Guide

Willy, step by step

Inhoudsopgave

[Inleiding Willy 2](#_Toc24708070)

[Wiki 3](#_Toc24708071)

[Bijdrage projectgroep 2019-1 4](#_Toc24708072)

[Samenvatting 4](#_Toc24708073)

[Netwerktekening en technische rapportage 5](#_Toc24708074)

[How to start with Willy 7](#_Toc24708075)

[Startprocedure 7](#_Toc24708076)

[Manual driving 7](#_Toc24708077)

[Autonomous driving 7](#_Toc24708078)

[Leap motion control 8](#_Toc24708079)

[Start to learn Willy 9](#_Toc24708080)

[Directories of Willy 9](#_Toc24708081)

[Components 10](#_Toc24708082)

[Brain 10](#_Toc24708083)

[Human\_detection 10](#_Toc24708084)

[Keyboard 10](#_Toc24708085)

[Lidar 10](#_Toc24708086)

[Motor\_controller 10](#_Toc24708087)

[Navigation\_stack 10](#_Toc24708088)

[Hardware components 11](#_Toc24708089)

[Lidar 11](#_Toc24708090)

[Ultrasonic sensor 12](#_Toc24708091)

[IMU 13](#_Toc24708092)

[Power distribution 15](#_Toc24708093)

[Motor controller 16](#_Toc24708094)

[The laptop 17](#_Toc24708095)

[ROS tutorial 17](#_Toc24708096)

[Example launch file 17](#_Toc24708097)

[Troubleshooting 18](#_Toc24708098)

[Leap Motion Manual 19](#_Toc24708099)

# Revisions

Project team 2019 Q3/Q4:

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Changes** |
| V1.0 | xx-xx-2019 | Initial setup of full guide |

# Introduction Willy

Willy first saw light after a competition held by foundation Art of Robotics. The idea behind Willy was to create an autonomous trash-collecting robot. Willy was then bought by Windesheim University. Willy’s goal has since been shifted to become an autonomous robot for promotional purposes. Willy has been worked on by different project teams. The end goal of Willy is to safely drive autonomously on expo’s while interacting with the public.

# Wiki

First of all, Willy has a Wiki which is linked below. While the Wiki has some interesting information on Willy, it is not an extensive guide on how to use and start with Willy. Every subject not explained on the Wiki is explained in this guide. It is advised to read the Wiki along with this guide before implementing any changes on Willy.

**Wiki:** <https://artofrobotics.github.io/WillyWiki/>

## Drive

Documentation from every group can be found in Google Drive. In this folder it contains a document filled with passwords for the project. To gain access to the Google Drive folder, you have to contact previous project group.

## Network design and technical report

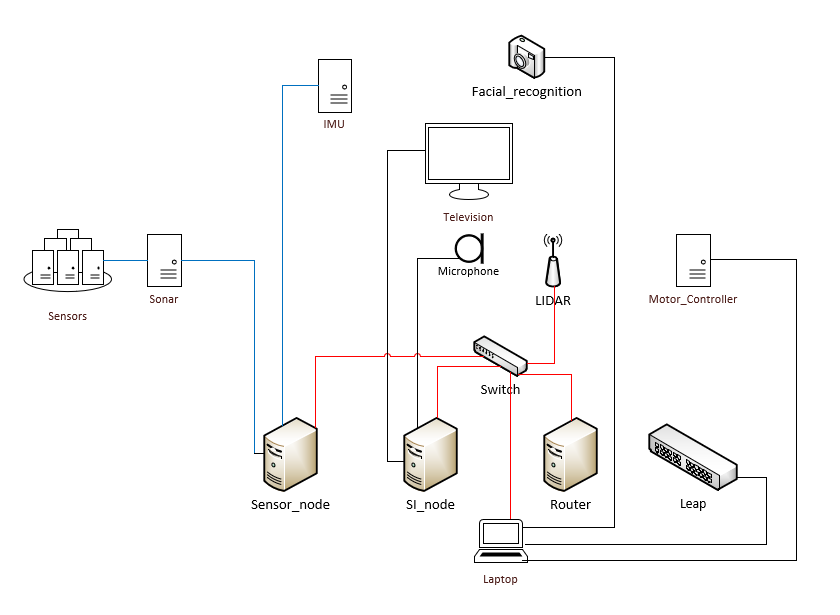
Project team 2019 Q3/Q4 had difficulties starting with Willy, especially regarding Willy’s Hardware. There was wiring everywhere and no structure. To simplify this, the team created a network schematic. 

Figure 1 Network design

The red network shows the internal communication network of Willy. This network deals with the communications between the Raspberry Pi’s and the laptop. The blue network shows the communication regarding the sensors which handle the autonomous driving of Willy.

Below, some information is shown for all the different nodes within Willy.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nodes | IP-address | Operating-System | ROS | Packages in workspace | DNS name |
| Sensor\_node | 192.168.0.13 | Ubuntu 16.04.05 LTS | ✓  1.12.14  Kinetic | * Aliencontrol * Pose\_tracking * Sensor\_node * Sonar | sensornode |
| Si\_node | 192.168.0.12 | Raspbian  9 | 🗶 | 🗶 | sinode |
| Brain\_node(physically removed) | 192.168.0.11 |  |  |  | brainnode |
| Router\_node | 192.168.0.1 | OpenWrt  4.14.120 | 🗶 | 🗶 | OpenWrt |
| Laptop | Public: 145.44.211.14  Private: 192.168.0.10 | Ubuntu  16.04.6 LTS | ✓  1.12.14  Kinetic | * Brain * human\_detection * keyboard * lidar * motor\_controller * navigation\_stack * Leapd | willy |

# How to start with Willy

Before understanding how Willy works, it is helpful to drive around with it for a bit. To do this you could choose between autonomous driving, manual or through your hand movements.

This is a short guide in driving Willy.

