

# User Manual

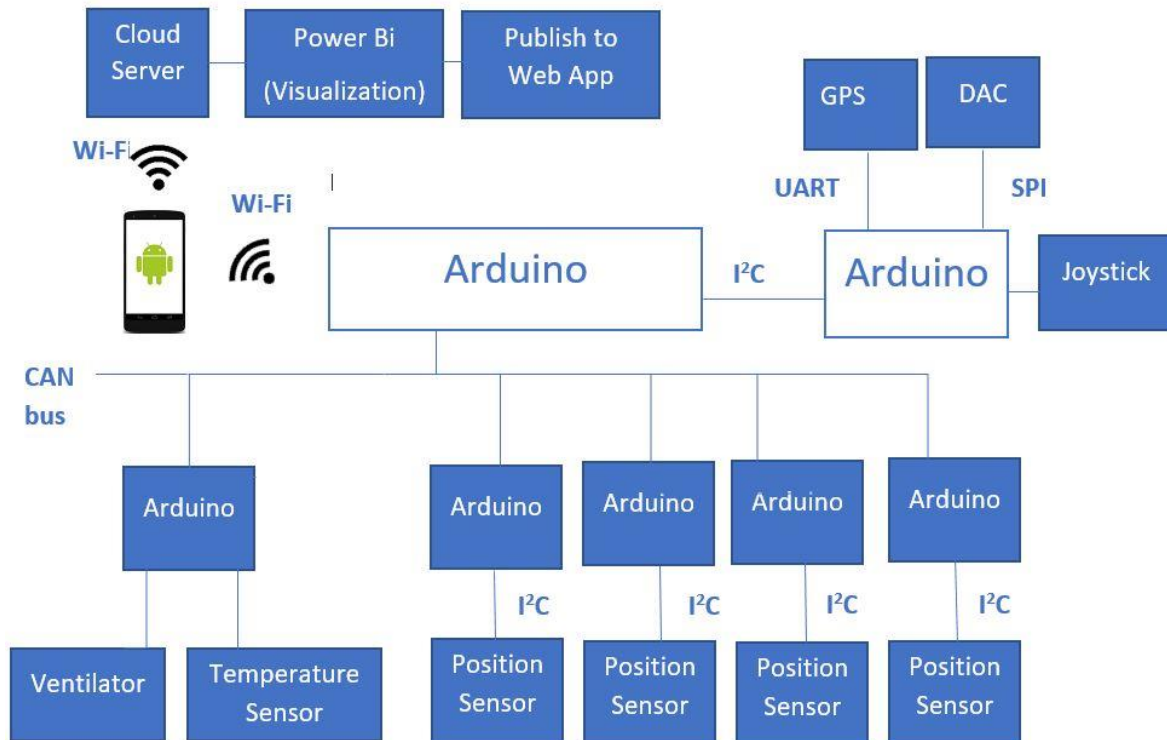
EE5: MOWING MACHINE 1  
FOR AWB SCHOTS

## Table of Contents

Introduction.....	2
Useful links .....	2
Installation .....	3
Power Bi.....	3
Azure Cloud Services .....	4
Mobile Application .....	6
Installing the App.....	6
Adapting the App.....	7
Hardware .....	9
Joystick .....	9
Connecting Master and Slave Arduino .....	9
Position Sensors.....	9
Ventilator.....	10
CAN Bus .....	12
Power .....	12
User Manual.....	13
Android Application .....	13
To start.....	13
To finish .....	14
Power BI .....	14
Using Our Report .....	15
Ventilator Control Configuration.....	18

## Introduction

This document contains all the necessary information for the implementation and future development of our project. It contains an installation guide as well as a user manual.



## Useful links

GitHub Repositories:

<https://github.com/Djensonsan/EE5>

[https://github.com/NathanHerrebosch/AWB\\_App](https://github.com/NathanHerrebosch/AWB_App)

OpenProject Wiki:

<https://openproject.groept.be/projects/mowing-machine-1/wiki/wiki>

# Installation

## Power Bi

Power Bi is an interactive data visualization tool made by Microsoft. Power Bi has a paid subscription of 8,40 euro/month. For students of the KUL, it is provided for free. You can try this software suite with a free 60-day trial.

Power Bi has 2 parts, a web-application and Power Bi desktop. Power Bi Desktop is an application you download where you can make reports. These reports can be published to the Power Bi Web-application. In this web-application, you cannot change the reports but you can share it with people in your company, publish it to a website and do a lot more.

To make an account head over to:

<https://powerbi.microsoft.com/en-us/>

(When you click sign in you will be redirected to the web application)

Download Power Bi Desktop:

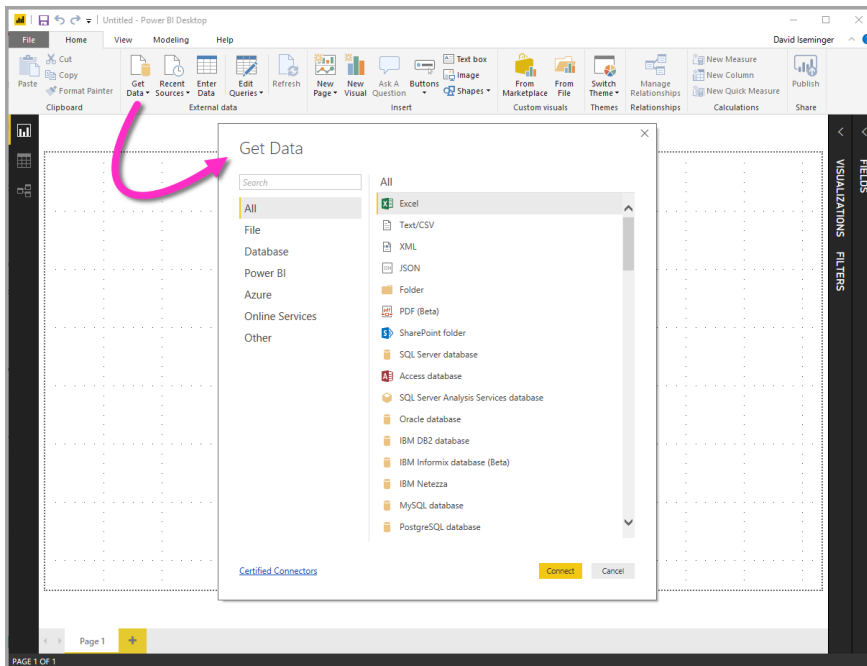
<https://powerbi.microsoft.com/en-us/desktop/>

