



Design for AI-based greenhouse pest control

Author: Benjamin Aartsen

Student number: 4804139

Date: 20-09-2024

SUMMARY

This report details the development of a device that utilises image detection and machine learning to predict slug infestation in greenhouses. Increasing pesticide regulations have led to a rise in slug infestations, causing significant crop damage. Although alternatives to pesticides exist, they are expensive and labour-intensive to implement across large greenhouses. The goal of this project is to reduce labour by automating slug monitoring through image detection and machine learning. The focus is narrowed to detecting the Spanish Earthslug (*Lehmannia Valentiana*) in Cut Cymbidium, a type of orchid, as this market is particularly affected, and one slug species makes image detection more manageable.

Context research and experiments were done to understand the problem. Slugs are nocturnal creatures that emerge from the soil periodically for food. Their activity is mainly influenced by temperature and humidity, preferring moist environments at around 17°C.

Since greenhouse conditions vary by plant growth stages and change throughout the year, it is worthwhile to monitor activity on a longer time scale. Image detection and computer vision are already used in pest control and other applications within greenhouses. For efficient image detection, the looked after objects must be clearly identifiable and the images should preferably be of consistent quality.

To achieve this in a greenhouse a good suggestion would be a controlled environment with minimal to no leaf occlusion. In our case, another important factor is having low data images with a high contrast between the slug and their background. This can be achieved by adjusting lighting conditions, converting images to black and white, and tracking changes between images to detect slug movement. After these results, the direction was chosen to predict slug infestation based on environmental data using counted slugs through image detection as a reference.

Based on these insights, a device concept was developed using a Seeeduno Xiao ESP32S3 Sense microcontroller and Grove sensors. The system lures slugs with wheat bran, a proven bait, and uses a camera to capture images. Photos are stored on an SD card along with environmental measurements, such as temperature and humidity. Field tests and prototyping demonstrated that a 12 cm-high container with four 1 cm-diameter entryways was effective for attracting slugs and capturing quality images. To enhance contrast and prevent mould growth, a white mesh was placed between the bait and the slugs. For easy processing and low data images, the images are processed into binary images, where pixels are either black or white, and Hough Circle transform is used to crop the images around detected circles.

For slug population prediction, a regression algorithm was selected, with environmental variables serving as the input and slug counts as the output labels. Spatial interpolation can be used to estimate environmental values for the entire greenhouse based on measurements of the device therefore getting the data to make predictions for slug infestation.

The final version of the device, called Cephal, was tested in the field and used to build an image detection model. During field tests, Cephal successfully captured environmental data and collected over 485 images, although no slugs were detected, indicating the need for further research. Additionally, improvements such as a longer cable for the moisture sensor and a more accurate DHT sensor are required.

With Cephal, a dataset of 482 images was used for machine learning. Reducing the input size to 64x64 binary images, a CNN model was made with around 90% accuracy.

In conclusion, Cephal has the potential to enhance slug monitoring in greenhouses. With improvements in sensor performance and further optimisation of the machine learning model, Cephal could accurately predict slug infestation, helping greenhouse owners implement timely pest control strategies. Future work should focus on optimising the electronics and training and optimising the slug prediction model based on environmental values.



Image 1: Cymbidium flowers

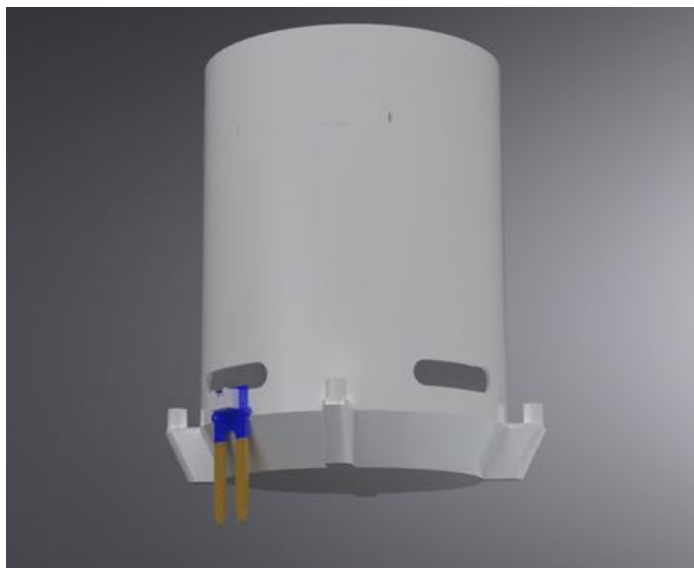


Image 2: Cephal

TABLE OF CONTENTS

ch.		page
1.	Introduction	7
2.	Context	10
	2.1 The Spanish Earthslug infestation	10
	2.2 The Cut-Cymbidium cultivation	12
	2.3 Machine learning in greenhouses	14
3.	Solution space exploration	17
	3.1 Data compression	17
	3.2 Adjusting colour values	20
	3.3 The first dataset	21
	3.4 Light test	22
4.	Requirements	24
5.	Design direction	25

ch.		page
6.	Physical design	28
	6.1 Building electronic prototype	28
	6.2 Testing the bait used	30
	6.3 Device shape experiment	31
7.1	Image detection system	35
	7.1 Corner detection	35
	7.2 Image processing	37
8.	Algorithm	39
9.	Update requirements	42
10.	Intermediary concept	43
11.	Showcase	46
12.	Final shape design	49
13.	Training image detection	54
14.	Field testing	57
15.	Conclusion	61
	15.1 Limitations and recommendations	62
16.	References	63
	Appendix	

1. INTRODUCTION

In the greenhouse industry, pest control presents a significant challenge. Due to increasing pesticide regulations the effectiveness of some pesticides is reduced due to limited allowed dosage. Alternatives either do not exist or have limitations in effectiveness, efficiency, and or costs (Han van Tilburg, personal communication, 19-03-2024). One pest that has become more prominent in crops due to these regulations is the slug. Due to a lack of suitable alternatives to pesticides slugs have been able to thrive in certain crops, causing a substantial amount of damage. One innovative solution that has recently been introduced to the market, that I have been involved in personally, is the use of slug traps (Aartsen et al., 2022). However, the maintenance and costs associated with these traps make it necessary to apply a selective application strategy to maximise their efficiency and cost-effectiveness.

Machine learning could potentially be an attractive option to prevent the use of manpower in monitoring damages to crops by applying selective slug control. The field of machine learning concerns the ability of computers to recognise patterns and learn from data.

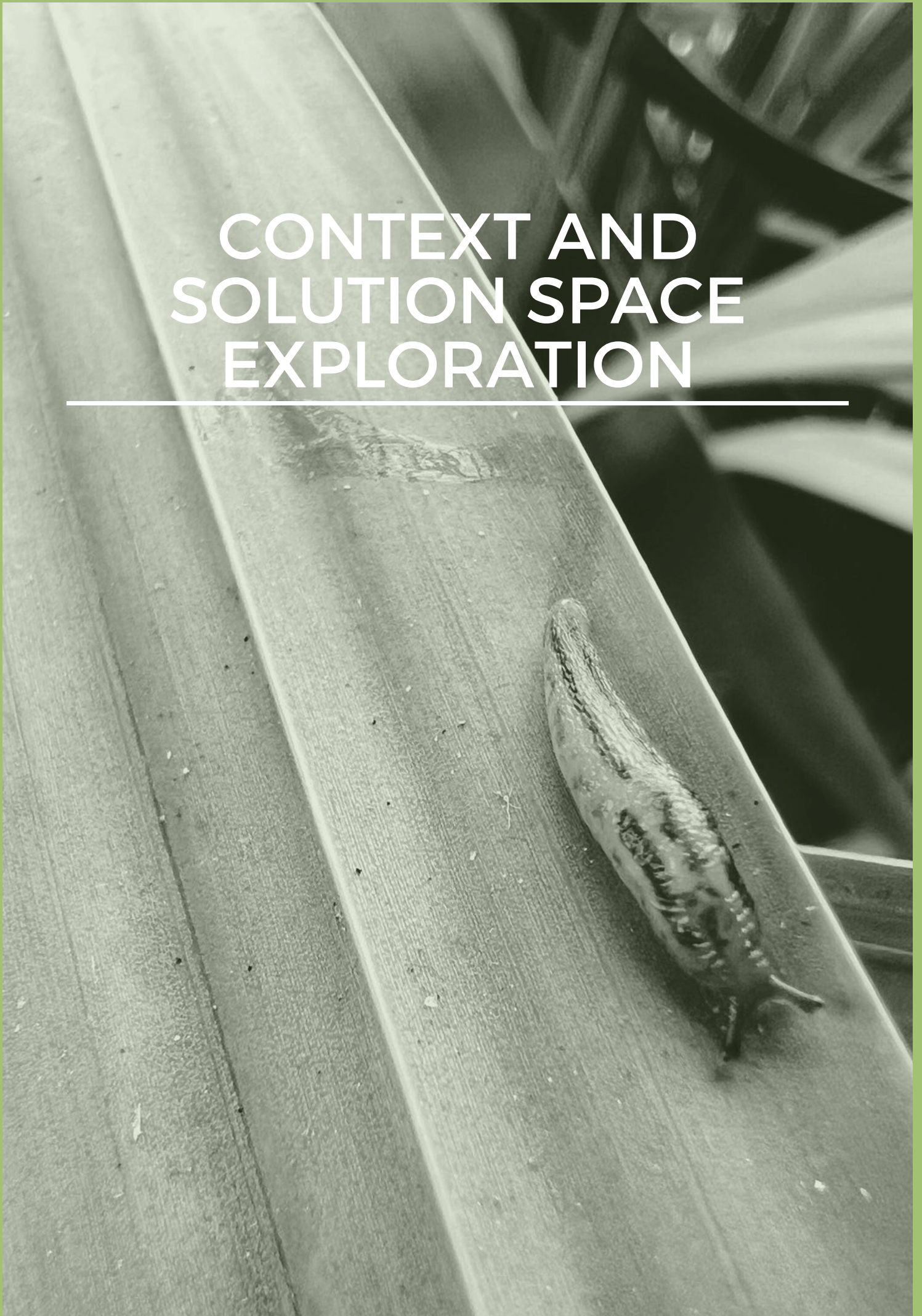
Machine learning models have already been successfully implemented in combating other pests such as Trips and Whitefly (Xia et al., 2015). With the combat of these pests, the models have learned to identify pests using images. This specific part of machine learning falls under the field of computer vision, which involves learning information from visual data. Within computer vision, the main part that this report will focus on is image detection, which involves a model being able to detect several objects within an image without the need for classification of the objects.

This report aims to develop a product that uses computer vision to identify and count slugs in a greenhouse setting. The focus is on creating a product capable of capturing a large number of images to train an optimal image detection model. This model will be designed to accurately predict slug infestations, providing greenhouse owners with valuable insights into where pest control measures are most needed. The ultimate goal is to offer a reliable tool for effectively managing slug populations and minimising crop damage in greenhouses.

The approach in this report can be put into 3 separate parts: First, the context of the slug infestation and machine learning within greenhouses will be explored. Simultaneously a series of small experiments will be done to explore the possibilities and limitations of image detection on slugs. The result of phase one will be a program of requirements and a design direction. In the next phase, another series of explorations and detailing will be done to create a more detailed concept. At the end of this phase, the primary working principle and main device shape should be ready. The final stage will detail this concept into a working prototype and detailed design. After this stage, the design will be detailed in technical drawings and validated within the context.

To properly develop a machine learning algorithm in the greenhouse setting, some restrictions will be made to keep the scope of this graduation manageable during the specific timeframe. The scope will be limited to monitoring the Spanish EarthSlug (*Lehmannia Valentiana*) within cut-Cymbidium greenhouses. The reason for this is that within the Netherlands the Cymbidium market is especially affected by the slug infestation. They are specifically bothered by the Spanish Earthslug which helps as the model only has to identify one species and colour of slug.

CONTEXT AND SOLUTION SPACE EXPLORATION



2. CONTEXT

2.1 The Spanish Earthslug infestation

The Netherlands is well known for its greenhouse industry. The municipality of Westland is responsible for a large part of the Dutch export. One lucrative trade is the Cymbidium (orchid) market, of which The Netherlands is the biggest exporter ("The Orchid Genome", 2021). However, this crop is jeopardized by the Spanish Earthslug (*Lehmanna Valentiana*). This small species of slug causes severe damage to the flowers by eating the soft parts of the plant (leaves and flowers), making them unsellable.

While in the past specific pesticides aimed at slugs were available (Holger Nennmann, 2021), due to regulations on the usage of these pesticides, combating slugs has become increasingly difficult (Ministerie van Algemene Zaken, 2024). So far no suitable alternatives are on the market.

Methods that were used by greenhouse owners involved luring the slugs with cucumber on a dish and catching them by hand as well as non-lethal slug detergents. No suitable alternatives to pesticides were there until a slug trap was developed which uses wheat bran to lure and kill the slug (Aartsen et al., 2022). The trap is reusable therefore eliminating waste. However, handling the traps is labour-intensive therefore applying them on every plant is not effective for greenhouse owners. The traps are now used either on their own in sections of the greenhouse or in combination with the detergent (Saponin). Traps can also be applied in the whole of the greenhouse when a sufficient workforce of high school students is present but most greenhouses in the sector do not have that (Han van Tilburg, personal communication, 19-03-2024).

To gain more insight into the slug monitoring behaviour of greenhouse owners, a small survey was done amongst Cymbidium Growers (results in Appendix A). The monitoring is done exclusively by the greenhouse owners themselves by looking at the damages and slime trails left on the plants. This is done weekly and is usually done to get a good perception of the damages done and to see if the applied pest control is working. While with good pest control, there is less need to do this as the effectiveness is known, with less certainty of good pest control this requires time and energy of the greenhouse owner that usually would be spent on other activities that would increase revenue.

Slugs are molluscs that come in various shapes and sizes. The Spanish Earthslug can go from a couple of millimetres to a maximum of 8 cm. They can live up to 6 years old and their eggs take 3 weeks to hatch (Wikipedia contributors, 2023). As slugs are hermaphrodite they can therefore quickly spread in greenhouses. In general, the defining factors that determine the habitability of an environment for slugs are moisture and temperature (Douglas & Tooker, 2012). They prefer very moist places as slugs lose moisture constantly (Stephenson, 1968) and for temperatures, they prefer cool places.

The Spanish Earthslug thrives most at 17 degrees Celsius. Besides this, they are mostly active during the night. In greenhouses, this means that during the day, slugs hide in the moist soil, and during the night they tend to go out to eat the soft parts of the plant. They don't do this every day as slugs usually go for food once every 5 days (Peter Zwinkels, personal communication, 08-04-2024).



Image 3: The Spanish Earthslug (above) and the Backie Slug trap (below) (Aartsen et al., 2022)

2.2 The Cut-Cymbidium cultivation

The Cut-Cymbidiums are a species of orchid where one stem bears multiple flowers. For harvesting, the stem gets cut from the plant so a new stem may grow. In the following chapter, the way of growing and cultivating these flowers is described.

An adult Cymbidium plant can stand in a greenhouse for multiple years and even decades. They start out their lifecycle as saplings in a small vial or pot in a substrate. After 6 to 8 months they go to a larger pot of 2l where they remain for about a year. Afterwards, they go to a 5l pot where in 2 years they go to a 7/10l pot, and after 2 more years a 10/12l pot. The exact moments of transferring and sizes of pots differ between breeds that bear flowers earlier in the year and breeds that bear flowers in the mid/ late year.

For light conditions in the greenhouse, it again depends per stage: The sapling stage takes 20-30k lux with additional artificial light of 3.5k lux needed in winter months. When older the plants need at least 35k lux but with light above 50k protection is needed to keep additional light out. The day cycle of a Cymbidium is at most 16 hours. Greenhouses for Cymbidium usually use natural light and no LED lighting. At night the greenhouse is dark with no light sources present.

The conditions that have to be maintained in the greenhouse depend on the phase in the lifecycle and the season. During the sapling stage of the plant 18 C at night and between 20-25 C during the day is recommended and 16-18 C at night and 18-20 C during the day in winter periods. After this, the temperature depends on the stage in the flowering cycle. The lowest range for a cycle is 10-13C and the highest is 20-22 C. The stages are usually for around 15/20 weeks.

Temperatuur	Per stadium	Tijdsduur in weken	
		Grootbloemig	Kleinbloemig
10 - 13 °C	Takaanleg	12 - 16	12 - 16
20 - 22 °C	Scheutgoei	14 - 19	10 - 14
13 - 20 °C	Takstrekking	16 - 27	12 - 23
13 - 19 °C	Bloei	4 - 8	4 - 8
10 - 14 °C	Combine tak- aanleg/strekking	22 - 27	18 - 23

Bloeiperiode	bloeitijd	jan	feb	mrt	apr	mei	jun	jul	aug	sep	okt	nov	dec
zeer vroeg	aug-okt												
vroeg	nov-dec												
middeelvroeg	jan-mrt												
laat	mrt-apr												
zeer laat	mei-jun												

Figure 1: Growth schedule of Cymbidium with recommended temperatures and number of weeks (Floricultura b.v., 2021)

A humidity between 50-80% should be upheld. Cymbidium do tend to be hardy so they can handle a lower relative humidity for a while but that should not be the norm.

As the different stages of growth require different circumstances, the greenhouses need to have different sections where those circumstances are maintained. Every section is equipped to go along with the different stages of growth. In the greenhouses, the plants are housed on trays or beds with an open bottom so the drain water can flow away freely, otherwise, mould can develop. For watering, a drip system is applied where either rainwater or water from reverse osmosis is given to the plants. With larger pots up to 6 drip systems can be applied per plant. The average man hours needed per year per hectare to work in the greenhouse is 800/900 hours. (Floricultura b.v., 2021)

In conclusion, especially temperature needs are quite variable in greenhouses that cultivate Cymbidium. For slug monitoring, this can indicate that monitoring on a larger timeframe is necessary to compare slug density with temperature fluctuations. Other factors that can influence slug density are the humidity, which is generally high but can fluctuate and soil moisture, which is directly influenced by the drip system. For a device that potentially uses a camera it is good to note the absence of any artificial light in the greenhouses.



Image 4: The Cut-Cymbidium flower (Aartsen et al., 2022)

2.3 Machine learning in greenhouses

In greenhouses, various applications for computer vision give us details about what does and does not work and what to take into account. One application for instance is in harvesting robots. Using computer vision, harvest ripe produce can be detected and then harvested. However, a challenging factor that inhibits commercial application is (partial) occlusion and clutter of the produce making detecting and isolating ripe fruits difficult. Ways to prevent this for instance has been to move the leaves and plants with a gripper (Kootstra et al., 2021).

A different research in the same use case developed a model to detect cherry fruits (Gai et al., 2021). They used an algorithm named YOLO-v4 (You Only Look Once) with a Densenet-based structure. With a dataset of 400 photos of which 50 were non-cherry and with augmenting the dataset (editing the photos to get more training data) they got an accurate model that was able to deal with occlusion. Another research found that changing the shape of the boundary box from a rectangle to a circle helped increase the accuracy in detecting tomatoes (Liu et al., 2020).

Another use of computer vision in farming is selecting fruit based on surface defects and blemishes. In this application, there are various possibilities for cameras that have their pros and cons. For instance, a widely used instance is colour cameras however blemishes outside of the visible light spectrum can not be spotted. Monochrome cameras using non-linear light alleviate this issue, but the photos produced by these cameras tend to have darker edges which can sometimes result in false diagnostics. Lighting setups can help this but it does give an increase in costs. Also removing the edges may work but this is at the risk of losing possible blemishes and therefore also cause false diagnostics. A 3D-based solution is structured light, where parallel lines get projected on a surface and the curvature of the lines is what gets measured (Li et al., 2014).

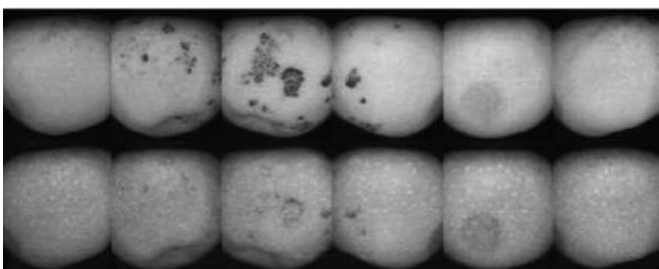


Image 5: Monochromatic camera used in detecting blemishes in fruits (Li et al., 2014)

When it comes to pest control, computer vision is also used already. For instance, by capturing flying animals with sticky traps and then counting them with computer vision (Xia et al., 2015). In this study from 2015, they used the watershed algorithm to build a model that was able to distinguish Aphids, Thrips and Whitefly. They found that the YCrBr colour space was the best way to distinguish the animals from the yellow background of the sticky trap. Also ironically enough a low resolution of the input image resulted in higher accuracy of the model as high-resolution images provided additional noise that caused inaccurate predictions of the model.

For slugs specifically research (Kozłowski et al., 2016) has been done that was able to track their movements in a controlled lab environment in 3 stages: Movement detection, object detection and movement tracking. The movement detection was done by making a grid of the images and taking the average pixel value in grayscale per square. This was compared frame by frame and differences in values were taken as movement. Then after a classifier identified the slugs using image detection, a so-called CamShift was used to track the movements. Here the centre of an object gets identified and then the results of the next frames get compared to construct a trajectory.

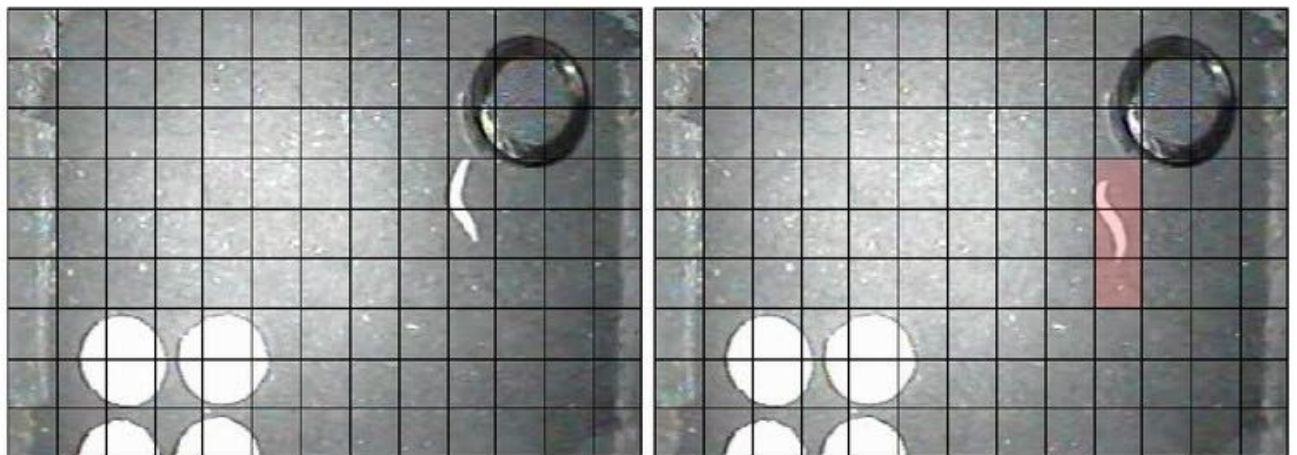


Image 6: Motion detection phase using a grid (Kozłowski et al., 2016)

In conclusion, computer vision has a variety of uses in greenhouses. What is important to take into consideration is the circumstances for your specific use case. In a controlled environment, such as with the glue strips to count flying pests, it is easier to deal with factors such as occlusion and methods such as finding the mean differences in pixel values between frames will also become more applicable.

If computer vision is necessary to use in the greenhouse itself outside of a controlled environment, then make sure factors such as occlusion become less of a problem by either influencing the environment through grippers or by using an optimal algorithm to build your model. Another thing that is important to note is what camera/ lighting setup is to be used, some will reduce computing power but might come at an increased cost.

3. SOLUTION SPACE EXPLORATION

To reiterate: the goal for the assignment is to make a device that is able to provide a valid dataset that serves as training for a model that can predict slug infestation in a greenhouse. The chosen field for this is computer vision, where a model can detect the contents of an input image.

In the context of greenhouses, computer vision will bring some complications as mentioned earlier, such as occlusion of leaves and the absence of light sources at night when the slugs are active as these factors prevent you from taking a good photo for your dataset. As well, you want to provide data to an algorithm that is not too big and can easily be interpreted (Supervised Learning, n.d.). To explore the solution space, a variety of small experiments were done.

3.1 Data compression

In order to save computation time and space, the photos will need to be properly compressed while still maintaining enough details for a model to recognise the slugs. To do so a proper proximity to the slug must be maintained to get them on the photo big enough to not be reduced to a couple of pixels. In practice, this is hard to do if a slug must be located within a plant.

A possible method that was designed was to do object detection in a larger image that contains one or more slugs. Based on the findings, cutouts can be taken from the image where the slugs are supposedly located. These can then be compressed and processed. Originally objects that belonged to a certain colour with a minimal area of 2000 pixels were selected. On simple images with high contrast, this worked very well, but with more complex objects such as slugs, colour was not a sufficient measure to make the distinction as slugs were frequently missed in photos where other objects were misidentified. (Code in Appendix B.) Therefore instead, a machine learning-based model is a better use for this purpose.



Image 7: The result of the blue detection algorithm

Another way to compress the data is to take a sequence of images and deduct all of the pixels that are different from each other. This way you will only register any moving objects. Using this, if the camera is in a stable environment the slugs can be registered when moving, using the difference in the images as an input for the model. Using the test setup for the light test, recordings of slugs moving have been made on a 1x1 cm grid

. Two screenshots were made from that video and used as input for a program. The resulting image was compressed to a 224x224 pixel image and thresholded to black-and-white values for even easier processing. The result could be used for a model (code in Appendix C). One downside to this method is that it will not be easy to track different slugs this way.



Image 8: Movement data of the slug and resulting difference image

3.2 Adjusting colour values

A way to make slugs more recognisable in a photo of a plant is by manipulating the colour values of an image. This was done by manipulating the values in the application Procreate of images shot in a greenhouse environment. It was during the daytime with photos of slugs placed on leaves. Changing the hue towards magenta on the spectrum made the slugs green on a blue leaf which could improve recognizability.

The downside to this method is that you are still working with coloured images, therefore being larger in data than binary images (images of solely black and white pixels). This method can therefore be useful if coloured images provide a more beneficial tradeoff compared to binary images. (Images are in Appendix D.)



Image 9: Slug image with altered magenta values

3.3 The first dataset

For the first iteration of a machine learning model based on image detection, a model from the internet was used as a basis to train a new model. This model is based on a convolutional neural network (CNN) and has been trained on a dataset of stock photos of both slugs and other pests frequent on farms such as beetles and caterpillars.

In order to enrich the dataset, photos of the Spanish Earthslug have been taken in varied contexts, shapes and sizes. For this, the slug was photographed in the greenhouse with varying visibility, the focus of the image and the amount of slugs in a photo. As well, photos without slugs have also been taken. Besides this, photos were taken in a photo studio and outside. To get more data, the dataset including the original slug dataset was then subjected to image augmentation, where the brightness of the photo was heightened and lowered and the images were also rotated at various angles. The model was meant to be retrained with this dataset.

Due to a change in the direction mentioned later on in the report, this model alongside the dataset was not necessary anymore. However, the direction of image augmentation to get a larger dataset is still very useful for later use. (Model in Appendix E and Dataset in Appendix F.)

3.4 Light test

The intention is to process slugs using a computer vision algorithm. As slugs are most active at night, when there is no light source in the greenhouse (Han van Tilburg, personal communication, 27-03-2024) a solution must be found to get the slugs photographed. One option is to use a light source on the visible spectrum to provide light in a traditional manner. In order to see what different light sources do with the visibility of slugs on camera a specific test setup was built (Appendix G). This setup is meant to provide a clean and stable photo of the slug while it is exposed to different wavelengths.

The lights used were: Red, yellow, blue, green and white configurations of a 12V LED spot with a GU5.3 fit. The photos were taken with an iPhone 13 pro. Based on the results, white light captures the details of the slugs best followed by the yellow light. The other light sources tend to give the slugs a more black appearance. Higher details might give more for an algorithm to properly identify a slug in comparison to other elements in a photo, whereas with a black slug the processing of an image might go more flawlessly if there is enough contrast between the slug and the background. Therefore a tradeoff can be made between more detail and processing. (Photos in Appendix H.)