## Start procedure

To get Willy up and running, the following procedure must be followed.

1. Twist the red handle on the left side of Willy vertically. This is the switch for the central power circuit of Willy. Be aware of the red emergency button above the laptop. (This button enables the emergency brakes when pushed down)
2. Charge the battery of Willy and the laptop. Both chargers can be found on or near Willy. Willy’s charging port can be found underneath the previously mentioned red handle. Willy can only be charged if the handle is positioned vertically. (You should be able to hear when Willy is turned on)
3. Start the laptop.
4. Open a terminal.
5. Make your choice between manual, LEAP motion or autonomous driving. The explanation on how to use these modes will be explained below.

## Manual driving

Willy can be controlled manually with the PS2 controller. This is the procedure:

1. Make sure the controller is plugged into Willy’s left USB port on the laptop.
2. Open a terminal on the laptop.
3. Enter the command: “startjoystick” in the terminal.
4. Open a new terminal.
5. Enter the command: “startmotorcon” in the terminal.
6. Press the PlayStation button in the center of the controller to turn it on. A red light should now burn on top of the controller. Whenever the controller is activated, the laptop’s touchpad might act weird.
7. Disengage the emergency brakes. (By releasing the emergency button)
8. Willy can now drive by moving the left joystick while holding the R1 button. There is some delay, so caution is necessary.

## Autonomous driving

Willy has the capability to drive autonomously. This is the procedure:

1. Open a terminal.
2. Enter the command: “startwilly” and enter the password.
3. A new window opens called Rviz. This is used to control Willy and visualize its position.
4. On this window you can see a map of T5.
5. Disengage the emergency brakes. (By releasing the emergency button)
6. In Rviz, set Willy’s pose estimate within the T5 environment.
7. In Rviz, drag a destination arrow on the map. This will enable Willy to try and drive to the location autonomously.

## Leap motion control

Willy can be controlled with hand movements using the LEAP motion sensor . This is the procedure:

1. Open a terminal.
2. Enter the command “startleap”.
3. A few terminals will now open with the input that ROS gets from the LEAP controller.
4. Disengage the emergency brakes. (By releasing the emergency button)
5. Control Willy with hand movements(Roll and pitch). This might require some practice.

# Start to learn Willy

To learn about Willy, it’s important to know what kind of data there is, how to use it and which hardware you are working with. At first, you will be taken on a tour through the directories of Willy and which components these directories populate. After this, the hardware will be explained globally. To get more specific specification about the hardware, take a look at the Wiki of Willy. After the hardware has been explained, the focus will be on how to use the software installed on Willy.

## Directories of Willy

The figure below shows the directory of Willy’s main process. In red the components currently not in use. The command ‘startwilly’ is called from the console to start Willy. This command calls the script ‘start-willy.sh’ located in the ‘Documents/willy’ folder. The script opens a roscore process and turns on the different components in the component folder. The information header below explains these components and their function.



# Components

## Brain

*This component is not in use at the moment*

The functionality of this component was to control the different states Willy is in. There are a few different states Willy can be in: Manual user control, autonomous control, emergency state, social interaction state and a state where Willy moves to a given position in relation to its surroundings.

## Human\_detection

*This component is not in use at the moment*

This component gives Willy the ability to recognize upper human body parts using a camera. It uses the OpenCV Haar Cascade algorithm to track the rectangles from the camera output and counts the frames it has seen and missed. Based on those frames, it calculates an accuracy. When this accuracy drops below 30% or the rectangle has not been seen for 50 frames, it will be removed from the tracking. Due to the specific output of the human\_detection, it has its own ROS message type with parameters.

## Keyboard

This component handles the joystick connection. This component consists of a few functionalities. There is a launch file that creates the joystick and teleop\_twist\_ joy nodes. There is also a python script for moving Willy with the keyboard.

When the command ‘startjoystick’ is called, the launch file is called and the joystick device is enabled.

## Lidar

This component handles the LIDAR. There is a package within this workspace that directs the LIDAR sensor. This component calls the tim511 launch file and creates the necessary nodes for the LIDAR product in use. These files are from the manufacturer’s product package.

## Motor\_controller

This is the only component that is not started by the command ‘startwilly’. This component contains a main.ino file that handles the motor’s velocity and movements. This file has been uploaded to an Arduino board connected to the motor encoders.

## Navigation\_stack

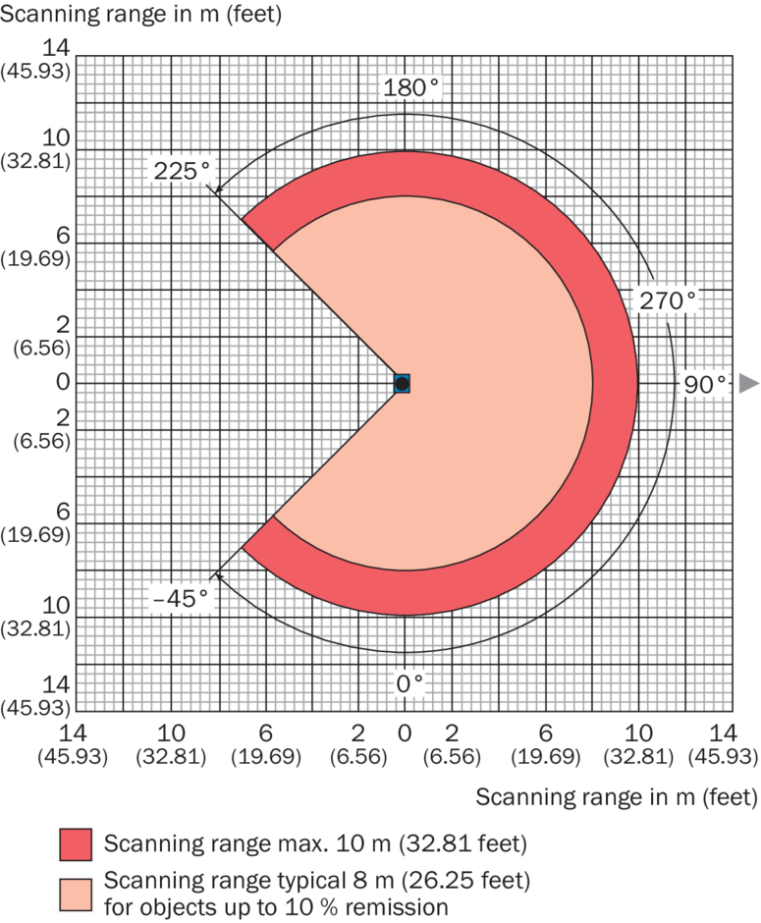
This component is in control of Willy’s navigation. It controls the mapping of Willy’s location and visualizes several types of information about the navigation on an RVIZ panel.

