KU LEUVEN

FACULTY OF SCIENCE

One Robotic Flower Field to Study Them All

A wireless system for behavioural experiments with pollinators

Kamiel DEBEUCKELAERE

Thesis presented in
fulfillment of the requirements
for the degree of Master of Science
in Biology

Supervisor: dr. M. I. Pozo Romero
KU Leuven

Co-supervisor: *Prof. dr. Ir. H. Jacquemyn*
KU Leuven

Academic year 2020-2021

Preface

First things first. Why did I pick a topic on pollinating insects from the list with proposals for master dissertations in Biology? Insects have been attracting my attention in some of the courses, especially during the course 'Behavioural Ecology' it was the part on social insects that intrigued me. When during professor Jacquemyn's course 'Global change, Ecosystems and Sustainability' the subject of pollination was brought up, the link between plants, insects and the human population became clear. This lecture was given by Dr. Maria Isabel Pozo Romero, who focused on pollinator decline and the global impact, but also showed some possible solutions. This problem-solving approach settled my choice, and I was eager to be a part of the solution by joining the research on pollinator behaviour.

When I started, I had in mind to build a robotic flower field containing multiple flowers and use it to perform an experiment. Along the way, it became clear that the development of these robotic flowers was much more work than anticipated. Moreover, we had some bad luck regarding an essential part, i.e., the LoRaWAN module for wireless data transmission. The global shortage of microchips partly caused by the corona pandemic, threw a spanner in the works. We were able to obtain only two of these modules, so that building a real 'field' was not possible, but at least two robotic flowers were sufficient to perform a proof of concept.

The development of the robotic flower inevitably led us to rely on skills that would rather be found with electrical engineers, computer scientists and software developers. For me this was a relatively unexplored territory but is has given me the opportunity to learn many new skills. Happily, the building of the prototype of our artificial flower was a success in the end.

The "we" and "our" in the preceding sentences stand for my supervisor, Dr. M. I. Pozo Romero, and myself. Therefore, my thanks primarily go to her. She stimulated me and helped lifting the quality of this dissertation with her feedback. Moreover, the idea of the robotic flower came from her, based on previous experience and tips from prof. Jef Rozenski for the new design.

I am also indebted to the advice of my co-supervisor, prof. H. Jacquemyn, who welcomed me in his lab the last year.

Lastly, without the help of Dirk Janssens of the ICTS service desk at the KU Leuven, my work would simply not have been possible. I therefore sincerely acknowledge his input.

Index

Summary

Pollinators are essential elements in successful sexual reproduction of many plants, including important agricultural crops. The last decades, an alarming trend of decline in the abundance of several insect taxa has been observed in different geographical areas. Among many other direct and indirect adverse effects, the loss of pollinators poses a threat to food supply, especially with a still increasing human population. Focussed research to bridge knowledge gaps in basic ecological features of pollinator species will be essential to take meaningful and effective measures both for biodiversity and food production. In particular, the influence of nectar composition and concentration on foraging behaviour is not fully unravelled despite many research efforts the last years. Besides the influence of nectar, pollinator foraging shows to be a complex behaviour influenced by many subtle factors that are not discovered or fully comprehended yet.

A current issue in the field of behavioural ecology of pollinators is that collecting data by manual observation and censuses is very labour intensive and time consuming, while gathered data are mostly insufficient to adequately describe and analyse insect behaviour. Furthermore, the human ability for observing fast-moving insects over a long period is rather limited, and invasive human actions can lead to a bias. Generating large datasets is thus a big challenge. A first solution might be to make video recordings, but this is still very time consuming afterwards and has the downside that there is no chance to adjust factors during the trial. To get more complete and ecological relevant information on insect behaviour, controlled experimental set-ups are needed that can approach natural environmental factors. Data should be gathered automatically and presented to the user in real time, reducing labour intensity and making data less error prone.

For this purpose, we developed a robotic flower, in which each flower can be seen as a mini lab as they each have a separate microcontroller and are powered by a battery. Its first function is the automatic offering of nectar rewards in discrete doses to simulate natural conditions. The second function is the non-invasive monitoring of visitation rate and duration. These data are then wirelessly transferred using Internet of Things (IoT), making it possible to get real-time data of the flowers during the experiment without interfering or even getting close to the experimental set-up. Here the design is focused on *Bombus terrestris*, but the conducted proof of concept showed that other species can be investigated with the same robotic flower. With some easy changes, the robotic flower can be adjusted to address new research questions and therefore is widely applicable. The use of this new automated system will result in much bigger datasets whilst avoiding the stochastic effects of observational snapshots. With this dissertation, we have shown how behavioural studies on pollinators can be improved and facilitated.

List of used abbreviations and symbols

Abbreviations

3D	Three dimensional
ABS	Acrylonitrile Butadiene Styrene
CPU	Central processing unit
CSV	Comma separated values
EEPROM	Electrically erasable programmable read-only memory
EUI	Extended unique identifier
FIFO	First in first out
GPIO	General purpose input/output
IDE	Integrated development environment
IR	Infrared
I	current
IC	Integrated circuit
IoT	Internet of Things
JSON	JavaScript object notation
LED	Light emitting diode
Li-Ion	Lithium-ion
LoRaWAN	Long-range wide area network
MCU	Microcontroller unit
MQTT	Message Queue Telemetry Transport
OTAA	Over the air activation
PCB	Printed circuit board
PLA	Polylactic acid
PWM	Pulse width modulated
R	Resistance
RAM	Random access memory
RF	Radio frequency
RFID	Radio Frequency identification
RGB	Red, green, blue
SJ	Solder jumper
SMA	SubMiniature version A
RX/TX	Receive and transmit
SF	Spreading Factor
SMD	Surface mount device
UPS	Uninterruptable power supply
USB	Universal serial bus
UV	Ultraviolet
Vf	Forward voltage
VSC	Visual Studio Code