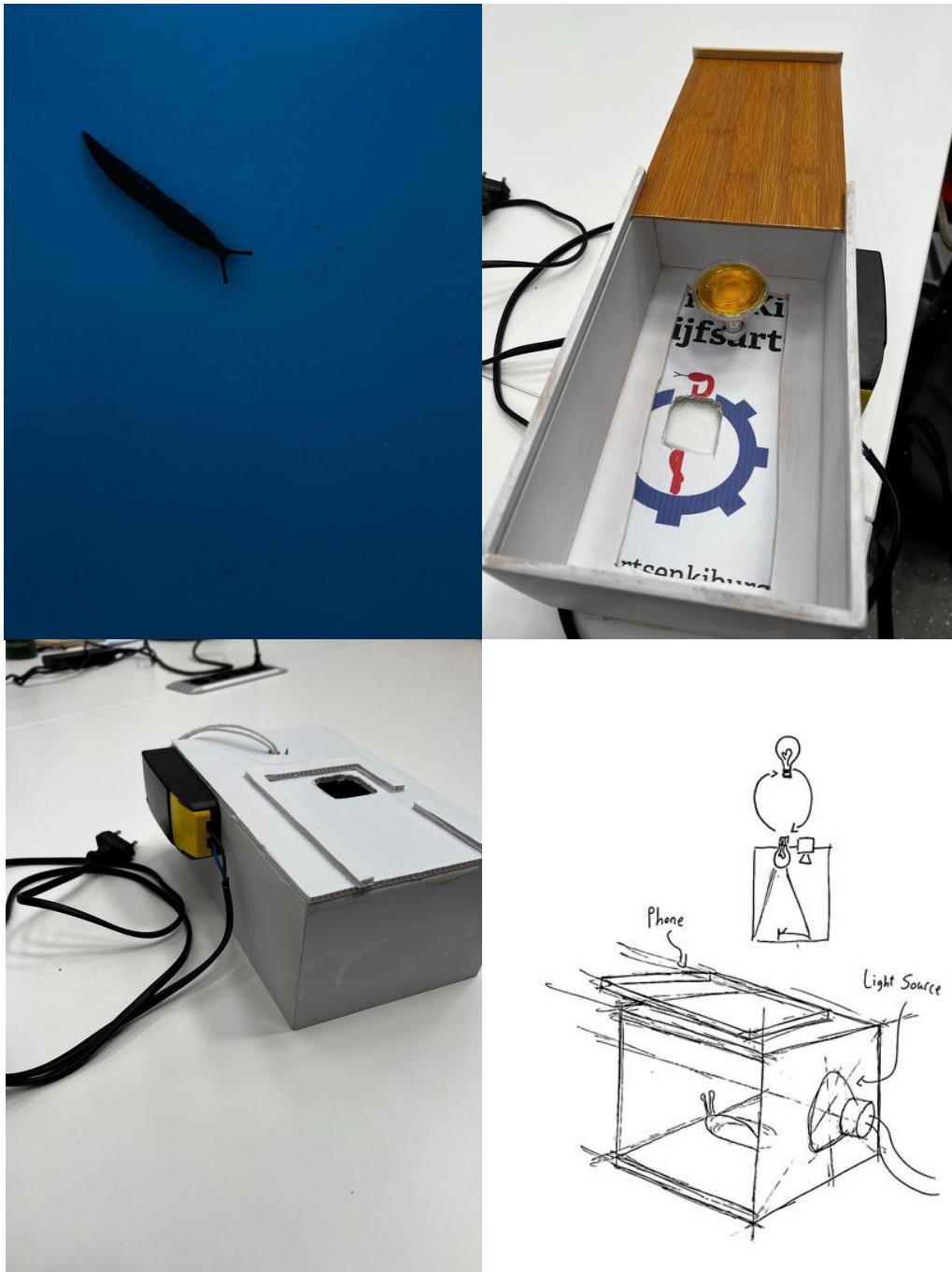


Image 10: Schematics for the light test setup and test setup + result

4. REQUIREMENTS

Based on the findings of the prototyping and context research, the following design requirements can be formed.

- The device should function with temperatures varying between 10-22C
- The device should function with a humidity between 50-80%
- The device needs to function with a drip watering system
- The device should be able to operate at night with no artificial lighting
- The device should be able to operate during the day with no artificial lighting with values of 20-50k lux
- A slug should be able to be detected regardless of occlusion by plants
- The data given to the ML model should be as low in storage space as possible
- The device should capture trends over a larger time scale (years)
- The device should not disrupt greenhouse operations

5. DESIGN DIRECTION

Looking at the aforementioned requirements and the habit of slugs to only come out once every couple of days for food and their ability to go for months without food (Peter Zwinkels, personal communication, 08-04-2024), a survey of just one night will not give an accurate estimation of the slug infestation within a greenhouse. Instead, a timeframe of 3 to 5 days monitoring one plant is more feasible. Also, the occlusion of leaves and the small size of a slug provide a serious obstacle for an image detection model to recognise slugs in photos taken from plants.

The difficulty of finding slugs mechanically for a machine learning model gives us the need to predict slug infestation another way. A way that can indicate slugs is by looking at the damage on a plant, such as eaten leaves and slime trails on the stems and leaves of the plant. However, this requires an automated system that checks every plant thoroughly, which can be resource-intensive. Two environmental factors that have been deemed most important in determining slug activity are moisture and temperature, as slugs prefer cool and moist places (Gerard van Wijk, personal communication, 08-04-2024). These can be used to make predictions for slug activity, as a couple of measurement points can be used to make predictions for the rest of the crops.

Taking the above findings into account, the environmental factors of temperature and moisture can be used to predict slug activity while physically counting the slugs in plants can be a way to confirm the predictions.

Therefore the new direction will be to make a device that measures the environmental factors and counts the slugs to then predict slug density based on those factors. In order to accurately count slugs without the hindrance described in the text above, one could lure slugs to a separate location to count them there.

As mentioned earlier greenhouse owners already do this using cucumber to lure slugs to a dish and then catch them from there (Peter Zwinkels, personal communication, 08-04-2024). The direction will therefore be to make a construction similar to the dish with some wheat bran, as this is a similarly effective slug bait with a longer expiration date (Peter Zwinkels, personal communication, 08-04-2024). Once lured, the slugs will be counted using Image Detection and a camera, as this is in line with the learning objectives of the assignment. For measuring the temperature a sensor can be used that also takes humidity into account, and for soil moisture, a sensor needs to be inserted in the soil (see Figure 3).

In the following stage of the project, this concept will be explored and elaborated on. In this process, 3 different trajectories can be distinguished: the design of the algorithm that predicts the slug density in the greenhouse, the design of the physical device that lures slugs to the photo-taking area and the image detection itself.

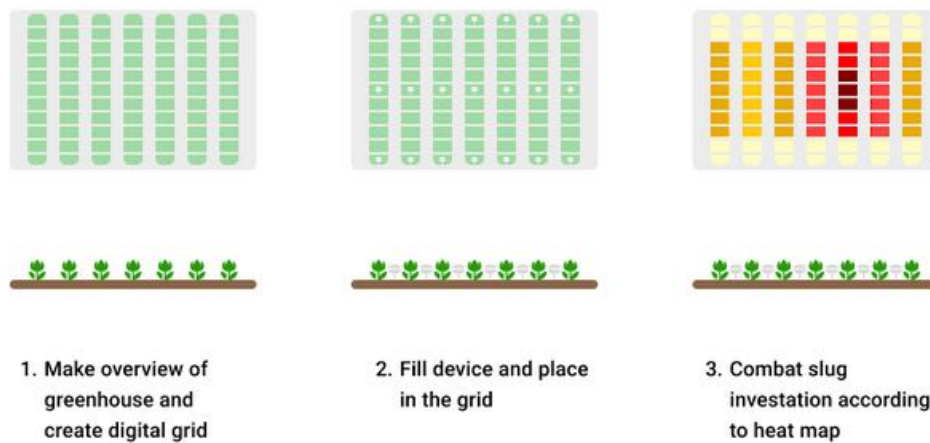


Figure 2: Envisioned user journey of the new direction

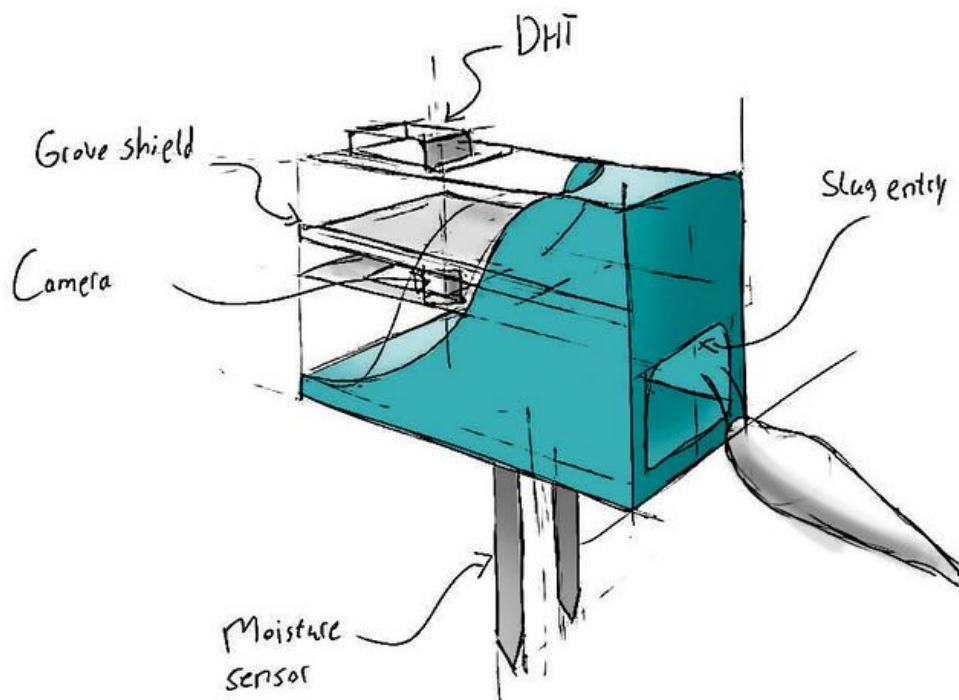
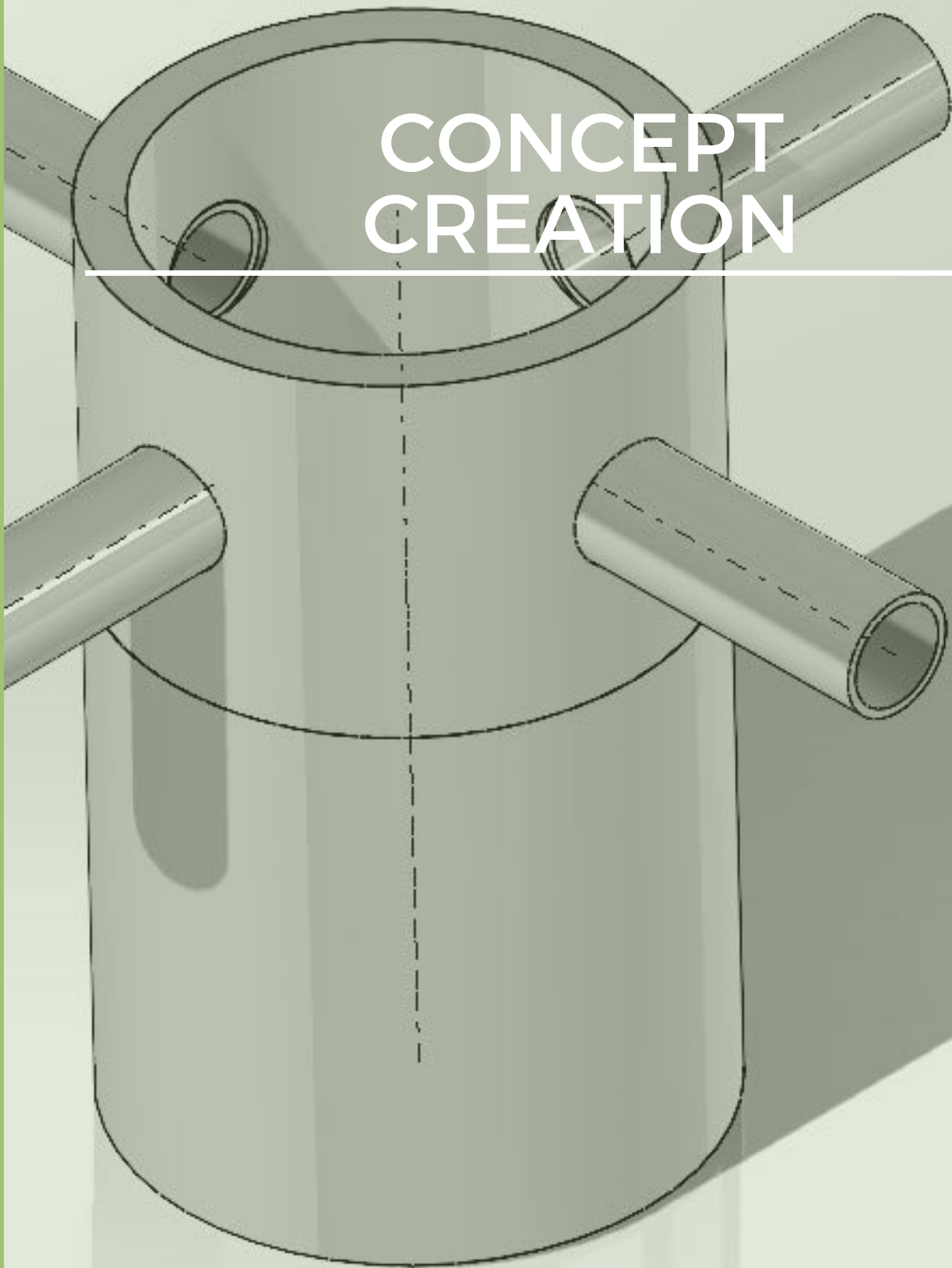


Figure 3: A quick sketch of the new direction with the specific hardware components

CONCEPT CREATION



6.

PHYSICAL DESIGN

6.1 Building electronic prototype

For the device, it is important to have an easily modifiable prototype that can be used for various tests. As well for machine learning to work, a dataset needs to be built with the device for training, testing and validating the performance of the model.

To start out with this, a device is being constructed that uses a camera to take periodic photos of the device for slug count and saves it to an SD card. To make sure the dataset does not get filled with unnecessary data and to save battery time the camera will take a photo once every 3 minutes as this is deemed a sufficient time frame based on the speed and feeding habits of slugs (Aartsen et al., 2022). Besides this, a soil moisture meter and a temperature and humidity sensor are also present. Simultaneously with the photos, these values get written to a CSV file that is to be used for machine learning later.

To start out, a Seeedstudio XIAO (nRF52840) was chosen with a Grove shield, DHT20, Grove vision AI module and a moisture sensor as a system. The reason for this is that a Grove module has plug-and-play sensors that are very easy to program and implement. Besides this, the XIAO meets the requirements for the performance of the device while simultaneously using significantly less power than for instance a Raspberry Pi. Due to compatibility, the XIAO was eventually replaced alongside the AI module with a Seeedstudio XIAO ESP32S3 sense for a better camera module and the possibility for wifi connectivity (Martin Verwaal, personal communication, 01-05-2024).

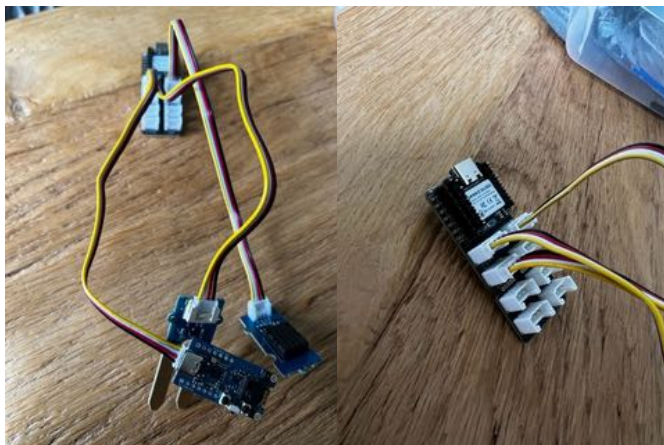


Image 11: Prototype hardware version 1 with the XIAO nRF52840, soil moisture sensor, DHT20 and the Grove Vision AI module

For power supply, a 3.7V battery with a capacity of 1200mh was chosen. With the system, it was able to last for 8 hours. Two separate tests have been conducted with the system, one with a photo taken every minute and one for every five minutes. The battery duration remained unchanged throughout both tests. To increase battery power either a sleeper module could be implemented or the device could be connected to a power bank. (Arduino code in Appendix I.)

As stated in the requirements, the device should be operable at night without any light source present. The prototype utilises a regular camera for simplicity and availability. For this purpose, a light source needs to be present to make sure the photos are properly lit. As to not disturb the usual activities of the slugs the light should only go on periodically when a photo gets taken. For this purpose, a WS2813 mini LED ring was chosen and implemented into the design.

To make the housing, first, a set of cable boxes was selected, as the waterproof design would be beneficial for the electronics in a greenhouse setting. After exploring however it became clear that the height of the boxes was too low to match with the focal length of the camera. After measuring it showed that about 2 times the height of the box would be necessary for a sufficient photo of the bottom side of the box. So as a replacement, a plastic food container was picked with a bottom area of 100 cm^2 and a height of 12 cm. This container provided sufficient footage but still mostly of the centre. Therefore the wheat bran must be laid out in the middle. The container was then modified to have entry holes of 1 cm in diameter, lined out in a way that improves airflow and luring (Aartsen et al., 2022).

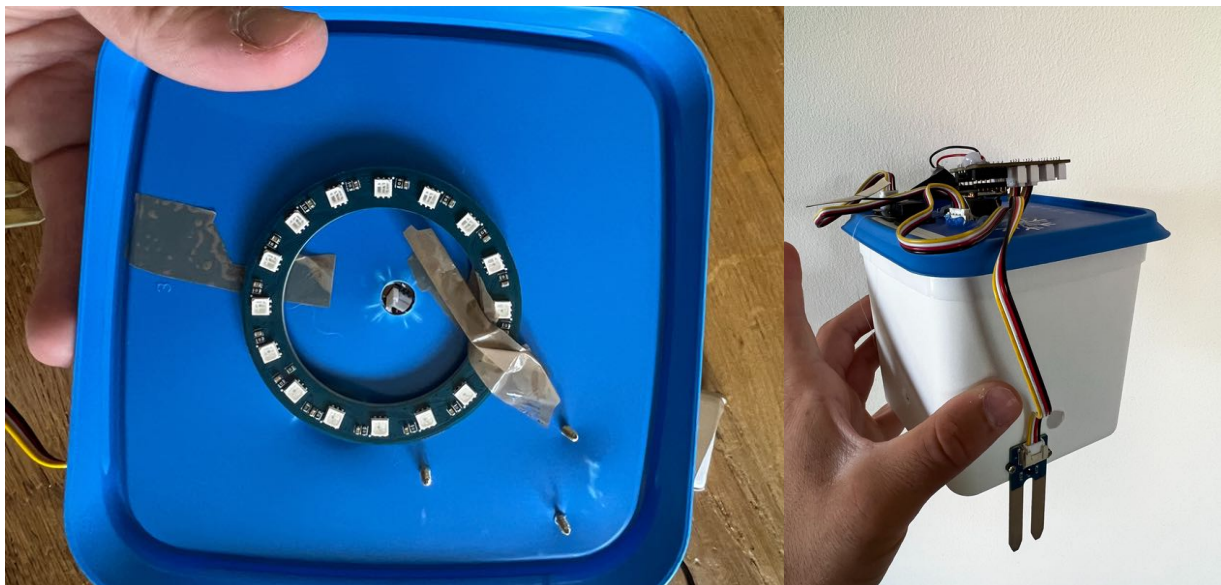


Image 12: Prototype hardware version 2, mounted on the container with the XIAO ESP32s3 Sense, WS2813 mini and the battery

6.2 Testing the bait used

The bait to be used is wheat bran, which is a long-lasting and effective bait to catch slugs (Aartsen et al., 2022). As a proof of concept, the container that was used for the electronic prototype was placed in an enclosure with slugs for a week with wheat bran inside. After a week no slugs were present in the container, however, due to slug slime in the container it was clear that slugs had been inside the container. But there was also significant mould growth in the container. This is most likely due to the moisture of the slugs interacting with the usually dry bran (Aartsen et al., 2022).

A way to solve this is by placing a mesh between the bait and the slugs. To verify that the effectiveness of the bait is not negatively impacted by the mesh the mesh was taken into account in the experiment that is described in the next paragraph.



Image 13: Mould buildup on the bran (left) and the mesh solution (right)

6.3 Device shape experiment

Method

The principle of using 4 holes of 1cm in diameter as an entryway for slugs into the device is based on previous research in the development of the slug trap (Aartsen et al., 2022). The slug trap uses 3 holes starting from 1cm in diameter and then decreasing in radius to make it hard for slugs to come back in from the way they came. However, as the application now is different to the intended purpose of the trap, the decreasing radius does not have to be implemented as slugs should not be trapped or killed by the device. To see how well the design can lure slugs, it was compared with different configurations in a greenhouse context.

In this test setup, a grand total of 14 lures among 7 different designs were deployed in slug-infested areas and consequently compared. The different designs that were compared were: A completely open container, a container with 4 holes drilled out of 1 cm diameter and a container with larger slots cut out. Each design has a version with accessible wheat bran and the bran behind a mesh to compare bran quality after a set amount of days. As a control, an open dish with cucumber is left out, as this is what was used to catch slugs by hand.

The lures were checked at two moments: within 24 hours after placement and 96 hours after placement. These moments were picked as slugs are usually active at night (Peter Zwinkels, personal communication, 08-04-2024).



Image 14: The 3 configurations of entryways (left) and a container with and without mesh (right)




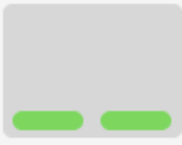
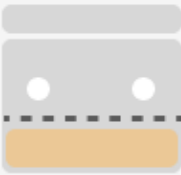
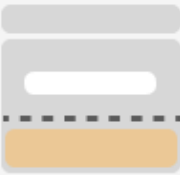

	4 Entries	Slots	Open	Cucumber
No mesh				
	1	2	3	4
Mesh				
	5	6	7	

Figure 4: Table with the different lure configurations

Results

After 24 hours 5 lures contained a low number of slugs, 4 did not contain any slugs but did have evidence of slug presence (slime trails slime in the wheat bran) and 5 remained presumably empty. After 96 hours half of the lures contained slugs and half were empty but did contain evidence of activity.

Additional findings were that the lures were hard to deploy in the foliage, as the foliage can be quite dense and the boxes were 10.7x 7.7x 3.5 cm which is a bit too large to properly place in the plant. (Table with slug count in Appendix J.)

Looking at the data in more detail some interesting facts could be found:

- Some lures contained fewer slugs at the second checkup than at the first checkup, indicating some slugs escaped.
- Some slugs also escaped when I made a second round past the lures directly after the first checkup.
- The mesh lures generally contained no slugs but did contain evidence of activity.
- The bran under the mesh was generally unaffected unless there was a way for the slugs to go under the mesh due to improper attachment of the mesh.
- There was no massive difference between types of entries when it came to slug presence in the lures.



Image 15: Test results showing a slug under the mesh (left), evidence of slug damage on the control (middle) and the effect of slugs on wheat bran (right)

Conclusion

From these results, it became clear that slugs are able to escape all of the designs, which is not necessarily bad as the main objective of the device is to monitor slugs. The mesh still attracts slugs while saving the bait but makes the slugs less interested in staying at the lure.

Also, a completely open design makes it again easy for slugs to escape and they tend to be more inclined to. Lastly, it is important not to disturb the lures, especially at night, as interference tends to scare the slugs off.

7. IMAGE DETECTION SYSTEM

As mentioned earlier, due to the scope of the assignment, the counting of the slugs within the device will be done using a camera and a machine learning model. For this purpose, it is important that the data (images) provided give a very clear indication of a slug being present while also being as low in data size as possible for faster computing. This tradeoff can be made by modifying the images digitally as well as by modifying the environment in which the photo is taken.

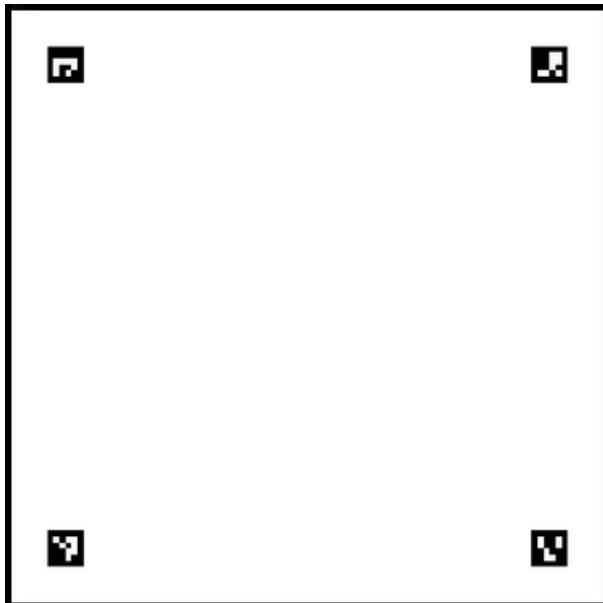


Figure 5: The bottom plate for the electronic prototype with Aruco markers implemented

7.1 Corner detection

With image detection, it is useful to provide not only small images but also consistent images to make identifying patterns easier. To further help this function it also helps to eliminate any unnecessary elements in the photos that could otherwise be taken into account during the model-making process. To achieve this, it is possible to crop the image to just show the area where the slugs are attracted to rather than the entire device.

To get photos which are always cropped in the same way, a program has been developed that detects Aruco markers and then crops the image around those markers. Aruco markers are small black-and-white symbols that can be detected and identified on camera (Wolf Song, personal communication, 07-06-2024). The reason for these markers is that they are predefined and labelled with a certain number. Therefore it is very easy to crop every image the same way as each marker is labelled and identifiable.

To apply this in the prototype, a small graphic was made that employs the markers labelled from 0-3 which could be laid on the bottom. After one trial, the markers were moved closer to the centre so that they could always be found by the camera. (Cropping code in Appendix K.)

One very real problem with these markers is the fact that slugs can cover these markers by sliding over them and causing occlusion. A possible solution is to keep the slugs separate from the markers by making the bait and the entry towards the bait a separate entity from the markers. Therefore making sure the slugs don't come close.

However, this also gives rise to new possibilities of image processing. By making this entity circular, one could also use Hough circle transform (Wolf Song, personal communication, 19-06-2024). This is an operation that detects circular shapes within the photo and is able to crop the image around that.

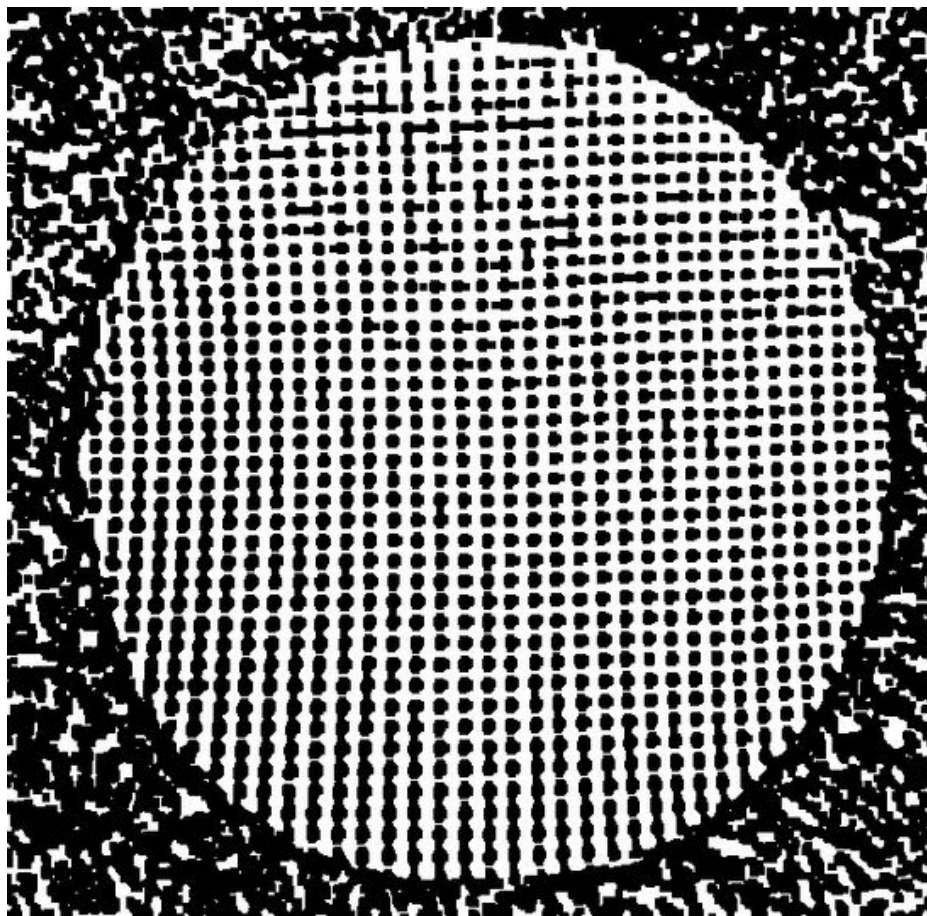


Figure 6: An example of Hough circle transform with an additional border of pixels to make sure the circle is fully captured in the image

7.2 Image processing

To count the slugs within the device, the intention is to use image detection on camera footage obtained from the device. In order to achieve this, a model needs to be trained on images from the device with and without slugs. To train this model, a simple dataset has been made using the container used for the prototype, an iPhone with flash function and the base template with the Aruco markers. In total, 3 versions were made, one with just the slugs and the wheat bran, one with the mesh (as per the redesign) and one with the mesh repainted to white to test if this would increase the contrast between the slugs and the surrounding environment (Appendix L).

As a result, the plain version had slugs that were harder to recognise, as they took a lot of the bran with them on their bodies, camouflaging themselves. The normal mesh made it easier as the slugs were elevated above the bran and the bran became less easy to photograph, creating a more controlled environment. The white mesh worked the best as the contrast of the slug against the background was by far the largest.

Next, a series of augmentations were made to make the images lower in data and even easier to process and recognise (Appendix L). As found in the earlier research, images with merely black and white pixel values (binary images) in a low resolution can be achieved and are desirable. This is more favourable than merely adjusting the colour values of an image as with a binary image the amount of data is significantly less. The following steps were used to increase the contrast between the slug and the background and consequently convert the image to a binary image:

- Convert the image to grayscale
- Apply histogram equalisation to enhance the contrast
- Apply Gaussian blur to smoothen the image
- With thresholding the image becomes binary (either black or white pixel value)

After these steps, again the white mesh provided the images with the highest contrast between the slugs and the background. This can be explained by the fact that the white mesh obscures the wheat bran better, therefore providing less clutter in the image.