# Hardware components

In this part of the manual, the components on Willy are explained.

## Lidar

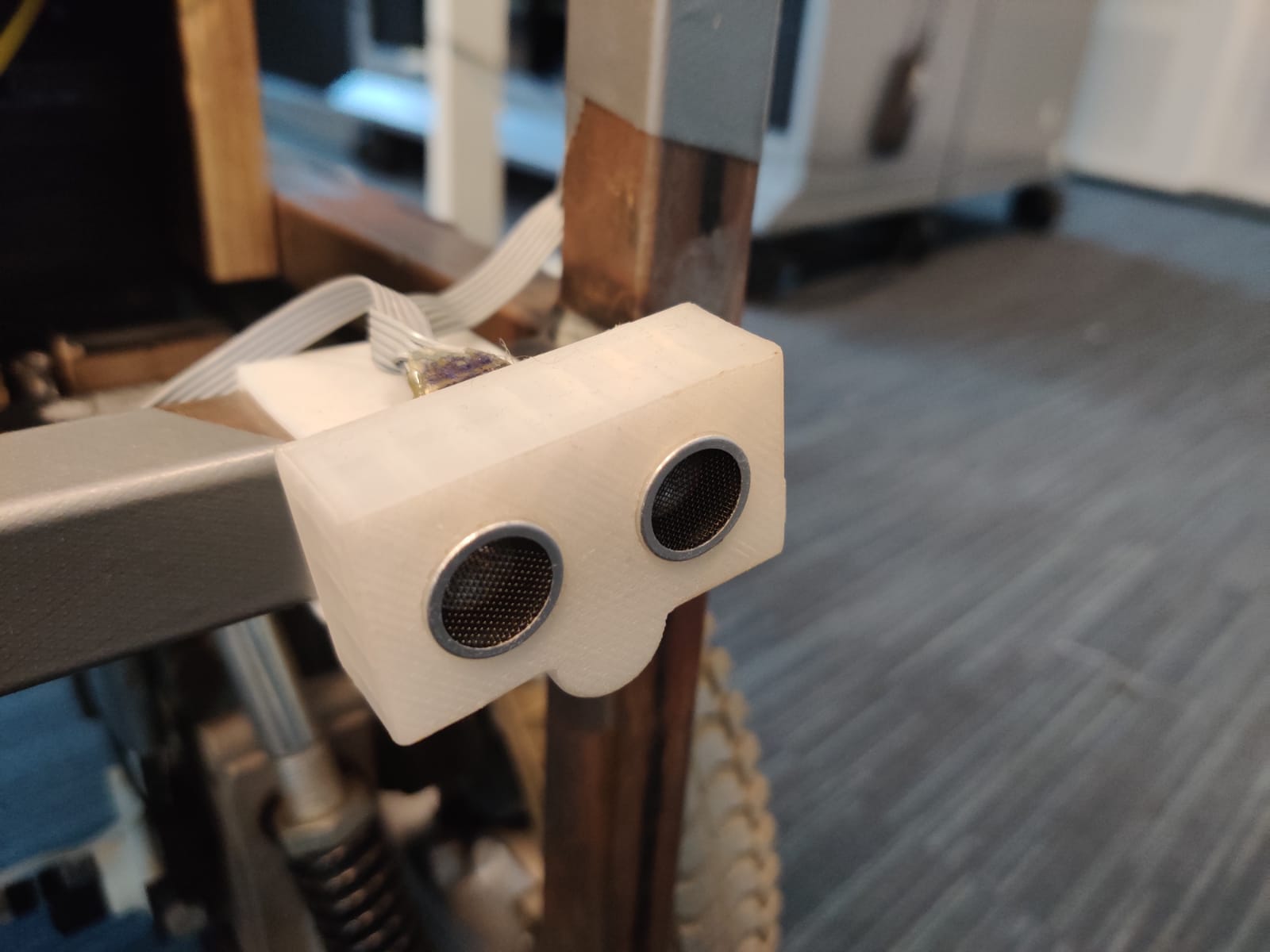
The lidar is a device that sends IR beams while rotating and measures how long it took to get feedback on these points. With this information, it can now calculate the distance to a particular object. The lidar does have trouble detecting see-through objects, because the IR beams go through them. The lidar is a SICK TIM551-2050001, the specifications about the range of this device’s scans are shown below. The lidar has a scan frequency of 15hz and is able to measure at an angle of 1 degree in height. The Lidar is connected through a network cable to the switch, and sends data to the laptop of Willy.