Symbols

A	Ampère
mAh	milliampère hour
cm	centimeter
F	Farad
KB	Kilobyte
Kg	Kilogram
G	gram
KΩ	Kilo-ohm
MHz	megahertz
mg	milligrams
mm	millimetres
ms	milliseconds
nF	nanofarad
nM	nanometer
Oz	ounce
s	seconds
V	Volts
VAT	Value-added tax
W	wattage
Ω	ohm
°C	degrees Celsius
µA	microampere
µF	microfarad
µg	micrograms
µL	microliters

1. Introduction

1.1 Pollinator decline

Pollinators are essential elements in the successful sexual reproduction of many plants. An estimated 87.5% of current living flowering plants are depending on these animals for pollen transmission (Ollerton et al., 2011). The importance of the ecosystem service provided by pollinators is reflected by their impact on wild plants as well as on crop species (Potts et al., 2010). The global average economic value of pollination has been estimated at 117 billion dollars per year (Costanza et al., 1997). Although the group of species that are considered pollinators consist of both vertebrates and invertebrates, insects and especially bees are the primary pollinators (Potts et al., 2010). Given the important services provided by many insects, the dramatic decline both in biomass and in diversity of flying insects in general during the last decades is definitely alarming (Hallmann et al., 2017). Sanchez-Bayo & Wyckhuys (2019) have predicted that 40% of global insect species might go extinct in the coming decades. More specific for wild bees, Zattara & Aizen (2021) showed a reduction of around 25% reported species globally between 2006 and 2015 compared to the number of species reported before 1990. In a specific example, Kosior et al. (2007) showed the decline in bumble bees (*Bombus*) in 11 countries of Western and Central Europe: between 1950 and 2000 13 out of the 60 species under investigation went extinct in at least one country. The same pattern of decline also occurs in North American bumble bee species (Cameron et al., 2011), reinforcing the global nature of this issue.

"If insects were to vanish, the environment would collapse into chaos." - E.O. Wilson

Decline of insects goes hand in hand with many adverse effects on a multitude of natural systems, among which pollination. Local extinction of pollinators has shown to be causally linked to local extinction of host plants in Britain and the Netherlands (Biesmeijer et al., 2006). The effect of a single plant species disappearing could cascade further down the food chain and affect a multitude of other species. Besides this direct and indirect loss of biodiversity, the food security of the human population is also at risk. Since 1950 the world population has grown from about 2.5 billion to over seven billion today and this trend is expected to continue until mid 21th century reaching a plateau estimated at nine billion people (Godfray et al., 2010; Steffen et al., 2015) and possibly even higher. As human welfare will also grow, the demand for food will also increase and although the past decades the food production has been rising, still many people suffer from malnourishment (Godfray et al., 2010). With regards to combining higher agricultural yields and biodiversity conservation, two contrasting strategies can be followed: land sharing or land sparing (Phalan et al., 2011). In the land sparing strategy, agricultural and conservational purposes are completely separated, while in land sharing the

two are combined on the same land. There is debate on which strategy is the best, but in practice it is a complex challenge that requires case-specific solutions rather than just using one of both strategies overall (Tscharntke et al., 2012). To reduce global food insecurity, increased yields will be needed in areas where the food is needed the most. Research of Phalan et al. (2011) indicates the potential of land sparing to least harm species compared to land sharing in southwest Ghana and India. Land sparing implicitly means striving for more yields on less land. This should be achieved not by using conventional but agroecological intensification, taking determining factors for biodiversity into account, and pollinators play an important role in this (Tscharntke et al., 2012). Crops that depend on animal pollination for fruit, vegetable or seed production make out up to 35% of the total crop yield (Klein et al., 2007). For some important agricultural crops, pollination by insects is even essential, cocoa being a prime example (Tscharntke et al., 2012). Currently, it has become common practice to use human reared pollinators such as bumble bees to enhance agricultural production and a well-known example is the use of bumble bees for greenhouse tomato production (Greenleaf & Kremen, 2006; Gallai et al., 2009). In South-Western China, the loss of pollinators forced apple farmers to pollinate their crops by hand when the production of apples dropped by 50% in 1990 compared to 1980 (Partap et al. , 2001). All those factors indicate the present shortage of natural pollination to sustain sufficient agricultural yields and stress the importance of a sufficient number of pollinators to satisfy the future demand for food.

There is a growing body of literature that recognises anthropogenic stressors such as land-use change and habitat loss, human spreading of pathogens and parasites, pesticide use and introduction of alien species as main drivers for the current pollinator decline (Potts et al., 2010; Goulson et al., 2015; Sanchez-Bayo & Wyckhuys, 2019). Besides those factors, climate change can negatively influence pollinators as well. The UN just recently released the sixth assessment report of the IPCC (Intergovernmental Panel on Climate Change) which states that human activities are driving the observed unprecedented rate of climate warming that has impacts on ecosystems all over the globe (IPCC, 2021). Burkle et al. (2013) demonstrated that climate change has been causing phenological mismatches between pollinators and plant species during the past 120-year period in a temperate forest system. On the other hand, they also report that the plant-pollinator network has been resilient in the past with the formation of new interactions to buffer extinctions. However, the redundancy, quality and quantity of interactions has decreased and the increasing stress on the system might lead to abrupt changes and a rapid collapse of this invaluable ecosystem service provided by pollinators (Burkle et al., 2013). Memmott et al. (2007) predict that loss of pollinators will mainly affect the more specialised species as they are more vulnerable to shifting phenology. Plants are partly protected by the presence of the remaining generalist pollinators, which may be more resilient to the aforementioned causes of pollinator decline.