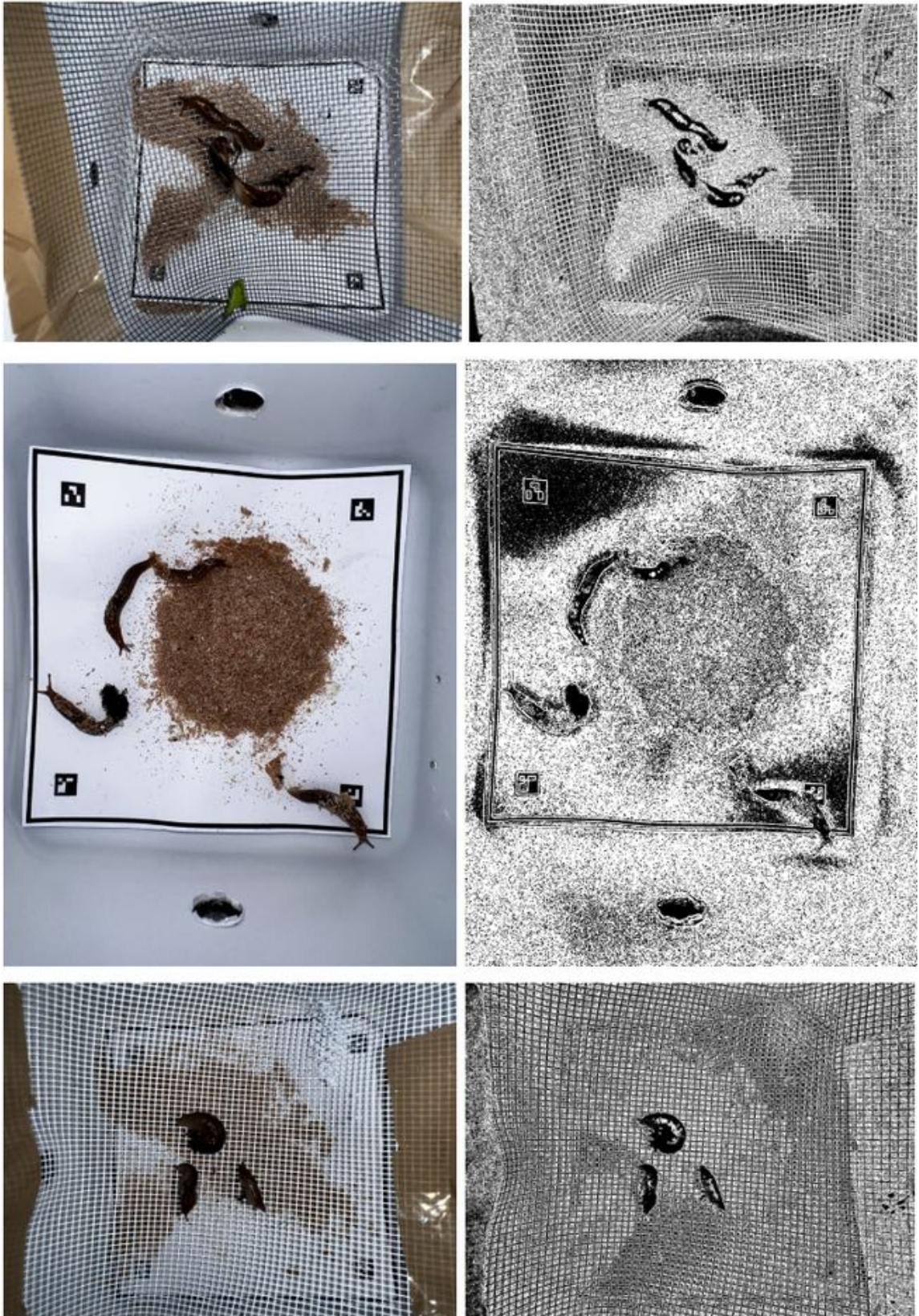


Image 16: Results of the dataset tests with augmentations

8. ALGORITHM

To train a model with the purpose of predicting slug density in a greenhouse, an algorithm is necessary to develop this model. When selecting an algorithm, there are various factors to take into consideration.

Two distinct ways of machine learning are supervised versus unsupervised learning. With supervised learning, the dataset used for training is labelled, meaning that the model assigns a label to new data based on what it has learned from the training dataset. Two important kinds of supervised learning are regression and classification. With regression, the labels are continuous values and are regularly used for the prediction of values. With classification, the labels can be categorised in several classes that get predicted based on the input data. Looking at unsupervised learning, the data does not contain labels, and it is up to the algorithm to find patterns in the data. This can for instance be done by making clusters of the data based on similarity in features. (Supervised Learning, n.d.)

In the use case of the device, the main purpose is to make a prediction of slug density in several locations in the greenhouse. For this purpose, a supervised regression algorithm makes the most sense. This however does mean that a lot of data will be needed and that training will take a lot of time. Looking at what to label the data with, a choice can be made to either compute a new value of slug probability based on the slugs counted in the device and the environmental values or the slug count itself can be used (with maybe a margin added to account for slugs not counted in the device.) This means that the environmental values will be the data used to make a prediction. Now to get the environmental values of the data points not counting slugs we could either compute the variables or put small sensors there.

A possibility for computing the environmental values is by using spatial interpolation (Wolf Song, personal communication, 07-06-2024) where missing values in a grid can be computed by looking at the existing values. This would allow for a drastic decrease in measurement stations needed in the greenhouse. To employ this, the devices can be placed in a certain grid within a greenhouse and the values for the plants in between the ones with a device can be computed using spatial interpolation.

Using the DHT20 sensor and the soil moisture sensor of the electronic prototype described earlier, a series of measurements were made in one section of the greenhouse in a variety of locations to get a better idea of the performance of the sensor and the distribution of the temperature and humidity. Measurements were made on the left, right and middle sides of the greenhouse with additional measurements being taken on the front and back sides on the middle line. As well, on the left side of the greenhouse, a total of 5 measurements were done to see the fluctuation in measurements.

Looking at humidity, the variation is quite high, with an average humidity on the left side of 57.166%, in the middle it is 71.002% and the measurement on the right is 64.100%. The temperature on average was 20.02 C with a median value of 20.12 C. The variation from front to back is 0.60 C and from left to right 0.37 C.

However, looking at the accuracy it is apparent that especially in humidity there is some fluctuation (See Figure 7). It is therefore important to take that into account when looking at the data and to perhaps take multiple measurements per point and take the average to get a more accurate reading.

When it comes to evaluating moisture the measurements varied greatly, giving frequent measurements that amounted to 0. This can be explained by the fact that the soil was not always accessible and therefore on some occasions the sensor was put between the leaves of the plant. Therefore it is very important that the soil is reached by the sensor to give adequate readings. (Dataset in Appendix M.)

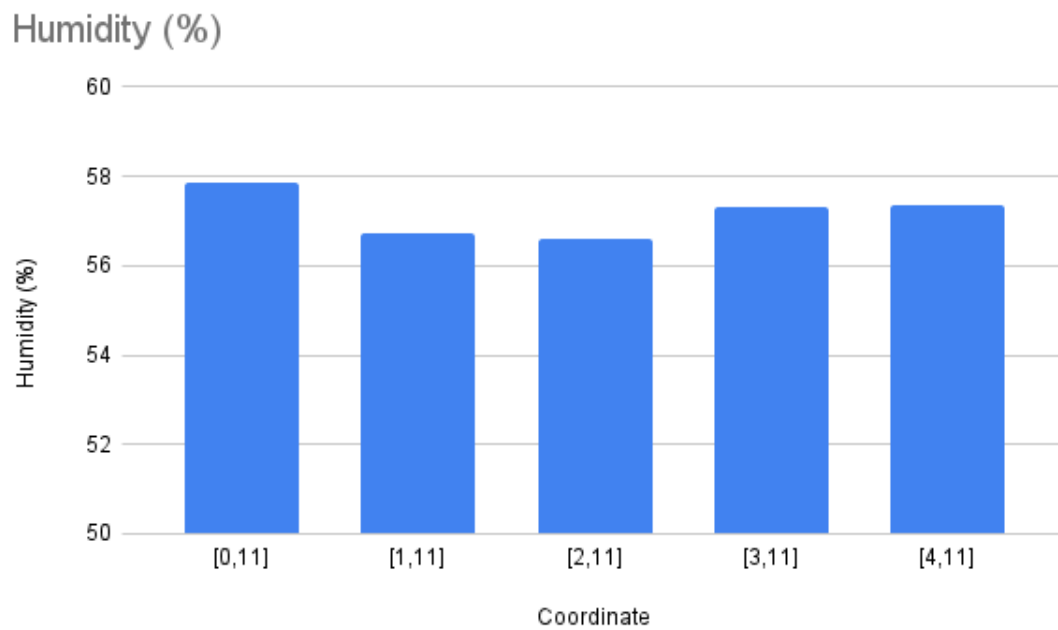
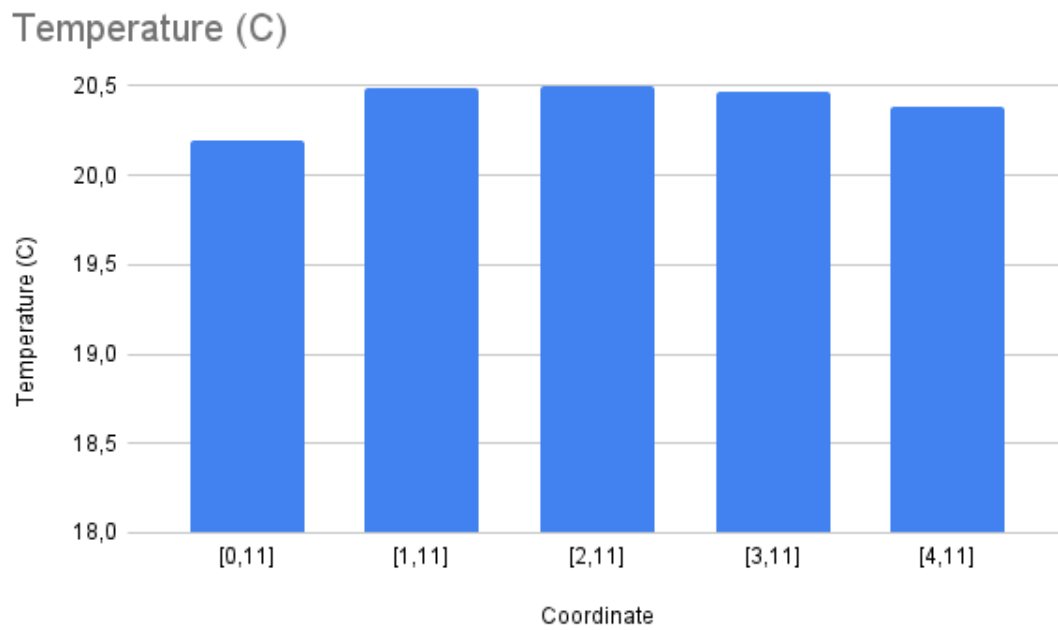


Figure 7: Fluctuations in both the temperature and humidity measurements in one row of plants

9. UPDATE REQUIREMENTS

With the research done in this stage of the project, it is wise to revisit the program of requirements. As it is seen no requirement needs updating after the new findings yet it would be wise to add some additional requirements to make it easier to see what the device must adhere to. Therefore the requirements are now the following:

- The device should function with temperatures varying between 10-22C
- The device should function with a humidity between 50-80%
- The device needs to function with a drip watering system
- The device should be able to operate at night with no artificial lighting
- The device should be able to operate during the day with no artificial lighting with values of 20-50k lux
- A slug should be able to be detected regardless of occlusion by plants
- The data given to the ML model should be as low in storage space as possible
- The device should capture trends over a larger time scale (years)
- The device should not disrupt greenhouse operations
- The device should not grow mould during regular use
- The device should let slugs move freely and not be aimed at trapping them
- The device should be able to function for 3 days without charging

10. INTERMEDIARY CONCEPT

With the findings surrounding the physical design and the image detection system, an intermediary design has been made that integrates the most important design elements and can be used for testing in the final design stage of the project. The designed part is meant to be placed into the prototype container made earlier, separating the slugs from the rest of the container therefore giving the opportunity to use the Aruco markers without occlusion. However, due to the circular shape, Hough circle transform can also be used to crop the images. The wheat bran is separated by the mesh with a good distance between the slugs and the bran to prevent mould growth. As well, the mesh has been made white to ensure maximum contrast between the slug and the environment. The slugs enter through the 4 entry tubes with a 1 cm diameter. Finally, a transparent window can be added on the top for the camera to photograph the slugs.

With a physical 3D printed prototype of the device (Image 17), a first dataset has already been made that is to be cropped using the Hough circle method. One problem was that the circular shape of the device where it was originally attached to the print bed was not completely even. Causing the Hough circle method to not always recognise that particular circle and cropping the image accordingly. (Code in Appendix N and Dataset in Appendix O.)

These findings will be taken into the next stage of the project where the final shape and details of the device will be established.

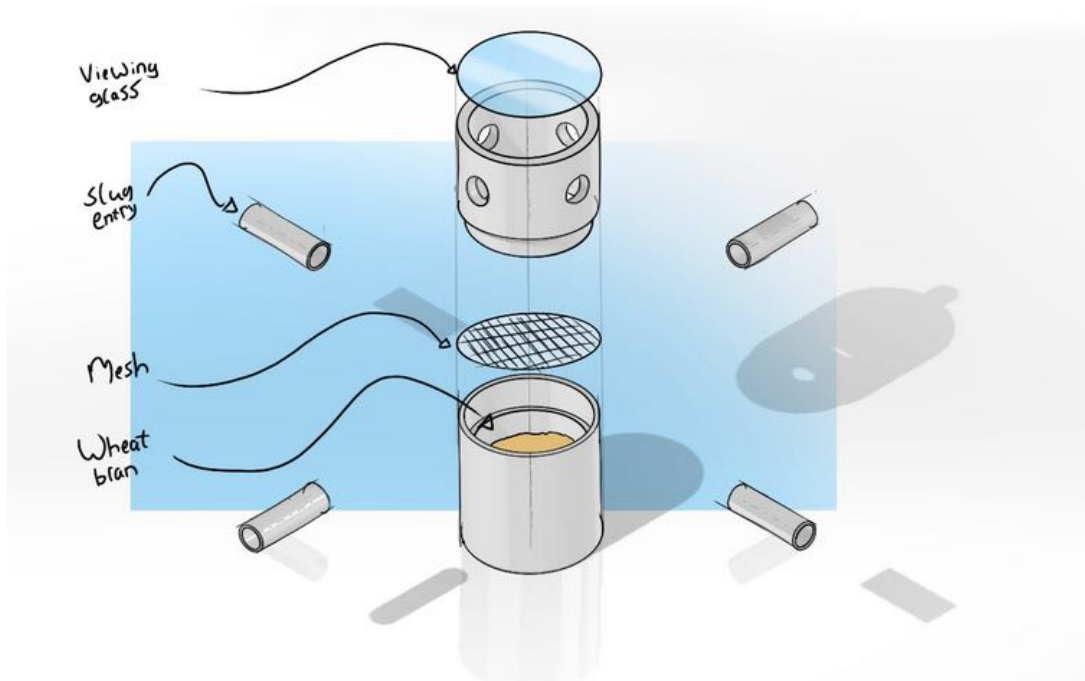


Figure 8: Drawing of the new model



Image 17: Physical model of the new concept

DESIGN DETAILING



11. SHOWCASE

In the final chapter, we will elaborate on the final version of the device. First, the device itself will be introduced to then go more into detail about the process of creating the definite features of the device.

Introducing Cephal, the AI-powered slug measuring device. Cephal refers to the feature of slug eyes being located inside of the head, only extracting when used. Just like a slug eye, this device has an internal camera which forms one of the main features of the device.



Image 18: Cephal

The main purpose of the device is to gather data on several locations within your greenhouse to then predict slug density for other areas within the greenhouse. It does so by gathering temperature and humidity data using a DHT sensor and a moisture sensor gathers the moisture levels of the soil. This data will then be used as parameters for machine learning against the data label slug count. The slug count will be measured by luring slugs to a wire mesh located over wheat bran where a picture will be taken every 3 minutes. From there the images get compressed and processed to 64x64 binary images which serves as input for a machine learning model which is meant to count the slugs.

In the greenhouse, multiple Cephal devices must be deployed to gather data over several locations in the greenhouse. Then, using spatial interpolation, the soil moisture, temperature and humidity will be estimated over the entire greenhouse. Using the measured data by Cephal, a machine learning model is created that estimates slug count for the estimated environmental values.



Image 19: Render of Cephal

The product is based on parts that are easily available and allow for rapid prototyping. The main housing consists of a bran container, a holder for the mesh, the middle part that hosts the entries for the slug and the housing for the electronics.

On the electronics front, the parts that were used earlier have been reused in the final model. These parts mainly revolve around the Seeed Studio XIAO ESP32s3 sense with the provided camera. Alongside this is a DHT20 sensor, a moisture sensor for the environmental values, a 3.7 V battery for power and an LED light to provide consistent photos throughout the night. The data for now gets saved on a 32 GB SD card where the data can get processed later on a separate pc (product schematics on Appendix P).

In the coming chapters there will be documentation on the forming of the final shape, the image detection and deployment in the field.

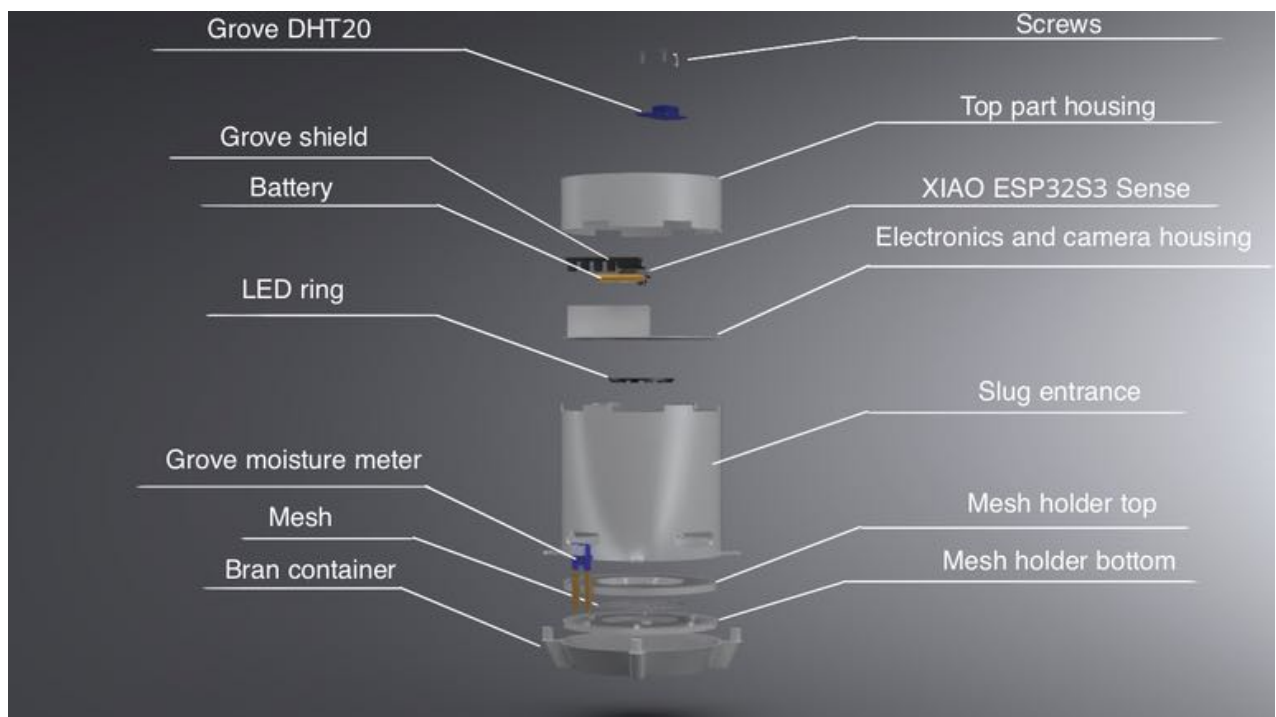


Image 20: Parts list

12. FINAL SHAPE DESIGN

To come up with the final shape of Cephal, some iterations were made to determine the optimal shape and dimensions of the device. Firstly, the electronic prototype and the 3D printed intermediary design were combined, as the idea was that the electronics are mounted on the outer container and the slugs mainly interact with the internal design. From here it quickly became apparent that the bran was hard to access for the user, as well the internal structure was not properly aligned to provide high-quality photos.

To remedy this, a new more open device was made where the mesh was placed underneath a cardboard frame, made to fit into the container and leave a circular shape with the mesh underneath to attract the slugs and for Hough circle transform to properly crop the images. Making several iterations in a trial and error fashion made it clear that a diameter of 6 cm for the mesh and a height for the camera of 10 cm was optimal for making quality images. This model was also used to explore the first proper dataset with slugs, more on that later.



Image 21: The old prototype (left) And the new prototype alongside a previous iteration (right)

From here, quickly some sketches were made that were used as the basis for a Solidworks design (sketches on Appendix Q). The main principle was to make a device of several parts that are to be stacked on top of each other. The body of the device is the middle part, where the slug entryways are located and the mesh. The entryways are slots, as the results from the field testing in chapter 6 pointed out that while all the designs of entryways attracted the slugs, the more open your design the easier the access for the slugs. Therefore the middle part is housed with 4 slots with a height of 1 cm.

The mesh is placed between two discs and lodged firmly in the bottom of the middle part just below the entries. The upper disk is slanted towards the mesh for a better connection between the mesh and the disk and the diameter of the exposed mesh is 6 cm. The whole construction is then mounted on the bran container, which is filled with the bait for the slugs but can't be accessed by the slugs themselves to keep the bait fresh for longer. One feature of the earlier model that has been omitted from the final device was the adding of a transparent barrier between the part accessible by slugs and the camera. This is because the barrier could give a reflection of the LED ring, reducing the visibility and quality of the images.

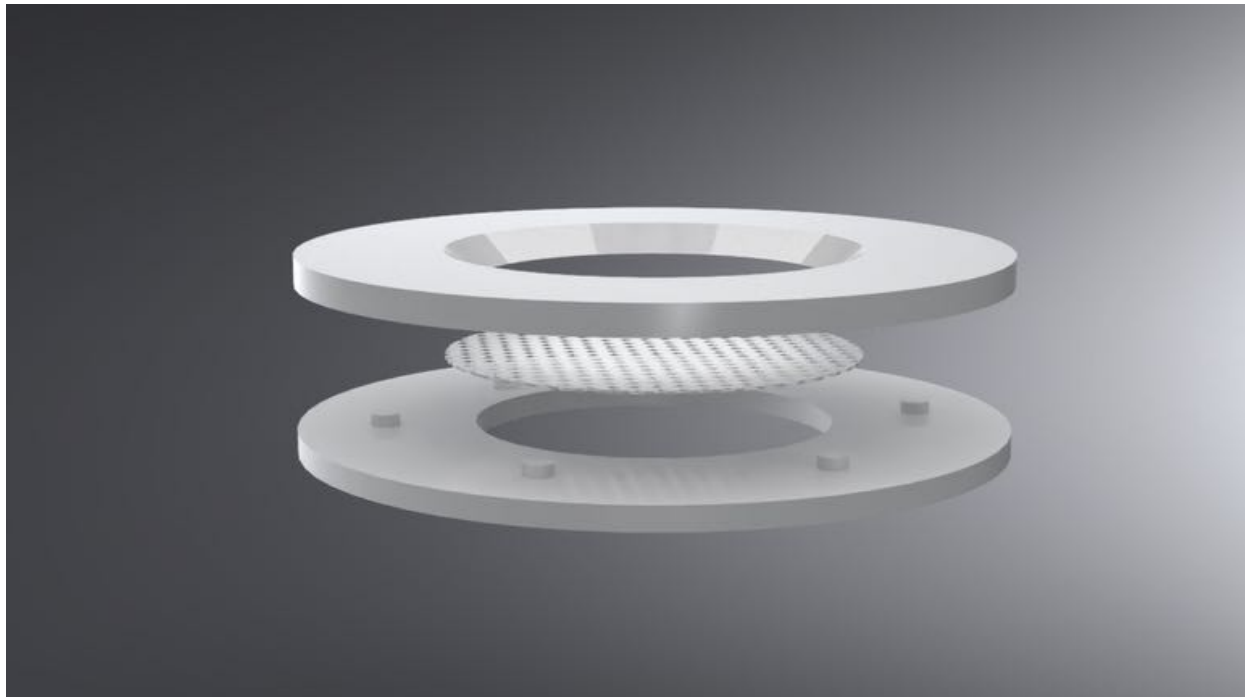


Image 22: Subassembly of the mesh between the top and the lower half

At the top of the device is the electronics housing which is made of two parts. The bottom part houses the XIAO, LED ring, battery and camera. The camera is located in the middle of the device and can access the inner part of the device through a hole which fits the camera lens. On the other side of the housing is the LED ring which is hot glued to the device. The wiring connects through a hole to the rest of the device which needs to be sealed off to prevent slugs and other external factors accessing the electronics. Due to the dimensions of the shell to which the XIAO is connected the diameter of the total device is 12 cm. Earlier research pointed out the benefit of having smaller devices for ease of placement in foliage. With a small test, it is clear that the device can still be placed in the foliage but further research could be done to reduce the size.

The top part of the housing shields the electronics but also houses the DHT20 sensor which is screwed on. As well, it also connects the moisture sensor to the rest of the electronics. The moisture sensor is not secured to the device but instead needs to be connected via a 100 cm cable to ensure free placement into the soil. These factors resulted in the shaping of Cephal, a robust device that is able to function freely on the battery for a couple of hours. If needed for longer measurements, a power bank can be connected to the xiao through a USB-c cable.

In the following two pages, the assembly instruction of Cephal is given.

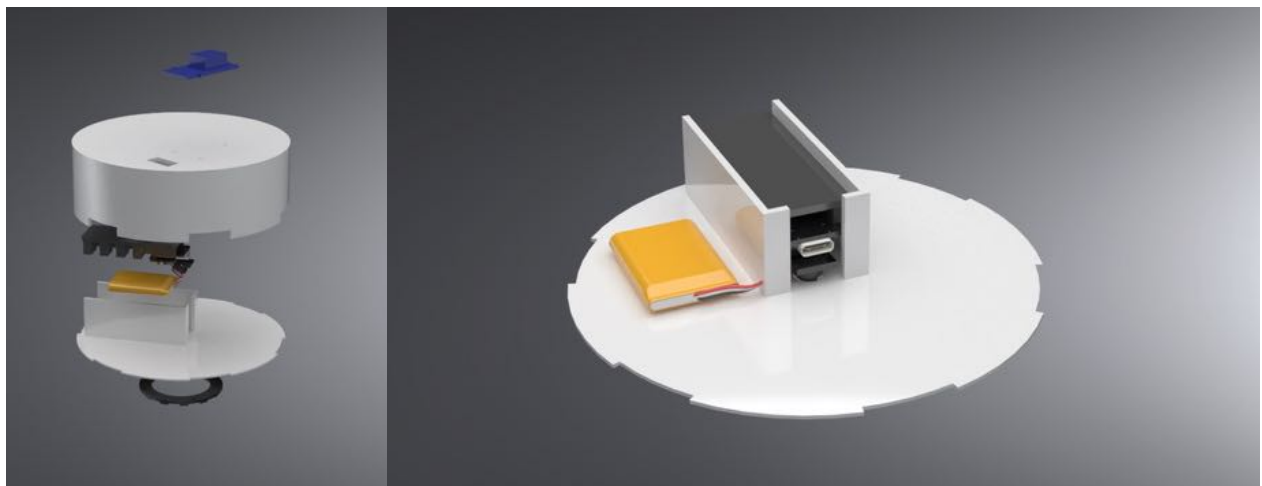


Image 23: Exploded view of the housing, alongside a render of the XIAO and battery mounted inside the housing



Secure mesh between holders, use hotglue if needed.



Secure mesh in middle part of Cephal.



Fill container with wheat bran.



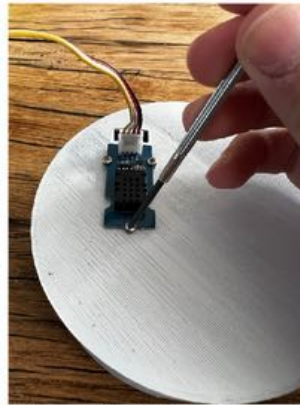
Secure container to Cephal.



Hot glue LED ring to bottom of the housing plate.



Fit cable through hole until it comes out on the other side.



Screw DHT20 on top part of housing.



Put cables of both the DHT20 and moisture meter through hole.



It should look like this on the other side.



Snap housing plate to Cephal.



Snap XIAO to grove shield.



Solder battery to grove shield.



Add camera to XIAO



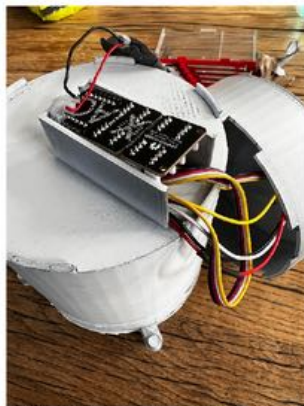
Add 32GB SD to XIAO



Connect electronics, top left DHT20, top right moisture meter and LED bottom right.



Secure Grove shield to housing, make sure the camera fits in the middle hole.



Fit all the wires out of the back side.



Connect Cephal to PC to upload the program.



