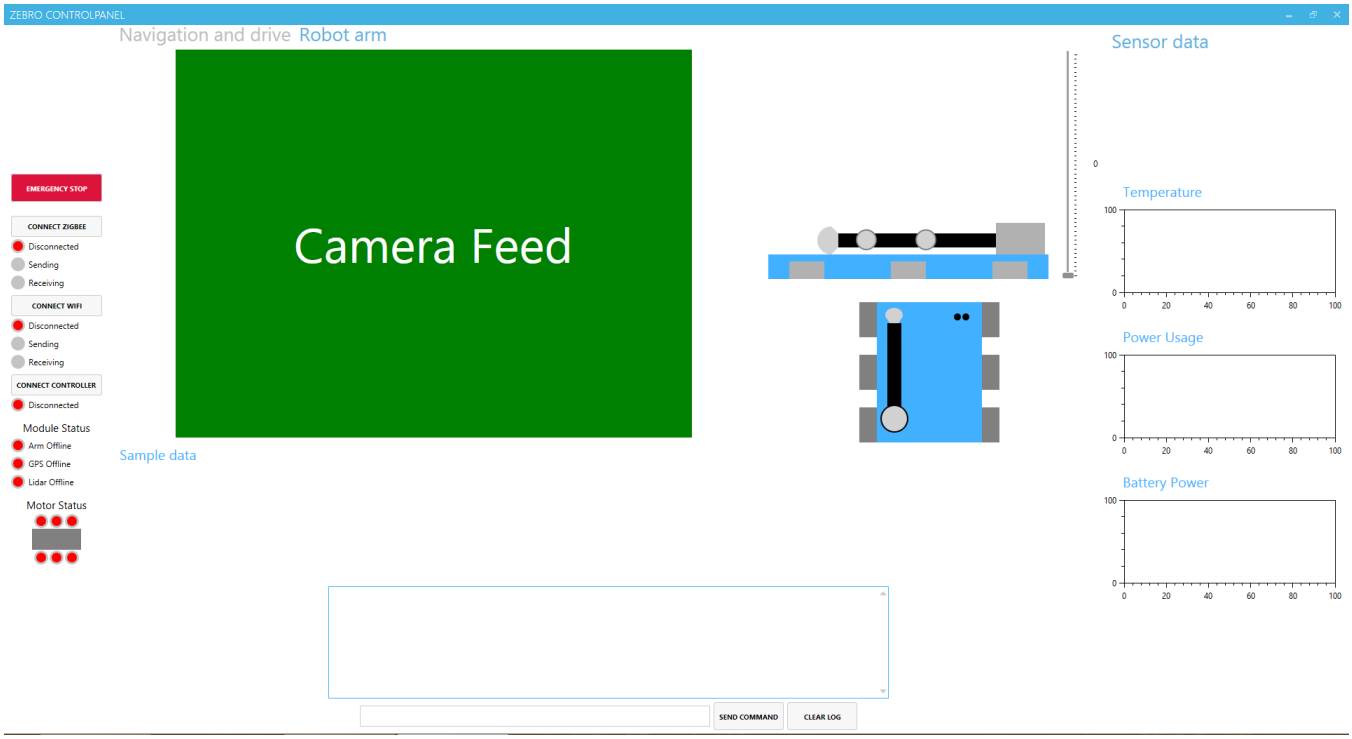


Figure 22: The implementation of the design for the Robot arm window.

Here too it is evident that the layout was inspiration for the end product. The top view of the arm is able to rotate clockwise and counterclockwise to show the position of the arm. The side view can rotate all its individual joints and extend its first part, together with the top view the complete position of the arm can be modeled.

5 Modularity and external systems

The most important feature of the ONS, next to being able to navigate, is its ability to pass through communication with all the other subsystems , since it is the wireless communication hub. This communication must be well defined. An overview of the total zebro system with the modules being used now can be seen in figure 23. We have one external connection to and from the ONS to the ZPU using I2C. This way it is possible to use the system not only on the Zebro Explorer, but also on other zebro robots, and even completely different systems.

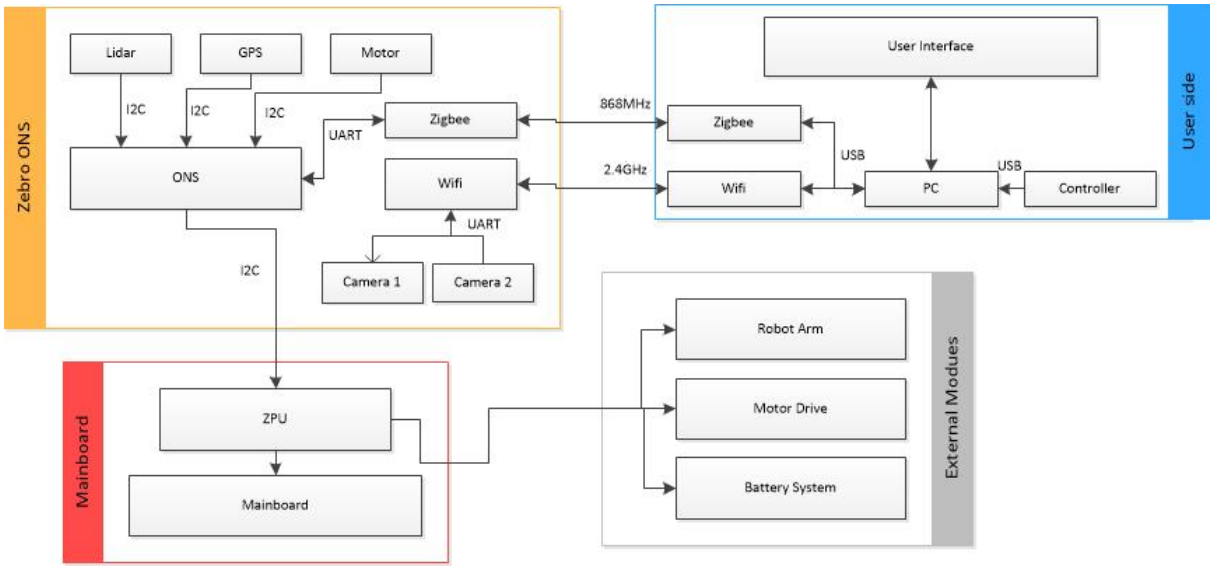


Figure 23: System overview

5.1 Modular PCB

The place of the ONS board on the Zebro wil be on top of the ZPU which also has multiple connections to it of other external modules like the robot arm, motor drive and the battery system. Most of these systems are also modular, making it available for every robot with the same interface. Further are all the connections with peripherals are based on I2C except for the XBee and WiFi module, which are connected via UART. All the peripherals can be changed out with improved versions as long as they support I2C. The XBee modules too are placed on headers and not soldered on the PCB itself making them interchangable.