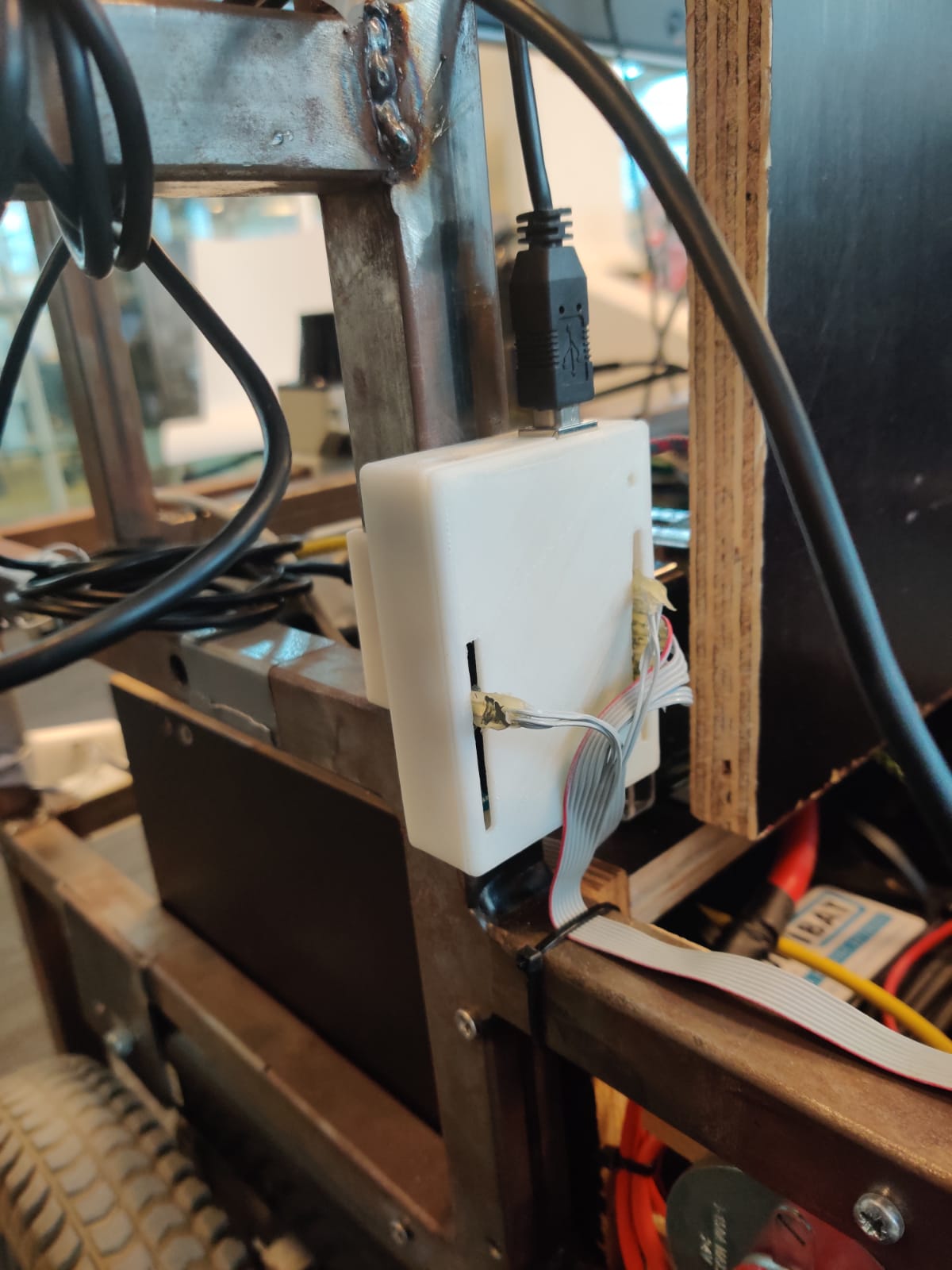
<https://www.sick.com/nl/nl/meet-en-detectieoplossingen/2d-lidar-sensoren/tim5xx/tim551-2050001/p/p343045>

## Ultrasonic sensor

This chapter will give information about the Ultrasonic sensors(HC-SR04) on willy. When project team 2019 Q3/Q4 was working on Willy, it was equipped with 8 of them. These sensors can measure up to 30 degrees horizontally, but has an effective range of 15 degrees. They can detect objects at a distance of up to 4 meters. These sensor are primarily used to detect when objects are too close and for objects which the lidar cannot detect. This helps Willy stop on time before a collision. The sensors have one side which receives the ultrasonic sounds and one which sends them.



The case below holds the arduino where all of the ultrasonic sensors report to. This arduino is connected to the sensornode.