Disconnect cable and put top on the housing.

Image 24: Assembly instructions

13. TRAINING IMAGE DETECTION

As mentioned in the previous chapter, Cephal is using image detection to count the number of slugs within the device. To construct and validate this, a dataset has been made with both the prototype Cephal as well as the actual device. Next, these datasets were processed and labelled and used as input for a machine learning model based on a Convolutional Neural Network.

The first dataset was a small one made with the prototype of Cephal based on a food container. With this device, a dataset was made with a total of 56 usable images of which 36 contained slugs (dataset on Appendix R). These slugs were placed manually to ensure a varied dataset. Using this dataset, it was found that the photos made using the electronics also used in the final devices were of good enough quality to provide clear identifiable data after processing. To double verify this, the device also shot photos in a dark environment with no light source except its own and provided photos with the same quality that was indistinguishable from the ones shot in daylight.

Using the processing and Hough circle transform, it also became clear that Hough circle transform provided cropping of sufficient quality, even if there was (partial) occlusion of the circle due to slugs covering it the cropping was still done. By including an additional border of 50 pixels the circle is always fully visible in the image.

This dataset was used as input for a machine learning model, of which the base code was taken from Kaggle (Rmishra, 2020). Here it was found that the computational time was too large for practical use, this was both due to the images being too big (uncropped the images were 1600x1200) and the neural network for training being too large and too complex for a smaller dataset. With the small dataset and the long computation time, the result was a rather inaccurate model (Code on Appendix S).

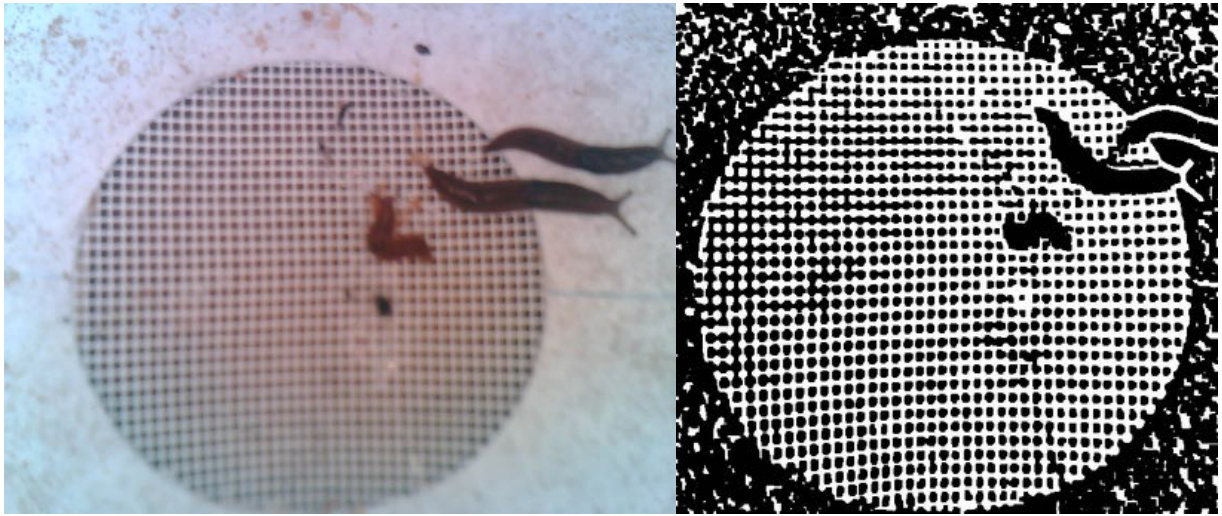


Image 25: Photo from the first dataset before and after processing. Look how the circle is partially occluded yet the cropping still works

With the new device constructed, a larger dataset was made to get a more accurate model. This dataset consisted of 322 usable images, of which 30 contained slugs. As this ratio is not very suitable for machine learning, some augmentation was done on the photos with slugs to create more instances of the images. This was rotation and mirroring and created 5 clones of each image, resulting in a total dataset of 482 images which were subsequently cropped and processed.

To make machine learning possible, the images were compressed to 64x64 pixels and the network has been simplified to contain fewer layers and take smaller input data (code on Appendix T). After these steps, a model with around 90% accuracy has been trained on the new dataset, showing the possibilities for training a proper model based on the data gathered by Cephal (dataset on Appendix R).

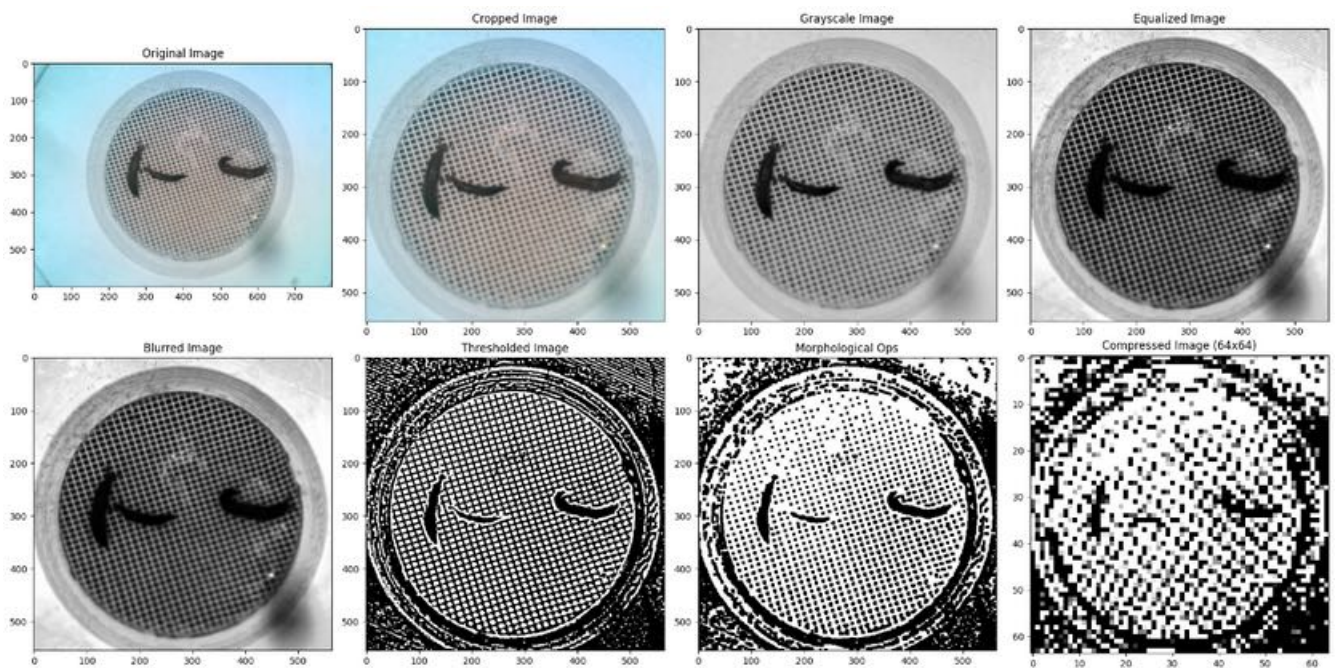


Figure 9: steps of the image processing

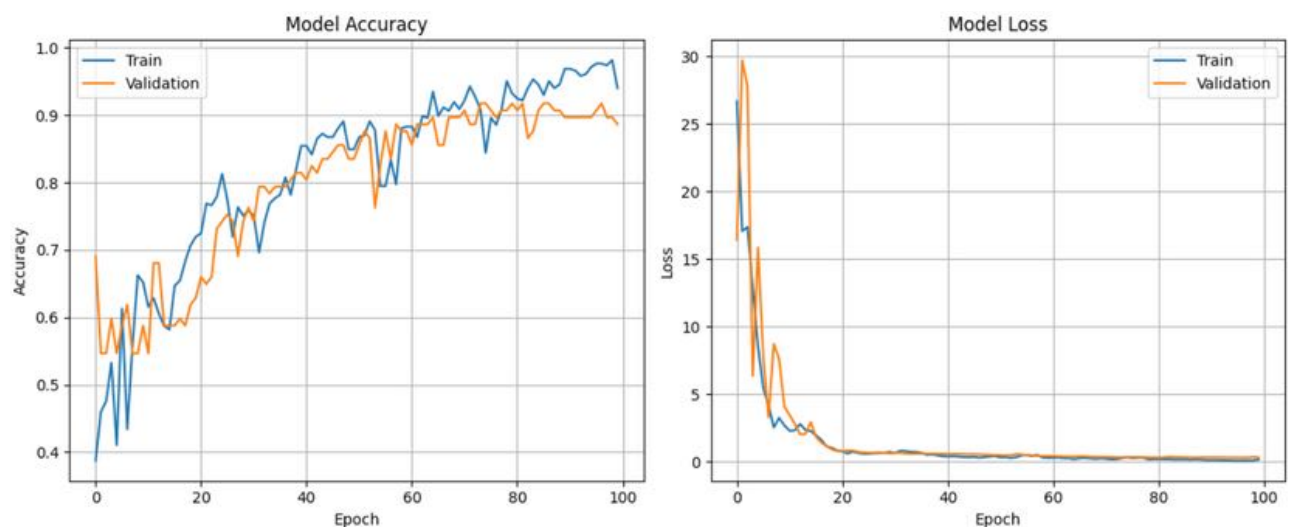


Figure 10: Model performance trained with the Cephal dataset

14. FIELD TESTING

With the prototype of Cephel constructed the device should be tested in context to see if it can provide suitable data over a longer time period. To do this a test setup was made to verify the ability of Cephel.



Image 26: Test setup

Method

Cephel has been placed within a plant with a reported high slug activity. The bran container has been filled with wheat bran and the device has been set to take a photo every 3 minutes. The interval was picked based on the fact that in the final product, this interval would save crucial battery time and data while still providing a reliable overview. For the test, the device was connected to a power bank of 25000 mAh. The device was placed for a total timespan of 24 hours to visualise both night and day. The photos are then taken and analysed to see any slug activity and observe the environmental values. To compare the environmental values, the owner of the greenhouse has also made their own measurements available for that section of the greenhouse (Han van Tilburg, personal communication, 05-09-2024).

Results

Looking at the results a couple of things have become clear. First of all, no slugs have visited the device unfortunately. Furthermore, the moisture sensor did not give any successful measurements as the cable was originally 50 cm long. This proved to still be too short to reach the soil of the plant due to the larger dimensions of the device.

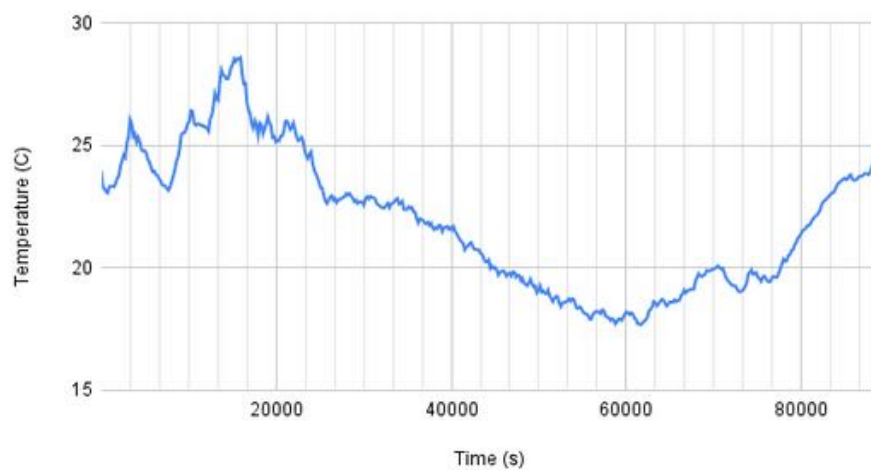
Despite these findings, the device did successfully generate a database of 485 images with the corresponding environmental measurements. The environmental data that was found with Cephal compared to the measured data is found in Figure 11.

As seen there is some discrepancy in the data as the measured temperature by Cephal seems to be higher than what was measured by the greenhouse while the humidity level seems to be lower than the humidity level measured by the greenhouse owner. Despite this, what can also be seen by looking at the graphs is that the trends in temperature and humidity throughout time do seem to match with both measurements with both stations depicting a spike in temperature and drop in humidity right around the 3 pm mark (results on Appendix U).

		Min	Max	Avg
Temperature	Measured	17,68	28,59	21,92131959
	Greenhouse	15,9	24,7	19,8
Humidity	Measured	51,85	77,49	68,04552577
	Greenhouse	67	93	85

Figure 11: Table with measured values compared to greenhouse values

Temperature (C) versus Time (s)



Humidity (%) versus Time (s)

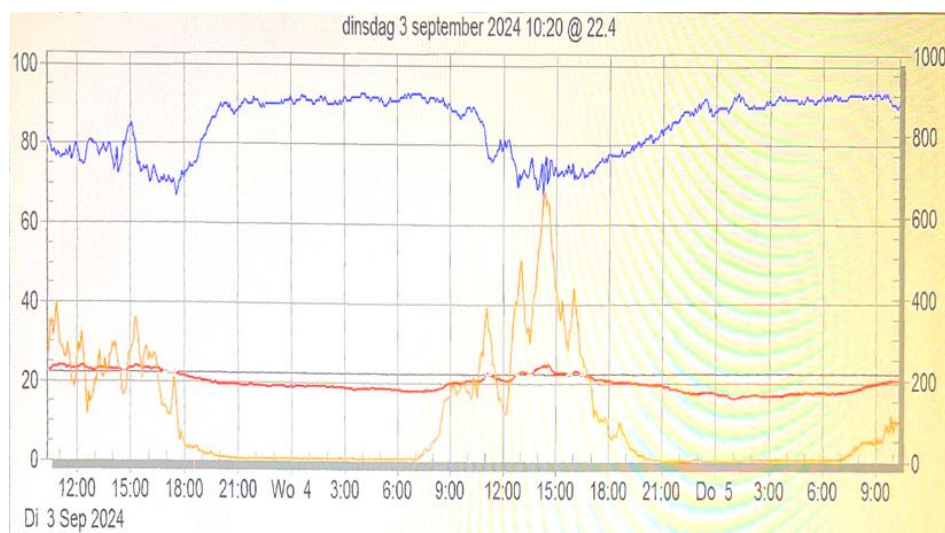
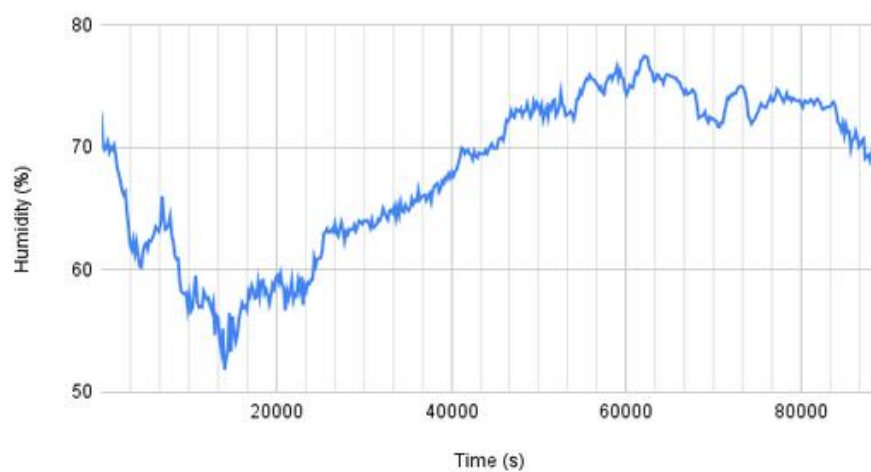


Figure 12: Measured values (top 2 graphs) compared to greenhouse values (bottom graph) with blue being humidity and red being temperature. The 20.000 second mark corresponds with 15:00 on Wednesday (Han van Tilburg, personal communication, 05-09-2024)

Interpretation

Looking at the device's performance it is safe to assume that the device is able to gather data and capture the trends present in the greenhouse. The big difference between the measurements of the greenhouse and of Cephal when it comes to humidity and temperature however does indicate that there might be an issue with the performance of the sensor or that there is a large fluctuation of environmental values throughout the greenhouse that causes the average to be vastly different from locally measured values.

Besides this, it is interesting that there were no slugs found in the device. This can most likely be explained that while the plant generally showed slug activity in the past, as of late the slug activity may have died down. As well the device had a layer of paint on it which may have masked the scent of the wheat bran.

Conclusion

Looking at the findings the following conclusions can be drawn: First of all in order to be secure about the environmental measurements more tests should be done on multiple locations to compare it to the measurements of the greenhouse to look for any trends and differences. As well it would be wise to compare different sensors and look at the performance of sensors over time. As well, the moisture cable needs to be even longer, 100 cm should suffice. Lastly, it would be a good idea to deploy unpainted models, printed in white filament, on multiple plants with recorded high slug intensity over a couple of days and compare it as well to the lure designs tested earlier in this research to look at the performance.

15. CONCLUSION

Looking back, the original purpose of this report was to design a device capable of generating an image dataset meant to train an image detection algorithm with the purpose of predicting slug infestation in a greenhouse. Through research of the context and exploratory experiments, it became clear that a device was needed that was capable of predicting slug behaviour by environmental values while also being capable of luring and identifying slugs using image detection. While doing so the device should be capable of measuring the environmental values generally present in a greenhouse. As well the device should be operable during day and night in the dark regardless of occlusion by plants all the while providing low-volume data suitable for machine learning.

With the creation of Cephal it has shown to be possible for a device to gather this type of data in a greenhouse suitable for image detection based machine learning resulting in a model of about 90% accuracy. This is done by luring slugs using wheat bran and taking photos periodically with a flash function to keep consistent data through day and night. By converting the photos to a 64x64 pixels binary image the model is able to quickly and efficiently estimate the number of slugs on the input image and use this as a label for the environmental data gathered with the other sensors on Cephal. With the dataset generated by Cephal, it should be possible to make a machine learning model capable of predicting slug density in a greenhouse and converting that to a heat map indicating where control is needed.

15.1 Limitations and recommendations

Due to the time constraints and scope of the assignment there are a couple of limitations and suggestions for further research to improve on the concept of Cephal. The selection of the electronics was partially based on the requirements but also on availability. Given that these are off-the-shelf components it gives room for further optimization. For instance, a camera could be picked with a more suitable focal length to decrease the required height of Cephal and simultaneously a custom pcb could reduce the required diameter of the device.

Likewise, the battery can be improved if the program successfully works with a sleeper module to increase the runtime of the device. Lastly there should be more research on the sensors to critically evaluate the performance both on short term runtime and longer runtime.

machine learning to predict slug density invites further research in order to gain an efficient model. To do this, there should be research to determine the optimal density and distribution of the Cephal measuring stations in order to provide enough data while also being able to give a reliable prediction of the rest of the environmental values through spatial interpolation. With those values known and enough data gathered a regression model should be trained and optimised that is capable of prediction.

To recap and summarise, Cephal has shown the potential of a ML based slug measuring station that could benefit from further optimization and invites for the training of a regression based model.

16. REFERENCES

1. Supervised learning. (n.d.). Scikit-learn. https://scikit-learn.org/stable/supervised_learning.html#supervised-learning

Aartsen, B., Drost, L., Lolling, N., Van Kampen, K., Verdoes, F., & Wienke, S. (2022). BACKie: “The solution to catching slugs in orchid growing greenhouses”.

Chen, F., & Chin, S. (2021). The orchid genome. In *Compendium of plant genomes*. <https://doi.org/10.1007/978-3-030-66826-6>

Douglas, M. R., & Tooker, J. F. (2012). Slug (Mollusca: Agriolimacidae, Arionidae) Ecology and Management in No-Till Field Crops, With an Emphasis on the mid-Atlantic Region. *Journal Of Integrated Pest Management*, 3(1), C1-C9. <https://doi.org/10.1603/ipm11023>

Floricultura b.v. (2021). *Cymbidium Snijbloem groot- en kleinbloemig Teelt Substraat*. In *Floricultura*. https://www.floricultura.com/media/1042/teeltadvies_cymbidium_snijbloem_nl.pdf

Gai, R., Chen, N., & Yuan, H. (2021). A detection algorithm for cherry fruits based on the improved YOLO-v4 model. *Neural Computing And Applications*, 35(19), 13895-13906. <https://doi.org/10.1007/s00521-021-06029-z>

Holger Nennmann. (2021). Orchideeën ongedierte: Slakken. In www.hark-orchideen.de. <https://www.hark-orchideen.nl/wp-content/uploads/2021/03/hark-schaedlinge-slakken-nl.pdf>

Kootstra, G., Wang, X., Blok, P. M., Hemming, J., & Van Henten, E. (2021). Selective Harvesting Robotics: Current Research, Trends, and Future Directions. *Current Robotics Reports*, 2(1), 95-104. <https://doi.org/10.1007/s43154-020-00034-1>

Kozłowski, R. J., Kozłowski, J., Przybył, K., Niedbała, G., Mueller, W., Okoń, P., Wojcieszak, D., Koszela, K., & Kujawa, S. (2016). Image analysis techniques in the study of slug behaviour. *Proceedings Of SPIE, The International Society For Optical Engineering/Proceedings Of SPIE*. <https://doi.org/10.1117/12.2244533>

Li, J. B., Huang, W. Q., & Zhao, C. J. (2014). Machine vision technology for detecting the external defects of fruits — a review. *The Imaging Science Journal*, 63(5), 241-251. <https://doi.org/10.1179/1743131x14y.00000000088>

Liu, G., Nouaze, J. C., Mbouembe, P. L. T., & Kim, J. H. (2020). YOLO-Tomato: A Robust Algorithm for Tomato Detection Based on YOLOv3. *Sensors*, 20(7), 2145.
<https://doi.org/10.3390/s20072145>

Ministerie van Algemene Zaken. (2024, 21 march). Afzetgegevens gewasbeschermingsmiddelen in Nederland. Publicatie | Rijksoverheid.nl.
<https://www.rijksoverheid.nl/onderwerpen/bestrijdingsmiddelen/documenten/publicaties/2022/05/19/afzetgegevens-gewasbeschermingsmiddelen-in-nederland>

Rmishra. (2020, June 18). Counting crowd with CNN- social distancing project. Kaggle.
<https://www.kaggle.com/code/rmishra258/counting-crowd-with-cnn-social-distancing-project>

Stephenson, J. W. (1968). A REVIEW OF THE BIOLOGY AND ECOLOGY OF SLUGS OF AGRICULTURAL IMPORTANCE. *Journal Of Molluscan Studies*.
<https://doi.org/10.1093/oxfordjournals.mollus.a065036>

Wikipedia contributors. (2023, 11 december). *Ambigolimax valentianus*. Wikipedia.
https://en.wikipedia.org/wiki/Ambigolimax_valentianus#cite_note-Rowson2014b-7

Xia, C., Chon, T., Ren, Z., & Lee, J. (2015). Automatic identification and counting of small size pests in greenhouse conditions with low computational cost. *Ecological Informatics*, 29, 139–146. <https://doi.org/10.1016/j.ecoinf.2014.09.006>



Name: Dr. Song, Y. (Wolf)
Chair

Name: Ing. Verwaal, M.
Mentor



Appendix

Appendix A- Questionnaire results

Results:

https://docs.google.com/spreadsheets/d/1V13V2YxCpkg1yqJXH8ALBuPpVON82n_nzQ0oTlhA9IM/edit?usp=sharing

Appendix B - Object detection code using colour

```

import cv2
import numpy as np

def compress_image(image, max_width=800):
    """Compresses the input image to a maximum width."""
    height, width = image.shape[:2]
    if width > max_width:
        ratio = max_width / width
        new_height = int(height * ratio)
        return cv2.resize(image, (max_width, new_height))
    return image

def isolate_brown_objects(image):
    """Isolates brown objects in the input image."""
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    lower_brown = np.array([5, 50, 50]) # Adjusted lower
threshold for brown
    upper_brown = np.array([30, 255, 255]) # Adjusted upper
threshold for brown
    mask = cv2.inRange(hsv, lower_brown, upper_brown)

    # Morphological operations to fill gaps and smooth edges
    kernel = np.ones((5, 5), np.uint8)
    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)

    # Dilate the mask to include nearby pixels of the same
color
    mask = cv2.dilate(mask, kernel, iterations=1)

    brown_objects = cv2.bitwise_and(image, image, mask=mask)
    return brown_objects, mask

def find_contours(mask, min_area=100):
    """Finds contours in the binary mask."""
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    # Filter contours based on area
    large_contours = [cnt for cnt in contours if
cv2.contourArea(cnt) > min_area]

```

```
return large_contours
```

```
def is_slug(contour, aspect_ratio_thresh=0.2,
solidity_thresh=0.7):
    """Checks if the contour has slug-like characteristics."""
    # Calculate the bounding rectangle and aspect ratio
    x, y, w, h = cv2.boundingRect(contour)
    aspect_ratio = w / h

    # Calculate the area of the contour and its convex hull
    area = cv2.contourArea(contour)
    hull = cv2.convexHull(contour)
    hull_area = cv2.contourArea(hull)

    # Calculate the solidity (contour area / convex hull area)
    solidity = area / hull_area

    # Check if aspect ratio and solidity meet slug-like
criteria
    return (aspect_ratio > aspect_ratio_thresh) and (solidity >
solidity_thresh)
```

```
def crop_objects(image, contours):
    """Crops brown objects from the original image."""
    cropped_images = []
    for i, contour in enumerate(contours):
        x, y, w, h = cv2.boundingRect(contour)
        cropped_images.append(image[y:y + h, x:x + w])
    return cropped_images
```

```
def main(input_image_path):
    # Load input image
    original_image = cv2.imread(input_image_path)

    # Compress the image for easier processing
    compressed_image = compress_image(original_image)

    # Isolate brown objects
    brown_objects, mask =
isolate_brown_objects(compressed_image)
```

```
# Find contours of brown objects
contours = find_contours(mask, min_area=2000) # Adjust
min_area as needed

# Filter contours for slug-like shapes
slug_contours = [contour for contour in contours if
is_slug(contour)]

# Crop slug-like objects from the original image
cropped_images = crop_objects(original_image,
slug_contours)

# Save cropped images
for i, cropped_image in enumerate(cropped_images):
    cv2.imwrite(f"slug_{i}.jpg", cropped_image)

if __name__ == "__main__":
    input_image_path = input("Enter path to the input image: ")
    main(input_image_path)
```


Appendix C - Difference in image computation code

```

import cv2
import numpy as np

def image_difference(image1, image2):
    # Read the images
    img1 = cv2.imread(image1)
    img2 = cv2.imread(image2)

    # Check if images are loaded successfully
    if img1 is None or img2 is None:
        print("Error: Could not read the images.")
        return

    # Convert images to grayscale
    gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
    gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

    # Calculate absolute difference
    diff = cv2.absdiff(gray1, gray2)

    # Thresholding to keep only pixel values above 200
    _, thresholded = cv2.threshold(diff, 100, 255,
cv2.THRESH_BINARY)

    # Resize the image to 224x224
    resized_image = cv2.resize(thresholded, (224, 224))
    #resized_image = cv2.resize(diff, (224, 224))
    return resized_image

def main():
    # Path to the input images
    image1_path = 'D:\Afstudeer documenten\Afstudeer
documenten\Images\slug1.PNG'
    image2_path = 'D:\Afstudeer documenten\Afstudeer
documenten\Images\slug2.PNG'

    # Compute the difference between the two images
    difference_image = image_difference(image1_path,
image2_path)

    # Display the difference image

```

```
cv2.imshow('Difference Image', difference_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
cv2.imwrite('D:\Afstudeer documenten\Afstudeer
documenten\Images\BLOB.jpg', difference_image)
```

```
if __name__ == "__main__":
    main()
```