5.2 Package details

In order to communicate in a safe and reliable way we have established a standard protocol which will be used throughout the whole Zebro. The first part of this protocol is that the communication functions through a master-client way of communication. Here the ONS will work as a master and other systems will only send info to the ONS when it has requested it. The packages sent to other modules, and from other modules are defined as following:

$[Number][Destination][Error][Datalenght][Data][Checksum]$

Where the length of each part is divided as

$[8bits][4bits][1bit][8bits][Xbits][?bits]$

5.2.1 Number bits

In this packaging standard *Number* stands for which package number it is that has been sent from a certain module. Thus each module keeps track of it's own packages sent. The sender will keep a list of every package sent. When the receiver receives a package an acknowledgement will be sent to the sender that the package has been received, and the number will be crossed off the list. When the sender does not receive an acknowledgement within a second, the package will be resent. If then again there is no acknowledgement or a request to resend the package because of an error, it could be the result of the wireless module being out of range.

5.2.2 Destination bits

The *Destination* bits will be linked to a list of numbers, in which each number represents a module. There are 4 bits available which allow for 16 different modules. After the checksum has been executed the package will be acknowledged and checked what the destination is. When the destination is the same as that of the receiver, it will read the rest of the package. If the destination is different, it will repackage it again as the data part of a new package, and send it to the end destination. The module acting as this hub is in this case the ZPU. This can be seen in figure 23 where it acts as the hub between the different modules. The different numbers given to the modules can be seen in table 5. These numbers are kept in the ONS, UI and ZPU to keep track where a package has to go to.

Table 5: Corresponding modules

0	UI
1	ONS
2	ZPU
3	Arm
4	Motor Drive
5	Battery System

5.2.3 Error bit

There is an *Error* bit included in the package, so it is possible to quickly identify if there is an error going on in a certain module. Since the error needs to be presented to the user, it is not needed to attach a sender to it. The ONS will request the status from a certain module and if its response has the error bit set to 1, it is clear that the system it requested status was from has a malfunction. When this bit is set, the *Data* part will include the error in the form of a number. This number is kept within a lookup table in the UI so it is easy to indentify what error number corresponds to what condition. By using this bit we can manipulate the *Data* entry and give the package priority so the error can be handled as soon as possible.

5.2.4 Datalength bits

The decision to include a *Datalength* byte was the fact that the system is dealing with a lot of different possible lengths of data. An option was to set the length to the largest amount of data, however this length is not known. And on top of that every packet would be that same lenght, which is just very ineffiecient. Next to that the way the Zigbee module is read out does not guarantee that there is a complete package. If there are different data lengths, it isl not known when the package ends. So this is required to have stable and reliable communication assuming the package is received correct.

5.2.5 Data contents

The *Data* part will contain the data sent and received from the different modules. In the master-client setup that is used, the data sent to other modules, except from the UI, from the ONS will always contain commands, and data sent to the ONS from other systems but the UI will contain the sensor data and feedback requested by the ONS.
The data sent from the UI to the ONS will always contain commands for the ONS, or commands that the ONS needs to pass to the ZPU. In which case passes it through to the module of destination. All the commands sent by the UI to the different modules are kept in a list within the code and can be found in appendices C through E. So to conclude the length of *Data* is variable, since it can simply contain a few bits, namely the command number or a whole array of sensor data.

5.2.6 Checksum

The checksum is needed to be able to check if the received package is still the same as when it was sent. There are multiple methods to do a checksum, but the most common ones are CRC and parity check. We have chosen to use the parity check instead of CRC since we are talking about small packages, and not much computational power on the different modules. A parity check however works by computing a XOR over a package. This is done by dividing the package in 2 parts, and computing a XOR operation over those 2 parts. The result of it is sent with the package itself, which is an array of 0 and 1 bits. When the package arrives, the receiver will repeat exact the same operation, and compare that with the attached checksum which is computed at the client. If this gives only 0, we can assume that the package has kept its integrity. The upside of this method is that it is simple and quick to compute. The downside is that only odd number of errors can be detected.

5.3 Communication with other modules

This communication is for the biggest part one sided, where the ONS system functions as the master in this system onboard of the Zebro. Other systems will only send information to the ONS when it asks for specific information.

5.3.1 User side

The user side is where all the manual controls happen. This side will be sending all the commands, and is the master over the ONS system on the Zebro. The UI has a list with commands for all the different modules on the Zebro. The UI will send a command, which is received at the ONS. Then the data will be sent to the end receiver, or if it is meant for the ONS itself it will execute the command. When there is data expected back the ONS will wait for a response. The different commands that the UI can send will be listed in the following paragraphs. The command list which is meant for the ONS can be found in appendix C.

5.3.2 Robotic arm

The robotic arm has serveral joints, which in turn can be added and removed in a modular way as seen in figure 24. The UI can send commands to the robotic arm, manually controlling each joint with a controller. These commands will be received at the ONS, which send them to the ZPU, which in turn sends it to the robotic arm. The different commands for the arm are for each joint a up, down and stop command. When an up command for a joint is sent, that specific joint wil keep moving up until a stop command is sent. This system works for every joint, and thus will keep its modularity. Futhermore there is a status command, which in response sends

back the angles of every joint, and other statusses like errors, on/off. A full list of commands and the exact responses can be found in appendix D.

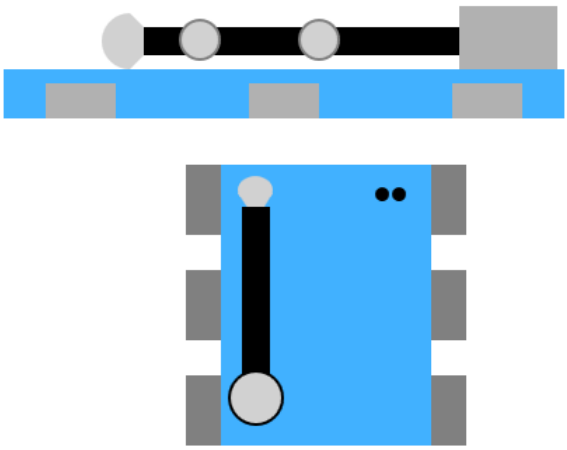


Figure 24: User interface representation of the orientation of the arm.

5.3.3 Motor drive

The motor drive takes care of the 6 different legs on it's own microprocessor. This has as a result that only the walking direction is needed to be sent. This walking direction exists of 8 directions. Next to the walking direction, a stop command can be sent and a command to let the robot "sit down" so the arm can be used safely. The commands and exact responses for the motor drive can be found in appendix E.

Figure 25: User interface representation of the motor drive directions

5.3.4 Battery system

The only command sent here is for a status report, whichl in return will give the current battery status and the amount of battery power left.

5.4 Error protocols

The package that needs to go to one of the modules sent from the ONS will be confirmed using an acknowledgement system. When a package is sent, it will be checked at the receiver with a checksum. If this comes out correct, a package containing an acknowledgement will be sent back which is illustrated in figure 26

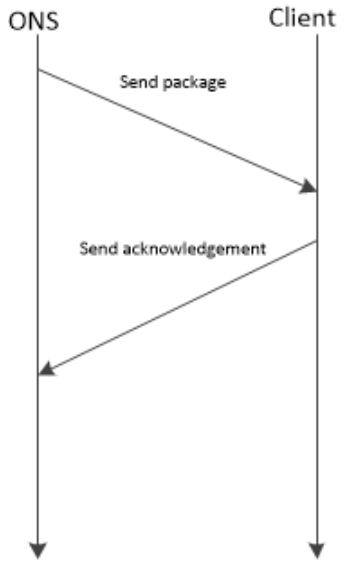


Figure 26: Acknowledgement system

This way the ONS will know if a package has been received correctly.