Habitat loss is one of the strongest drivers of pollinator loss, especially social bees are sensitive to human disturbances, with exclusion of managed honey bees of course (Winfree et al., 2009). Not only the pure loss of habitat, but also fragmentation and degradation can lead to reduced availability of food and nesting sites besides the possible lethal or sublethal presence of pesticides or other environmental contamination (Potts et al., 2010). Moreover, monoculture farming reduces the diversity of available plants and concentrates the resources in a very little time window. Therefore, although food might be abundant on average, pollinator populations cannot be sustained (Kremen et al., 2007). Introduction of alien pollinators, for instance for managed crop pollination, can impact native species in different ways: competition for resources, genetic dilution, and the spread of viruses, mites and other diseases (Potts et al., 2010). These stressors do not act separately from each other, in reality multiple pressures are present simultaneously and could even act synergistically (Tylianakis et al., 2008). For example, pesticides and lack of food render bees more vulnerable to parasites as immune responses are affected (Goulson et al., 2015), or pesticides could have an increased toxicity under global warming (Verheyen et al., 2019).

The well-established fact that pollinator decline poses a real threat inspired scientists to direct their research towards this problem. In the last decades, several publications have suggested roadmaps to insect recovery in general (Harvey et al., 2020), for pollinator species (Kearns et al., 1998; Potts et al., 2016) or for taxa such as *Bombus* specifically (Goulson et al., 2008). They all point to the need for focused research to bridge knowledge-gaps in basic ecological features of pollinator species to take meaningful and effective conservation measures which will be needed to sustain agricultural yields and safeguard wild plant diversity. Plant-pollinator relationships have been shaped during a long period of co-evolution and specialisation with selection pressure from pollinators on flowers and vice versa. Although visitor diversity is high in the vast majority of flowering plant, they often show some degree of specialisation in a certain functional group of similar pollinators (Fenster et al., 2004). Most research efforts are directed towards managed pollinators such as honeybees as these species are more readily available for lab experiments compared to wild pollinators. However, Garibaldi et al. (2013) have shown that wild pollinators increase fruit set with a factor two compared to honey bees, making them more effective pollinators. This means that research that was previously done is not always representative for other, wild pollinators as each species or even population develops their own set of traits over time depending on the specific interactions with flowers and the pollinator community (Fenster et al., 2004). There is therefore an urgent need for more and thorough studies to fully comprehend these interactions in less-studied species.

1.2 Pollinator foraging behaviour

Insect pollinators can be found in many taxonomic groups and have a big biogeographical distribution. This leads to the assumption that the evolution of this insect-plant mutualism did not occur in a single event but has evolved multiple times in history (Bronstein et al., 2006). Ancient interactions between plants and animals, e.g., herbivorous insects feeding on plant tissue, are at the origin of what we now know as pollination. Evidence suggests that plants were able to take advantage of these interactions by using these insects for pollen transfer. Before any other specialist traits evolved to attract and reward these floral visitors, they were already feeding on pollen (Labandeira, 1997; Bronstein et al., 2006). Judging from palaeobiological information on feeding strategies, the first reward to lure visitors to the flower was probably excess pollen (Labandeira, 1997). Later in the evolution of floral traits, nectar appeared as the main floral reward. As insects evolved to feed on nectar and pollen, co-evolution led to flowering plants being more and more reliant on these pollinators for successful reproduction (Poinar, 2016). Over time, plants showed suites of traits that facilitate the attraction of particular pollinators (or at least functional pollinator groups, e.g., long tongued pollinators or pollinators big enough to reach down in the flower) and many insects only visited specific flower types (Bronstein et al., 2006; Fenster et al., 2004). Flowers and insects can be specialists or instead go for a more generalist strategy, i.e., attracting multiple pollinators or using multiple plants respectively (Waser et al., 1996). Adding to complexity, pollinator foraging can also show some degree of flexibility, with context-dependent preferences. For example, innate colour preference can change through associative learning, contrast between flowers and their surroundings, presence of similar flowers or competition with other pollinators (Briggs et al., 2018). However, some insects, such as beetles did not develop highly specialised pollination mechanisms and are thus sometimes labelled as ‘primitive’ pollinators, in contrast to others that adapted fully to a life as pollinator with bees as the main example.

All floral traits focused on attraction of specific pollinator groups are known as a pollinator syndrome. They include production and composition of floral rewards and traits that affect the potential pollinator’s ability to locate a flower and exploit its food resources. For example, red flowers attract hummingbirds while bees commonly prefer yellow or purple flowers more (Briggs et al., 2018). Lower concentration and thus less viscous nectar enables Lepidoptera to feed on it with their long proboscis (Kim et al., 2011), and the orchid *Epipactis veratrifolia* gets pollinated by mimicking aphid alarm pheromones to lure Syrphidea (Stoekl et al., 2011). What started as simple interactions evolved into a multitude of complex interactions that affect pollinator foraging behaviour and attractiveness of plant species. As pollinators often cannot directly see the reward hidden inside the flower, many plants display guides that help pollinators to find it, e.g., the anthers can have a signalling function for a reward in the form of pollen (Lunau, 2000). Besides of this specific reward guides, flowers can also invest in features that

are related to the magnitude of the reward, such as flower size, shape or scent (Makino & Sakai, 2007; Sun et al., 2018; Wright & Schiestl, 2009). Although plants would benefit from attracting pollinators without spending resources on providing a reward, the learning ability of the pollinators often helps to keep these signals honest (Sun et al., 2018). Still, dishonest signals were able to persist in some cases, this is also known as deceit pollination. For example, some orchids developed ingenious strategies with mimicry of patterns and odours of potential mates to trick insect pollinators to visit the flower without providing any reward (Givnish et al., 2015). The cost of possibly ignoring a mate is bigger than the cost of going to the flower without getting a reward.