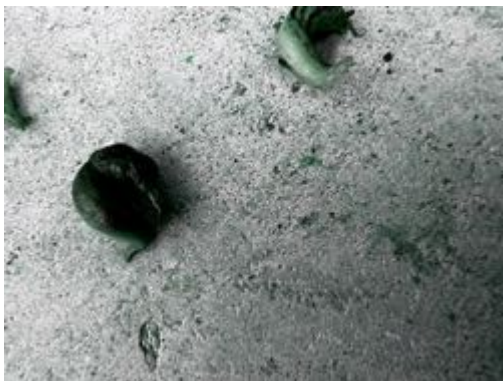
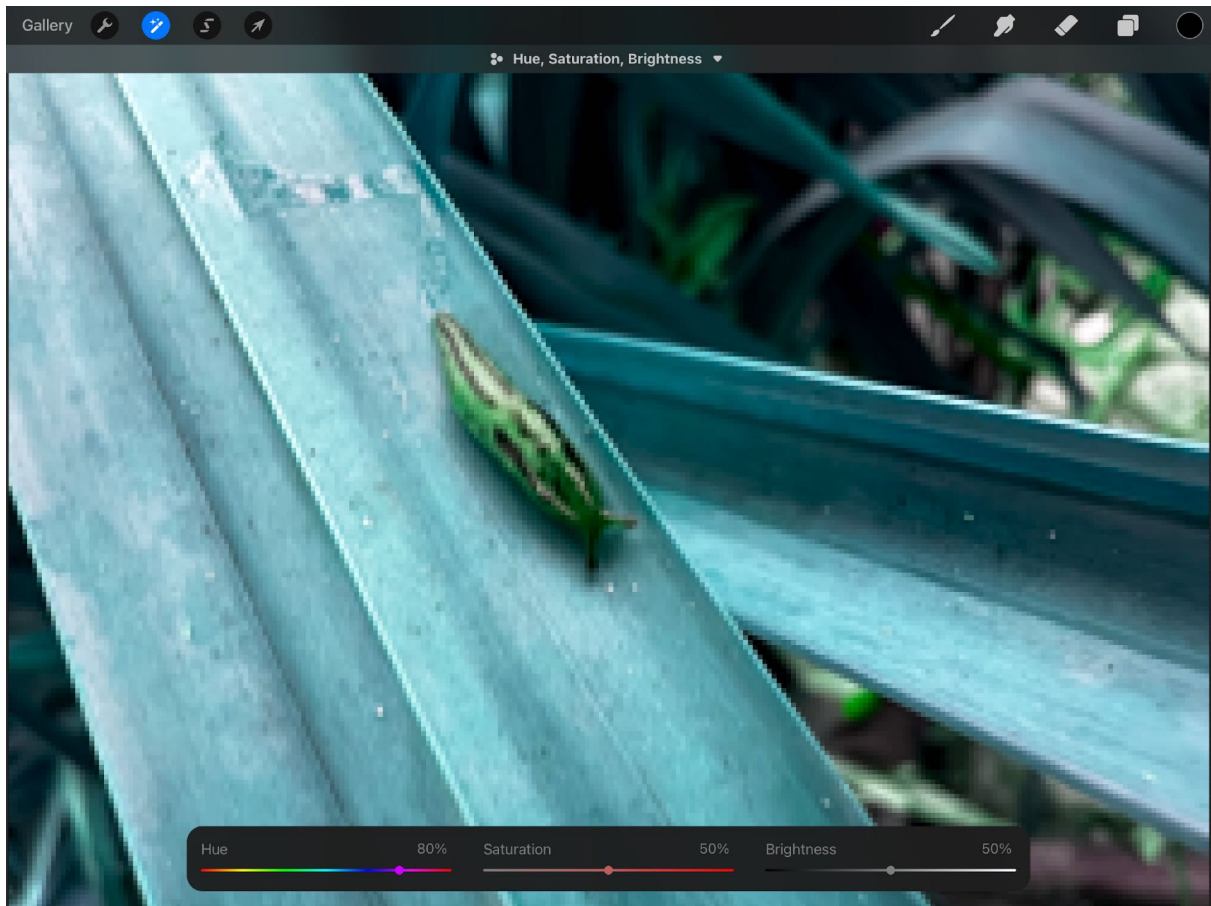
Appendix D - Playing with images











<https://drive.google.com/drive/folders/1kFyPD8XML3BwzGo6AF9uAMMCQEirttxO?usp=sharing>

Appendix E - First ML model


```
# Import Data Science Libraries
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
import itertools
import random
```

```
# Import visualization libraries
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import cv2
import seaborn as sns
```

```
# Tensorflow Libraries
from tensorflow import keras
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import
ImageDataGenerator
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import Callback,
EarlyStopping, ModelCheckpoint
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras import Model
from tensorflow.keras.layers.experimental import preprocessing
from keras.layers import Dense, Flatten, Dropout,
BatchNormalization
```

```
# System libraries
from pathlib import Path
import os.path
```

```
# Metrics
from sklearn.metrics import classification_report,
confusion_matrix
```

```
#extra help: Import series of helper functions for our
notebook
```

```

from helper_functions import create_tensorboard_callback,
plot_loss_curves, unzip_data, compare_historys,
walk_through_dir, pred_and_plot

# Seed Everything to reproduce results for future use cases
def seed_everything(seed=42):
    # Seed value for TensorFlow
    tf.random.set_seed(seed)

    # Seed value for NumPy
    np.random.seed(seed)

    # Seed value for Python's random library
    random.seed(seed)

    # Force TensorFlow to use single thread
    # Multiple threads are a potential source of
    non-reproducible results.
    session_conf = tf.compat.v1.ConfigProto(
        intra_op_parallelism_threads=1,
        inter_op_parallelism_threads=1
    )

    # Make sure that TensorFlow uses a deterministic operation
    wherever possible
    tf.compat.v1.set_random_seed(seed)

    sess =
tf.compat.v1.Session(graph=tf.compat.v1.get_default_graph(),
config=session_conf)
    tf.compat.v1.keras.backend.set_session(sess)

sns.set(style='darkgrid')

seed_everything()

BATCH_SIZE = 32
TARGET_SIZE = (224, 224)

# Walk through each directory
dataset = "./data/"
walk_through_dir(dataset)

def convert_path_to_df(dataset):

```

```

image_dir = Path(dataset)

# Get filepaths and labels
filepaths = list(image_dir.glob(r'**/*.JPG')) +
list(image_dir.glob(r'**/*.jpg')) +
list(image_dir.glob(r'**/*.png')) +
list(image_dir.glob(r'**/*.PNG'))

labels = list(map(lambda x:
os.path.split(os.path.split(x)[0])[1], filepaths))

filepaths = pd.Series(filepaths,
name='Filepath').astype(str)
labels = pd.Series(labels, name='Label')

# Concatenate filepaths and labels
image_df = pd.concat([filepaths, labels], axis=1)
return image_df

image_df = convert_path_to_df(dataset)

# Check for corrupted images within the dataset
import PIL
from pathlib import Path
from PIL import UnidentifiedImageError

path =
Path("/kaggle/input/grape-disease-dataset-original").rglob("*.
jpg")
for img_p in path:
    try:
        img = PIL.Image.open(img_p)
    except PIL.UnidentifiedImageError:
        print(img_p)

label_counts = image_df['Label'].value_counts()

plt.figure(figsize=(10, 6))
sns.barplot(x=label_counts.index, y=label_counts.values,
alpha=0.8, palette='rocket')
plt.title('Distribution of Labels in Image Dataset',
fontsize=16)
plt.xlabel('Label', fontsize=14)
plt.ylabel('Count', fontsize=14)

```

```

plt.xticks(rotation=45)
plt.show()

# Display 16 picture of the dataset with their labels
random_index = np.random.randint(0, len(image_df), 16)
fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(10, 10),
                          subplot_kw={'xticks': [], 'yticks':
[]}))

for i, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(image_df.Filepath[random_index[i]]))
    ax.set_title(image_df.Label[random_index[i]])
plt.tight_layout()
plt.show()

def compute_ela_cv(path, quality):
    temp_filename = 'temp_file_name.jpeg'
    SCALE = 15
    orig_img = cv2.imread(path)
    orig_img = cv2.cvtColor(orig_img, cv2.COLOR_BGR2RGB)

    cv2.imwrite(temp_filename, orig_img,
[cv2.IMWRITE_JPEG_QUALITY, quality])

    # read compressed image
    compressed_img = cv2.imread(temp_filename)

    # get absolute difference between img1 and img2 and
multiply by scale
    diff = SCALE * cv2.absdiff(orig_img, compressed_img)
    return diff

from PIL import Image, ImageChops, ImageEnhance

def convert_to_ela_image(path, quality):
    temp_filename = 'temp_file_name.jpeg'
    ela_filename = 'temp_ela.png'
    image = Image.open(path).convert('RGB')
    image.save(temp_filename, 'JPEG', quality = quality)
    temp_image = Image.open(temp_filename)

    ela_image = ImageChops.difference(image, temp_image)

    extrema = ela_image.getextrema()

```

```

max_diff = max([ex[1] for ex in extrema])
if max_diff == 0:
    max_diff = 1

scale = 255.0 / max_diff
ela_image =
ImageEnhance.Brightness(ela_image).enhance(scale)

return ela_image

```

```

def random_sample(path, extension=None):
    if extension:
        items = Path(path).glob(f'*.{extension}')
    else:
        items = Path(path).glob(f'*')

```

```

    items = list(items)

```

```

    p = random.choice(items)
    return p.as_posix()

```

```

# View random sample from the dataset
p = random_sample('./data/ants')
orig = cv2.imread(p)
orig = cv2.cvtColor(orig, cv2.COLOR_BGR2RGB) / 255.0
init_val = 100
columns = 3
rows = 3

```

```

fig=plt.figure(figsize=(15, 10))
for i in range(1, columns*rows +1):
    quality=init_val - (i-1) * 8
    img = compute_ela_cv(path=p, quality=quality)
    if i == 1:
        img = orig.copy()
    ax = fig.add_subplot(rows, columns, i)
    ax.title.set_text(f'q: {quality}')
    plt.imshow(img)
plt.show()

```

```

train_df, test_df = train_test_split(image_df, test_size=0.2,
shuffle=True, random_state=42)

```



```
train_generator = ImageDataGenerator(  
  
    preprocessing_function=tf.keras.applications.efficientnet_v2.p  
    reprocess_input,  
    validation_split=0.2  
)
```

```
test_generator = ImageDataGenerator(  
  
    preprocessing_function=tf.keras.applications.efficientnet_v2.p  
    reprocess_input  
)
```

```
# Split the data into three categories.  
train_images = train_generator.flow_from_dataframe(  
    dataframe=train_df,  
    x_col='Filepath',  
    y_col='Label',  
    target_size=(224, 224),  
    color_mode='rgb',  
    class_mode='categorical',  
    batch_size=32,  
    shuffle=True,  
    seed=42,  
    subset='training'  
)
```

```
val_images = train_generator.flow_from_dataframe(  
    dataframe=train_df,  
    x_col='Filepath',  
    y_col='Label',  
    target_size=(224, 224),  
    color_mode='rgb',  
    class_mode='categorical',  
    batch_size=32,  
    shuffle=True,  
    seed=42,  
    subset='validation'  
)
```

```
test_images = test_generator.flow_from_dataframe(  
    dataframe=test_df,  
    x_col='Filepath',  
    y_col='Label',
```

```
target_size=(224, 224),
color_mode='rgb',
class_mode='categorical',
batch_size=32,
shuffle=False
)
```

```
# Data Augmentation Step
augment = tf.keras.Sequential([
    layers.experimental.preprocessing.Resizing(224,224),
    layers.experimental.preprocessing.Rescaling(1./255),
    layers.experimental.preprocessing.RandomFlip("horizontal"),
    layers.experimental.preprocessing.RandomRotation(0.1),
    layers.experimental.preprocessing.RandomZoom(0.1),
    layers.experimental.preprocessing.RandomContrast(0.1),
])
```

```
# Load the pretrained model
pretrained_model =
tf.keras.applications.efficientnet_v2.EfficientNetV2L(
    input_shape=(224, 224, 3),
    include_top=False,
    weights='imagenet',
    pooling='max'
)
```

```
pretrained_model.trainable = False
```

```
# Create checkpoint callback
checkpoint_path = "pests_cats_classification_model_checkpoint"
checkpoint_callback = ModelCheckpoint(checkpoint_path,
                                     save_weights_only=True,
                                     monitor="val_accuracy",
                                     save_best_only=True)
```

```
# Setup EarlyStopping callback to stop training if model's
val_loss doesn't improve for 3 epochs
early_stopping = EarlyStopping(monitor = "val_loss", # watch
the val loss metric
                             patience = 5,
                             restore_best_weights = True) #
if val loss decreases for 3 epochs in a row, stop training
```

```
#model training
```

```

inputs = pretrained_model.input
x = augment(inputs)

x = Dense(128, activation='relu')(pretrained_model.output)
x = Dropout(0.45)(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.45)(x)

outputs = Dense(12, activation='softmax')(x)

model = Model(inputs=inputs, outputs=outputs)

model.compile(
    optimizer=Adam(0.00001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

history = model.fit(
    train_images,
    steps_per_epoch=len(train_images),
    validation_data=val_images,
    validation_steps=len(val_images),
    epochs=100,
    callbacks=[
        early_stopping,
        create_tensorboard_callback("training_logs",
"pests_cats_classification"),
        checkpoint_callback,
    ]
)

results = model.evaluate(test_images, verbose=0)

print("    Test Loss: {:.5f}".format(results[0]))
print("Test Accuracy: {:.2f}%".format(results[1] * 100))

accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

```

```
epochs = range(len(accuracy))
plt.plot(epochs, accuracy, 'b', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'r', label='Validation
accuracy')
```

```
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
```

```
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

```
# Predict the label of the test_images
pred = model.predict(test_images)
pred = np.argmax(pred,axis=1)
```

```
# Map the label
labels = (train_images.class_indices)
labels = dict((v,k) for k,v in labels.items())
pred = [labels[k] for k in pred]
```

```
# Display the result
print(f'The first 5 predictions: {pred[:5]}')
```

```
# Display 25 random pictures from the dataset with their
labels
random_index = np.random.randint(0, len(test_df) - 1, 15)
fig, axes = plt.subplots(nrows=3, ncols=5, figsize=(25, 15),
                        subplot_kw={'xticks': [], 'yticks':
[]})
```

```
for i, ax in enumerate(axes.flat):
```

```
ax.imshow(plt.imread(test_df.Filepath.iloc[random_index[i]]))
    if test_df.Label.iloc[random_index[i]] ==
pred[random_index[i]]:
        color = "green"
    else:
        color = "red"
```

```

    ax.set_title(f"True:
{test_df.Label.iloc[random_index[i]]}\nPredicted:
{pred[random_index[i]]}", color=color)
plt.show()
plt.tight_layout()

```

```

y_test = list(test_df.Label)
print(classification_report(y_test, pred))

```

```

'''
make_confusion_matrix(y_test, pred, list(labels.values()))

```

```

# Display the part of the pictures used by the neural network
to classify the pictures
fig, axes = plt.subplots(nrows=3, ncols=5, figsize=(15, 10),
                          subplot_kw={'xticks': [], 'yticks':
[]})

```

```

for i, ax in enumerate(axes.flat):
    img_path = test_df.Filepath.iloc[random_index[i]]
    img_array = preprocess_input(get_img_array(img_path,
size=img_size))
    heatmap = make_gradcam_heatmap(img_array, model,
last_conv_layer_name)
    cam_path = save_and_display_gradcam(img_path, heatmap)
    ax.imshow(plt.imread(cam_path))
    ax.set_title(f"True:
{test_df.Label.iloc[random_index[i]]}\nPredicted:
{pred[random_index[i]]}")
plt.tight_layout()
plt.show()
'''

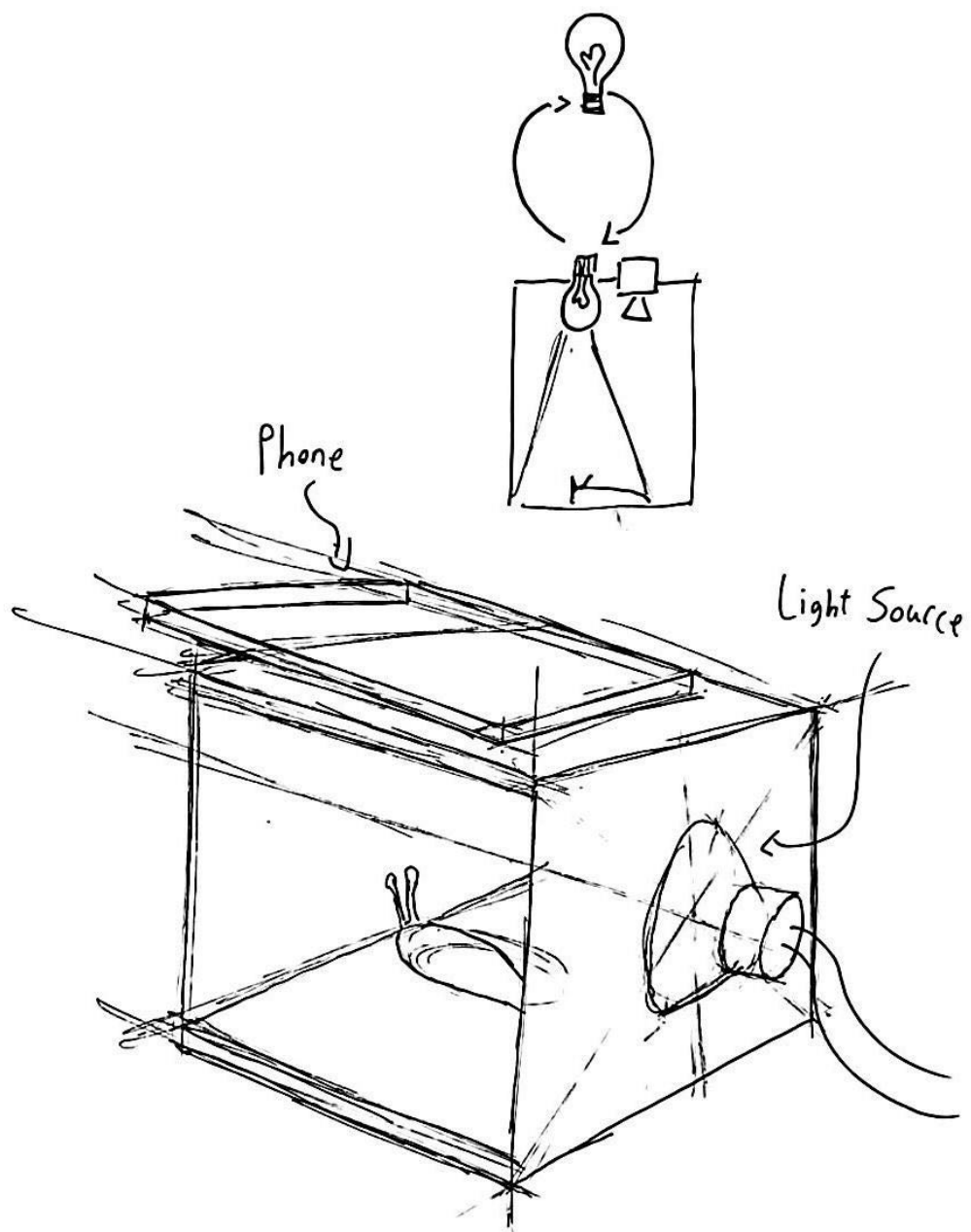
```


Appendix F - First Dataset

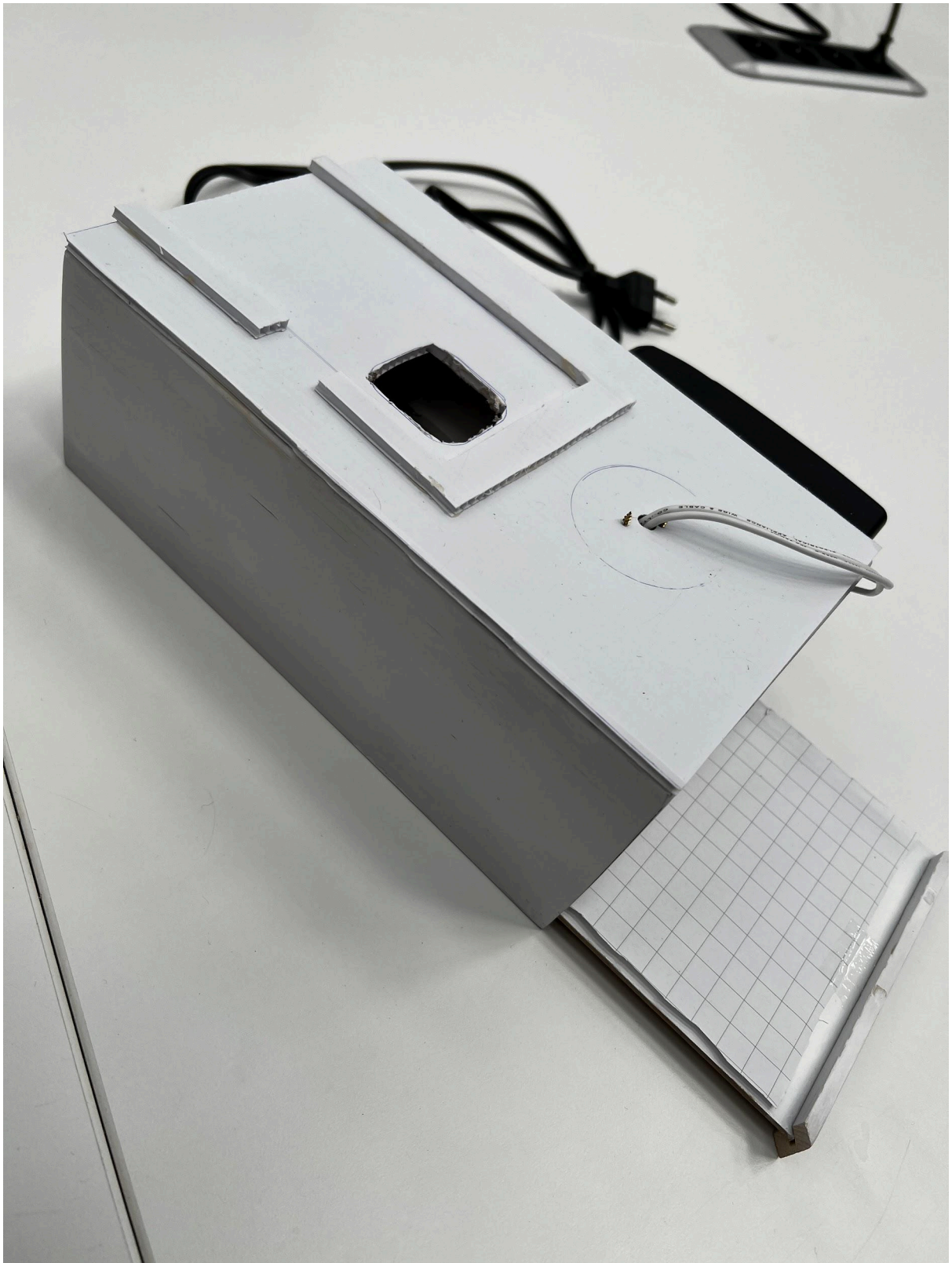
Dataset:

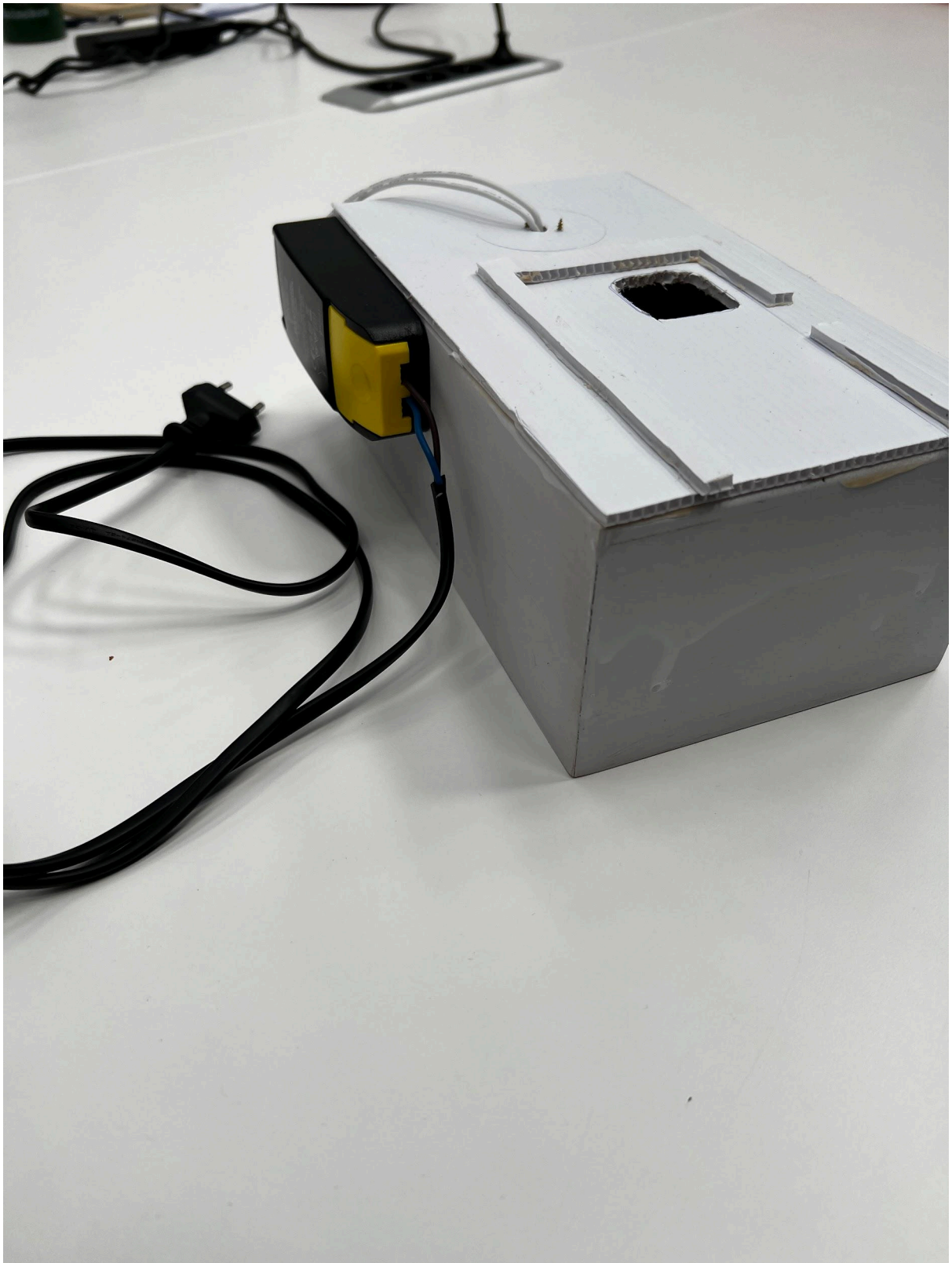
<https://drive.google.com/drive/folders/11HIUdIFV5KnFr8rLp2jXUUR9Wg9rxbO?usp=sharing>

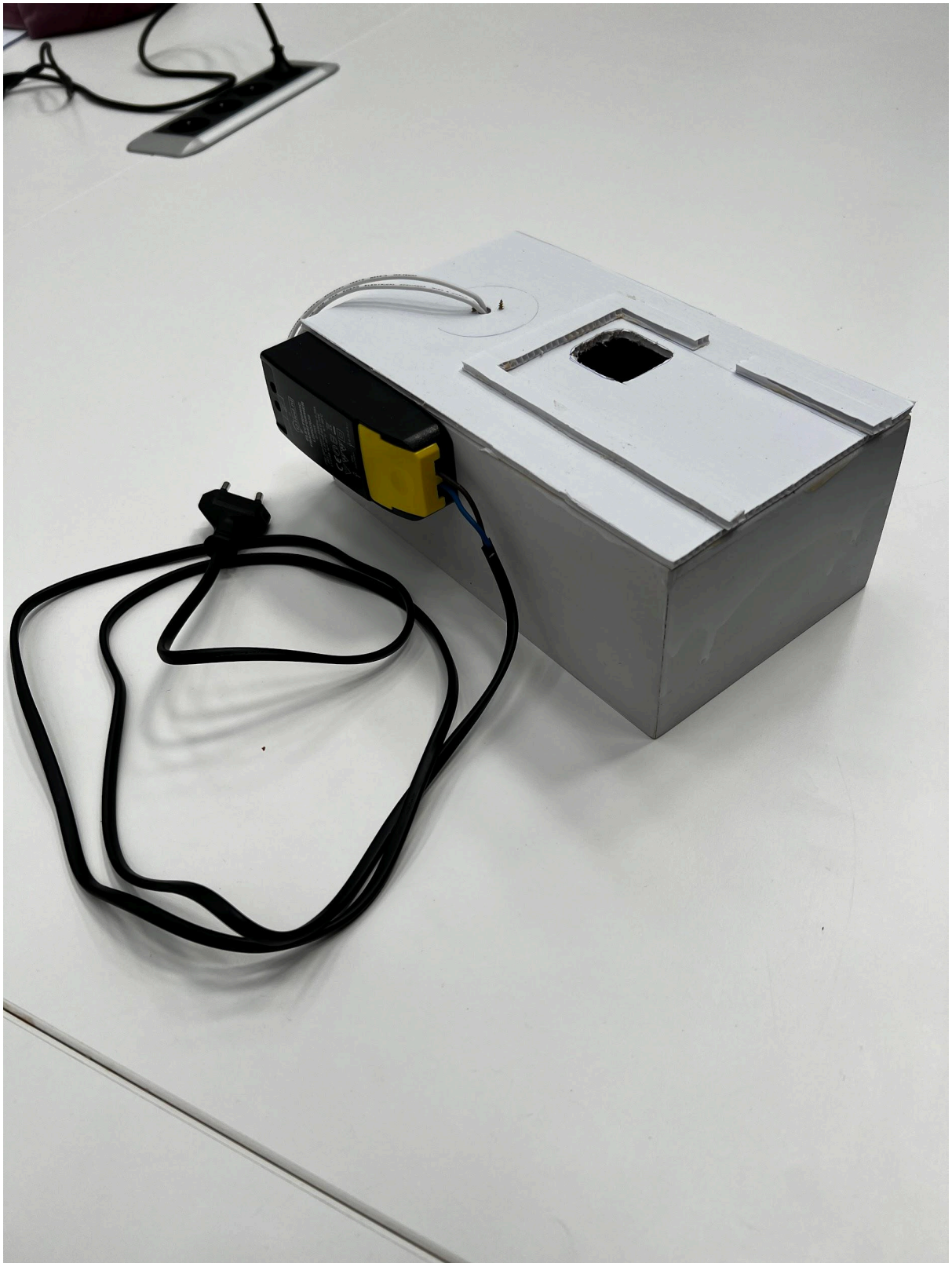
Appendix G - Light test setup











Appendix H - Light test dataset

Dataset:

https://drive.google.com/drive/folders/1RQjN-kAGb2qiAHHtXvUOLtOxjr6Ndwrc?usp=drive_link

Appendix I - Arduino code electronic prototype


```

#include <DHT20.h>
#include <Grove_Temperature_And_Humidity_Sensor.h>
#include <Wire.h>
#include <DHT.h>
#include "FS.h"
#include "SD.h"
#include "SPI.h"
#include "esp_camera.h"
#include "Adafruit_NeoPixel.h"
#include "esp_sleep.h"