Time out

There are some things that can go wrong while sending the package. First of all the package could not be received at all. First the package will be resent 3 times. Resending a package is shown in figure 27, where it is anwsered after the first time resending. If after 5 times resending there is still no response the system can be assumed disconnected and appropriate measures need to be taken.

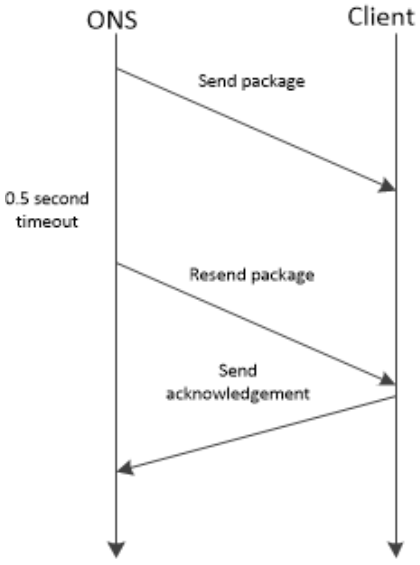


Figure 27: Resent package

Wireless timeout

A timeout of the wireless communication could happen because of a variety of reasons. First of all on top of the rover there is only 1 antenna and an arm. It could happen that the rover is far away, which means more data loss, and the arm is in the way of the signal. In previous chapters the link budget was analysed, but this is only applicapable when assuming free space loss. When the arm is in the way, the theory of free space loss no longer complies without taking the loss that happens because of the signal needing to pass through the material of the arm. The first step towards regaining signal is letting the ONS send a signal to the arm, which is inside the rover, to move the arm in base position. If it then still does not receive a signal within 5 seconds, it could mean that the arm, even if it is in base down is still in the way because of a slope the rover is standing on. The next step is to turn the Zebro slowly 360 degrees untill the signal is regained. If the Zebro is still disconnected it means it is out of range or some other object is between the sending station and the rover. The ONS will turn the rover around and let it move back to the base station using its autonomous walking algorithm, until it reconnects again. If it doesn't reconnect it will walk all the way back until it is within 10 meters of the endpoint. This way it is able to always come back even if the signal is completely lost. The steps to handle a wireless timeout are summed up in steps below.

1. Send command to arm to move into base position.
2. Wait 5 seconds.
3. Turn around 360 degrees or until signal is regained.
4. Walk back towards last known base station location until connection is regained or it arrives.

Checksum error

Between receiving a package and sending an acknowledgement as shown in figure 26 the checksum is commputed. As previously explained a parity check is computed from which is detectable wether a package is received correct or not. If it is not received correctly a package will be sent back not containing an acknowledgement but a request for resending the package. After receiving this package, the checksum will be computed again.If it is correct an acknowledgement will be sent. Otherwise the client will request for the same package again until it is received correct. This whole overview is shown in figure 28.

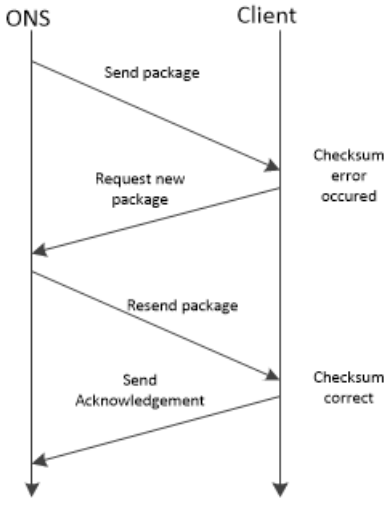


Figure 28: Checksum error

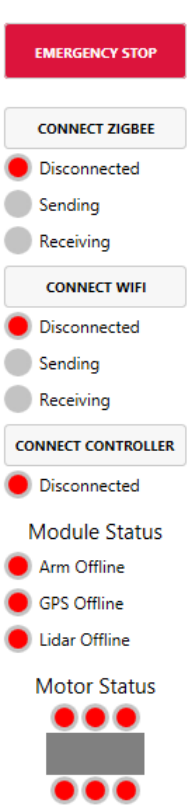
6 Results

6.1 ✓ User Interface

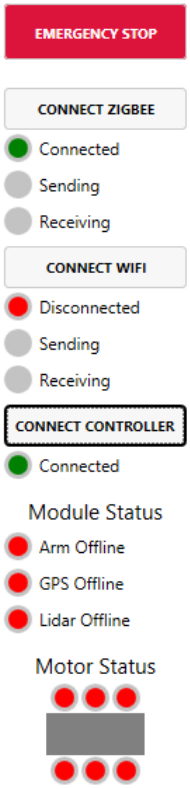
It has already been shown how the total interface looks like in figures 21 and 22. However below are all the results of the user interface. All the subsystems of the interface are shown and explained how they give feedback to the person controlling the Zebro explorer.

✓ Status lights and info

An important aspect of debugging the Zebro explorer and knowing during the challenge what modules are holding up or are failing is having the status lights work porperly. An even bigger help is building in error messages to show what is wrong with a certain module. In figure 29a and firgure 29b a few lights are shown on and off to indicate if there is a connection or not with the XBee module.



(a) The status lights all turned off.



(b) The XBee and controller status lights turned on when they are connected properly.

Figure 29: The status lights turned on and off.

When a new module is implemented an extra status light and error messages can be added.

✓ Command window

In figure 30 the command window is shown. It is displaying the input from the Xbox controller.

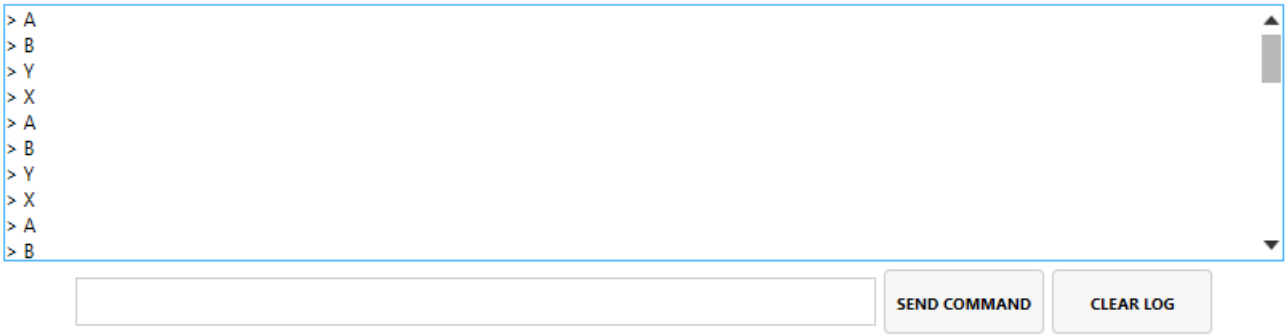


Figure 30: The command window showing the input from a Xbox controller.