We have made a report in Power Bi Desktop that visualizes the movement of the arm of the mower and the route of the mower. This Report is a file with the extension “.pbix” and should be handed over to AWB Schots. If you would like to make changes to the visualization, open the report in Power Bi desktop. If you want to share this report, you first must ‘Publish’ it. More information on publishing and sharing Power Bi reports can be found here:

<https://docs.microsoft.com/en-us/power-bi/guided-learning/publishingandsharing?tutorial-step=2>

## Azure Cloud Services

Power BI needs a data source for its visualizations, a lot of different data sources are supported as shown below.



We use a Microsoft Azure SQL Database to store our lawn mower data and a Mobile service app on Azure to push data from the mobile phone to the SQL database.  
The Price for a cloud SQL database varies with the size of the database. A 2GB SQL database costs 2 USD per month. A 250 GB SQL database costs around 25 USD per month.

To make an Azure account of your own, head over to:

<https://azure.microsoft.com/en-us/>

To log in with my student account:

Mail: [jens.leysen@outlook.com](mailto:jens.leysen@outlook.com)

Password: fuIFT67o

The account has 100 USD of free credit available.

To connect a new Azure database to the visualization report in Power BI:

- Click on Get Data>Azure>Azure SQL database
- Fill in the credentials you chose when setting up the SQL database
- Select the table you would like to import and import the data via “direct import”

You should then be able to use the provided report to visualize this data.

In case you would like to make changes to the database:

- Click SQL databases on the right, and select EE5\_Test.
- To query the Database, go to the Query editor. You will be prompted for a username and password:  
    Username: Djensonsan  
    Password: p64EsRG3NcVy.
- You should then be able to query the database using the Transact-SQL syntax.

If you would like to make a new mobile application that connects to your database:

- Go to App Services
- Click AWBApp>Quickstart.
- Select the operating system the app should run on (normally Android).

You will then find an explanation on how to set up a new Mobile Application or connect an existing one to your database.

In case you prefer not to use any of the resources we have created in Azure and would like to start from scratch, please follow the steps presented here:

<https://docs.microsoft.com/en-us/azure/app-service-mobile/app-service-mobile-android-get-started?fbclid=IwAR0vwXqr9IL WsnO3nBctshF8rcL4M1z2eJzC2hrZxDyeIOMhjO6B392YOg>

You would then have an SQL database with one table (ToDoItem) and a Mobile App that can add new items in the table.

You can enable offline synchronization for the app by following this tutorial:

<https://docs.microsoft.com/en-us/azure/app-service-mobile/app-service-mobile-android-get-started-offline-data?fbclid=IwAR0Cy0n3frVsH-ztr-u1Y6G2UZz5QcHieVmUfYtMUag9x0WRJ34YcDcebyw>

Note:

If you would like to add tables or columns, go to the query editor as described above and write the appropriate SQL queries. However, these new tables or columns must also be added in the Android Studio project, otherwise, the app will not work properly anymore. To update your own Android Studio project to your preferences, you need a thorough knowledge of Android development, so it will not be discussed here. However, at the end of the next section of this Installation Guide, you can find a stepwise explanation on how to make the most important basic adaptations to our Android Studio project that already exists.

## Mobile Application

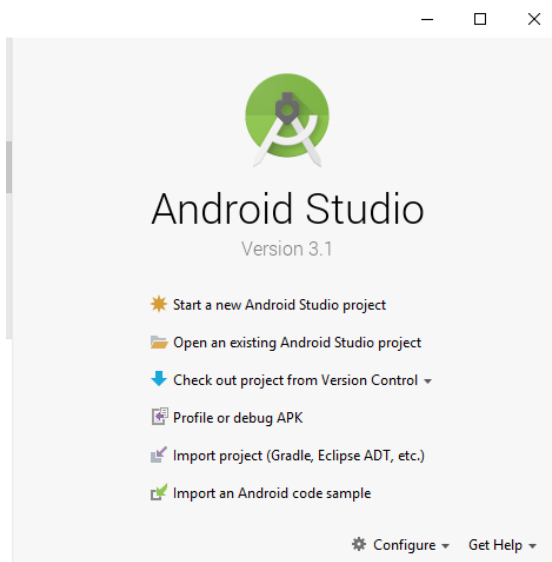
### Installing the App

The mobile application code can be found on our GitHub page (See link above). We did not publish it to the Play Store. The Application can be found on the Motorola we received at the start of the project; should you want to download the app on a different phone, there is no other choice than to download Android Studio and run the code on an Android phone that has developer options enabled.