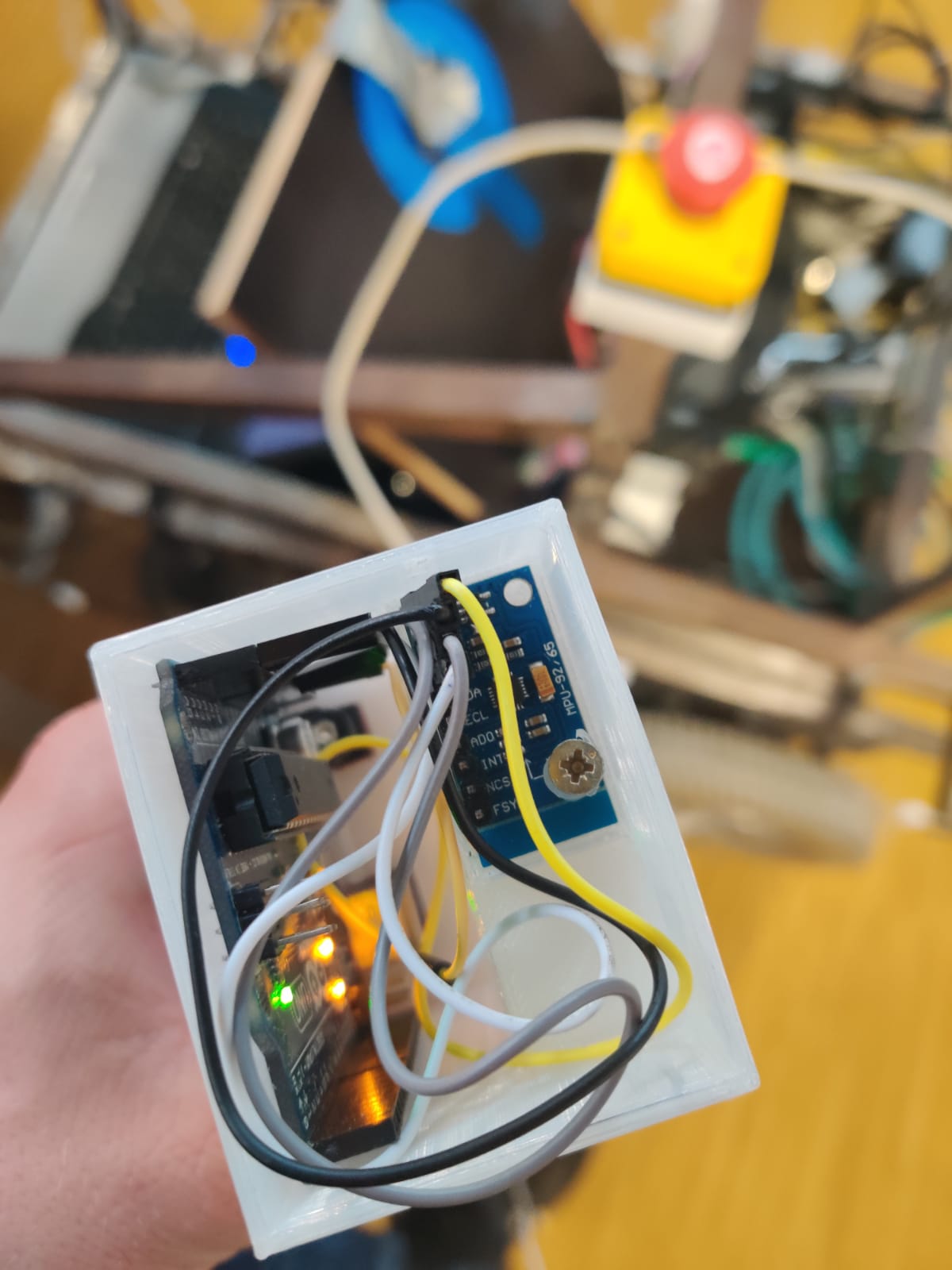
## IMU

*This component is not in use at the moment*

The IMU is a component which aids in measuring the speed and angle of Willy. The device measures this by use of a 3-Axis gyroscope, a 3-Axis accelerometer and a 3-Axis magnetometer. These sensors measure the angle, speed and magnetism to aid in finding the location of Willy. The sensor used is a MP-92/65, paired with an Arduino which is connected to the sensornode. This sensor is located on top of the screen.

**The IMU’s USB port(ACM0, bottom-right) is hardcoded to the sensor-node.**

Beware of any changes in USB-ports on the sensornode. Even plugging something in and out again has proven to lead to issues regarding the sensornode. The project team 2019 Q3/Q4 has plugged out the IMU because is causes a lot of problems and the team has installed a back-up.