Further the user can type commands into the command window to request certain data or execute a command manually. So this part of the interface is checked off.

✓ Graphs

The graphs which show the temperature, battery power and power consumption can plot the data retrieved from the Zebro. The graphs are updated everytime new data is available. For now it has only been simulated with test data which is shown in figure 31.

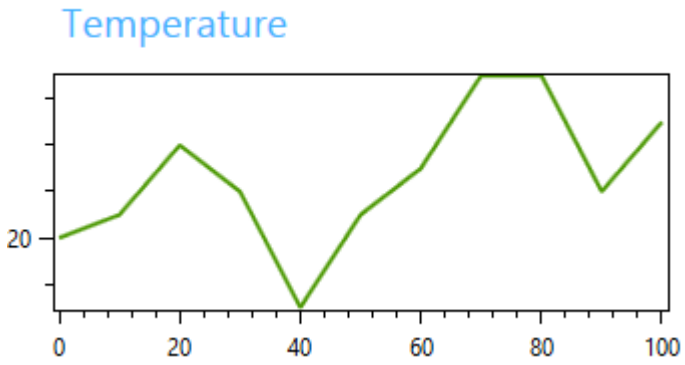


Figure 31: The graphs in the interface simulated with test data.

When extra data is required for permanent display it can easily be added to the interface.

✓ Switchs tabs

One of the ways to keep the design clear was to introduce different tabs for different tasks. This way different information can be displayed on the screen and the environment to fulfill the task can be optimized. For now the different tabs are 'Navigation and Drive' and 'Robot arm'. When the tab is switched the interface fades to the new layout and the task can be completed. In figure 32 and 33 it can be seen that the different tabs correspond with different layouts.
Here the design also lends itself to implementing an extra tab when for example other tasks need to be done or when a completely new module is added to the Zebro explorer.

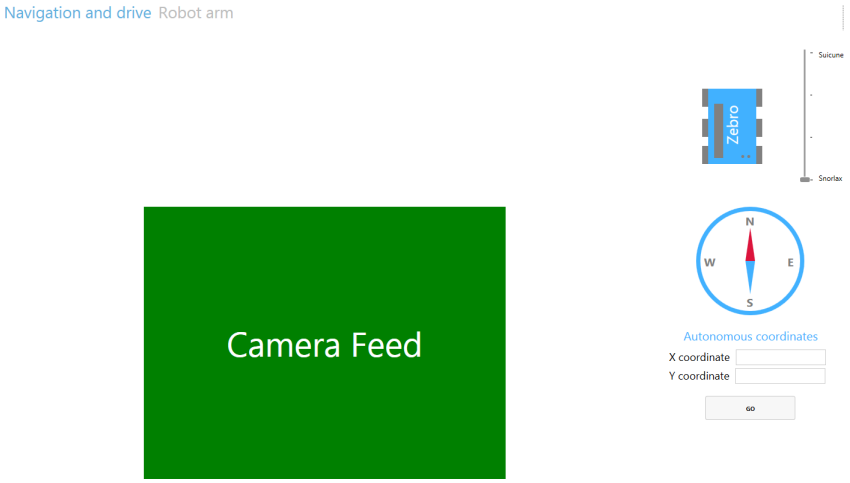


Figure 32: The Navigation and Drive tab is active.

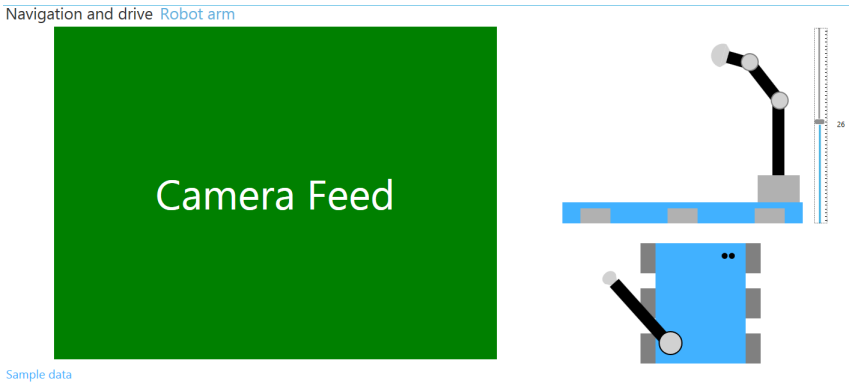


Figure 33: The Robot arm tab is active.

- **Navigation and Drive**

First of is the compass, it should point to the direction the Zebro is aimed at. On the Zebro explorer there already is a compass, so it is possible to collect that information and display it. Below in figure 34a and 34b the compass is shown pointing in the North East direction and the West direction respectively.
The second part of the Navigation and Drive window is the mini Zebro which indicates the direction the user is driving. In figure 35a and 35b the user is driving the zebro slightly to the right and backwards respectively.
This function of the user interface is directly linked to the controller and therefor gives instant visual feedback to the user. It can also help with the debugging of the Zebro, for example when the Zebro does not drive in same direction as the controller is steering.
The next part is the camera feed, it has been tested with just a regular video. But unfortunately the camera did not make it on time.
The GPS map and the LIDAR map too have not been finalized in the design, for the GPS only the asynchronous polling needs to be implemented. For the LIDAR map the code that plots the readings still has to be made. So for this part of the interface there are still a few things that need to be finished.

(a) Compass pointing in the North East direction.

(b) Compass pointing to the West.

Figure 34: The compass of the user interface.

(a) Zebro driving slightly to the right.

(b) Zebro driving backwards.

Figure 35: The Zebro driving direction.

✓ **Robot arm**

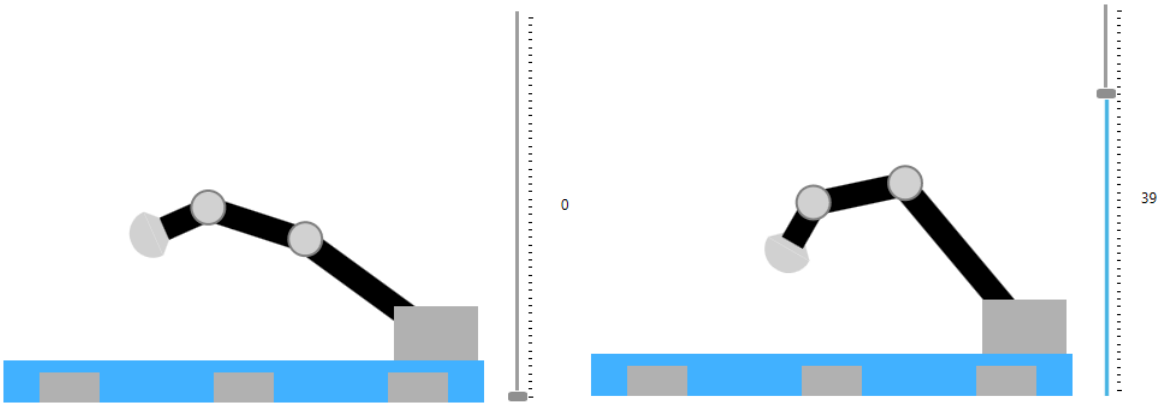
In this tab the camera feed is larger. However the same goes for the Navigation and Drive window, it has only been tested with a prerecorded video and not with the actual camera. What has been completed in this tab are the top and side view of the arm.
As seen in figures 36a and 36b the the arm can be rotated a full circle to show its position relative to the Zebro explorer.