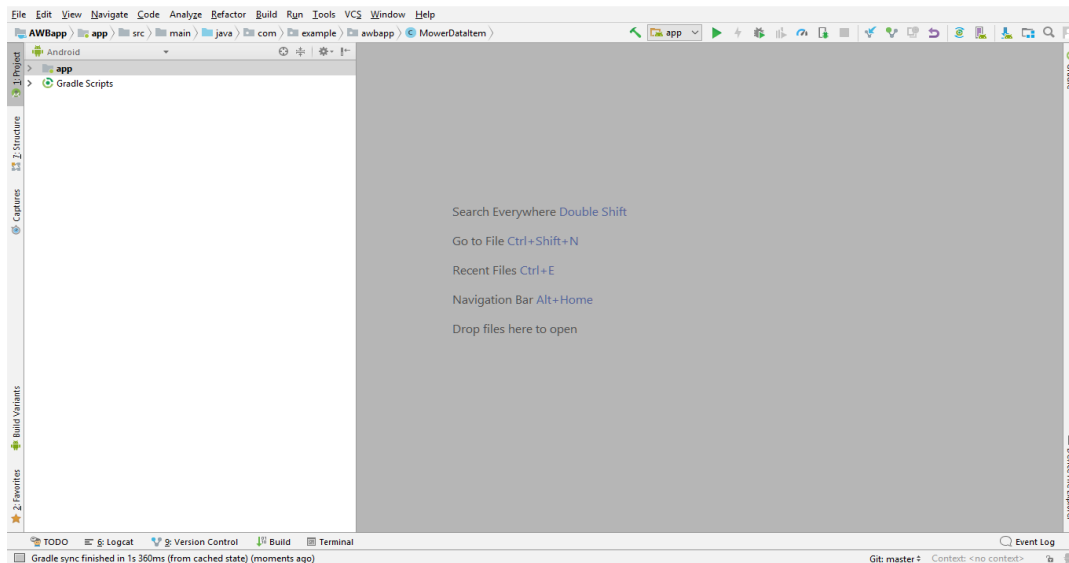
Instructions on how to enable developer options on your phone:

<https://developer.android.com/studio/debug/dev-options.html>


1. To install the app on a new device, first, open Android Studio on your computer. It can be downloaded from here: <https://developer.android.com/studio>
2. If there is no project currently opened, the following screen pops up upon launching Android Studio:



3. Click Check out project from Version Control > Github.
4. Paste this URL in the URL bar: <https://github.com/NathanHerrebosch/AWBApp>, and select the directory on your computer where you would like to save it.
5. Click Clone-the Android Studio project of the application will open.
6. If, on the other hand, the screen below pops up upon opening Android Studio, it means there was still a project open.



Click File>New>Project from Version Control>Github and follow steps 4 and 5

7. If you have successfully opened the AWBApp project, plug the Android Phone in to your computer using a USB cable. Make sure the phone has USB debugging enabled in Developer Options.
8. Go back to Android Studio on your computer and run the app on the connected device. Select Run>Run in the menu bar or click Run  in the toolbar. If Android Studio asks you to select a deployment target, select the connected device and click Ok. If you have problems with getting the app to run on your device, go to <https://developer.android.com/studio/run> for more information.

## Adapting the App

In this section, you can find an explanation on how to make some of the most important basic changes to the application. This requires some basic knowledge of Java.

To add a column to the table:

- Add  

```
tableDefinition.put("yourColumnName", ColumnDataType.YourColumnType);
```

to the `InitLocalStore()` function in `MainActivity.java`, where `YourColumnType` is either Boolean, Integer, Real, String, Date, DateTimeOffset or Other
- In `MowerDataItem.java`, add  

```
@com.google.gson.annotations.SerializedName("yourColumnName")
private yourColumnType mYourColumnName;
```

where `yourColumnType` is a java datatype (int, String, float,...)
- Add a setter and a getter for the new column in `MowerDataItem.java`
- In the `parseData()` method of `MainActivity`, add a line that sets the new column value. The precise implementation of this depends on the nature of the column.



When the Android app tries to put data in a non-existing column in the cloud database, Azure will automatically generate a new column that can hold the data. It is also not necessary to change anything in PowerBI, unless you would like to visualize this new data.

To add a new table to the database:

- In the `InitLocalStore()` method of `MainActivity.java`, make a new `HashMap<String, ColumnDataType>` and put a new entry in it for every column of the new table, and to the image of the table that is already defined there.  
Also add a line `localStore.defineTable("tableName", hashMapName);`

The added code looks approximately like this:

```
Map<String, ColumnDataType> newTableDefinition = new HashMap<String,
ColumnDataType>();
tableDefinition.put("id", ColumnDataType.String);
tableDefinition.put("columnName", ColumnDataType.String);
// one line for every column of the table
localStore.defineTable("newTable", newTableDefinition);
```

- Add a new class that represents a row in the table. Its implementation should be similar to that of `MowerDataItem.java`, so a field declaration, a setter and a getter for every column, plus a constructor and some general functions if needed.
- Add a field in `MainActivity.java` that represents the table and contains instances of the newly defined class. So for example :  
`private MobileServiceSyncTable<NewTableItem> mNewTable;`
- In the `OnCreate()` method of `MainActivity`, add a line that initializes the field:  
`mNewTable= mDbClient.getSyncTable("NewTableItem", NewTableItem.class);`
- In the `sync()` method of `MainActivity`, add this line in the try block:  
`mNewTable.pull(null).get();`
- In `MainActivity`, you may need to change the implementation of the methods `addItem()`, `parseData()`, `addItemInTable` and `refreshItemsFromMobileServiceTableSyncTable()`. However, the precise implementation depends on what the new table represents.

The new table must also be added in Azure. To do this, go to the query editor as explained above and write the appropriate SQL query. In Power BI, the new table must be added as well. We refer to the part on Power BI for information on how to do this.

## Hardware

### Joystick

#### Connections

The joystick board has a screw terminal with 8 connections that are labelled by colour:

Red (R): VCC

Black (Bk): GND

Green (G): Z-Axis

Yellow (Y): Y-Axis

Blue (B): X-Axis

White (W): Digital Common

Purple (P): Button 1

Orange (O): Button 2

#### Voltage Configuration

To change the desired output voltage of the joystick, the values saved in the file 'voltage-config.h' need to be changed as follows:

For output voltages between 0-5V:

AMPLIFY : 0

MAX : maximum voltage desired for the axis outputs

MIN: minimum voltage desired for the axis outputs

MAXBUTTON : maximum voltage desired for the button outputs

MINBUTTON : minimum voltage desired for the button outputs

For the output voltage range 6-9V:

AMPLIFY : 1

\*other values are automatically set

## Connecting Master and Slave Arduino

The Master and Slave are connected by only two wires for SCL and SDA. Each one needs to be connected to the corresponding connection on the other board.