#define CAMERA_MODEL_XIAO_ESP32S3 // Has PSRAM
#include "camera_pins.h"

#define DHTTYPE DHT20 // DHT 20
#define SD_CS_PIN 21 // SD card CS pin for ESP32-S3

#define PIN 2 // NeoPixel pin
#define NUMPIXELS 16 // Number of NeoPixels

DHT dht(DHTTYPE); // DHT10 DHT20 don't need to define Pin
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(NUMPIXELS, PIN, NEO_GRB +
NEO_KHZ800);

int sensorPin = A0;
int sensorValue = 0;
String deviceId;

unsigned long lastCaptureTime = 0; // Last shooting time
int imageCount = 1; // File Counter
bool camera_sign = false; // Check camera status
bool sd_sign = false; // Check SD status

// Function declarations
void writeFile(fs::FS &fs, const char *path, uint8_t *data, size_t len, bool append =
false);
void writeToCSV(float humidity, float temperature, int moisture);
String generateRandomID();
void photo_save(const char *fileName);
void lightUpLEDs();
void turnOffLEDs();
void goToDeepSleep();

```

```

void setup() {
  Serial.begin(115200);
  while (!Serial); // Wait for serial monitor to open
  pinMode(LED_BUILTIN, OUTPUT);
  Wire.begin();
  dht.begin();

  // Initialize NeoPixel
  pixels.setBrightness(255);
  pixels.begin();

  // Generate a random ID for the device
  deviceId = generateRandomID();

  // Initialize SD card
  if (!SD.begin(SD_CS_PIN)) {
    Serial.println("Card Mount Failed");
    return;
  }
  uint8_t cardType = SD.cardType();
  if (cardType == CARD_NONE) {
    Serial.println("No SD card attached");
    return;
  }

  Serial.print("SD Card Type: ");
  if (cardType == CARD_MMC) {
    Serial.println("MMC");
  } else if (cardType == CARD_SD) {
    Serial.println("SDSC");
  } else if (cardType == CARD_SDHC) {
    Serial.println("SDHC");
  } else {
    Serial.println("UNKNOWN");
  }

  sd_sign = true; // SD initialization check passes

  // Create and write header to CSV file if it doesn't exist
  if (!SD.exists("/sensor_data.csv")) {
    String header = "ID,Time (s),Humidity (%),Temperature (C),Moisture\n";
    writeFile(SD, "/sensor_data.csv", (uint8_t *)header.c_str(), header.length());
  }
}

```

```

// Camera configuration
camera_config_t config;
config.ledc_channel = LEDC_CHANNEL_0;
config.ledc_timer = LEDC_TIMER_0;
config.pin_d0 = Y2_GPIO_NUM;
config.pin_d1 = Y3_GPIO_NUM;
config.pin_d2 = Y4_GPIO_NUM;
config.pin_d3 = Y5_GPIO_NUM;
config.pin_d4 = Y6_GPIO_NUM;
config.pin_d5 = Y7_GPIO_NUM;
config.pin_d6 = Y8_GPIO_NUM;
config.pin_d7 = Y9_GPIO_NUM;
config.pin_xclk = XCLK_GPIO_NUM;
config.pin_pclk = PCLK_GPIO_NUM;
config.pin_vsync = VSYNC_GPIO_NUM;
config.pin_href = HREF_GPIO_NUM;
config.pin_sscb_sda = SIOD_GPIO_NUM;
config.pin_sscb_scl = SIOC_GPIO_NUM;
config.pin_pwdn = PWDN_GPIO_NUM;
config.pin_reset = RESET_GPIO_NUM;
config.xclk_freq_hz = 20000000;
config.frame_size = FRAMESIZE_UXGA;
config.pixel_format = PIXFORMAT_JPEG; // for streaming
config.grab_mode = CAMERA_GRAB_WHEN_EMPTY;
config.fb_location = CAMERA_FB_IN_PSRAM;
config.jpeg_quality = 12;
config.fb_count = 1;

// Camera initialization
esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
}

camera_sign = true; // Camera initialization check passes

Serial.println("Initialization done.");
}

void loop() {
    float temp_hum_val[2] = {0};
    // Read DHT20 temperature and humidity

```

```

if (!dht.readTempAndHumidity(temp_hum_val)) {
    Serial.print("DHT20 Humidity: ");
    Serial.print(temp_hum_val[0]);
    Serial.print(" %\t");
    Serial.print("Temperature: ");
    Serial.print(temp_hum_val[1]);
    Serial.println(" *C");
} else {
    Serial.println("Failed to get DHT20 temperature and humidity value.");
}

// Read moisture sensor
sensorValue = analogRead(sensorPin);
Serial.print("Moisture: ");
Serial.println(sensorValue);

// Write to CSV
writeToCSV(temp_hum_val[0], temp_hum_val[1], sensorValue);

// Capture photo
char fileName[37];
sprintf(fileName, "%s_image%d.jpg", deviceID.c_str(), imageCount);
photo_save(fileName);
Serial.println("Photo saved.");

// Increment image count
imageCount++;
digitalWrite(LED_BUILTIN, HIGH);
delay(1000);
digitalWrite(LED_BUILTIN, LOW);

// Go to deep sleep
goToDeepSleep();
}

void writeFile(fs::FS &fs, const char *path, uint8_t *data, size_t len, bool append) {
    Serial.printf("Writing file: %s\r\n", path);

    // Open file for writing (append if specified)
    File file;
    if (append) {
        file = fs.open(path, FILE_APPEND);
    } else {
        file = fs.open(path, FILE_WRITE);
    }

```

```

    }

    if (!file) {
        Serial.println("Failed to open file for writing");
        return;
    }
    if (file.write(data, len) == len) {
        Serial.println("File written");
    } else {
        Serial.println("Write failed");
    }
    file.close();
}

void writeToCSV(float humidity, float temperature, int moisture) {
    // Get current time
    unsigned long currentTime = millis();

    // Convert time to seconds
    unsigned long seconds = currentTime / 1000;

    // Create CSV line
    String dataLine = deviceId + ", " + String(seconds) + ", " + String(humidity) + ", " +
String(temperature) + ", " + String(moisture) + "\n";

    // Write to SD card in append mode
    writeFile(SD, "/sensor_data.csv", (uint8_t *)dataLine.c_str(), dataLine.length(),
true);
}

String generateRandomID() {
    String alphabet =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
    String ID = "";
    for (int i = 0; i < 8; ++i) {
        ID += alphabet[random(alphabet.length())];
    }
    return ID;
}

void photo_save(const char *fileName) {
    // Take a photo
    camera_fb_t *fb = esp_camera_fb_get();
    if (!fb) {

```

```

        Serial.println("Failed to get camera frame buffer");
        return;
    }
    // Save photo to file
    writeFile(SD, fileName, fb->buf, fb->len);

    // Release image buffer
    esp_camera_fb_return(fb);
}

void lightUpLEDs() {
    for (int i = 0; i < NUMPIXELS; i++) {
        // pixels.Color takes RGB values, from 0,0,0 up to 255,255,255
        pixels.setPixelColor(i, pixels.Color(255, 255, 255)); // Moderately bright white
        color
    }
    pixels.show(); // This sends the updated pixel color to the hardware
}

void turnOffLEDs() {
    for (int i = 0; i < NUMPIXELS; i++) {
        pixels.setPixelColor(i, pixels.Color(0, 0, 0)); // Turn off LED
    }
    pixels.show(); // Update the hardware to turn off LEDs
}

void goToDeepSleep() {

```


Appendix J - Slug count table

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Day	Slots (no mesh)	Open (mesh)	Holes (no mesh)	Slots (mesh)	Holes (mesh)	Control	Open (no mesh)	Slots (no mesh)	Open (mesh)	Holes (no mesh)	Slots (mesh)	Holes (mesh)	Control	Open (no mesh)
Friday 11:00	1	0 (no evidence)	0 (no evidence)	0 (no evidence)	0 (evidence)	0 (evidence)	3	0 (no evidence)	0 (evidence)	2	0 (no evidence)	2	0 (evidence)	2
Mon 10:00	4	0 (evidence)	0 (evidence)	0 (evidence)	0 (evidence)	0 (evidence)	1	2	0 (evidence)	0 (evidence)	2	1	1	1

Appendix K - Aruco cropping code

```

import cv2 as cv
import numpy as np

# Load the image file
image_path = 'D:\\Afstudeer documenten\\Afstudeer
documenten\\Images\\New Aruco\\image5.JPG' # Replace with
your image file path
# Replace with your image file path
frame = cv.imread(image_path)

# Check if the image was successfully loaded
if frame is None:
    print(f"Error: Unable to load image from path
{image_path}")
    exit()

# Convert the image to grayscale
gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)

# Load the predefined dictionary
dictionary =
cv.Aruco.getPredefinedDictionary(cv.Aruco.DICT_4X4_50)
parameters = cv.Aruco.DetectorParameters()
detector = cv.Aruco.ArucoDetector(dictionary, parameters)

# Detect the markers in the image
markerCorners, markerIds, rejectedCandidates =
detector.detectMarkers(gray)

# Check if 4 markers are detected
if markerIds is not None and len(markerIds) == 4:
    # Sort the markers based on their ids to get them in a
consistent order
    ids = markerIds.flatten()
    sorted_indices = np.argsort(ids)
    sorted_corners = [markerCorners[i] for i in sorted_indices]

    # Flatten the sorted_corners list and extract the points
    pts_src = np.array([corner[0] for marker in sorted_corners
for corner in marker], dtype='float32')

    # Define the coordinates for the desired output
width, height = 800, 600 # Define your output size

```

```
pts_dst = np.array([[0, 0], [width - 1, 0], [width - 1, height - 1], [0, height - 1]], dtype='float32')

# Compute the homography matrix
h_matrix, _ = cv.findHomography(pts_src, pts_dst)

# Warp the image using the homography matrix
warped_image = cv.warpPerspective(frame, h_matrix, (width, height))

# Display or save the result
cv.imshow('Warped Image', warped_image)
cv.waitKey(0)
cv.destroyAllWindows()
cv.imwrite('output_image.jpg', warped_image)
else:
    print("Could not detect exactly 4 corner markers.")
```

Appendix L - Datasets with image processing

No mesh:

https://drive.google.com/drive/folders/1v4W0Dt1VDq0PpESQU2y4fQ6GsuWWjqgH?usp=drive_link

Black mesh:

https://drive.google.com/drive/folders/1HBmsKUhZC6_ymivxu_on5J-pQj5yQVKK?usp=drive_link

White mesh:

https://drive.google.com/drive/folders/1HBmsKUhZC6_ymivxu_on5J-pQj5yQVKK?usp=drive_link

No mesh augmented:

https://drive.google.com/drive/folders/1CnUFxILwUhnZ6W8ITaVTrAam2BQVK9mB?usp=drive_link

Black mesh augmented:

https://drive.google.com/drive/folders/1XSX8y7c7V5ST0wQwgNU-heZ_AfW5uKOv?usp=drive_link

White mesh augmented:

https://drive.google.com/drive/folders/1QHRNPACwDIA0fyp-VmCinclUr1ta3wzS?usp=drive_link

Appendix M - Environmental values

ID	Coordinate	Time (s)	Humidity (%)	Temperature (C)	Moisture
150L	[60,19]	76	71.64	20.03	0
150R	[61,19]	112	70.48	19.64	1259
141R	[61,10]	160	70.24	19.72	1129
141L	[60,10]	196	71.54	19.54	141
131R	[61,0]	257	71.34	19.30	1198
131L	[60,0]	293	70.77	19.43	0
142R (end)	[120,11]	673	64.10	20.57	731
142L (end.1)	[0,11]	979	57.84	20.20	356
142L (end.2)	[1,11]	1040	56.72	20.49	0
142L (end.3)	[2,11]	1064	56.58	20.50	1287
142L (end.4)	[3,11]	1100	57.33	20.47	42
142L (end.5)	[4,11]	1136	57.36	20.38	1571

Appendix N - Image processing with Hough circle transform

```

import cv2
import numpy as np
import os

def crop_image_around_circle(image, circle, border=20):
    x, y, r = circle
    h, w = image.shape[:2]

    # Calculate the cropping boundaries with the border
    x_min = max(x - r - border, 0)
    x_max = min(x + r + border, w)
    y_min = max(y - r - border, 0)
    y_max = min(y + r + border, h)

    # Perform the cropping
    cropped_image = image[y_min:y_max, x_min:x_max]

    return cropped_image

def detect_circle(image):
    # Resize the image if it is too large
    max_dimension = 800
    scale_factor = min(max_dimension / image.shape[0],
max_dimension / image.shape[1], 1)
    if scale_factor < 1:
        image = cv2.resize(image, (int(image.shape[1] *
scale_factor), int(image.shape[0] * scale_factor)))

    # Convert image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Apply GaussianBlur to reduce noise
    blurred = cv2.GaussianBlur(gray, (9, 9), 2)

    # Apply Hough Circle Transform
    circles = cv2.HoughCircles(blurred, cv2.HOUGH_GRADIENT,
dp=1, minDist=50,
                                param1=100, param2=30,
minRadius=10, maxRadius=0)

    if circles is not None:
        circles = np.round(circles[0, :]).astype("int")
        if len(circles) > 0:

```

```

        return circles[0] # Return the first detected
circle

    return None

def process_image(image_path, output_path):
    # Load the image
    image = cv2.imread(image_path)

    # Convert to grayscale
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Apply histogram equalization to enhance contrast
    equalized_image = cv2.equalizeHist(gray_image)

    # Apply Gaussian blur to smooth the image
    blurred_image = cv2.GaussianBlur(equalized_image, (5, 5),
0)

    # Apply adaptive thresholding to create a binary image
    thresh_image = cv2.adaptiveThreshold(blurred_image, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY_INV,
11, 2)

    # Use morphological operations to remove small noise and
fill gaps
    kernel = np.ones((3, 3), np.uint8)
    morph_image = cv2.morphologyEx(thresh_image,
cv2.MORPH_CLOSE, kernel, iterations=2)

    # Detect circle on the original image
    circle = detect_circle(image)

    # Crop image around the detected circle with a border if
circle is detected
    if circle is not None:
        processed_image = crop_image_around_circle(morph_image,
circle, border=20)
    else:
        processed_image = morph_image # If no circle is
detected, use the morph_image

    # Save the final processed image

```



```

cv2.imwrite(output_path, processed_image)

def process_images_in_folder(input_folder, output_folder):
    # Create output folder if it doesn't exist
    os.makedirs(output_folder, exist_ok=True)

    # Loop through all files in the input folder
    for filename in os.listdir(input_folder):
        if filename.lower().endswith(('.png', '.jpg', '.jpeg',
        '.bmp', '.tiff')):
            input_path = os.path.join(input_folder, filename)
            output_path = os.path.join(output_folder,
            f"processed_{filename}")