Foraging behaviour of pollinators is influenced by many factors. There is a large diversity in floral displays, with flowers varying in combinations of colour, shape and scent aimed at attracting the pollinators that are most effective (Schiestl & Johnson, 2013). Floral signals must thus be tuned to the perceptual capacities from these specific pollinators. First, colour and size determine if a pollinator will be able to perceive the flower from a long distance, while scent is only giving away information at shorter distances (Goulson et al., 2007). Larger flowers with colours contrasting with the background are more easily perceived by foraging pollinators (Spaethe et al., 2001). Besides, flower size is often positively correlated with reward, meaning that large-sized flowers usually have higher visitation rates (Makino & Sakai, 2007). Second, floral guides (also known as nectar or honey guides) are specialized patterns found on flower surfaces designed to guide pollinators to the reward, being nectar and/or pollen (Lunau, 2000). Giurfa (2004) showed that if visited flowers give a reward, honeybees could learn and remember the colour patterns. This could be a simple colour differentiation, but if similar non-rewarding plants are present when the positive association is being made, honeybees are able to learn finer details in the colours. The finer discriminations between flowers will be forgotten after a couple of days, while the easy visual tasks can be remembered for many days (Dyer & Garcia, 2014). Social insect pollinators can also communicate their foraging experience with other workers, for example in honeybees with the famous waggle dance, in bumble bees with a foraging alert pheromone (Granero et al., 2005). Third, the shape, including the depth of flowers will determine which visitor will be able to reach the rewards, e.g., their tongue needs to be long enough, but they also need the necessary cognitive abilities to handle the three dimensional shape of the flower (or the landing platform in the case of the robotic flower) (Werner et al., 2016). Fourth, the scent of flowers consists of volatiles emitted by plants with a myriad of functions, many not known yet, but one of them is certainly to attract pollinators (Pichersky & Gershenzon, 2002). Insects have a well-developed sense for olfactory signals in most cases and are able to associate scent with food, even if the scent does not directly originate from the food itself (Wright & Schiestl, 2009).

Other than nectar and pollen, floral reward include oils, structures for nesting, heat sources and sleeping spots (Simpson & Neff, 1981). Nectar is a liquid substance produced in nectaries that is used by plants to interact with animals (Pacini et al., 2003). There are different kinds of nectar: floral nectar to attract pollinators and extrafloral nectar used to attract defenders, such as ants, against herbivores (Heil, 2011). Floral nectar, containing a lot of sugar, is of interest here as it was specifically designed to fuel the foraging by pollinators and thus increase pollen transmission and plant fitness (Pacini et al., 2003). The carbohydrates found in nectar are fructose, its monomers fructose and glucose, and other hexoses. Although it was long believed that nectar consisted only of these sugars, Baker & Baker (1973) pointed to the presence of amino acids in nectar as well. Together with the sugars, these free amino acids have the function of attraction as they are an important source for nitrogen in many adult pollinators (Heil, 2011). Some nectar is also scented, delivering an honest signal of availability to pollinators, but in the meantime it could also be used as a defence against unwanted visitors (Raguso, 2004). It seems that not only extrafloral nectar has a protective objective, but also components in floral nectar have this function. Proteins in nectar, also known as nectarines, are in most cases likely to have antimicrobial functions to protect plant tissue while secondary metabolites found in nectar are probably responsible for protection of the nectar itself against nectar robbers and nutrient degradation (Heil, 2011; Stevenson et al., 2017). An example of such a secondary metabolite is Gelsemine, which has toxic traits to keep away unwanted visitors, but can also affect pollinator behaviour, e.g., shorter probing times (Heil, 2011). Secondary metabolites can also have beneficial effects for pollinators, such as reduction in disease level (Stevenson et al., 2017).

Besides the composition that is largely phylogenetically determined, also concentration of nectar is of importance for pollinator foraging behaviour. The concentration of secondary compounds in nectar often determines the behaviour of pollinators. A low concentration could attract them while a higher concentration is repelling, e.g., nicotine (Stevenson et al., 2017). A large variation is possible in concentration and composition of nectar between species, between different populations of the same species, within the same species or even within the same plant. The sexual expression of flowers may display a difference in nectar, but also photosynthetic activity (sugar production) plays a major role in concentration and composition of nectar (Pacini et al., 2003). Fluctuations in light, water availability and temperature affect the photosynthetic rate, and thus also the nectar concentration will change according to this pattern, even within a single plant. Every sugar has its own nutritional value from the point of view of the pollinator, and different concentrations can result in changing energy reward (Nicolson, 2011). The previous cited factors for nectar variability are all directed by the plant itself, but other organisms may also have some influence. After inoculation through the air or using a visiting pollinator as vector, all kinds of microbes, but primarily yeasts and bacteria, can reside in nectar

and because of their metabolism change its composition, concentration, odour and temperature (Pozo et al., 2014). The pollinators themselves can thus be responsible for external changes in the nectar composition that was originally designed to attract them.

The functions of nectar appear to be much broader than first believed and that is why so many research in the field of plant-pollinator mutualism has been focussed on volume, concentration and composition of nectar (Alvarez-Perez et al., 2012; Nepi et al., 2012; Pozo et al., 2020; Richardson et al., 2015). Not only nectar, but also pollen can affect pollinator foraging behaviour. Hymenoptera, Lepidoptera, Diptera are holometabolous insects, which use the protein-rich pollen to feed their larvae, while adults are limited to nectar. In *B. terrestris*, a well know study system for pollinator behaviour, young foragers generally collect nectar, while older foragers switch to pollen, probably because this complex task requires more experience (Goulson, 2010). *Apis mellifera* is known to transform pollen into 'bee bread'. They add nectar and salivary gland secretions to the pollen and let it ferment, giving it higher nutritional values and thanks to the formed lactic acid also protection from spoilage for many months (Vasquez & Olofsson, 2009). Besides protein, pollen also contains lipids, fatty acids, sterols and volatile components (Nicolson, 2011).