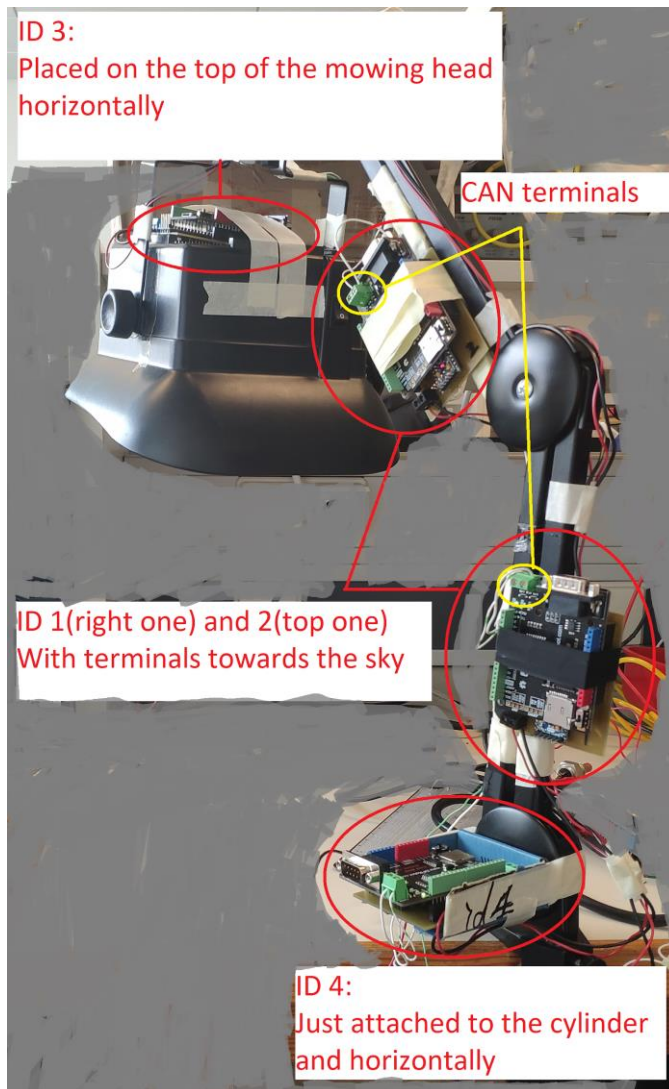
## Position Sensors

### Setup

To install the sensors, the user needs to follow the following instructions:

1. Boards with ID 1-3 should be launched in sequence from the segment closes to the cabinet to the head.
2. Boards with ID 1 and 2 should be placed in a way that the CAN terminals towards the sky.
3. Board with ID 3 should be placed horizontally on the mowing head.
4. Board with ID 4 should be placed horizontally on the rotary cylinder.

An example setup is shown in the picture below.



### Calibration

Due to different placement, there might be some error in the initial position. If so, the user can calibrate this error by testing and changing the offset value in the Arduino code, as shown below

```
#define MODE 0 //mode=0 kalman
#define DEGREE_DIFFERENCE 10
#define CAN_ID 0x01
#define OFFSET -3.3
```

To do so, the user needs first move the arms into a proper angle (eg. horizontal should read 0 and vertical should read 90), so that by comparing it with the angle from the Arduino, the users can adjust the offset.

### Ventilator

There are four different connections on the ventilator control unit, to let shields and Arduino work correctly, please refer to the following figures and make sure there is nothing wrong before applying power to it, otherwise, you may break the shield.

### Motor Shield Connections

The only white shield that underneath the PCB is the motor shield. *Figure A* is the top view of the shield. On the top, there are 6 connections for controlling two motors. Users need to connect an external power supply to VIN and GND, the maximum nominal voltage is 18V. Close to the power supply, on the left, are two ventilator connections: M1A, M1B.

### CAN Bus Connections

The black shield that directly connects to the Arduino board is the CAN Bus shield. *Figure B* is the right side view of the whole control unit, two terminals are left for the CAN\_High and CAN\_Low. For the same placement as *Figure B*, the left terminal is CAN\_High and the right terminal is CAN\_Low.

### Temperature Sensor and LED indicator

The PCB on the top of this unit is for the temperature sensor and an LED indicates whether the board is working. In *Figure C*, users can find the connection of this PCB. Plugging two connections of the temperature sensor into the female headers, the LED indicator would be on when the board is under correct working states, otherwise, would be off when in the setup state and would blink when the motor fails. In the last situation, the user needs to manually restart the whole program.