            # Process the image
            process_image(input_path, output_path)
            print(f"Processed {filename}")

# Example usage
input_folder = "D:\\Afstudeer documenten\\Afstudeer
documenten\\Datasets\\Circular slugtrap" # Replace with your
input folder path
output_folder = "D:\\Afstudeer documenten\\Afstudeer
documenten\\Datasets\\Circular slugtrap augmented with Hough"
# Replace with your output folder path
process_images_in_folder(input_folder, output_folder)

```

Appendix O - Hough dataset

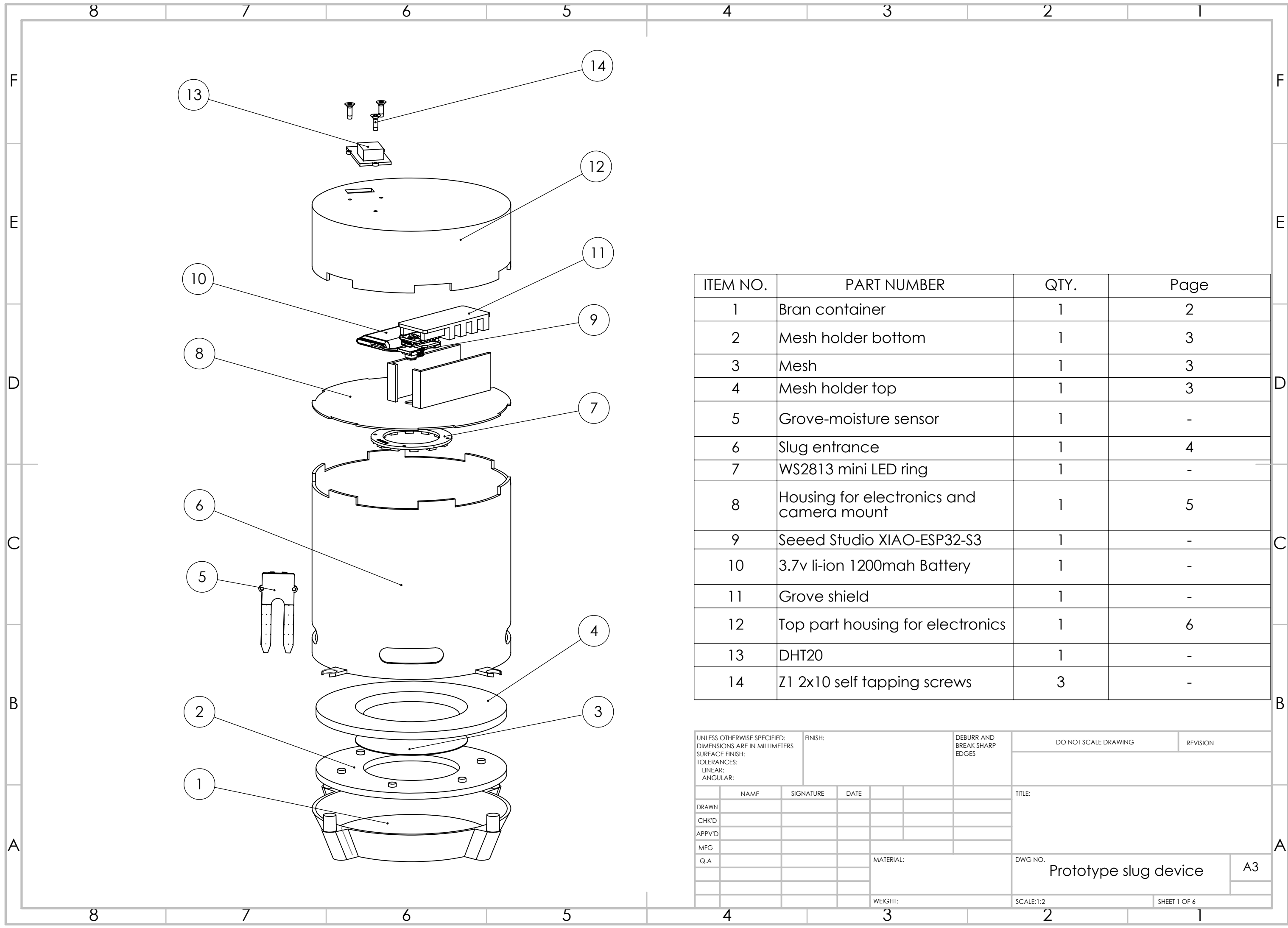
Base dataset:

https://drive.google.com/drive/folders/17EIXY771hFRsk3Clt9ebt5TTTzEBy9dZ?usp=drive_link

Augmented dataset:

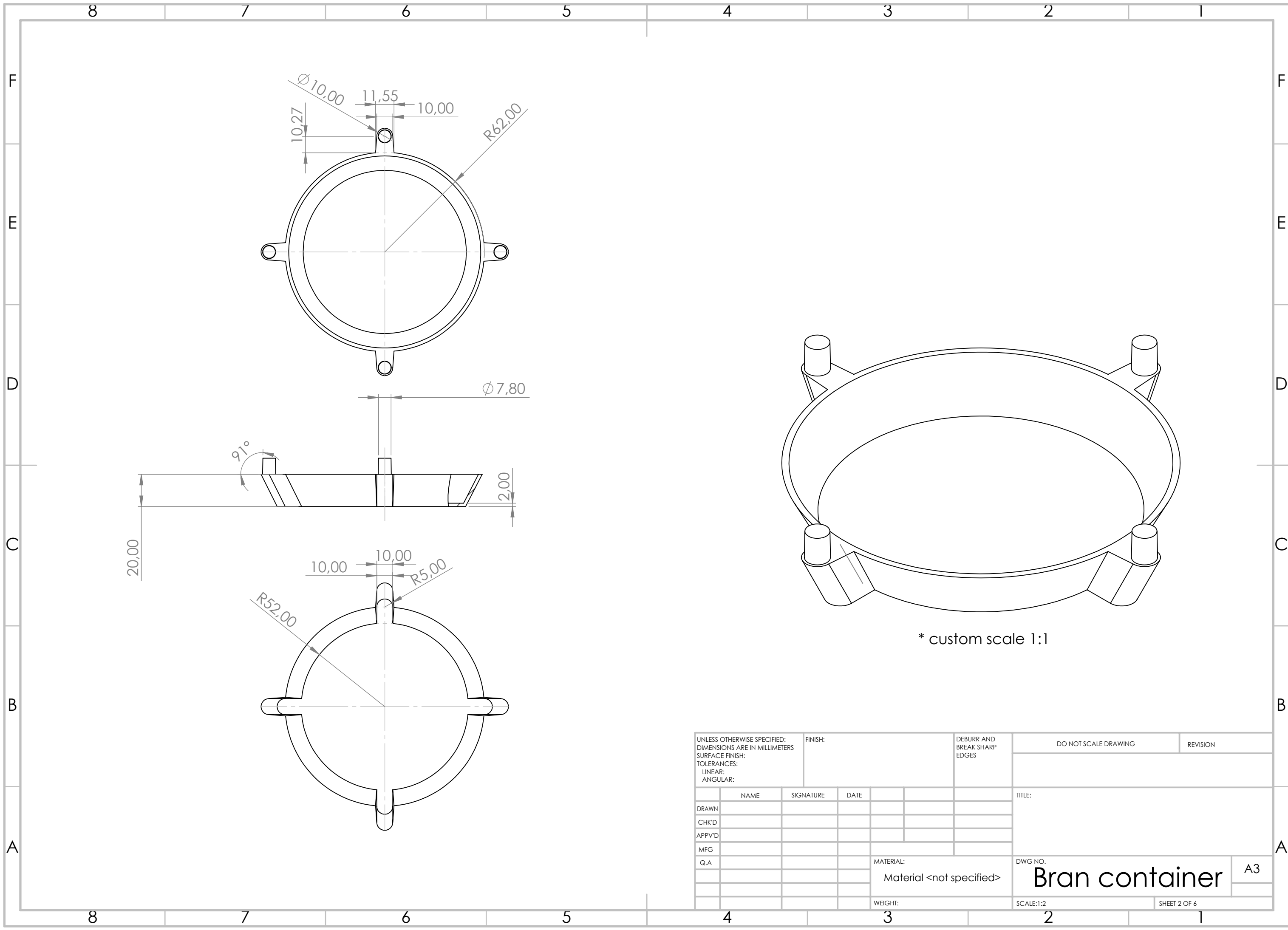
https://drive.google.com/drive/folders/1YqBjZM0Cg0x723u_0ttlNfhzROhLtYOB?usp=drive_link

Appendix P - Product schematics

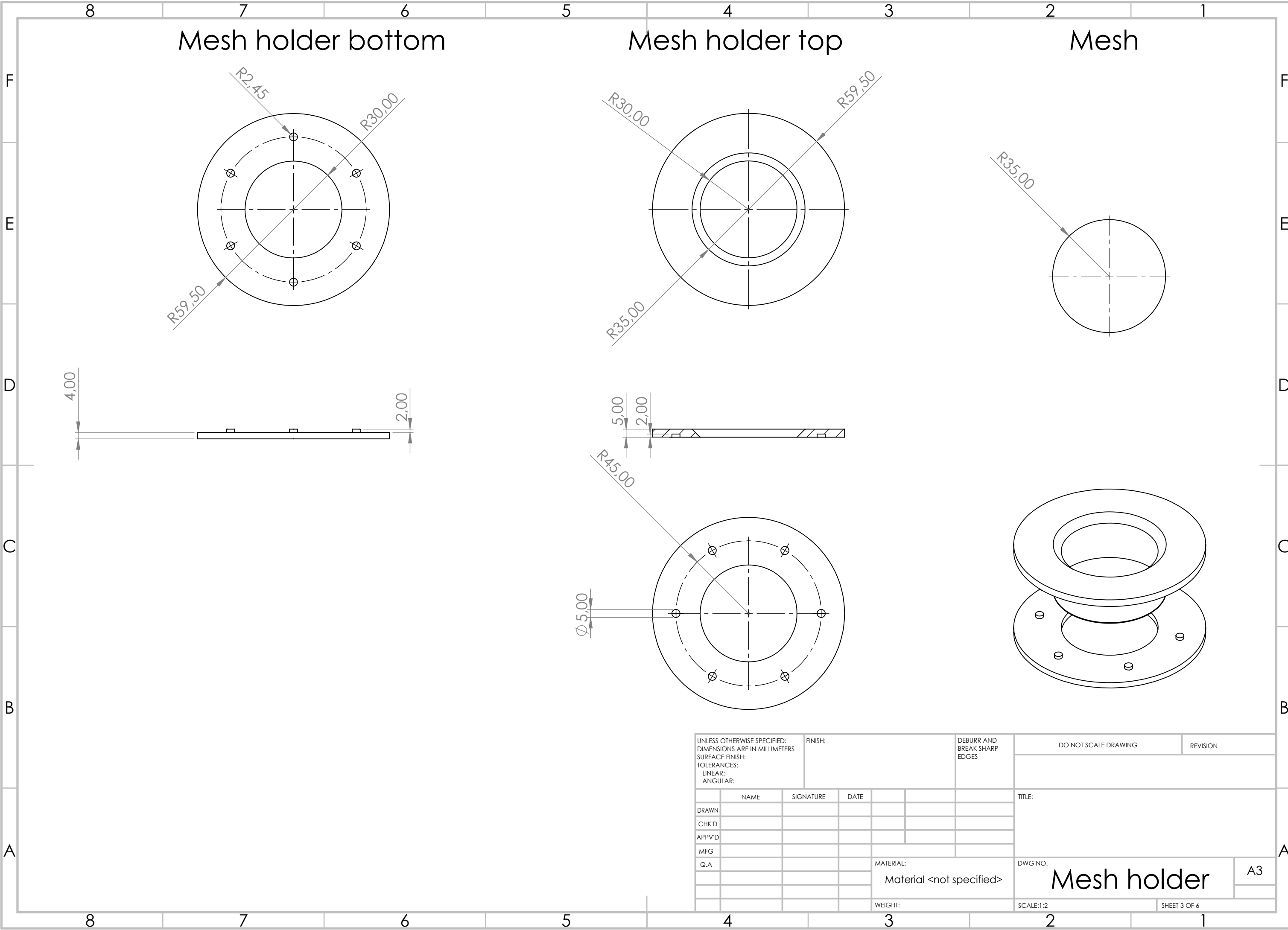


ITEM NO.	PART NUMBER	QTY.	Page
1	Bran container	1	2
2	Mesh holder bottom	1	3
3	Mesh	1	3
4	Mesh holder top	1	3
5	Grove-moisture sensor	1	-
6	Slug entrance	1	4
7	WS2813 mini LED ring	1	-
8	Housing for electronics and camera mount	1	5
9	Seeed Studio XIAO-ESP32-S3	1	-
10	3.7v li-ion 1200mah Battery	1	-
11	Grove shield	1	-
12	Top part housing for electronics	1	6
13	DHT20	1	-
14	Z1 2x10 self tapping screws	3	-

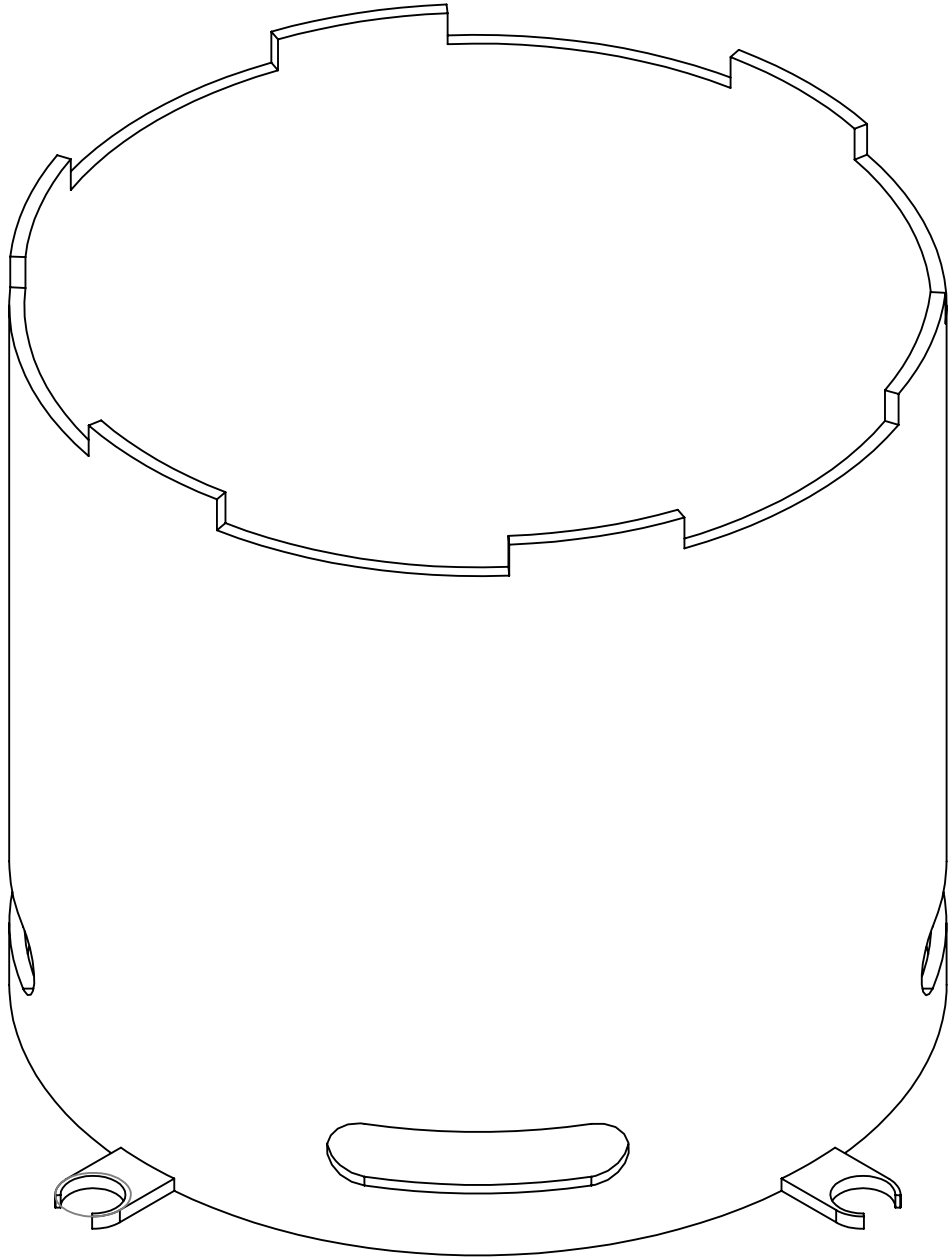
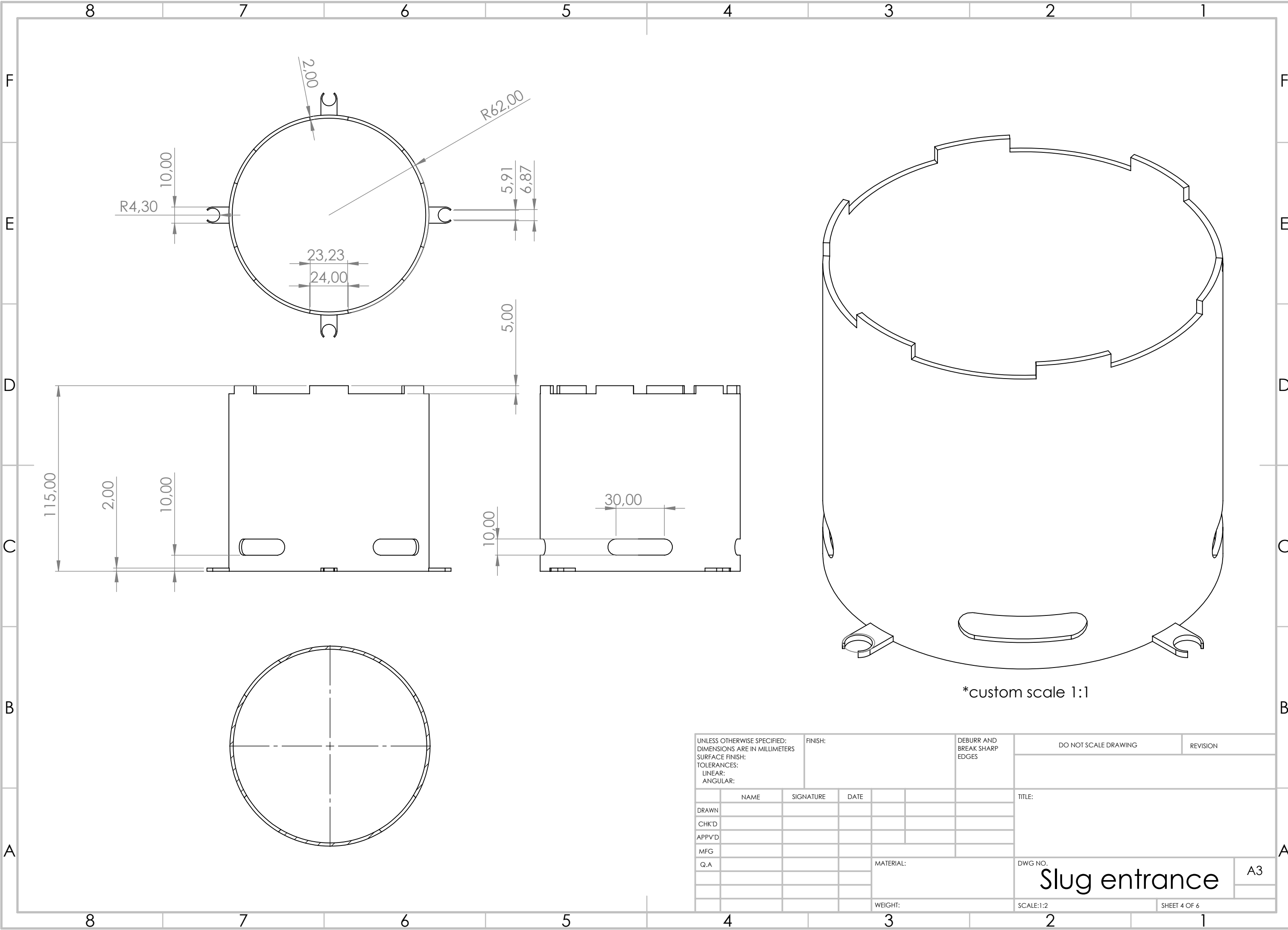
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING		REVISION	
DRAWN		NAME		SIGNATURE		DATE		TITLE:	
CHK'D									
APPV'D									
MFG									
Q.A						MATERIAL:		DWG NO.	
								Prototype slug device	
								A3	
						WEIGHT:		SCALE:1:2	
								SHEET 1 OF 6	



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:				FINISH:		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING		REVISION
	NAME	SIGNATURE	DATE				TITLE:		
DRAWN									
CHK'D									
APPV'D									
MFG									
Q.A							MATERIAL: Material <not specified>		DWG NO.
									A3
							WEIGHT:		SCALE:1:2
									SHEET 2 OF 6



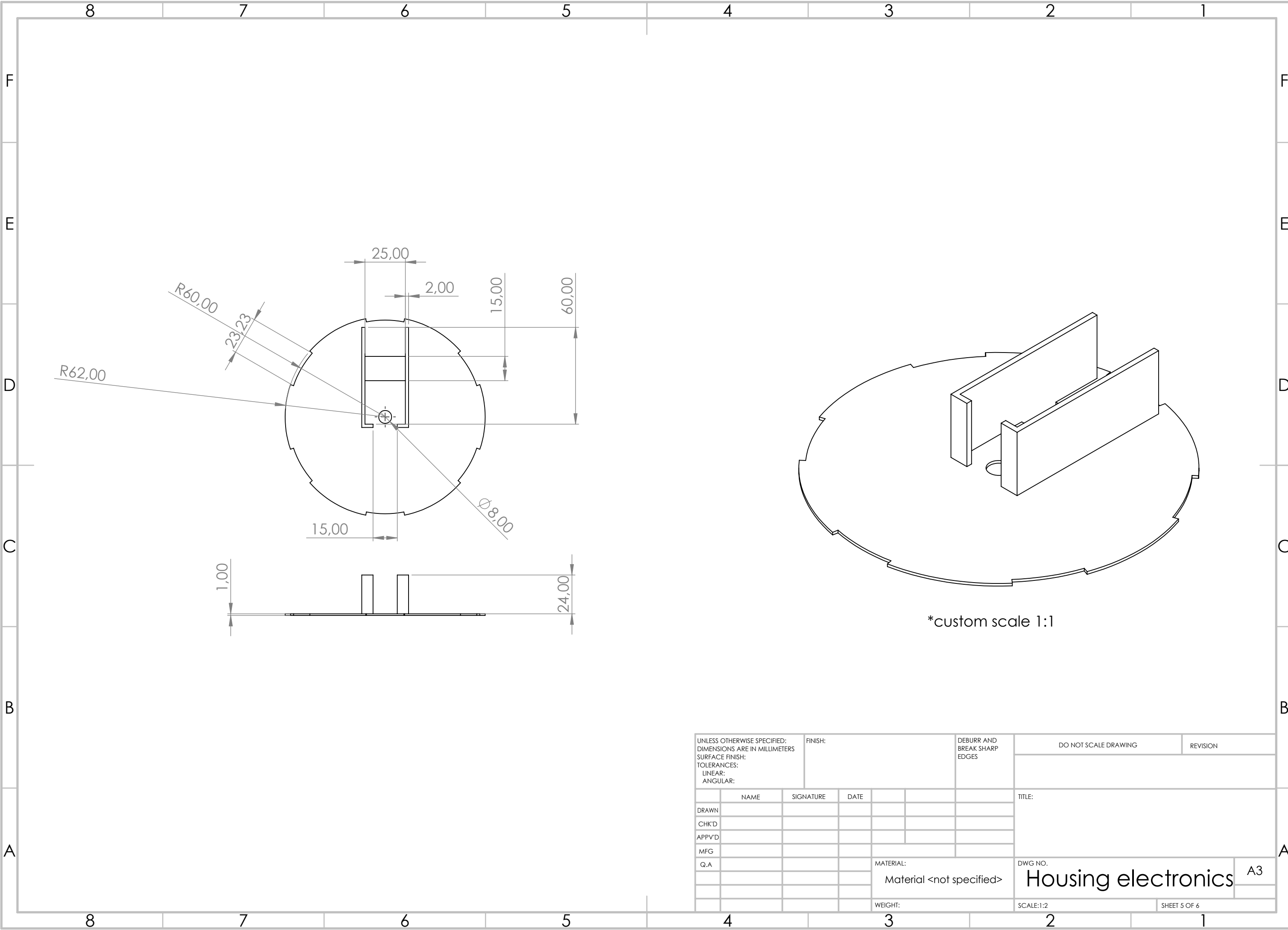
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:					FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING		REVISION		
	NAME		SIGNATURE		DATE				TITLE:				
DRAWN													
CHK'D													
APPV'D													
MFG													
Q.A					MATERIAL: Material <not specified>			DWG NO. Mesh holder				A3	
					WEIGHT:			SCALE:1:2				SHEET 3 OF 6	



*custom scale 1:1

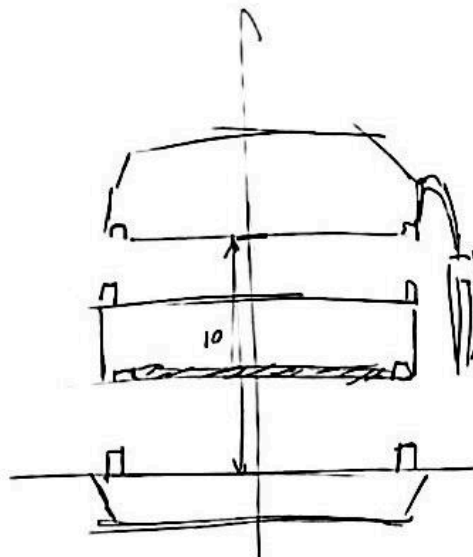
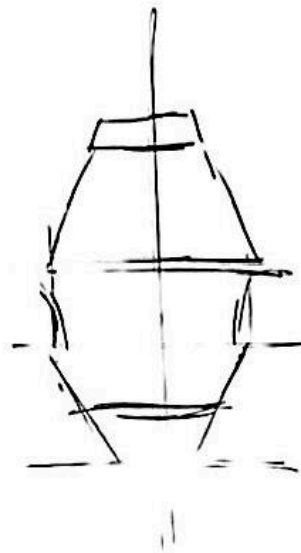
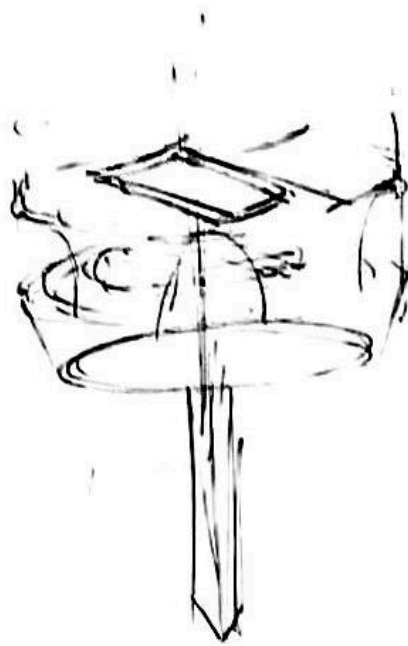
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:						FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING		REVISION	
		NAME		SIGNATURE		DATE						TITLE:	
DRAWN													
CHK'D													
APP'V'D													
MFG													
Q.A								MATERIAL:		DWG NO. Slug entrance		A3	
								WEIGHT:		SCALE:1:2		SHEET 4 OF 6	

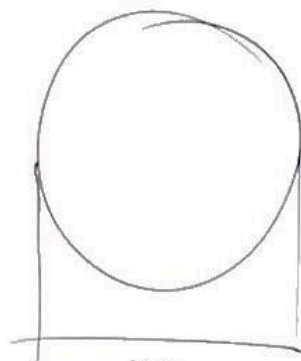
Slug entrance



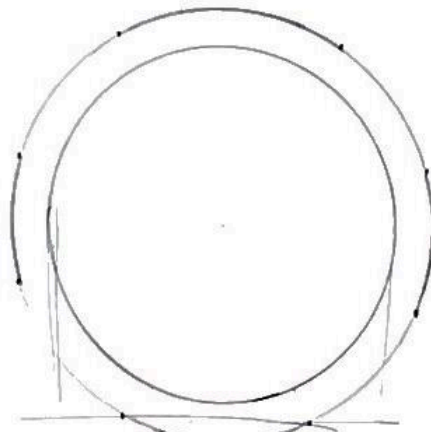
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:				FINISH:		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING		REVISION
	NAME	SIGNATURE	DATE				TITLE:		
DRAWN									
CHK'D									
APPV'D									
MFG									
Q.A							MATERIAL: Material <not specified>		DWG NO.
									A3
							WEIGHT:		Housing electronics
							SCALE:1:2		SHEET 5 OF 6

Appendix Q - Design sketches





7,2 cm
72n.



7,2



22-8

final stage
design

List:

light source

DHT

moisture

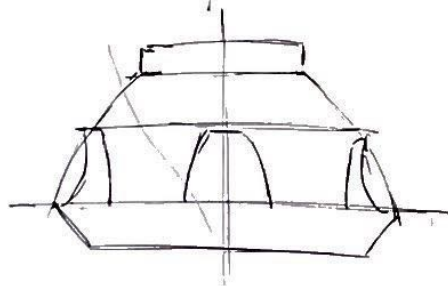
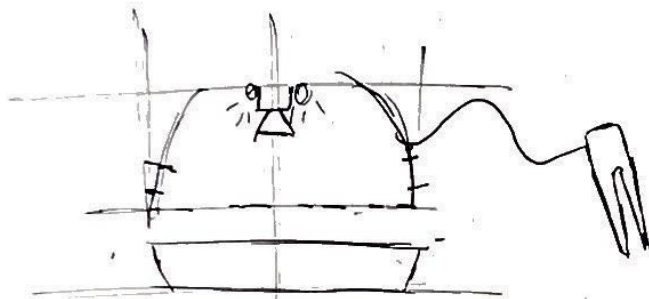
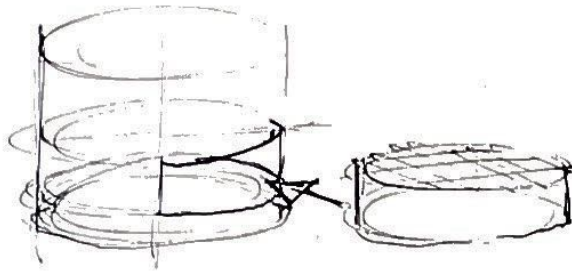
camera

mesh

bran

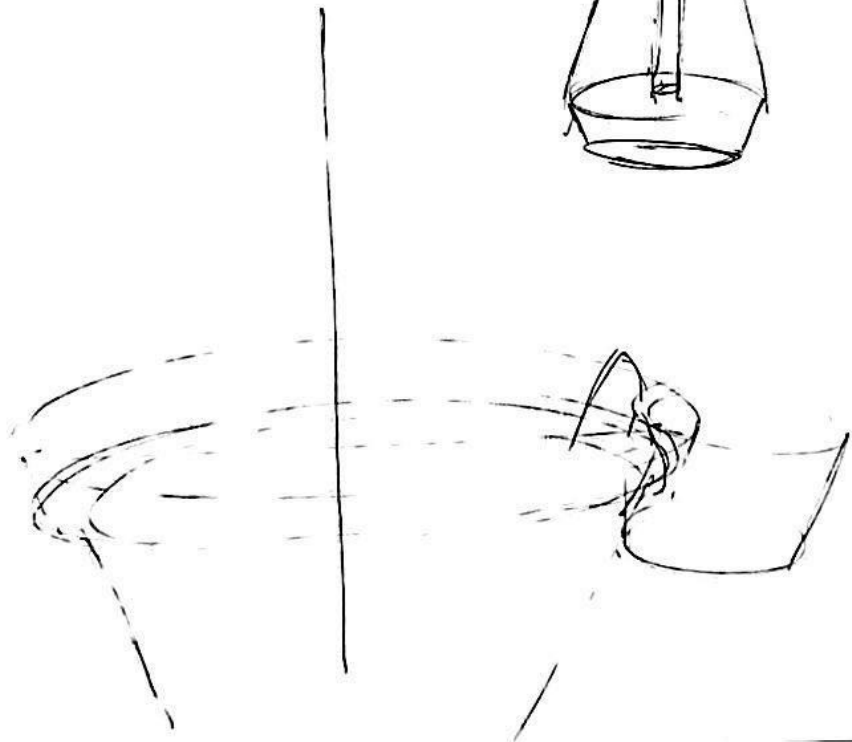
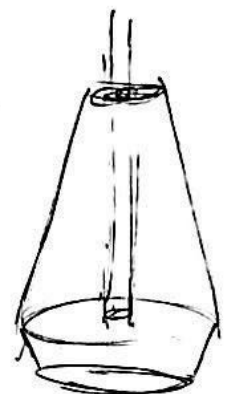
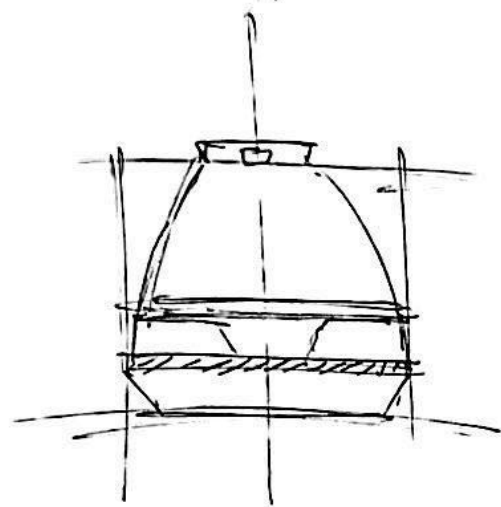
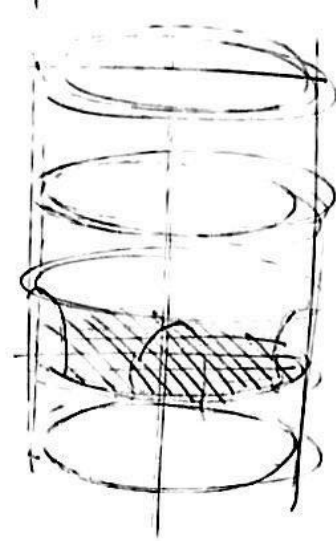
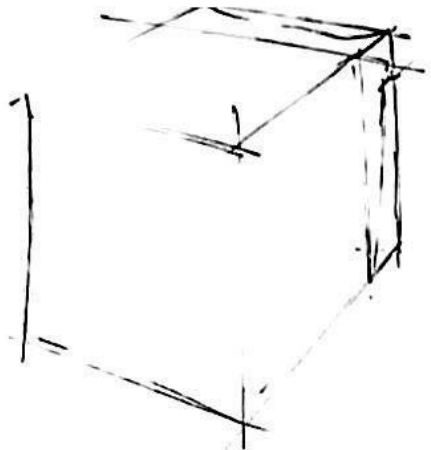
4 holes or

mo.



things:

bran replace, access slugs, take photo, access SD.



Appendix R - Machine learning datasets

Small dataset:

<https://drive.google.com/drive/folders/1P-OKDYjgZbbY1q7W8p0tWC-sP-dD4E7J?usp=sharing>

Labels small dataset:

<https://drive.google.com/file/d/1iz6hiDF7lr5A3qahwStMBcC718LAPdWh/view?usp=sharing>

Full dataset:

https://drive.google.com/drive/folders/1-sLMDJ45svZ-uY_eULruB2cqz5tPzAk2?usp=sharing

Labels full dataset:

<https://drive.google.com/file/d/1LtrPevKzlzyqt4DzqJNAQ-ZpEgR4Shic/view?usp=sharing>

Appendix S - Adapted code from Kaggle

```

import os
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g.
pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
import shutil
from sklearn.model_selection import train_test_split
import tensorflow as tf
from sklearn.model_selection import train_test_split

# Ensure the checkpoints directory exists
checkpoint_dir = 'checkpoints'
if not os.path.exists(checkpoint_dir):
    os.makedirs(checkpoint_dir)

# %%
# getting the labels corresponding to the image
label_df = pd.read_csv('C:/Users/baart/University/Afstudeer
documenten/Machine Learning/Labels dataset p2 test -
Blad1.csv')
label_df.columns = ['index', 'slug count']
print(label_df.head())
# %%
# loading the images in vector format
img = np.load('C:/Users/baart/University/Afstudeer
documenten/Machine Learning/images.npy')
print(img.shape)
# %%
labels = np.array(label_df['slug count'])
print(labels)
# %%
# setting features and target value

x_train, x_test, y_train, y_test = train_test_split(img,
labels, test_size=0.1)
print(x_train.shape[0])
print(x_test.shape[0])
# %%
"""
x_train, x_test = x_train / 255.0, x_test / 255.0
"""
# %%
# create model

```

```

model = tf.keras.Sequential([
    tf.keras.layers.InputLayer(input_shape=(1200, 1600, 1)), #
    Input layer
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    tf.keras.layers.MaxPool2D(2, 2),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPool2D(2, 2),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)
])

model.compile(loss=tf.keras.losses.Huber(),
optimizer=tf.keras.optimizers.Adam(), metrics=['mae'])
model.summary()

# %%
# add a learning rate monitor to get the lr with smoothest
prediction

lr_monitor = tf.keras.callbacks.LearningRateScheduler(
    lambda epochs: 1e-8 * 10 ** (epochs / 20))

# Adding EarlyStopping callback
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss', # Monitor the validation loss
    patience=10, # Number of epochs with no improvement after
which training will be stopped
    restore_best_weights=True # Restore model weights from the
epoch with the best validation loss
)

# Adding ModelCheckpoint callback for saving the best model
and checkpoints every 5 epochs
checkpoint_best = tf.keras.callbacks.ModelCheckpoint(
    os.path.join(checkpoint_dir, 'best_model.keras'), # Save
in the "checkpoints" folder
    monitor='val_loss', # Monitor the validation loss
    save_best_only=True, # Save only the model with the best
validation loss
    verbose=1 # Print a message when the model is saved
)

```



```

checkpoint_interval = tf.keras.callbacks.ModelCheckpoint(
    os.path.join(checkpoint_dir,
'model_epoch_{epoch:02d}.keras'), # Save in the "checkpoints"
folder
    save_freq=5 * x_train.shape[0] // 32, # Save every 5
epochs; adjust batch size
    verbose=1 # Print a message when the model is saved
)

# %%
# train the model

history = model.fit(x_train, y_train, validation_data=(x_test,
y_test), epochs=50, batch_size=32,
                    callbacks=[lr_monitor, early_stopping,
checkpoint_best, checkpoint_interval])

# %%
# plot mae
plt.semilogx(history.history['lr'], history.history['loss'])
plt.axis([np.min(history.history['lr']),
np.max(history.history['lr']),
np.min(history.history['loss']), 15])
plt.show()

# %%
np.max(history.history['lr'])

# %%
# change the learning rate to 1e-5 and re-run the model

model.compile(loss=tf.keras.losses.MeanSquaredError(),
optimizer=tf.keras.optimizers.Adam(lr=1e-6), metrics=['mae'])
model.summary()

# %%
# train the model with early stopping and checkpoints

history = model.fit(x_train, y_train, validation_data=(x_test,
y_test), epochs=100, batch_size=32,
                    callbacks=[early_stopping, checkpoint_best,
checkpoint_interval])

```

```

# %%
# plot mae
plt.plot(history.history['mae'])
plt.plot(history.history['val_mae'])
plt.legend(['mae', 'val_mae'])
plt.ylim(1, 4)
plt.xlim(0, 50)

plt.xticks(np.arange(0, 50, 5))

plt.xlabel('epochs')
plt.ylabel('mean absolute error')
plt.title('Mae in every epoch')
plt.show()

# %%
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import ImageGrid

# set figure size

fig = plt.figure(figsize=(15, 15))
grid = ImageGrid(
    fig, 111,
    nrows_ncols=(2, 2),
    axes_pad=0.5
)

for x in range(0, 4):
    grid[x].set_title('Number of people => ' + str(labels[x]))
    grid[x].imshow(img[x])

```

Appendix T - Final code for training

```

#%%-----
import os

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

from sklearn.model_selection import train_test_split

from tensorflow.keras.models import
Sequential, Model, Sequential
from tensorflow.keras.layers import
Dropout, Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras import regularizers
from tensorflow.keras.preprocessing.image import
ImageDataGenerator
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Input, Conv2D,
MaxPooling2D, Conv2DTranspose, concatenate,
GlobalAveragePooling2D, Dense
from tensorflow.keras.layers import Conv2D, MaxPooling2D,
Dropout, Flatten, Dense, BatchNormalization
from tensorflow.keras.callbacks import
ReduceLROnPlateau, EarlyStopping

#%%function for data augmentation, use once before everything
def image_augmentation():
    # 1. Load your image data
    images = np.load("images_newest.npy") # Shape (482, 64,
64, 1)

    # 2. Create an ImageDataGenerator for augmentation
    datagen = ImageDataGenerator(
        rotation_range=30, # Rotate images by up to 30
degrees
        width_shift_range=0.2, # Shift the image width by up
to 20% of the width
        height_shift_range=0.2, # Shift the image height by
up to 20% of the height
        zoom_range=0.2, # Zoom in/out by up to 20%
        horizontal_flip=True, # Randomly flip images
horizontally

```

```

        vertical_flip=False          # Set vertical_flip=True if
applicable
    )

    # 3. Fit the data generator (optional)
    datagen.fit(images)

    # 4. Augment the images in batches and collect all
augmented images
    augmented_images = [] # List to hold augmented images

    # Total number of iterations needed to cover all images
once
    steps_per_epoch = len(images) // 32 + 1

    # Loop through the batches of augmented images
    for i in range(steps_per_epoch):
        augmented_batch = next(datagen.flow(images,
batch_size=32, shuffle=False))
        augmented_images.append(augmented_batch)

    # Concatenate all augmented batches into a single numpy
array
    augmented_images = np.concatenate(augmented_images, axis=0)

    # Save the augmented dataset to a .npy file
    np.save('augmented_images.npy', augmented_images)

#%% get data
def get_data():

    # 1. Load the image data and label data
    images = np.load("images_newest.npy") # Assuming shape is
(482, 64, 64, 1)

    # Load the labels from the CSV file
    df = pd.read_csv('Label full compressed dataset -
Blad1.csv')
    labels = df.to_numpy()[ :, 1].reshape(-1, 1) # Extract
second column and reshape

    # 2. Convert labels to one-hot encoding for multi-class
classification
    num_classes = 4

```

```

    labels_one_hot = to_categorical(labels,
num_classes=num_classes)

    # 3. Perform train-test split
    x_train, x_test, y_train, y_test = train_test_split(images,
labels_one_hot, test_size=0.2, random_state=42)

    return x_train, y_train, x_test, y_test
%% Define a simple sequential model
def cnn_model(input_shape=(64, 64, 1), num_classes=4):
    model = Sequential()

    # First block
    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
padding='same', input_shape=input_shape))
    model.add(Dropout(0.5))

    model.add(BatchNormalization())
    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
padding='same'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.5))

    # Second block
    #model.add(Conv2D(64, kernel_size=(3, 3),
activation='relu', padding='same'))
    #model.add(BatchNormalization())
    #model.add(Conv2D(64, kernel_size=(3, 3),
activation='relu', padding='same'))
    #model.add(MaxPooling2D(pool_size=(2, 2)))
    #model.add(Dropout(0.3))

    # Flatten and Dense layers
    model.add(Flatten())
    model.add(Dense(256, activation='relu'))
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))

    model.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])

    return model

```

```

def unet_model():

    input_shape = (64, 64, 1)
    num_classes = 4

    inputs = Input(shape=input_shape)

    # Encoder
    conv1 = Conv2D(64, (3, 3), activation='relu',
padding='same')(inputs)
    conv1 = Conv2D(64, (3, 3), activation='relu',
padding='same')(conv1)
    pool1 = MaxPooling2D((2, 2))(conv1)

    conv2 = Conv2D(128, (3, 3), activation='relu',
padding='same')(pool1)
    conv2 = Conv2D(128, (3, 3), activation='relu',
padding='same')(conv2)
    pool2 = MaxPooling2D((2, 2))(conv2)

    # Bottleneck
    conv3 = Conv2D(256, (3, 3), activation='relu',
padding='same')(pool2)
    conv3 = Conv2D(256, (3, 3), activation='relu',
padding='same')(conv3)