The foraging behaviour of pollinators is driven by their need for energy. Bumble bees, for example, are endothermic, meaning they need a lot of energy to maintain their body temperature. Moreover, they also need to keep the nest warm for the larvae, and therefore they need large amounts of nectar to compensate this energy loss (Heinrich, 1974). Under optimal foraging conditions, pollinators in general will try to maximise their net energy intake and therefore will try to select the best possible foraging strategy (Pyke et al., 1977). In this way, they try to increase their fitness, or that of the hive. There is a general trade-off between nectar concentration, which determines the amount of energy and viscosity, that determines how much nectar can be ingested and/or transported (Kim et al., 2011). Bumble bees are able to make associations with floral traits and food reward when interactions are repeated, and this enables them to optimize their foraging efficiency (Sun et al., 2018). Another consideration foragers can be faced with is the trade-off between a source of high quality at a long distance versus a lower quality source nearby. Factors like predator avoidance and mating behaviour also influence foraging behaviour (Ne'eman et al., 2006). Solitary bees, who make up the biggest part of the bee species, as opposed to social bees, are not divided in castes and thus each individual has to forage to provide for itself (Michener, 2000). When foraging in a patch of flowers, the individual forager must decide to stay or go and search for a new and possibly richer patch after every visit. The rewards gotten from the last visited flowers determine the behaviour of the bee: if it was highly rewarding, the bee will fly short distances with many

turns, while in a low rewarding patch longer distances in a straight line will be flown to reach a new patch (Ne'eman et al., 2006).

The concept of flower constancy describes the habit of pollinators to visit only flowers of a single species, which can lead to a higher chance to get pollinated for the flower as the chance of receiving conspecific pollen is high (Grüter & Ratnieks, 2011). There is an exception when only one morph is visited, because in that case outcrossing will not occur, rendering the flowers susceptible to inbreeding (Charlesworth et al., 1990). At the other side, the advantage pollinators get from this strategy is less clear, as they possible pass by other, more rewarding flowers when looking for that one other flower (Gegear & Laverty, 2005). Different theories have been developed to explain this, with many of them claiming cognitive limitations are at the base of flower constancy (Chittka et al., 1999). Indeed, the rate of flower constancy is observed to be related to the trade-off difference in floral traits between co-occurring flowers: the more different flowers are, the more pollinators show flower constancy (Waser, 1986). This could imply that cognitive limitations prevent flower constancy as flowers that resemble too much cannot be discriminated. Nonetheless, these hypotheses are all unsatisfactory according to Grüter & Ratnieks (2011) based on experiments that debunk these proposed cognitive limitations. They are instead pushing the 'costly-information hypothesis' forward. This theory states that if in any given field the pollinator has constancy for a relatively high rewarding flower, the chance is high a random other flower would be less rewarding. This means many flowers would need to be sampled by the pollinators to gather this information, making it too costly. Different species show different strategies, for example *A. mellifera* workers show a very high degree of flower constancy, while *B. terrestris* workers move more between plant species (Gegear & Laverty, 2005; Grüter & Ratnieks, 2011). The target number of visits needed to assure full fertilization is reached quicker with higher flower constancy. For example, apples need 10 visits per flower by honeybees or solitary bees to get a full fertilization (Garibaldi et al., 2020).

Pollinator foraging shows to be a complex behaviour this is subject to many subtle influences. To unravel the biotic and abiotic interactions between pollinating insects and their environment a lot of experimental information is needed. Many factors are not discovered or fully comprehended yet, making future research essential to improve knowledge on pollinators and the important ecosystem service they provide.

1.3 Previous research methods & limitations

To get the most complete and ecological relevant information on animal behaviour, controlled experimental set-ups are needed that can approach the natural environmental factors as close as possible. According to Calisi & Bentley (2009), results of behavioural experiments are strongly

dependent on the environment in which they are conducted. Unfortunately, performing behavioural field experiments with pollinators has proven to be challenging as they often imply collecting data from fast moving species and/or data from multiple individuals at the same time. Manual collection of such behavioural data relies on continuous observation and is thus very labour intensive and time consuming while at the same time error prone because of human observational limitation for dynamic objects (Crall et al., 2015; Simons & Chabris, 1999). Moreover, invasive human actions or equipment are often needed, which again have the potential to influence foraging behaviour of pollinators (Calisi & Bentley, 2009). Generating large, qualitative datasets for robust analyses is thus in many cases not feasible. A first solution might be to make recordings of the behaviour for later analysis to reduce error rate. Unfortunately, this process still needs a high input of labour and time, again resulting in a low sampling resolution (Ratnayake et al., 2021). Recent research proposed new techniques to improve observational studies: investigating individual pollinators in a non-invasive way with algorithms for image-based tracking (Crall et al., 2015; Ratnayake et al., 2021). Nonetheless, manipulative studies remain hard. For example, how to manipulate nectar, the main floral reward, to study the effect of microbes, pathogens, chemicals, or other components on pollinator foraging behaviour? The solution today would be to keep the focal species in captivity and offer them artificial nectar in vials that in the best case resemble a flower. This does not reflect realistic conditions for foraging and the nectar is not depleted and refilled as it would happen in real flowers. Cheap, reliable, automated, and easy to use methods are needed to lift the quality of observational studies for foraging behaviour in pollinators.

1.4 First artificial flowers

An answer to the apparent need for new research methods came in the form of artificial flowers that allow a degree of control over variables needed to study foraging behaviour (Essenberg, 2015). Many versions of artificial flowers have been constructed, with the aim of providing a self-refilling system for nectar. Although they all practically have the same function, many different designs were used, each having its own characteristics. Nectar can be delivered in continuous doses (Paldi et al., 2003; Makino & Sakai, 2007; Ohashi et al., 2010) or in discrete doses (Keasar, 2000; Cnaani et al., 2006; Sokolowski & Abramson, 2010; Essenberg, 2015). In some artificial flower designs it is possible to control the timing and/or the volume of the nectar as well, but the ability to automatically detect a floral visitor is still missing.