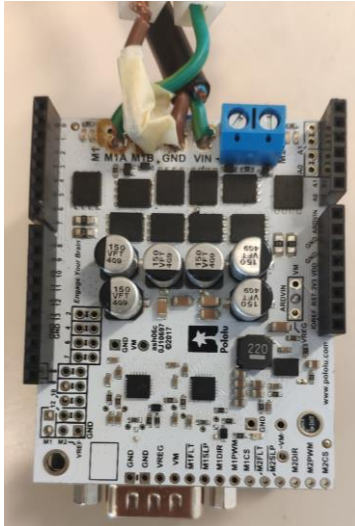


Figure A. The Motor Shield-top view

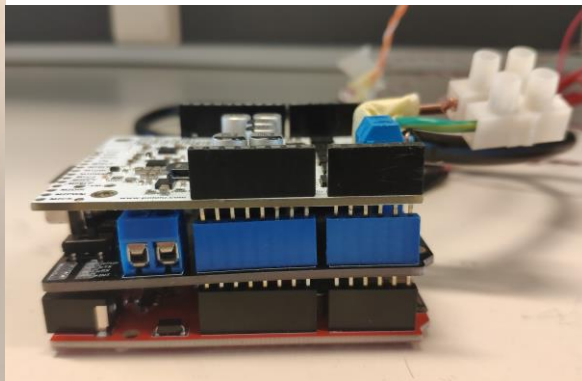


Figure B. The CAN Bus-right side view

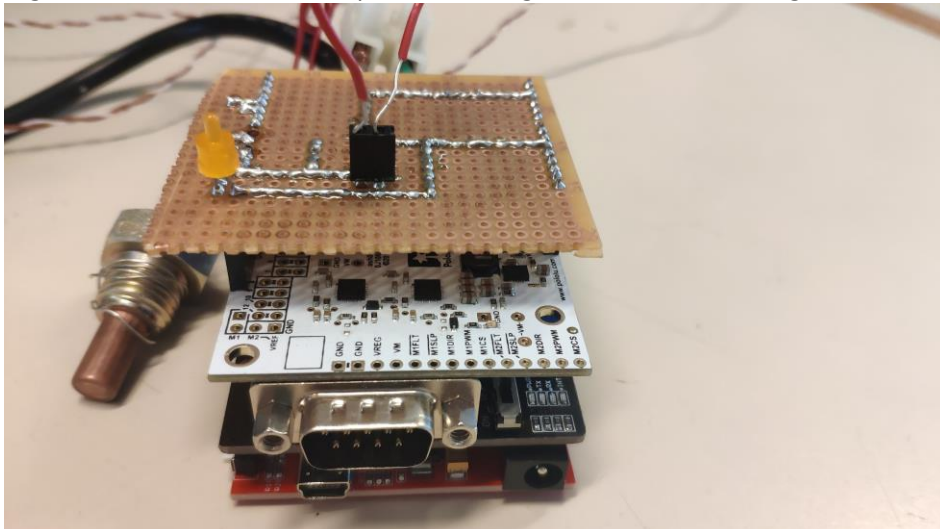


Figure C. The PCB and LED Indicator

## CAN Bus

All boards that require a CAN bus connection have a CAN shield with a screw terminal for the CANH and CANL lines. The CAN baudrate is set to 500Kbps but can be changed in `canSetup` function in the position sensor code. A shielded data-transfer line with twisted wires for the differential signals is the best choice since large motors and ventilators are present on the mower. These can generate significant electromagnetic fields and voltage disturbances in the bus. The terminating resistance on the CAN bus shields is 120 ohm. For more help on choosing CAN cables, see the provided PDF document: <http://canlab.cz/pages/download/CAN-Wiring.pdf>

## Power

- The position sensor boards can be powered by a supply between 5-12V.
- The amplifier board needs to be powered by a 12V supply.
- The Master (Mega) and Slave (Uno) Arduinos and the Ventilator board need an Arduino can be powered by a supply between 6-20V.
- The Motor shield needs to be powered by a source between 7.5V and 18V, the suggested voltage is 12V.

# User Manual

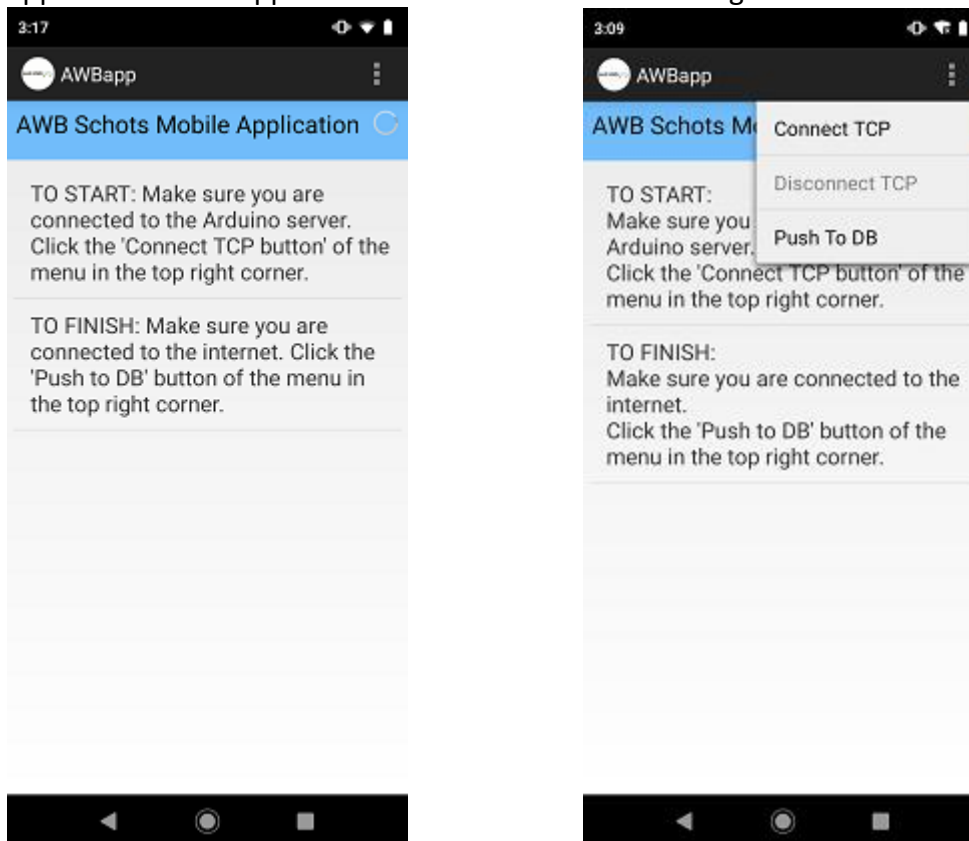
## Android Application

### To start

To connect with the Arduino that logs all the data, you should first make sure you are connected to the ARDUINO\_SERVER wifi network. To do this follow these steps (on Motorola g7):

- Go to settings on the Android phone
- Click on 'network & internet' > Wi-Fi
- If the Wi-Fi is turned off, turn it on by pushing the slider on the top right corner of the screen. After that, the slider should turn to the right and the ribbon becomes green.
- The ARDUINO\_SERVER network should now be visible. Click on it.
- If the phone has connected to the network before, it should connect automatically, otherwise, you have to enter a password. The password is 123456789. It can be changed in the Arduino code.