(a) The arm is pointing forward.

(b) The arm is pointing to the right.

Figure 36: The position of the arm as seen from above.

In the figures 37a and 37b two different configuration of the joints of the arm are shown. All the different joints can turn seperately and the first bar is a telescopic part which is also extendable in the interface itself. In figure 37a the first bar of the arm is shorter than the one in figure 37b.



(a) A possible position of the arm without the telescopic part extended.

(b) An other possible position of the arm, this time with the telescopic part extended.

Figure 37: Possible configurations of the arm as seen from the side of the Zebro explorer.

And these three parts make up the tab for the arm control. When the camera arrives it can be implemented and the whole tab is finished.

6.2 • PCB

When the PCB was finished it was manufactured by eurocircuits. They sent us the PCB as seen in figure 38.

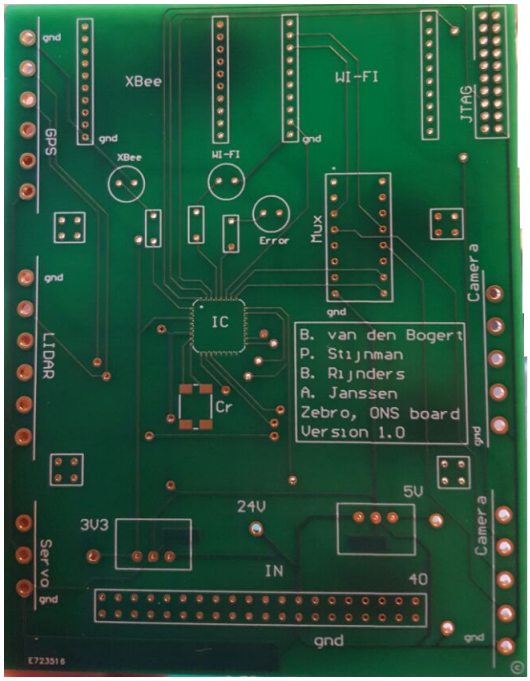


Figure 38: A picture of the manufactured PCB.

Since our components were not delivered it was not possible to test the PCB if it is working correctly. In the chapter 'Future plans ' the plans for assembling the PCB and testing it are explained.

6.3 ✓ Zigbee

✓ Stable and reliable communication

After implementing the Zigbee protocol using the 2.4GHz Xbee module we had a reliable connection where it is possible to send data to the arduino, and receive responses.

• 1000 meter distance

We can conclude that it is possible to send data over, but we can not send it over 1000 meters yet since we have not received the module we want to implement. When the new module arrives however it is plug and play, using the code already written at the moment.

7 Future plans

In the week of August teammembers can work on the Zebro project and finish their parts. This means the group needs to think on what to do and make a plan beforehand. Below the building of the prototype, the testing and the final integration in the Zebro explorer itself is discussed.

7.1 Building prototype

When in August all the parts have been delivered the actual building of the prototype can take place. First off the components have to be soldered on the PCB. Next is making the wires to connect all the peripherals to the PCB.

7.1.1 Soldering

For the soldering of the components first the SMD parts need to be done. These include the microcontroller and the external crystal. When these are on the PCB the rest of the components can be soldered regularly. The things to look out for is not to damage the PCB or burn away the silk on top. Further the components have to be soldered the right way around, luckly there are signs to where the parts should be and where pin number 1 is located on the PCB. This for all the components that can be the wrong way around.

7.1.2 Wires

For the wiring of the peripherals the size of the Zebro explorer needs to be known. By the time we can work on the module again the final concept of the chassis will be done. With these parameters it is possible to make the wires the right length instead of making them too long or short. This makes the Zebro less prone to issues with cables getting stuck behind other things or being just to short and falling loose during a task.

7.2 Testplan

7.2.1 PCB

When all the components are on the PCB it is time to test it. First thing to do is programming the microcontroller. The other subgroup has already made a code on an arduino so the only changes that have to be made are certain addresses that the microcontroller needs to write to or read from.
If the programming of the microcontroller works the following step is to simply control the LEDs that are on the board. This is to check if the microcontroller is actually executing the code. If it is all the other subsystems can be attached and checked if they work accordingly.
When the PCB is not working the next task is to find out why it is not. After that the design of the PCB should be updated and ordered again. If the PCB is working properly other parameters of the PCB can be tested. For example what happens when the Zebro explorer is above 60 degrees celsius inside. This to see what the board can handle and what it cannot. If the PCB fails to handle conditions that can be expected during operation it needs to be redesigned, or if a certain component is failing it can be replaced with another part that can handle these conditions like temperature.

7.2.2 Wireless communication

The communication is already working between an arduino board and using a low power cheap zigbee module from Digi. This module however has exactly the same footprint. Because of this it is possbile to simply interchange the current module with the new one and the code written will still be applicable. For WiFi however there were no possibilities to test since it was not possible to get our hands on a XBee WiFi module, so this still has to be done in the future. The following testplan needs to be done with both the WiFi and the zigbee modules.

Effect of objects

Since the theory of our link budget is based upon estimations. And our zigbee module has variable power output, which is desirable to be as low as possible. This is why some testing needs to be done to be able to understand in detail how much power is necessary for a certain distance. This can be done first by investigating what the effect of objects are. To test this first a sender and receiver must be placed in a direct line of sight, and measure the amount of dB that is transferred and received. After this is done the same test must be done but this time with an object in the way. By doing this it is possible to know which material and object have what effect, which can help make a needed estimation needed for a range of 1 km. The test setup can be seen in figure 39.

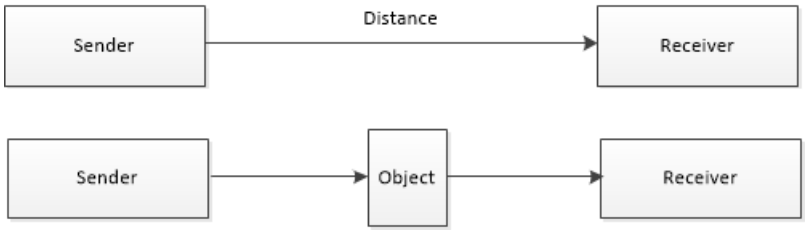


Figure 39: Setup object test

Range in weather

In the calculation of the link budget analysis the loss was calculated using free space loss. However weather has a certain effect upon data loss of the communication. This can be tested by first calculating the free space loss, and then measuring over the same distance in a direct line of sight. There will be a difference in the amount of dB. If this is done for a certain amount of time an average can be found for certain weather, and it is possible to discover the difference between free space loss and real weather effects.