1.5 Innovation with robotic flower by Kuusela & Lämsä

The Finnish researchers Kuusela & Lämsä (2016) developed a low-cost robotic flower system specifically for behavioural experiments on pollinators in lab environments. These artificial flowers facilitate experimental set-ups with controlled manipulations of nectar composition. Compared to previous artificial flower systems, the main innovation here is the automatic

detection of floral visitors that is coupled to the automatic refilling of nectar in discrete doses. In this way, invasive, costly and time-consuming manipulations such as manual nectar refilling are avoided and data collection is more reliable as it is no longer depending on visual observation (Kuusela & Lämsä, 2016). Recently, Pozo et al. (2020) made use of this system with 16 flowers in an experimental arena to test the impact of fungal metabolites in nectar on bumble bee behaviour. This application clearly proves the usefulness and effectiveness of such robotic flower system. However, the need to be connected to a central control unit and computer makes the system not versatile and widely deployable.

2. Objectives

We now attempt to make the logical extra step to design a robotic flower that is completely independent. The new robotic flower should also be wireless and energy efficient so that it can be used for longer experimental periods. Another innovation we aim for is the ability to access real-time data during an experiment. To build this new robotic flower, we could rely on previous experiences by Kuusela & Lämsä (2016) and Pozo et al. (2020), but as those innovations require a different approach we basically had to start from zero. The major aim of this thesis was to produce a robotic flower, covering the complete production process from start to finish, including mechanistic design, hardware selection and software development for the robotic flower.

3. Materials & Methods

The following text gives an overview of the robotic flower system (Figure 1). After a system overview, we discuss the mechanical design of the robotic flower, its internal hardware parts, assembly, firmware on the microcontroller and software of the application. This chapter is concluded with a proof of concept in which the functionality of the flower is examined.

3.1 System overview

The principal idea of the Robotic Flower is to automate the measurement of flower visitation rates and duration with wireless data transmission over a period of multiple days that can also be used in more remote areas. The robotic flower needs to present the visitors with a discrete dose of artificial nectar to mimic a realistic system and stimulate natural foraging behaviour. To achieve these goals, some requirements must be met: the device needs to be energy-efficient, and it must be able to make use of long-range communication tools. These demands fit perfectly in the concept of Internet of Things (IoT), a booming technology in which devices become interconnected and/or connected to the internet with many possible purposes, but in this particular case the automation of the system. The device typically has a unique identification,

a unit for computing, sensors to gather information and a way to communicate (Al-Fuqaha et al., 2015). In this case, the device is the robotic flower itself, which is built with an Arduino microprocessor, infrared-sensors and a microchip based on wireless LoRa® technology for communication with LoRaWAN (Long Range Wide Area Network) (see section 3.3.11 LoRaWAN module for more information). LoRaWAN is a low power networking technology that uses the LoRa® radio protocol in which a single gateway can receive data from thousands of devices in a radius of 2 to 5 km in urban areas and up to 15 km in more remote areas (Adelantado et al., 2017). For example, while testing the robotic flower inside the ICTS building of the KU Leuven in Heverlee, the message was received by the gateway placed on the tower of the central university library in the city centre of Leuven at almost 3.5 km. The airtime of this message, containing alarm statuses and battery voltage of the robotic flower (3 bytes), was only around 0.05 seconds. The airtime, or the speed at which a message is sent, depends on the spreading factor (SF). The standard is SF 7, the fastest possible send protocol to make airtime and energy consumption as small as possible. If there are no gateways close, the LoRaWAN module can change the SF (e.g., to SF 12) to have a slower transmission which

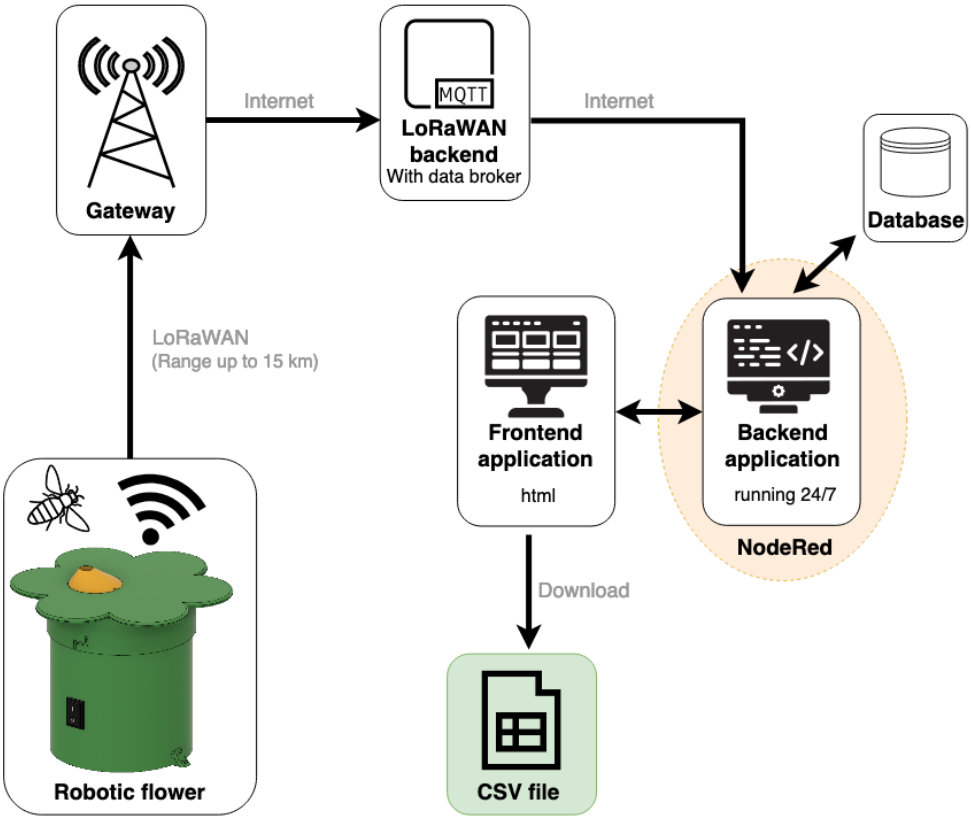


***Figure 1**: Overview of the robotic flower system. The system consists of the robotic flower, which sends data packages using LoRaWAN to the LoRaWAN backend through a gateway. The backend subsequently transfers the data using the internet network to the application. The application has a Backend, which is constantly running to receive and store data in the database, and a Frontend that enables the user to overlook all deployed robotic flowers and retrieve the daily CSV files. These files contain all the data that was gathered by the robotic flower field.*