When you are connected to ARDUINO\_SERVER, open the app by clicking the icon on the phone. The app is called AWBapp and the icon is the AWB Schots logo. The screen below (on the left) pops up.



Then, click on the three dots in the top right corner, the menu drops down like in the right picture. The only difference is that in the latest version, the Disconnect TCP button has been omitted. Click the Connect TCP button from the menu. The text 'TCP connection started' appears on the screen. You are now ready to go.

## To finish

To push all locally stored data to the cloud database, you should first make sure you have an internet connection. This can either be a W-Fi connection with internet access (ARDUINO\_SERVER does not have internet access) or via 4G.

Then, again click the three dots on the top right corner of the screen, followed by the Push To DB button. Now the text 'Disconnected from TCP server' appears on the screen. This simply means that you are no longer connected to the Arduino.

After a while, the spinning icon just below the three dots should stop spinning and start disappearing and reappearing quickly. This means the Android is pushing its locally stored data to the database. When all data is pushed, the icon completely disappears. If the icon does not stop turning, try again after a minute or so. Sometimes it can take a while for the phone to establish a connection to Azure.

A short summary of the steps to follow is also available on the User Interface.

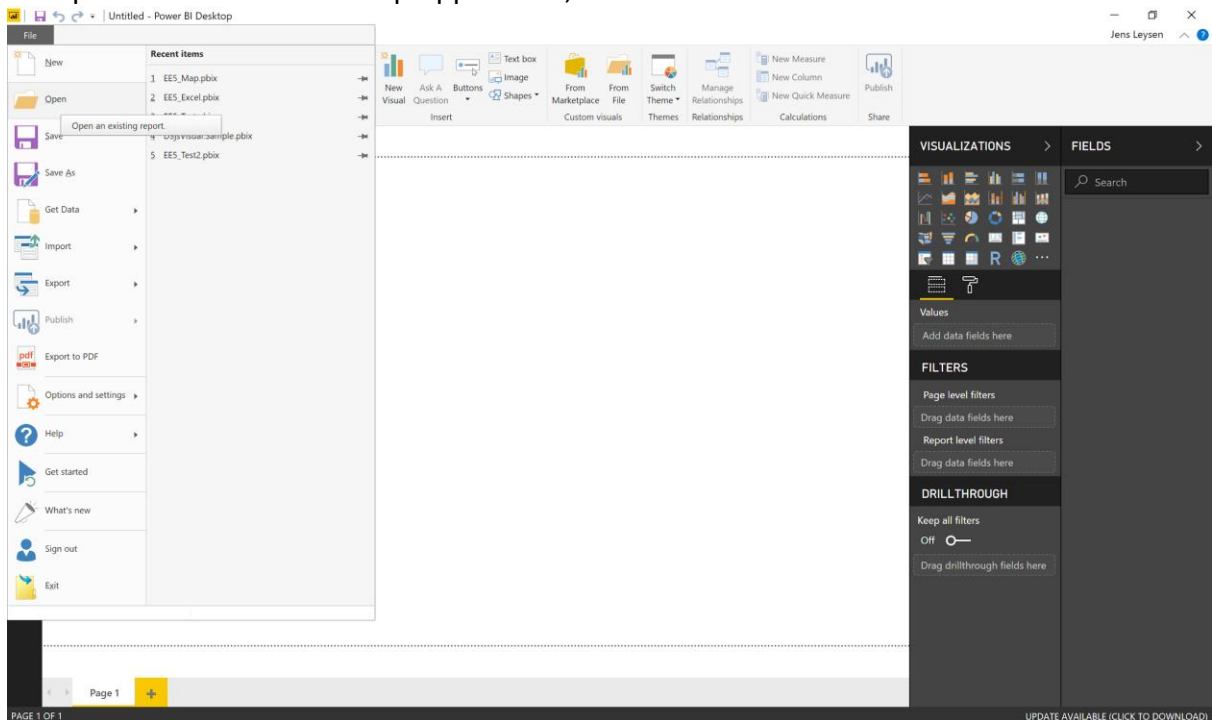
## Power BI

- Please download the Power BI report made by our team, also see the installation manual above.