    # Decoder
    up4 = Conv2DTranspose(128, (2, 2), strides=(2, 2),
padding='same')(conv3)
    up4 = concatenate([up4, conv2])
    conv4 = Conv2D(128, (3, 3), activation='relu',
padding='same')(up4)
    conv4 = Conv2D(128, (3, 3), activation='relu',
padding='same')(conv4)

    up5 = Conv2DTranspose(64, (2, 2), strides=(2, 2),
padding='same')(conv4)
    up5 = concatenate([up5, conv1])
    conv5 = Conv2D(64, (3, 3), activation='relu',
padding='same')(up5)
    conv5 = Conv2D(64, (3, 3), activation='relu',
padding='same')(conv5)

    # Classification head

```



```

gap = GlobalAveragePooling2D()(conv5)
output = Dense(num_classes, activation='softmax')(gap)

model = Model(inputs=inputs, outputs=output)

model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])

    return model

#%% plot
def plot_loss_accuracy(history):

    # Plot training & validation accuracy values
    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend(['Train', 'Validation'])
    plt.grid(True)

    # Plot training & validation loss values
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend(['Train', 'Validation'])
    plt.grid(True)

    plt.tight_layout()
    plt.show()

#%% main -----

batch_size = 128
no_epochs = 100
verbosity = 1

```

```
# data
x_train, y_train, x_test, y_test = get_data()

# model
model = cnn_model()

#
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
patience=3, min_lr=1e-6)
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)

# train
history = model.fit(x_train, y_train,
                    validation_data=(x_test, y_test),
                    batch_size=batch_size,
                    epochs=no_epochs,
                    verbose=verbosity,
                    callbacks=[early_stopping, reduce_lr])

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(x_test, y_test,
verbose=verbosity)
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")

# plot
plot_loss_accuracy(history)

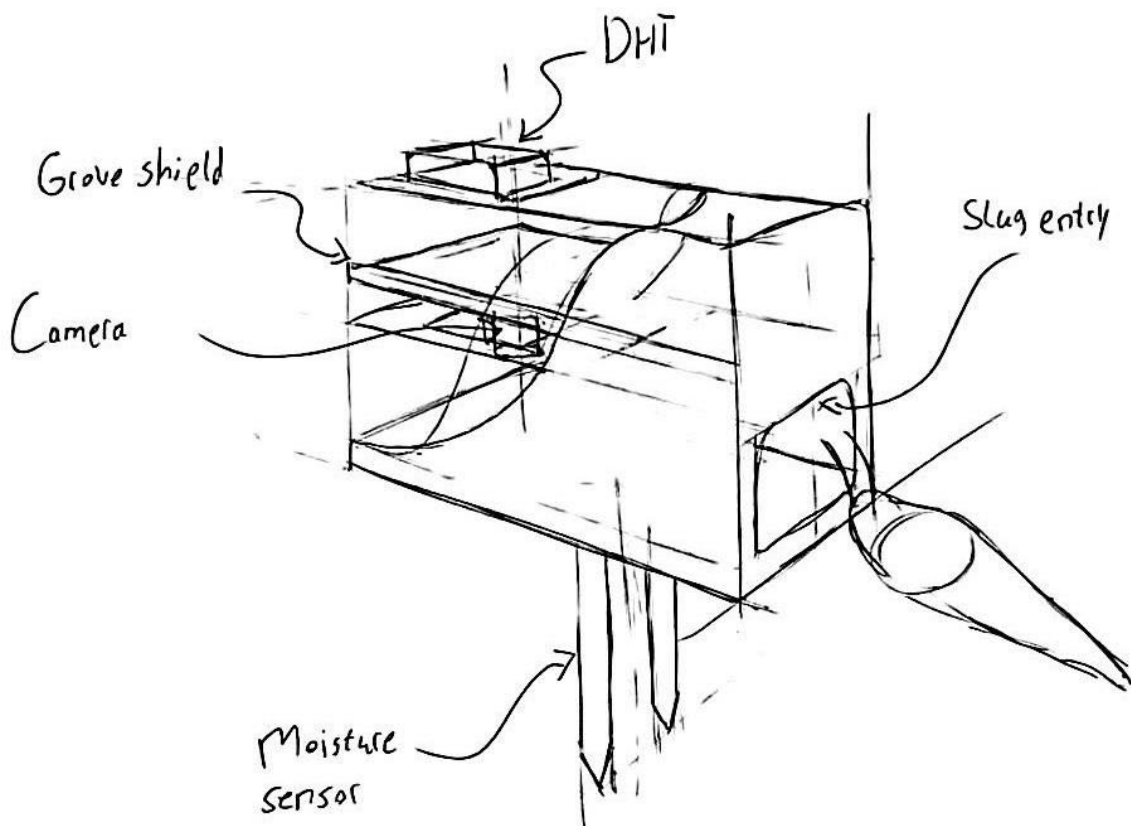
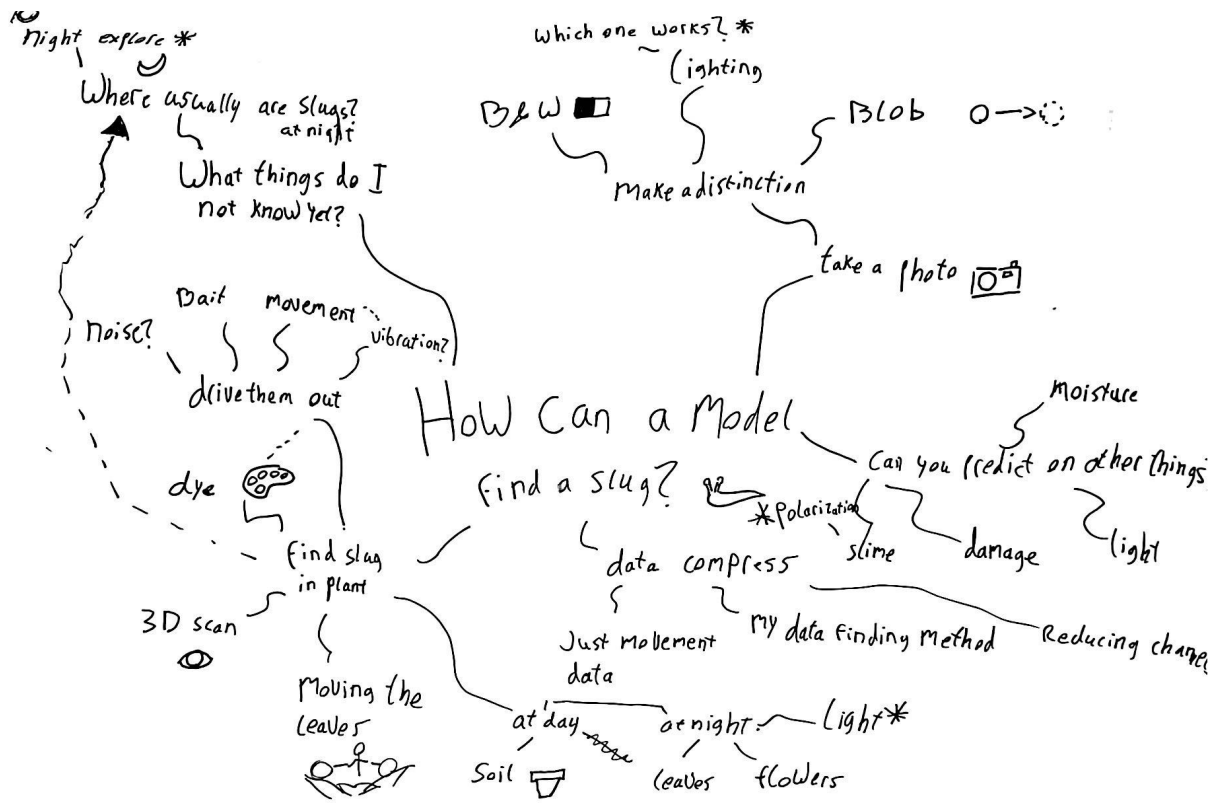
# save the model
model.save('slug_model.keras')
```

Appendix U - Results from fieldtesting

Results folder:

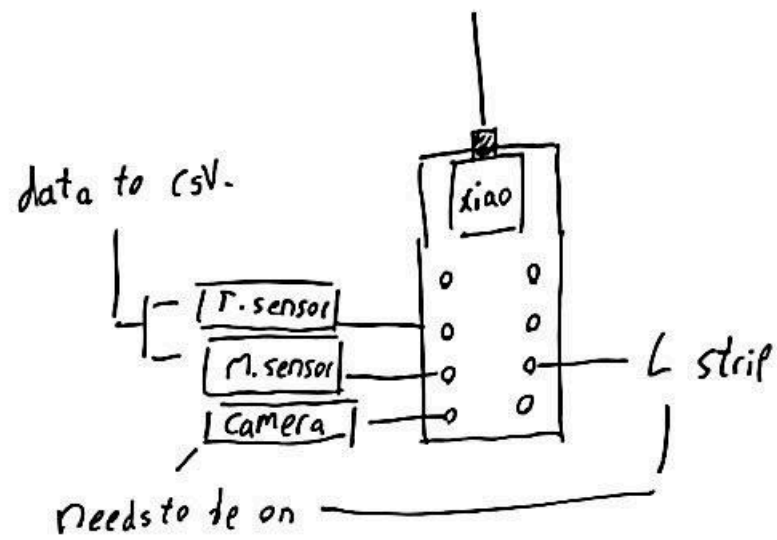
<https://drive.google.com/drive/folders/1vnZFcf3hsVARInu4J1-5pbnIGQs8pO87?usp=sharing>

Appendix V - Sketches from notebook



Set-up Pt.

244



Basic test

Moddable
PT is
needed!

1. build product
2. take photos \rightarrow train small model:
Just to count slugs!
3. deploy, make observations on slug behavior,
maybe mod the camera's to continuously
film
4. use these tests to make improvements.



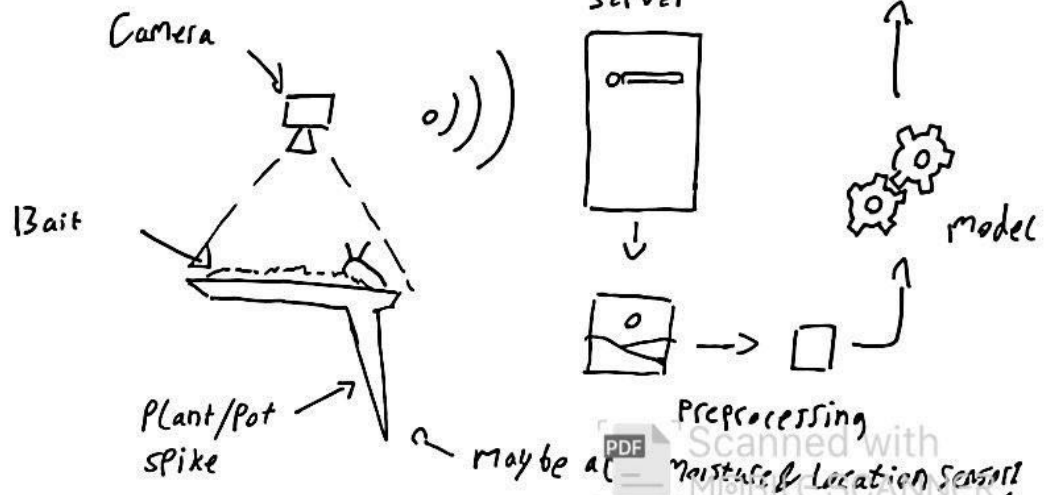
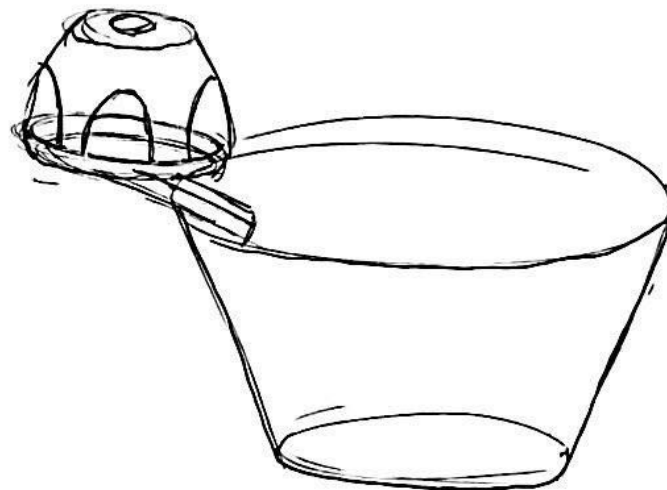
maybe a joint to ensure good
placing?
List:

idea

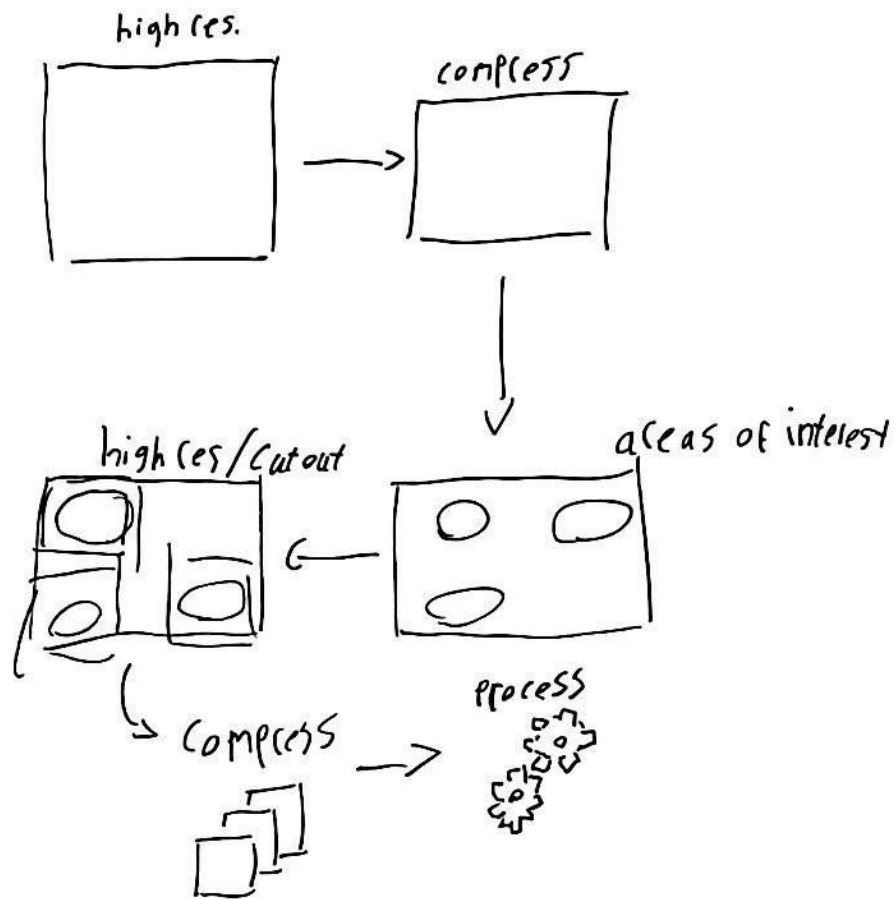
AS72626channel

Bme280
|
humidity
|
CO₂

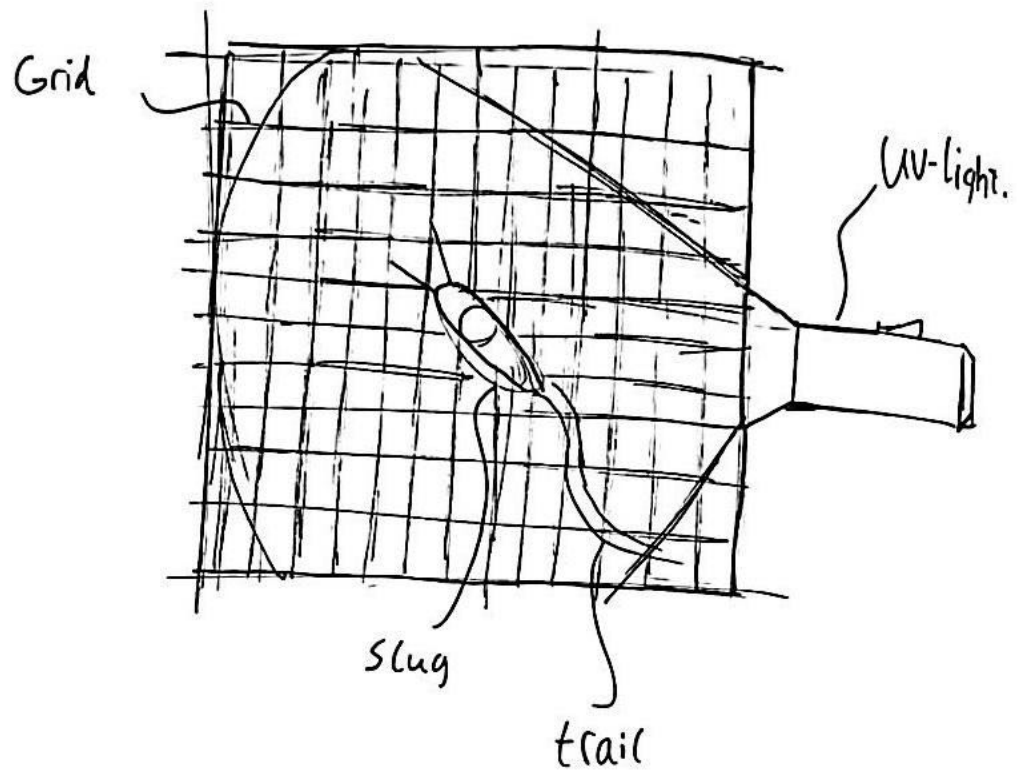
BMP 280.
Aliexpress
Batch ordering.

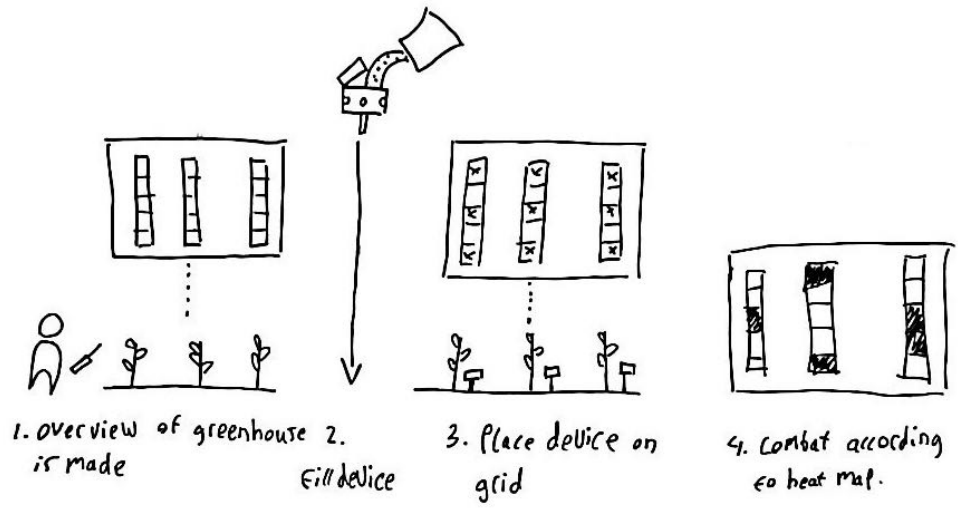


data compression idea:



direction test.





after a week,
maybe one
relocation.

Appendix X - Project Brief



Personal Project Brief – IDE Master Graduation Project

Name student _____

Student number _____

PROJECT TITLE, INTRODUCTION, PROBLEM DEFINITION and ASSIGNMENT

Complete all fields, keep information clear, specific and concise

Project title _____

Please state the title of your graduation project (above). Keep the title compact and simple. Do not use abbreviations. The remainder of this document allows you to define and clarify your graduation project.

Introduction

Describe the context of your project here; What is the domain in which your project takes place? Who are the main stakeholders and what interests are at stake? Describe the opportunities (and limitations) in this domain to better serve the stakeholder interests. (max 250 words)

→ space available for images / figures on next page

introduction (continued): space for images

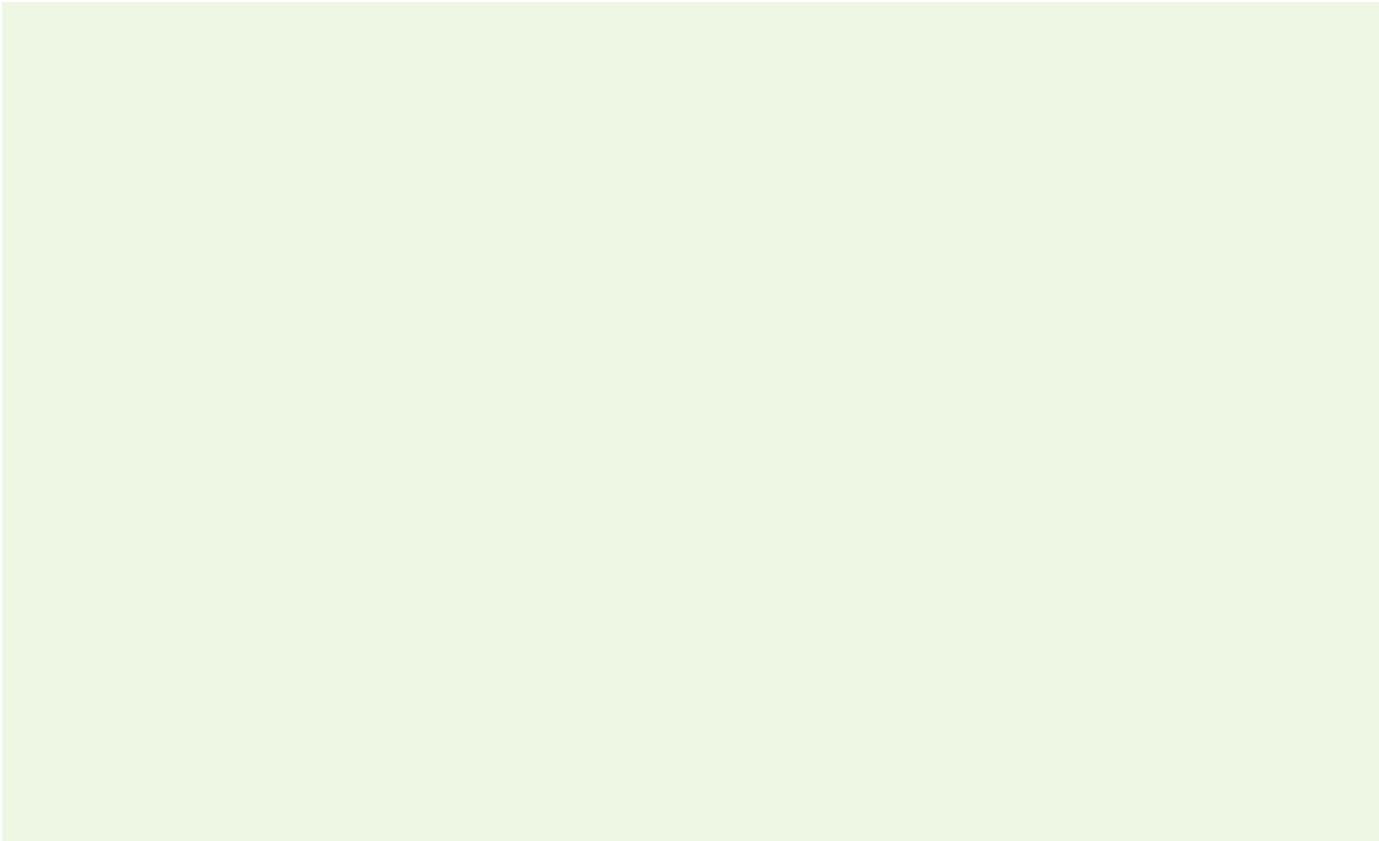


image / figure 1

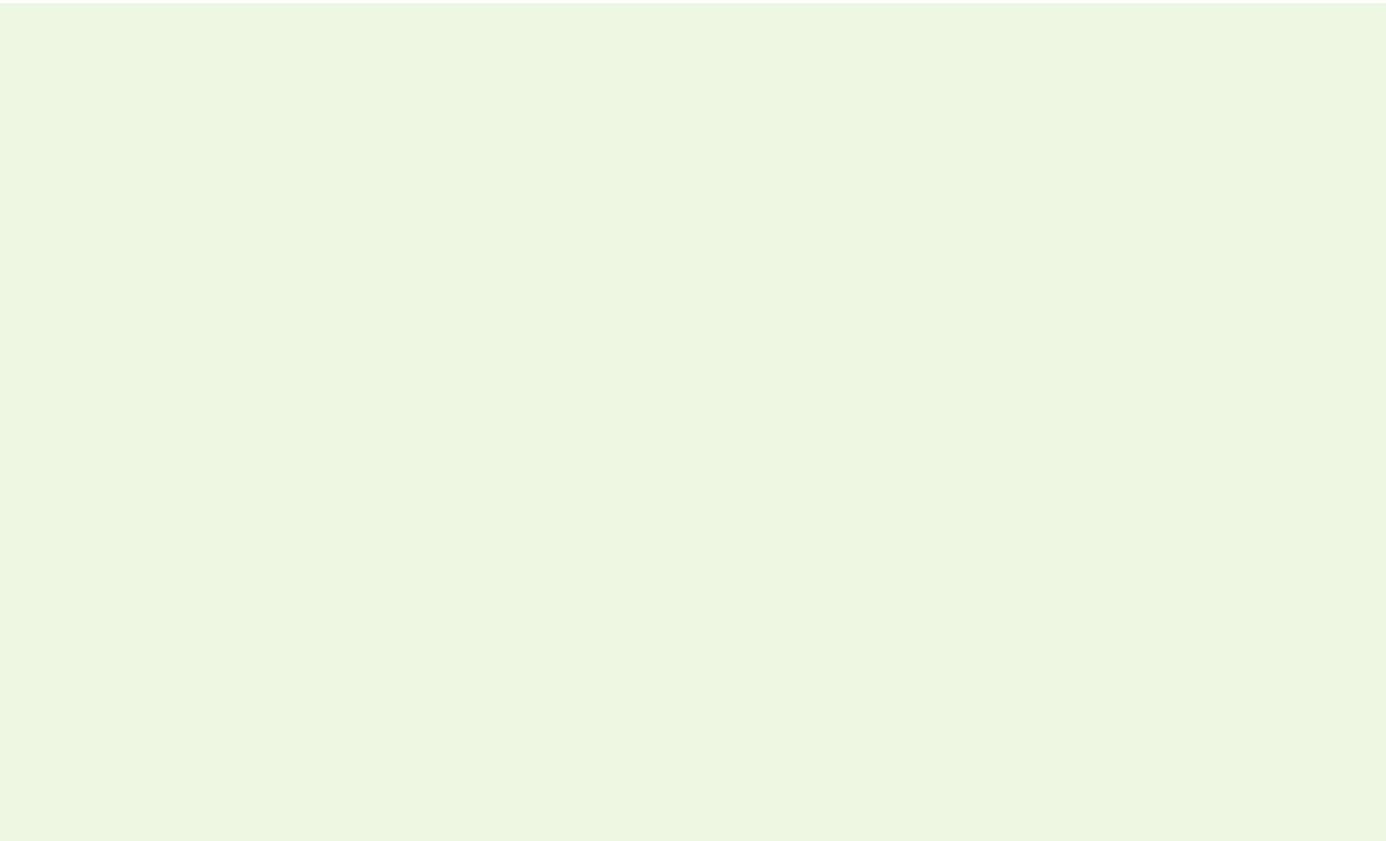


image / figure 2

Personal Project Brief – IDE Master Graduation Project

Problem Definition

*What problem do you want to solve in the context described in the introduction, and within the available time frame of 100 working days? (= Master Graduation Project of 30 EC). What opportunities do you see to create added value for the described stakeholders? Substantiate your choice.
(max 200 words)*

Assignment

*This is the most important part of the project brief because it will give a clear direction of what you are heading for. Formulate an assignment to yourself regarding what you expect to deliver as result at the end of your project. (1 sentence)
As you graduate as an industrial design engineer, your assignment will start with a verb (Design/Investigate/Validate/Create), and you may use the green text format:*

Then explain your project approach to carrying out your graduation project and what research and design methods you plan to use to generate your design solution (max 150 words)

Project planning and key moments

To make visible how you plan to spend your time, you must make a planning for the full project. You are advised to use a Gantt chart format to show the different phases of your project, deliverables you have in mind, meetings and in-between deadlines. Keep in mind that all activities should fit within the given run time of 100 working days. Your planning should include a **kick-off meeting, mid-term evaluation meeting, green light meeting** and **graduation ceremony**. Please indicate periods of part-time activities and/or periods of not spending time on your graduation project, if any (for instance because of holidays or parallel course activities).

Make sure to attach the full plan to this project brief.
The four key moment dates must be filled in below

Kick off meeting _____

Mid-term evaluation _____

Green light meeting _____

Graduation ceremony _____

In exceptional cases (part of) the Graduation Project may need to be scheduled part-time. Indicate here if such applies to your project

Part of project scheduled part-time	
For how many project weeks	
Number of project days per week	

Comments:

Motivation and personal ambitions

Explain why you wish to start this project, what competencies you want to prove or develop (e.g. competencies acquired in your MSc programme, electives, extra-curricular activities or other).

Optionally, describe whether you have some personal learning ambitions which you explicitly want to address in this project, on top of the learning objectives of the Graduation Project itself. You might think of e.g. acquiring in depth knowledge on a specific subject, broadening your competencies or experimenting with a specific tool or methodology. Personal learning ambitions are limited to a maximum number of five.

(200 words max)