Receiver power

The last thing that needs to be tested is how much power the receiver actually needs in order to be able to still receive the signal. According to the datasheet of the XBee zigbee module this is in the order of -112 dBm, but this is the absolute minimum. We can do this by connecting the sending and receiving modules together by using a wire, which has minimal loss. From that point we place dampener which is controllable and we can continue increasing the amount of dB that is lost until the data loses its integrity. Of course we need to measure the amount that is received in order to be able to make a clear conclusion. The test setup can be seen in figure 40

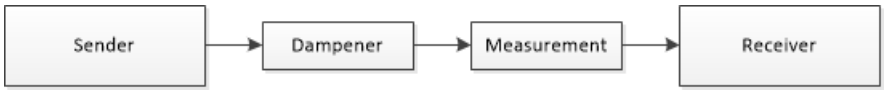


Figure 40: Receiver sensitivity test

Calculating sending power

It is important that in the worst case conditions the rover is still able to communicate at at least 1000 meters. Since the receiving and sending antenna gain are set, and using the results from the tests that are planned it is possible to accurately calculate the sending power that is needed from the Zebro. This can then be increased whenever a bigger range is needed, in trade for a lower battery life.

7.3 Integration in Zebro

If our module has passed the testing phase it needs to be integrated with the Zebro explorer. physically this is not challenging, however getting it work with the ZPU is. Therefor all the standards we made in for the Zebro need to be integrated in all the different modules. After this the Zebro should be tested in its entirety by trying the different ERC tasks and see where it struggles to complete one or more. The Zebro explorer can than be modified accordingly or maybe the user interface needs to display extra information. When the Zebro explorer can fulfill all the tasks normally it is possible to try the task in more extreme conditions or give it more difficult tasks to do. This should be all the steps for making the robot ready for the ERC, however that does not mean the team is done. There are more modules to make next to the ones that are already there. An example might be a charging system for the Zebro, this way it could be on the go for a longer period of time.

8 Discussion

8.1 ONS team

At the start of the project our only goal was to make a long range wireless communication system, and be able to navigate using LIDAR or Radar and GPS. However soon after that we needed a camera system as well, and it soon became clear that there was not enough time to make a wireless module by ourselves. This is when we decided that the project was too short and needed to use components off the shelf. This however gave us much more time, and we decided to also take the user side of the project on us. We have had a setback because of components that have not yet arrived after a long waiting time, and decided to use this time to build an eleborate user interface. At this point we also split up in two groups, group one with Alex and Peter working on all the communications outside and inside the robot and the user interface, and the other group, Bart B. and Bart R. working on all the integration of the peripherals and autonomous walking.
By having used a lot of components off the shelf we were able to build a robust system, of which each off the shelf component can be replaced by something self designed afterwards.

8.2 Zebro team

The team consisted out of 3 teams, Arm, Motor drive and ONS. Every morning we had a meeting which helped in understanding what other teams were doing. The standards set in the system integration chapter have been thoroughly discussed with other teams before we were able to set standards for everyone. The downside is being dependend upon choices, for example the arm team had to change their design a couple of times which is understandable but needs to be implemented in the UI as well. Partaking in a big team is a whole new level of working on a project and bringing its own advantages and disadvantages. One big advantage is that keeping the system modular is easier to do since integrating different modules takes a lot of time so it needs to be as modular as possible in order to be less dependend on eachother, which makes the final integration easier.

9 Conclusion

As seen in the chapter Results we have finished most of our project. Below is a short recap of all the specifications we have met or are guaranteed to meet when the parts arrive.

1. ✓ A communication system that is able to send and receive over 1000m.
2. ✓ A GPS module with with an accuracy of less than 10m.
3. ✓ A system that can detect objects in the surrounding at a distance of 10m.
4. ✓ A camera with 240p quality and around 20 fps.
5. ✓A user should be able to control the Zebro explorer manually.
6. ✓ The Zebro explorer should be able to walk autonmously from one point to another.
7. ✓An user interface containing all the necessary information to control the Zebro explorer.
8. ✓ A system where peripherals can be switched out, with either new ones or improved versions.

The orange checkmarks indicate that the system will work when everything has arrived, plus these systems are explained in the thesis of the other subgroup. For the communication system the code has been completed and when the new XBee modules arrive they can be directly connected to the system and they will work.
The green checkmarks indicate that those subsystems are completed and are already working. The user interface is completed except for the a few minor bugs and features that are almost ready to be implemented.
Specification number 6 is also in the other subgroups thesis. The rest of the specifications are discussed in the chapter results that they are finished and working.
A note on the modularity of the system is that all the peripherals can be swapped out and new or better ones can be connected as long as they have an I2C or UART interface. Which are probably two of the more used protocols. Therefor this specification can be checked off.

10 Bibliography

1. http://circuitcalculator.com/wordpress/2006/01/31/pcb-trace-width-calculator/
2. $http://www.siliconimaging.com/cmos_fundamentals.htm$
3. $http://www.cambridgeincolour.com/tutorials/camera-sensors.htm$
4. $http://www.globalspec.com/learnmore/semiconductors/microprocessors_microcontrollers/microcontrollers$
5. $Microcontrollersarchitecture, implementation\&programming$ $authors : kennethhintzanddanieltabak$
6. $Skolnik, M.2008, RadarHandbook, McGrawHill$
7. $ODonnel, RobertM.2002, SlidesIntroductiontoRadarSystems$ $, LincolnLaboratoryMassachusettsInstituteofTechnology$
8. $GlobalPositioningSystem : TheoryandPractice,$ $authorsB.Hofmann-Wellenhof, H.Lichtenegger, J.Collins.$
9. $DigitalandAnalogCommunicationSystems-LeonW.CouchII$
10. $StructuredElectronicDesign, byChrisVerhoevenetal.$
11. ODonnel, Robert M. 2002, Slides Introduction to Radar Systems, Lincoln Laboratory Massachusetts Institute of Technology
12. Skolnik, M. 2008, Radar Handbook, McGraw Hill
13. Marcoe, K. 2007, Slides LIDAR: an Introduction and Overview, Portland State University.
14. Shan, J. & Toth, C.K. 2008, Topographic Laser Ranging and Scanning: Principles and Processing, CRC Press Taylor & Francis Group
15. Orly Yadid-Pecht. Ralph Etienne-Cummings. 2007, CMOS imagers: From Phototransduction to image processing.