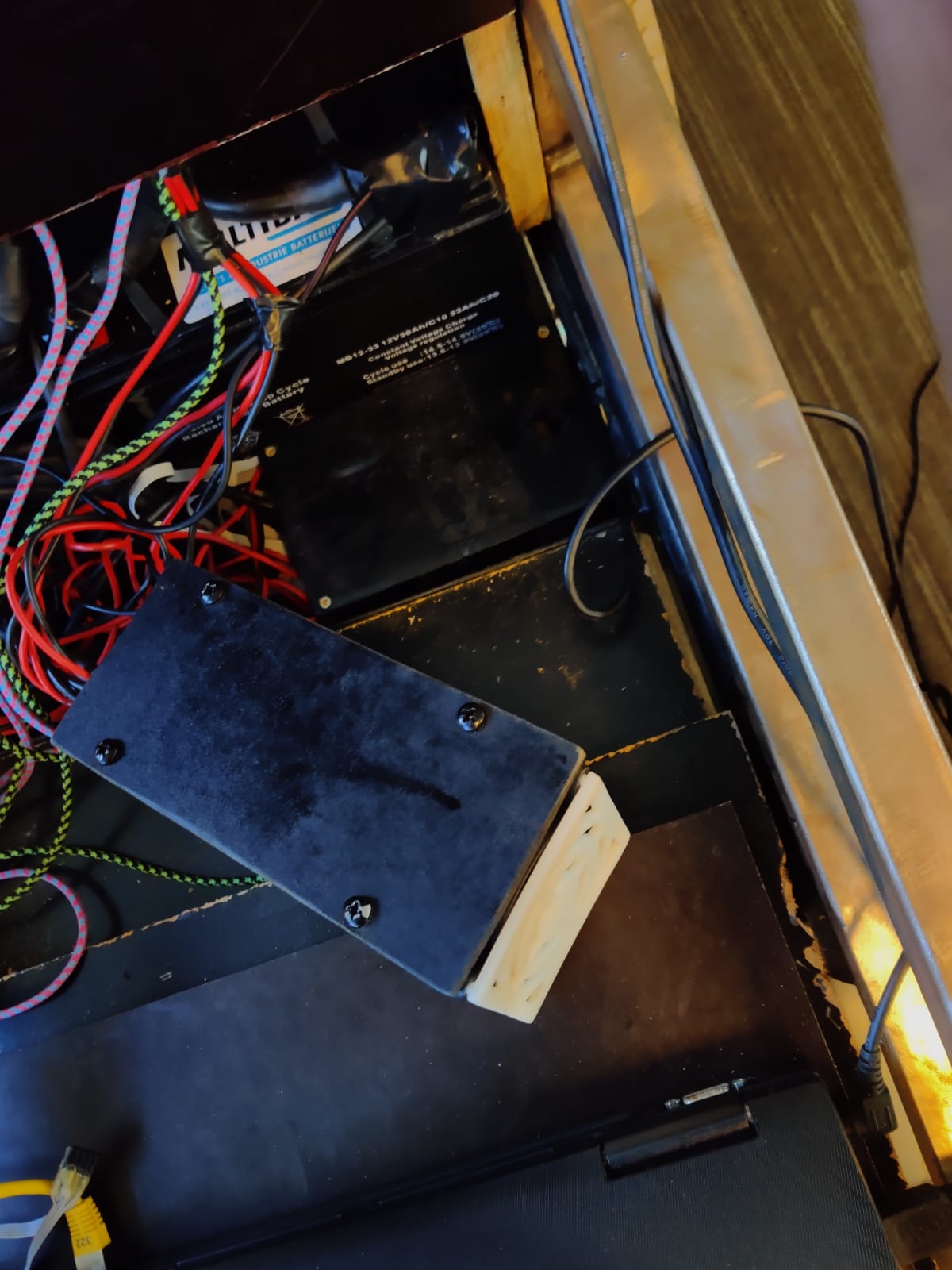




## Power distribution

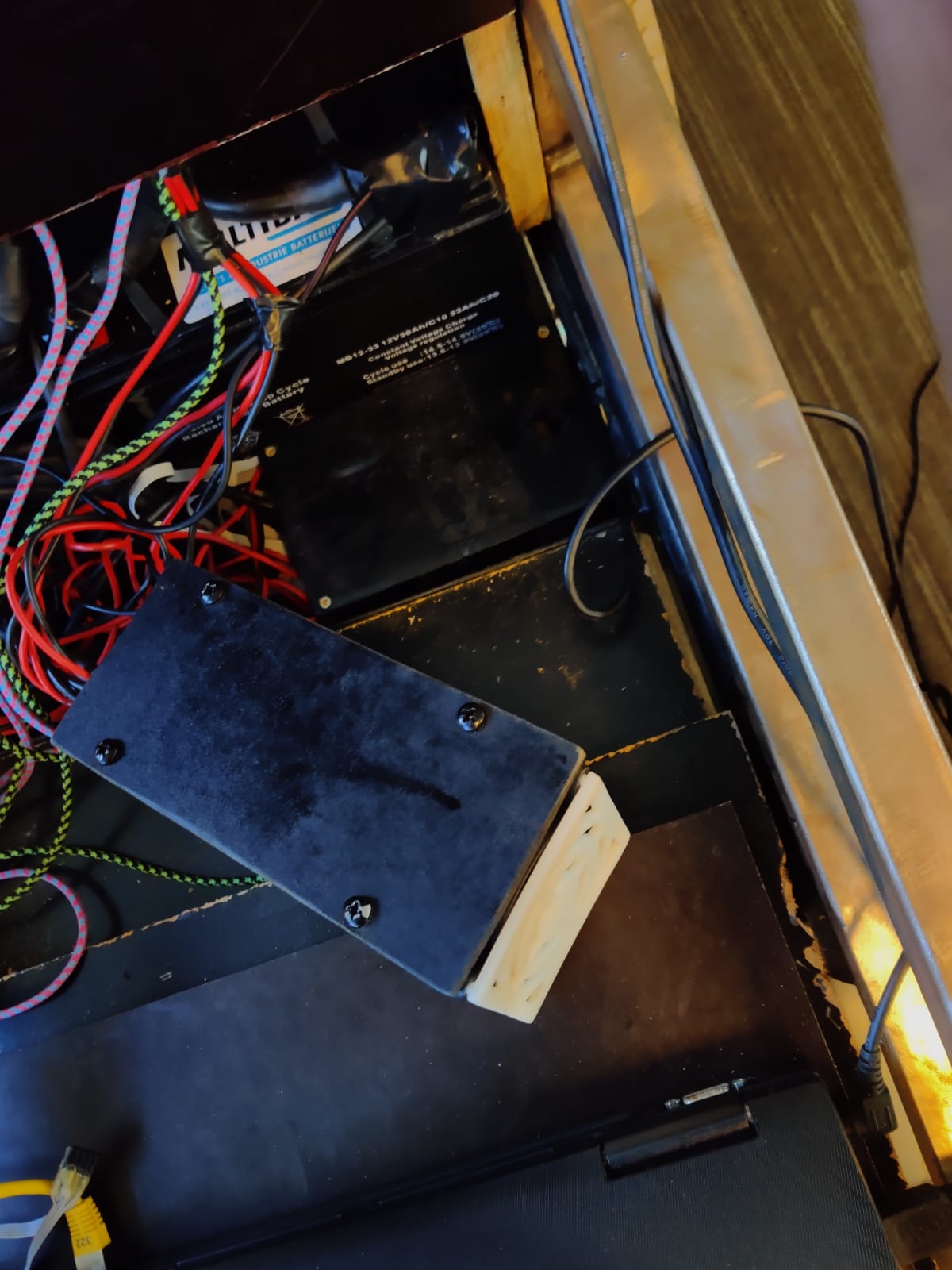
Power distribution on Willy starts with four 12V battery’s. These are located in the front of willy, under all of the wood. These batteries are placed in series so they create a 24V circuit. The batteries can be seen in the picture below above the highlighted area.

The box highlighted below is used to distribute power across Willy. This device holds different transformers to supply the TV, Raspberries, motor controller, etc. with power. This box is located under the flap behind the laptop.



## Motor controller

The motor controller is highlighted in the picture below. This is an Arduino which uses Twists messages from ROS to send commando’s to the computer of the base. This is located below all the wiring, in the original box which came with the mobility scooter. The motor controller connects directly to the laptop.



## Kinect

# The laptop

The laptop itself is running Ubuntu 16.04 with ROS kinetic. When logged in to this laptop, the user profile is called ‘willy’. Most of the ROS files are located in ‘/home/willy/Documents/willy’. To get more information about the files, take a look at chapter:

This laptop is used as the main hub for ROS to do its job. It is connected to all the Raspberry Pis through the switch located below the screen.