makes the chance bigger that further gateways can receive the message (Adelantado et al., 2017). The gateway antennas are listening to all the signals coming in and then translate the incoming radio waves to electrical signals. The gateway, which is connected to the internet, then relays the received message to a LoRaWAN backend server with an MQTT (message queue telemetry transport) broker. MQTT is a protocol based on publish/subscribe transport of messages, here used for communication between the gateway and the application (Provoost et al., 2019). The data broker acts like a post office: it receives all the messages sent from devices (publishers) and then forwards the right message to the right application (subscriber).

The message sent by the robotic flower contains two parts: the topic and the payload. The payload contains data about the time and duration of each visit for the period between two sending events, extra information such as the battery level of the flower and metadata. For this project, we made use of DingNet, the physical setup of IoT by the KULeuven which provides the gateways with antennas that ensure coverage across the whole city and the LoRaWAN backend. For projects outside the standard coverage of DingNet, portable gateways with antennas can be used. These have a range from 1 to 5 km depending on the environment (e.g., city versus open field situation), but they will need power supply and internet connection.

The application consists of a backend and a frontend part. Both are constructed using the browser-based programming tool Node-RED (copyright OpenJS Foundation). The backend application receives the message from the broker and Node-RED should therefore be running all the time when the robotic flowers are used in experimental setups. This could for example be on a server, on a Raspberry Pi or in the cloud with Front End for Node-RED (FRED). For developmental purposes, Node-RED was running on a local computer during the making of this thesis. After receiving the message of the broker, the backend does the processing and stores the visitation data in a daily CSV file on a temporary database. For every visit, a new line is added in the file containing time of visit, duration of visit and the name of the device that measured the visit (e.g., ‘flower 1’ or ‘flower 2’). The frontend application gives a visual representation of the battery life and some alarms that can be triggered for every robotic flower in the system. In the menu of the frontend application there is also an overview of files created by the backend application which allows users to download the separate CSV files for every day on a local device for further analysis.

When the application receives a message from the robotic flower, it is also possible to send a small message back to the flower (Adelantado et al., 2017). In the flow that is created, the robotic flower will receive a confirmation if the data transmission was successful, otherwise the flower will try to send the data again for a limited number of times. If the transmission was

not successful after these attempts, the data will be lost. The robotic flower receives the order to go to sleep at a pre-selected time in the same way, by receiving a message from the backend.

3.2 Mechanical design

The mechanical design of the robotic flower was based on the previous design by Kuusela & Lämsä (2016). A visit is recorded by a voltage drop of the infrared sensor when the emitter is blocked by a bumble bee in the feeding hole (Figure 2). This detection system was used multiple times before Kuusela & Lämsä (2016) and has proven to be robust, straightforward, and reliable (Keasar, 2000; Sokolowski & Abramson, 2010). The refilling of the nectar is done with a servo-arm with a small nectar cup connected to a servomotor (Figure 2), which makes it possible to present the pollinators with discrete amounts (see part 'nectar cup') of floral reward instead of a continuous supply of nectar used in other designs (e.g., Ohashi et al., 2010). Plants try to attract as many pollinators as possible, but long visits do not enhance pollen transfer among