Power BI desktop application icon

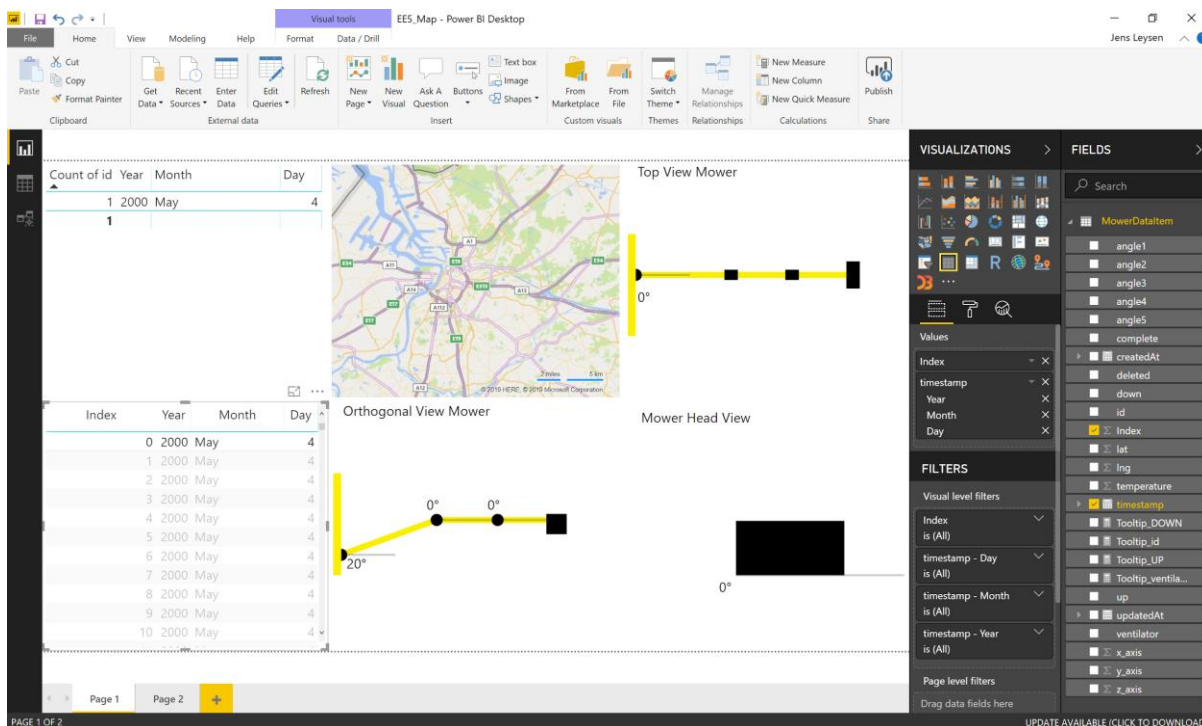
- Open the Power BI Desktop Application, icon shown above.



Power BI desktop application once opened



- Once Power BI has opened, navigate to the File Tab and open the report we provided.



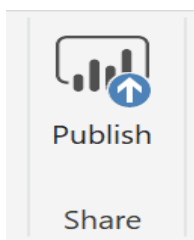
Power BI desktop application our report

- The Opened file should be similar to what is captured above. As noted before, the Power BI desktop application can be used to change anything related to the visualization. For more information on how to work with Power BI desktop, please visit the countless tutorials provided by Microsoft.

## Using Our Report

The table at the top left of the visualization shows the number of data points collected by day. The table underneath shows the distinct data points on a certain day. Clicking a distinct data point will show the angles for the segments of the arm in the 2 visualizations on the right, if you zoom in on the map, you will find the data point you selected indicated in blue. When hovering over a data point, additional columns are displayed such as the input of the joystick, whether the ventilator was on etc. Clicking on a day in the table in at the top left will show the route the Mower did on that day. Feel free to play around with the report, everything should feel familiar.

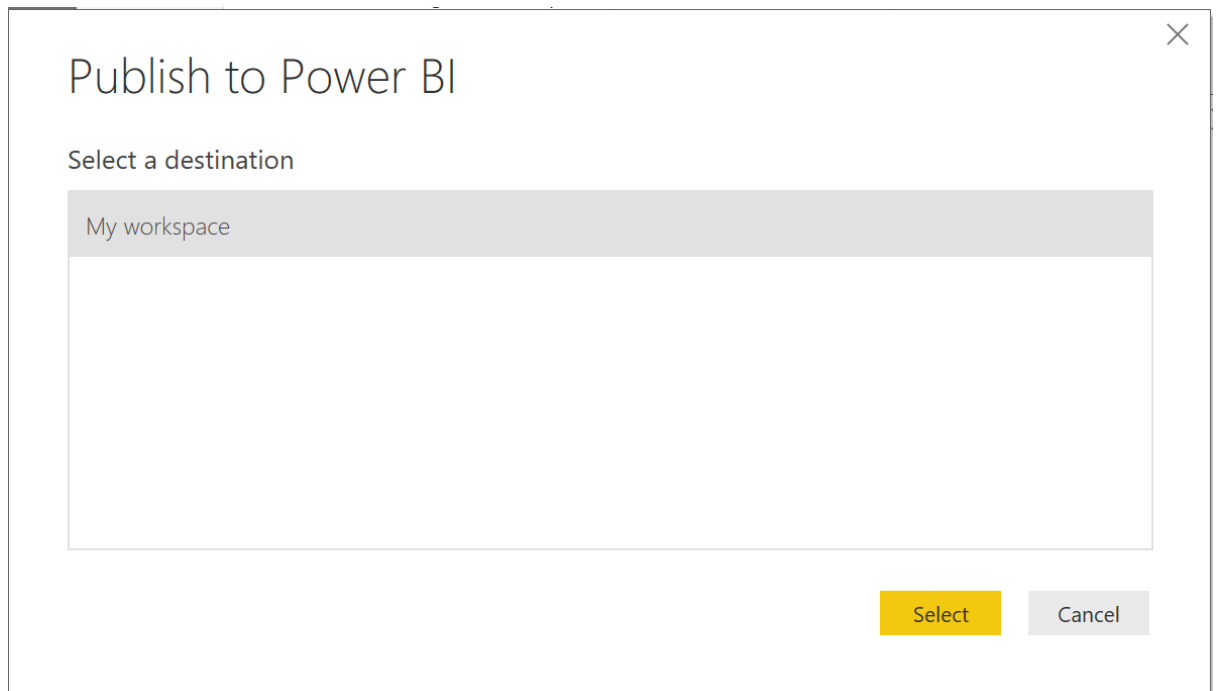
- If you would would like to share the report with other people in your company, or publish the report to your website. You have to “Publish” it. Publishing your report does not mean everyone will be able to see it! You still have to share it in order for other people to be able to see your visualisation.