This laptop has an i5 CPU, 16GB RAM and is equipped with an SSD to make it as fast as possible.

# ROS tutorial

To learn more about ROS, it is recommended to practice with it. To do this, the following tutorial is recommended: <http://wiki.ros.org/ROS/Tutorials>. You should start by setting up a virtual machine and walk through the beginner tutorial to get a general sense on how ROS works.

Some important commando’s to focus on are:

* rostopic
* rosnode
* roslaunch
* rosrun

# Example launch file

# Troubleshooting

When willy doesn’t drive after activating the autonomous or manually driving modes, you should press and release the emergency stop button. This reenables the connection to the motor controller.

# Leap Motion Manual

This manual is an example on how to install something on ROS and make it work. This is a step by step installation of the leap motion, making it work as intended.

1. Install the Leap motion drivers. This is done by downloading them from the leap motion website. Afterwards, install the drivers. Execute the following commando’s:
   1. Wget <https://leapmotion.cachefly.net/Protected/expiretime=1570701568/5a37d1f2d5ffd2bbb9fcf07a3363bfde/f9a7608/Leap_Motion_Setup_Linux_2.3.1.tgz>
      1. This link can change depending on the version.
   2. tar -xvzf Leap\_Motion\_Setup\_Linux\_2.3.1.tgz
      1. This wil unzip the leap motion drivers.
   3. sudo dpkg --install Leap-[VERSION]-x64.deb
      1. This will install the leap motion, replace [VERSION] with the correct version number.
2. Create the folder in which the Leap will be installed, this needs to be done in willy/components to keep it organized. And then put the ROS-drivers into the folder.
   1. mkdir -p ~/Documents/willy/components/leap\_ws/src
   2. cd ~/Documents/willy/components/leap\_ws/src
   3. git clone https://github.com/ros-drivers/leap\_motion.git
3. Append your LeapSDK location, this is a BASH terminal wide variable. You can put it in ‘/.bashrc’, which eliminates the need to execute it every time you want to run something that requires this export.
   1. Vi ~/.bashrc
   2. Copy the exports below to the bottom of the .bashrc file.
   3. export PYTHONPATH=$PYTHONPATH:$HOME/Documents/willy/components/leap\_ws/LeapSDK/lib:$HOME/Documents/willy/components/leap\_ws/LeapSDK/lib/x64
   4. export LEAP\_SDK=$LEAP\_SDK:$HOME/Documents/willy/components/leap\_ws/LeapSDK
4. Copy the libleap.so file to ‘/usr/local/lib’ using the following command:
   1. sudo cp $LEAP\_SDK/lib/x64/libLeap.so /usr/local/lib
   2. sudo ldconfig
5. Catkin make the workspace, this will install the leap\_motion driver.
   1. Cd ~/Documents/willy/components/leap\_ws
   2. Catkin\_make
6. Test if the driver works.
   1. Sudo leapd
   2. LeapControlPanel
      1. If an error occurs, execute the following: sudo service leapd restart
7. Initiate the workspace.
   1. source ~/Documents/willy/components/leap\_ws/devel/setup.bash
8. Try to run the demo.launch file to see if everything works correctly.
   1. Sudo leapd
   2. Sudo LeapControlPanel
   3. Roslaunch leap\_motion demo.launch

The steps from 9 to 14 are optional, they make it easier to troubleshoot the leap with ROS.

1. In this part of the manual, the leap client is going to be installed. Which will be needed to visualize the leap motion data inside of ROS. First, add another export to the /.bashrc file.
   1. Export LEAPSDK=$LEAPSDK:$HOME/Documents/willy/components/leap\_ws/LeapSDK
2. Add the leap\_client to the workspace, and install it.
   1. Cd ~/Documents/willy/components/leap\_ws/src
   2. Git clone <https://github.com/qboticslabs/leap_client.git>
   3. Cd ..
   4. Catkin\_make

1. Now the leap client is installed, make sure leapd and LeapControlPanel are running.
   1. roslaunch leap\_motion sensor\_sender.launch
   2. Open a new terminal
   3. roslaunch leap\_client leap\_client.launch
   4. rosrun rviz rviz -d ~/Documents/willy/components/leap\_ws/src/leap\_client/launch/leap\_client.rviz
2. The next step is to create a ROS package for this node. There are two ways to create this. By executing a command or coping all the files from a GitHub. Here, it’s going to be created from scratch.
   1. Cd ~/Documents/willy/components/leap\_ws/src
      1. catkin\_create\_pkg vr\_leap\_teleop roscpp rospy std\_msgs visualization\_msgs geometry\_msgs message\_generation visualization\_msgs
   2. cd ..
   3. catkin\_make
3. Create a script which will hold all of the configurations for the leap motion.
   1. cd ~/Documents/willy/components/leap\_ws/src/vr\_leap\_teleop
   2. mkdir scripts
   3. vi vr\_teleop.teleop.py

copy the script lines below:

#!/usr/bin/env python

\_\_author\_\_ = 'flier'

import rospy

from leap\_motion.msg import leap

from leap\_motion.msg import leapros

from geometry\_msgs.msg import Twist

teleop\_topic = '/cmd\_vel\_mux/input/teleop'

low\_speed = -0.3

stop\_speed = 0

high\_speed = 0.3

low\_turn = -0.3

stop\_turn = 0

high\_turn = 0.3

pitch\_low\_range = -10

pitch\_high\_range = 30

roll\_low\_range = 50

roll\_high\_range = -50

tempRoll = 0

tempCount = 0

# Callback of the ROS subscriber, just print the received data.

def callback\_ros(data):

global pub

global tempRoll

global tempCount

msg = leapros()

msg = data

roll = msg.ypr.x

pitch = msg.ypr.y

yaw = msg.ypr.z

# if current roll value is not the same as the previous roll value

# it will assign the current value to the temporary value

if(tempRoll != roll):

tempRoll = roll

tempCount = 0

# else it will count to 5

# when 5 has been reached, the yaw, pitch and roll will be set to 0

# this is to prevent leap from keeping values even after hands have disappeared

else:

if(tempCount == 5):

yaw = 0

pitch = 0

roll = 0

elif(tempCount < 5):

tempCount += 1

twist = Twist()