11 Appendices

11.1 Appendix A

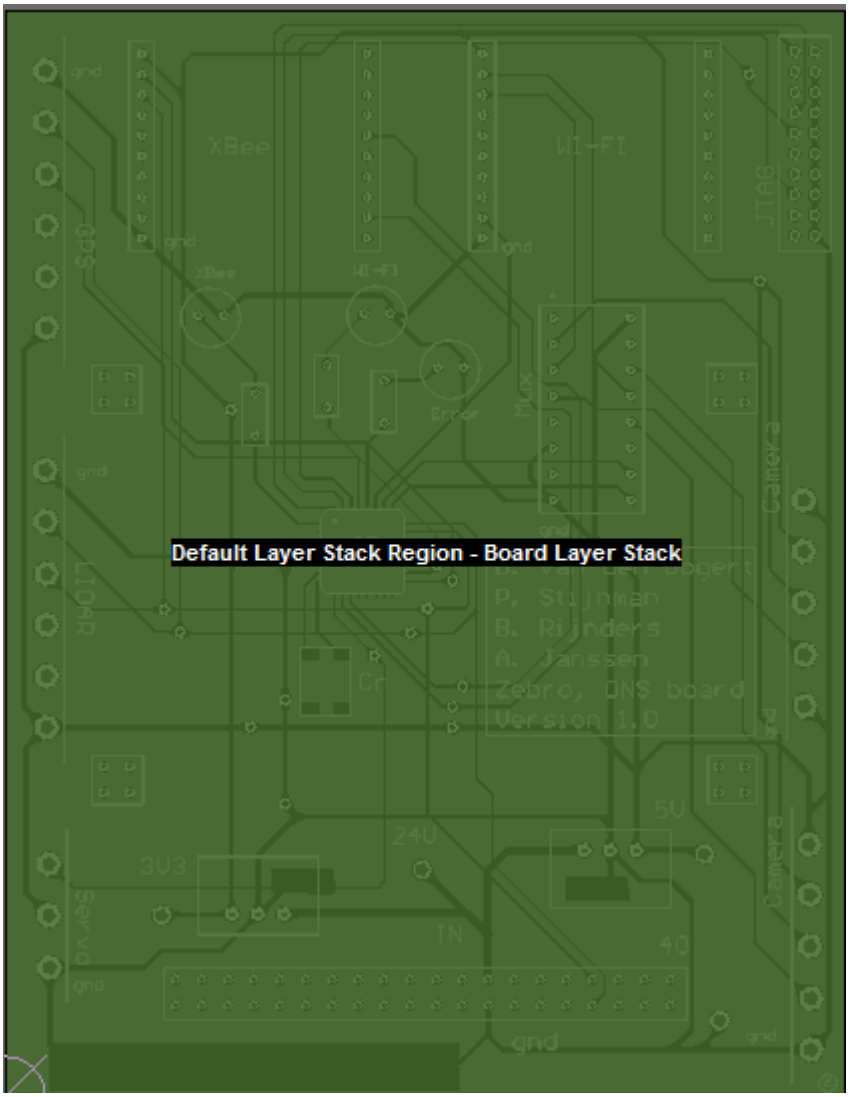


Figure 41: The board size.

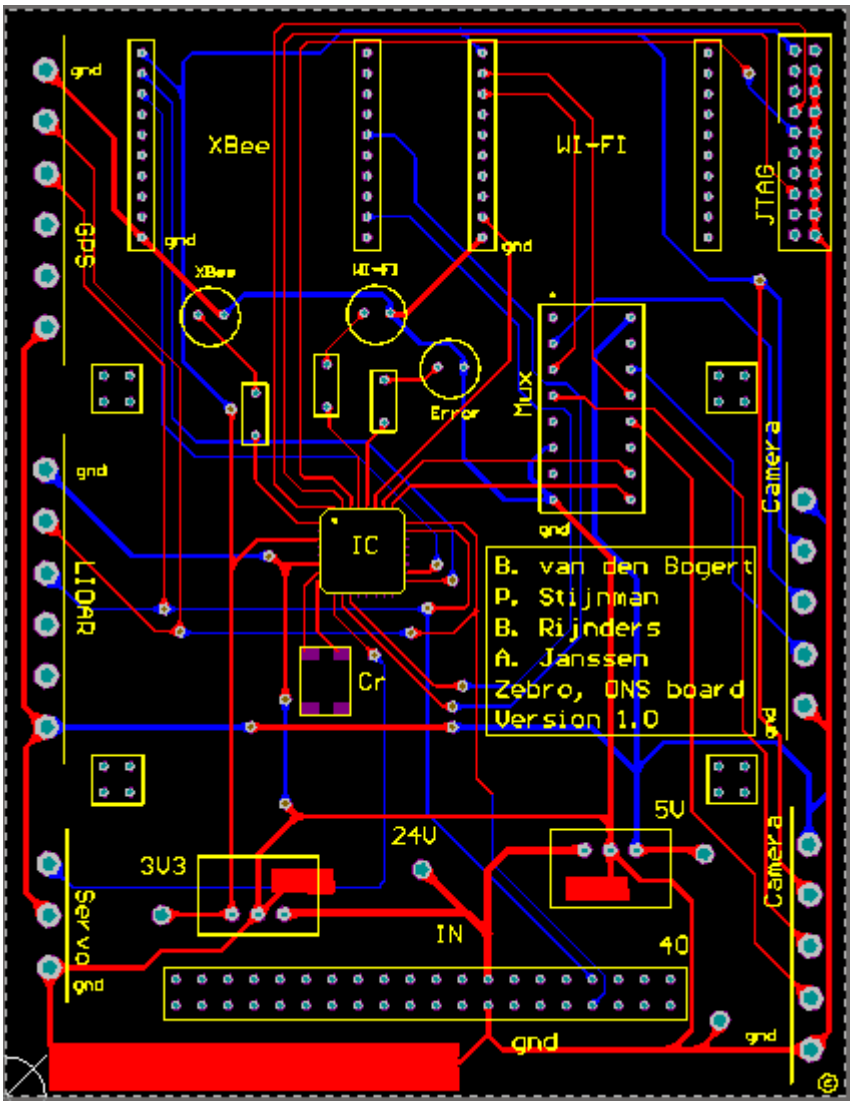


Figure 42: The PCB layout and routing.