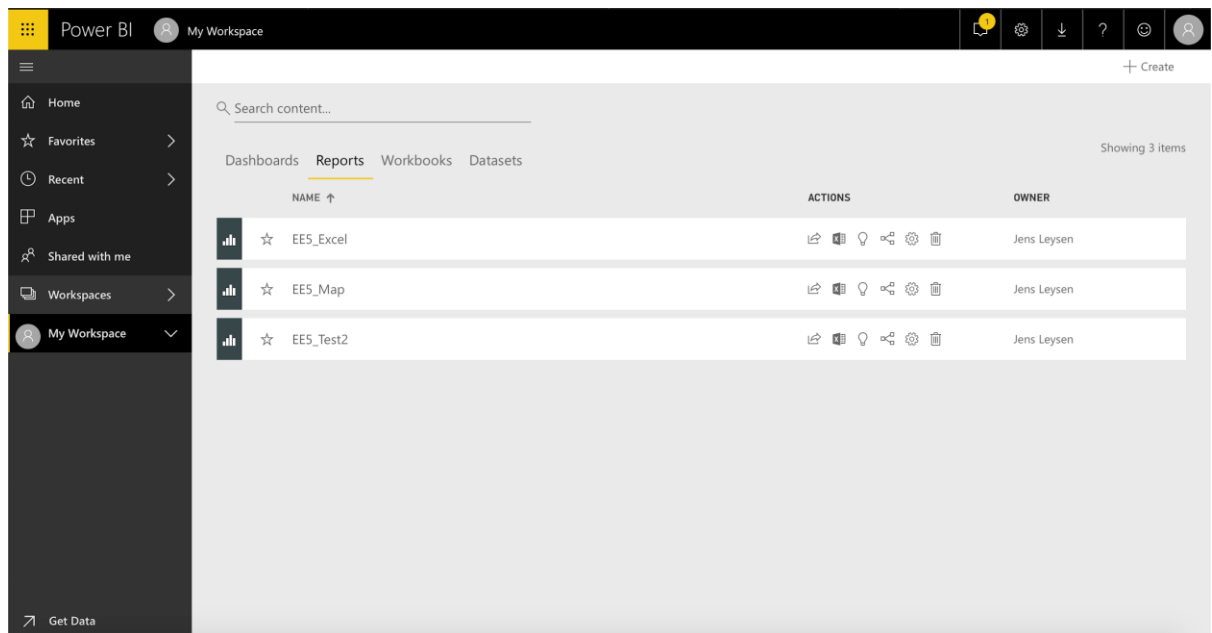
Publish option in Power BI



- You will be asked to sign in, please sign in.




















- When the Pop-Up above shows up, click select. Your Report will be published to your Power BI Workspace.
- Next, head over to the Power BI sign in page: <https://powerbi.microsoft.com/en-us/>
- Please Sign In again.



- Under Reports, you should be able to find the report you published. Click on it, the visualization will be opened in your browser.
- You can now decide to publish it to a webpage of your company, share it with other people. More information can be found on the following page: <https://docs.microsoft.com/en-us/power-bi/guided-learning/publishingandsharing>

- You can schedule a refresh of the data in the SQL database by going to the Datasets tab, to adapt the schedule please click on the 3rd icon under Actions. You can also see the next refresh and the time of the last refresh.

Dashboards Reports Workbooks <u>Datasets</u> <span>Showing 4 items</span>						
NAME ↑	ACTIONS	REFRESHED	NEXT REFRESH	API ACCESS		
EE5_Excel	    ...	5/10/2019, 11:56:43 AM	N/A	--		
EE5_Map	    ...	5/16/2019, 3:26:23 PM	N/A	--		
EE5_Test2	    ...	4/27/2019, 12:18:34 PM	N/A	--		
EE5_V2	    ... 	5/17/2019, 9:07:43 AM	5/17/2019, 9:30:00 AM	--		

- The schedule looks like the picture below.

## Settings for EE5\_V2

Next refresh: Fri May 17 2019 09:30:00 GMT+0200 (Central European Summer Time)

[Refresh history](#)

▶ Gateway connection

◀ Data source credentials

EE5\_Test-ee5.database.windows.net [Edit credentials](#)

▶ Parameters

◀ Scheduled refresh

Keep your data up to date

☒ On

Refresh frequency

Daily

Time zone

(UTC+01:00) Brussels, Copenhagen, Madrid, Paris

Time

9 30 AM

9 00 PM

[Add another time](#)

☒ Send refresh failure notification emails to me

Apply

Discard

## Ventilator Control Configuration

In this section, the way to control the ventilator is discussed. By changing the following parameters in the Arduino Program, users can adapt the codes to different scenarios.

### Temperature

```
#define threshold_high_temperature 90 // Start ventilator above this temperature
#define threshold_low_temperature 80 // Stop ventilator below this temperature
```

The ventilator will start work when the temperature is higher than **threshold\_high\_temperature** and will stop when the temperature is lower than **threshold\_low\_temperature**, users can set the value by changing the numbers, but be cautious, never set the **threshold\_low\_temperature** higher than the **threshold\_high\_temperature**, otherwise, the program would never start.

```
#define Temperature_difference 0.5 //Send temperature through CAN when there is a change > than this value
```

**Temperature\_difference** is used for collecting data on the cloud, Arduino would only send temperature data when a new reading has a difference larger than this value compared with the last reading.

### Time

```
#define clean_time 30000 //in milliseconds for ventilator: 30 seconds
#define cool_time 30000 //in milliseconds for ventilator: 30 seconds
```

**clean\_time** and **cool\_time** are the time the ventilator would run in the cleaning mode and cooling mode. This time is in milliseconds, so 30 seconds would be 30,000, as shown in the example sketch.

```
#define temperature_update_time 2000 //in milliseconds every X seconds read the temperature once
```

More advanced, users can define the time interval between two temperature readings, as shown in the example sketch, now the temperature value would be updated after every 2 seconds.

### Motor Speed

```
#define max_speed 200
#define min_speed -200
```

**max\_speed** and **min\_speed** determine the speed of the ventilator in cooling mode and cleaning mode respectively. The maximum speed user can set is 400 for max or -400 for min. Be aware that the higher the number, the less torque the motor would provide, in case that obstacles got stuck in the ventilator, a smaller number would rather be preferred.