# resets values

twist.linear.x = 0; twist.linear.y = 0; twist.linear.z = 0

twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = 0

pitch\_low = False

pitch\_high = False

roll\_left = False

roll\_right = False

# these statements will check the hand's position

# when the pitch and roll values of the hand are in the defined ranges,

# it will set the linear and angular twist values accordingly

# each statement sets a variable to True, it's counterpart to False

# this is to let other statements know in what state the hand is currently in

# when hand is tilted forward

if(pitch > pitch\_low\_range - 40 and pitch < pitch\_low\_range):

twist.linear.x = high\_speed; twist.linear.y = 0; twist.linear.z = 0

twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = 0

pitch\_high = True; pitch\_low = False

# when hand is tilted backwards

elif(pitch > pitch\_high\_range and pitch < pitch\_high\_range + 30):

twist.linear.x = low\_speed; twist.linear.y = 0; twist.linear.z = 0

twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = 0

pitch\_high = False; pitch\_low = True

# when hand is tilted left

if((roll > roll\_low\_range and roll < roll\_low\_range + 40) or (roll > roll\_low\_range + 30 and roll < roll\_low\_range + 80)):

# if pitch is high or low, it will not change the linear twist values

if not (pitch\_high or pitch\_low):

twist.linear.x = 0; twist.linear.y = 0; twist.linear.z = 0

# invert the angular z if hand is tilted backwards, else turn normally

if(pitch\_low):

twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = low\_turn

else:

twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = high\_turn

roll\_left = True; roll\_right = False

# when hand is tilted right

elif((roll > roll\_high\_range - 40 and roll < roll\_high\_range) or (roll > roll\_high\_range - 80 and roll < roll\_high\_range - 30)):

# if pitch is high or low, it will not change the linear twist values

if not (pitch\_high or pitch\_low):

twist.linear.x = 0; twist.linear.y = 0; twist.linear.z = 0

# invert the angular z if hand is tilted backwards, else turn normally

if(pitch\_low):

twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = high\_turn

else:

twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = low\_turn

roll\_left = False; roll\_right = True

# publishing to the twist topic

pub.publish(twist)

# console print statements

if(pitch\_high or pitch\_low or roll\_left or roll\_right):

rospy.loginfo(rospy.get\_name() + ": Roll %s" % roll)

rospy.loginfo(rospy.get\_name() + ": Pitch %s" % pitch)

rospy.loginfo(rospy.get\_name() + ": Yaw %s" % yaw)

if(pitch\_high):

rospy.loginfo("Pitch High")

if(pitch\_low):

rospy.loginfo("Pitch Low")

if(roll\_left):

rospy.loginfo("Roll Left")

if(roll\_right):

rospy.loginfo("Roll Right")

# Yes, a listener aka subscriber ;) obviously. Listens to: leapmotion/data

def listener():

global pub

rospy.init\_node('leap\_sub', anonymous=True)

rospy.Subscriber("leapmotion/data", leapros, callback\_ros)

pub = rospy.Publisher(teleop\_topic, Twist, queue\_size=1)

rospy.spin()

if \_\_name\_\_ == '\_\_main\_\_':

listener()

1. Create a launch file, this will make it easier to start. Because of different types of drive management, the leap sends its twist messages, which are the X, Y, and Z axes. Willy uses these axes to drive in a specific direction. Right now, the vr\_leap\_teleop publishes its data to ‘/cmd\_vel\_mux/input/teleop’. This will not work for Willy because the motorcontroller subscribes to ‘/cmd\_vel’. To do this, a remap of this needs to be created. But first, create the launch folder if it does not exist.
   1. Cd ~/Documents/willy/components/leap\_ws/src/vr\_leap\_teleop/launch
   2. Vi vr\_leap\_teleop.launch
   3. Copy the below code into the launch file.

<launch>

<node pkg="vr\_leap\_teleop" name="vr\_leap\_teleop" type="vr\_leap\_teleop.py" output="screen">

<remap from="cmd\_vel\_mux/input/teleop" to="cmd\_vel"/>

</node>

</launch>

1. Inside of the launch file you can see it calls to the node pakage ‘vr\_leap\_teleop’ and runs the python script created earlier. Now it remaps the cmd\_vel\_mux to the cmd\_vel which willy uses. Now willy can be started, this can be done by the following sequence of commands:
   1. Sudo leapd
   2. LeapControlPanel
   3. Source ~/Documents/willy/components/leap\_ws/devel/setup.bash && roslaunch leap\_motion sensor\_sender.launch
   4. Startmotorcon
   5. Source ~/Documents/willy/components/leap\_ws/devel/setup.bash && roslaunch vr\_leap\_teleop vr\_teleop.launch
2. Now Willy will be able to be controlled by the leap controller.

<https://github.com/juancamilog/leap_client>

<https://github.com/ros-drivers/leap_motion>

<https://learning.oreilly.com/library/view/ros-robotics-projects/9781783554713/ch11s05.html>

**RVIZ visualization**

* Sudo leapd
* Sudo LeapControlPanel (be aware of the capital letters)
* Rosrun leap\_motion sensor\_sender.py
* roslaunch leap\_client leap\_client.launch
* rosrun rviz rviz -d ~/leap\_ws/src/leap\_client/launch/leap\_client.rviz

<https://answers.ros.org/question/51474/can-i-run-a-bash-script-using-roslaunch/>