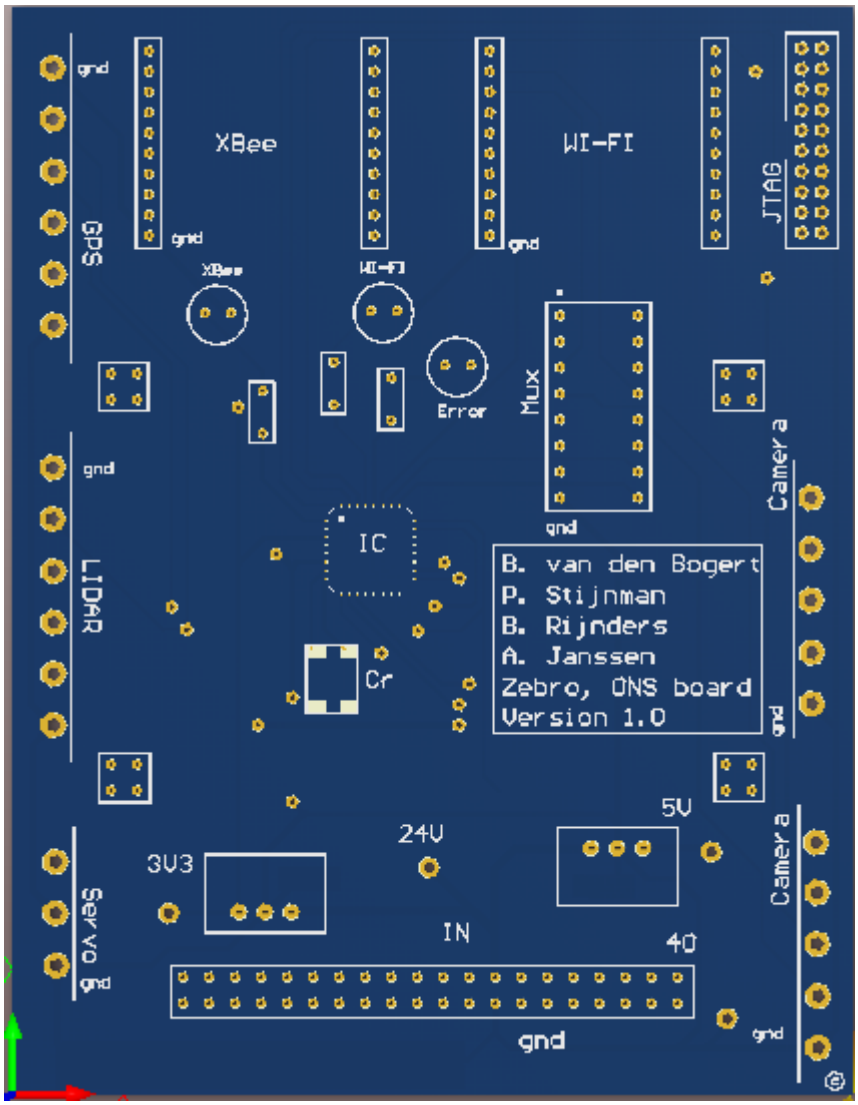


Figure 43: 3D look of the PCB.

11.2 Appendix B

Table 6: Microcontroller pin configuration

Pin number	Signal	Comment
Pin 1	Output	XBee LED
Pin 2	Not connected	
Pin 3	3v3 supply	
Pin 4	Not connected	
Pin 5	Not connected	
Pin 6	Ground	
Pin 7	External oscillator	32.768KHz
Pin 8	External oscillator	32.768KHz
Pin 9	PWM output	PWM signal for the servo
Pin 10	Output	Request to send for XBee
Pin 11	Input	Clear to send from XBee
Pin 12	Not connected	
Pin 13	Not connected	
Pin 14	Not connected	
Pin 15	Not connected	
Pin 16	Not connected	
Pin 17	UART-RX	connected to UART-TX of the XBee
Pin 18	UART-TX	connected to UART-RX of the XBee
Pin 19	Not connected	
Pin 20	Not connected	
Pin 21	Not connected	
Pin 22	Not connected	
Pin 23	I2C SCL	Connected to LIDAR,GPS and the ZPU
Pin 24	I2C SDA	Connected to LIDAR,GPS and the ZPU
Pin 25	Output	MUX B selector input
Pin 26	Output	MUX A selector input
Pin 27	Output	Sleepcontrol WiFi
Pin 28	Output	Error LED
Pin 29	Output	WiFi LED
Pin 30	SWD_CLK	Programming pin
Pin 31	RESET_B	Programming pin
Pin 32	SWD_DIO	Programming pin

Table 7: Multiplexer pin configuration

Pin number	Signal	Comment
Pin 1	Not connected	
Pin 2	UART-TX of camera	camera 1
Pin 3	UART-RX of WiFi	connected to UART-TX of camera 1 & 2
Pin 4	UART-TX of camera	camera 2
Pin 5	Not connected	
Pin 6	Ground	Inhib needs to be 0
Pin 7	Not connected	
Pin 8	Ground	
Pin 9	Input B	Connected to pin 25 of the microcontroller
Pin 10	Input A	Connected to pin 26 of the microcontroller
Pin 11	Not connected	
Pin 12	UART-RX of camera	camera 2
Pin 13	UART-TX of WiFi	connected to UART-RX of camera 1 & 2
Pin 14	UART-RX of camera	camera 1
Pin 15	Not connected	
Pin 16	5V power supply	

11.3 Appendix C: Command Table Arm

Command sent	Response	Command UI	Comments
01		BaseLeft	Whole arm turns left from base
02		BaseRight	Whole arm turns right from base
03		BaseStop	Whole arm stops turning from base
04		Up1	First joint from base keep going up
05		Down1	First joint from base keep going down
06		Stop1	First joint from base stop moving
07		Up2	Second from base joint keep going up
08		Down2	Second from base joint keep going down
09		Stop2	Second from base joint stop moving
10		Up3	Third joint from base keep going up
11		Down3	Third joint from base keep going down
12		Stop3	Third joint from base stop moving
13		WristLeft	Turn hand left
14		WristRight	Turn hand right
15		WristStop	Stop the hand from turning
16		TelescopicUp	Make telescopic part go up
17		TelescopicDown	Make telescopic part go down
18		TelescopicStop	Lock telescopic part into length
19		Grab	Grab the object
20	Angles, height, Status	Status	Ask for all statusses and response
21		Attachment	Change attachment
22	Attachment number	AttachmentStatus	Get which attachment is equipped
23		On	Turn the robotic arm on
24		Off	Turn the robotic arm off

11.4 Appendix D: Command Table Motordrive

Command sent	Response	Command UI	Comments
01		On	Turn on motors
02		Off	Turn off motors
03	Errors, statusses	Status	Receive all motor statusses
04		SitDown	Make robot sit down for stability
05		GoUp	Reactivate robot to continue walking
06		0Degrees	Walk forward
07		45Degrees	Slight turn forward-right
08		90Degrees	Turn around to the right
09		135Degrees	Slight turn backwards-right
10		170Degrees	Walk backwards
11		215Degrees	Slight turn backwards-left
12		260Degrees	Turn around to the left
13		305Degrees	Slight turn forward-left

11.5 Appendix E: Command Table ONS

Command sent	Response	Command UI	Comments
01	Distance in cm	GetDistance	Distance measurement with LIDAR
02	Distance array	ScanArea	Do a 90 degrees sweep with LIDAR
03	GPS Coordinates	GetPosition	Request the current position of the Zebro
04		FlushBuffer	Flushes the command input buffer of Zigbee
06		TurnL0	Turn LIDAR to 0 Degrees
07		TurnL10	Turn LIDAR to 10 Degrees
08		TurnL20	Turn LIDAR to 20 Degrees
09		TurnL30	Turn LIDAR to 30 Degrees
10		TurnL40	Turn LIDAR to 40 Degrees
11		TurnL50	Turn LIDAR to 50 Degrees
12		TurnL60	Turn LIDAR to 60 Degrees
13		TurnL70	Turn LIDAR to 70 Degrees
14		TurnL80	Turn LIDAR to 80 Degrees
15		TurnL90	Turn LIDAR to 90 Degrees

This page is left blank intentionally.