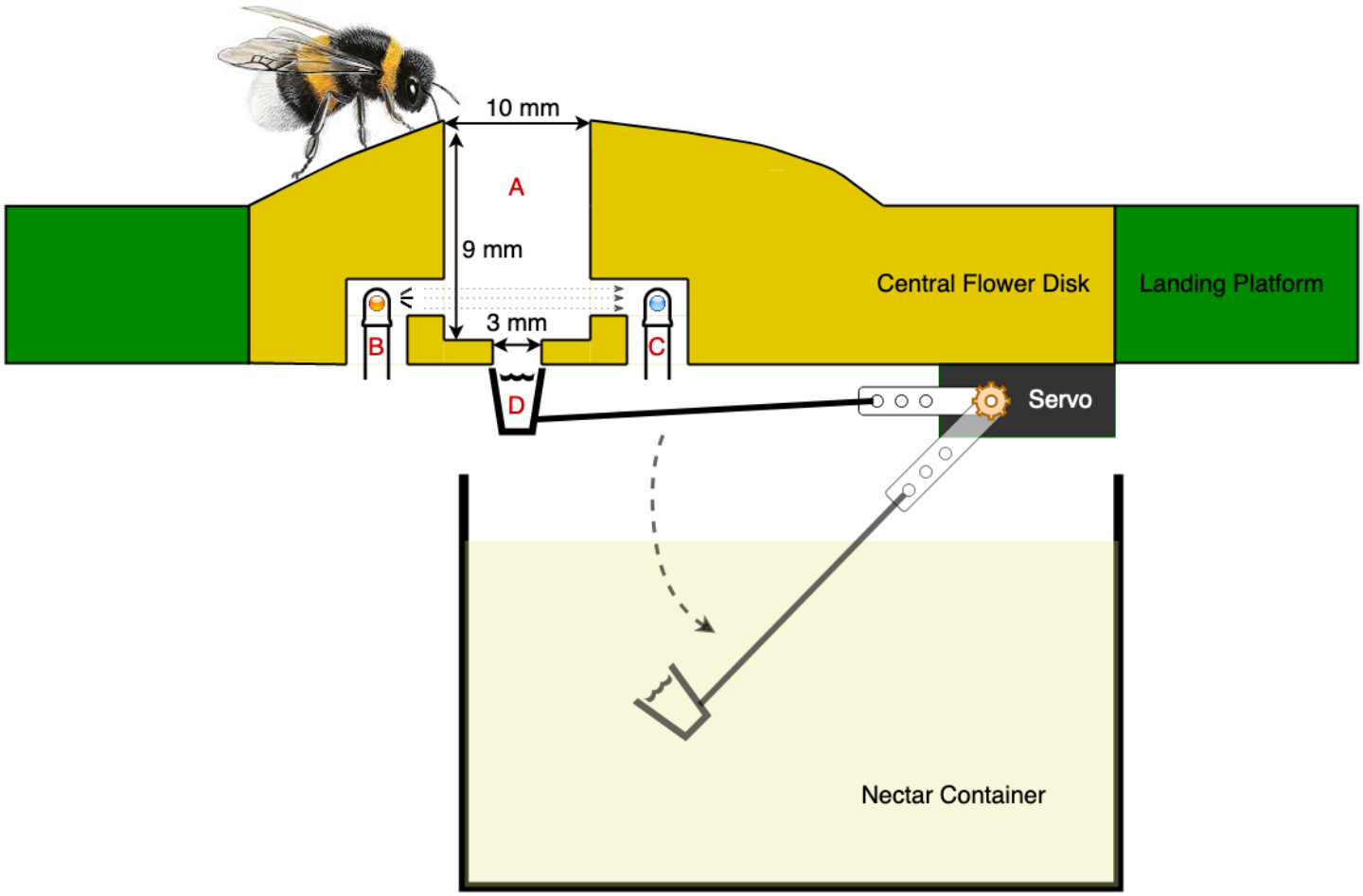


***Figure 2**: Mechanistic design of the robotic flower with **A)** the feeding hole, **B)** Infrared Emitter, **C)** Infrared sensor and **D)** the nectar cup. The servo-arm is used to refill the nectar cup and press it to the central flower disk, where visiting insects can reach the nectar.*

flowers and therefore favour self-pollination (Goulson et al., 1998). Therefore, most plants only provide discrete amounts of nectar so that pollinators must move to different flowers and plants to meet their high daily energy intake requirement. In doing so, they increase pollen transmission (Wolf et al., 1999; Spaethe et al., 2001). It takes some time to replenish nectar after depletion by a pollinator, so flowers can be empty when visited. For example, when sampling *Symphytum officinale* (Boraginaceae) showed 28% of the flowers contained no nectar (Goulson et al., 1998). Presenting discrete amounts with intervals is the best way to simulate the natural availability of nectar in flowers.

Making use of a servomotor for presenting the nectar also has the advantage that overfilling is avoided, which would not be the case if a pump system was used. Other systems are also possible, for example with an electromagnet (Keasar, 2000) but using a micro type servomotor (see section 3.3.4 Servomotor for more information) is the easiest, the most fail-proof and the cheapest solution for this purpose (Kuusela & Lämsä, 2016).

The mechanical design of the robotic flower was made using Autodesk® Fusion 360® under student license. The design consists of several components, of which five are meant to be 3D-printed: the stem, the landing platform, the central flower disk with feeding hole, the nectar cup, and the nectar reservoir case. The nectar reservoir itself was purchased in readily available format but could also be custom made if wanted or needed for a different project. The used 3D-printer was a Prusa I3 MK3S, available at FabLab KULeuven. The material used for the print is a mixture of PLA (polylactic acid), a thermoplastic with colourants and UV (ultraviolet) stabilisers. PLA is made of natural products and is biodegradable, making it much more environmentally friendly than alternatives such as ABS (Acrylonitrile Butadiene Styrene) which is a petroleum derivative. Because of the low melting temperature, PLA is a very user-friendly material for 3D-printing, but this also means that the printed parts should not be exposed to temperatures higher than 50°C to avoid damage. For more rigid, long-term use of the robotic flower it might be useful to consider printing in ABS instead of PLA. Both materials are splash-proof, which is a quality that is necessary to make the flower suited for outside use. ABS is completely waterproof, while PLA can show some leakage over time. This can be prevented by coating the parts with polyurethane sealer or wax. For the prototype we chose PLA with a conformal coating spray, as it is quick, cheap, and easy to print.

In making the files for 3D-printing there are some restrictions to consider. First, every component needs to have at least one flat surface, as the 3D-printer has a flat printer bed. Second, with the Prusa I3 MK3S that was used here, each part can only be printed in one colour. Other types of 3D-printers are capable of multi-colour printing but still, it is something to think about when designing. Third, the printer needs to use supports to print parts that are not supported by the print itself. After printing, these supports need to be taken off in a way

depending on the printer and material that was used. Sometimes it can be washed away with water, but in our case the supports had to be broken off from the print. These restrictions made it necessary to separate the central flower disk and the landing platform instead of combining them in one component. The flat surface of both components is not compatible, and the colour is different. The two components were later joined together to form one whole. The total weight of the 3D-printed components was around 300 g, including the supports needed for printing the parts. The weight of the total flower, with hardware parts and battery is around 358 g (without nectar in the reservoir).

3.2.1 Stem

The stem of the flower (Figure 3) serves as the housing part for the electronic components of the robotic flower to keep them protected against dirt, moisture and other substances that could impair the components. Moreover, the parts are also hidden for pollinators so that the electronic components and motion of the servo-arm don't interfere with foraging behaviour. At the bottom, four beams are in place to support the printed circuit board (PCB) so that it is elevated from the bottom. This was done to prevent the PCB being covered with water when condensation is formed in the flower and runs down the sides. A slit between the wall and PCB is kept free so that the condensation can run past he PCB to the bottom of the stem. To keep the PCB in place, two of the four supports have cylindrical protrusions that fit in the holes of the

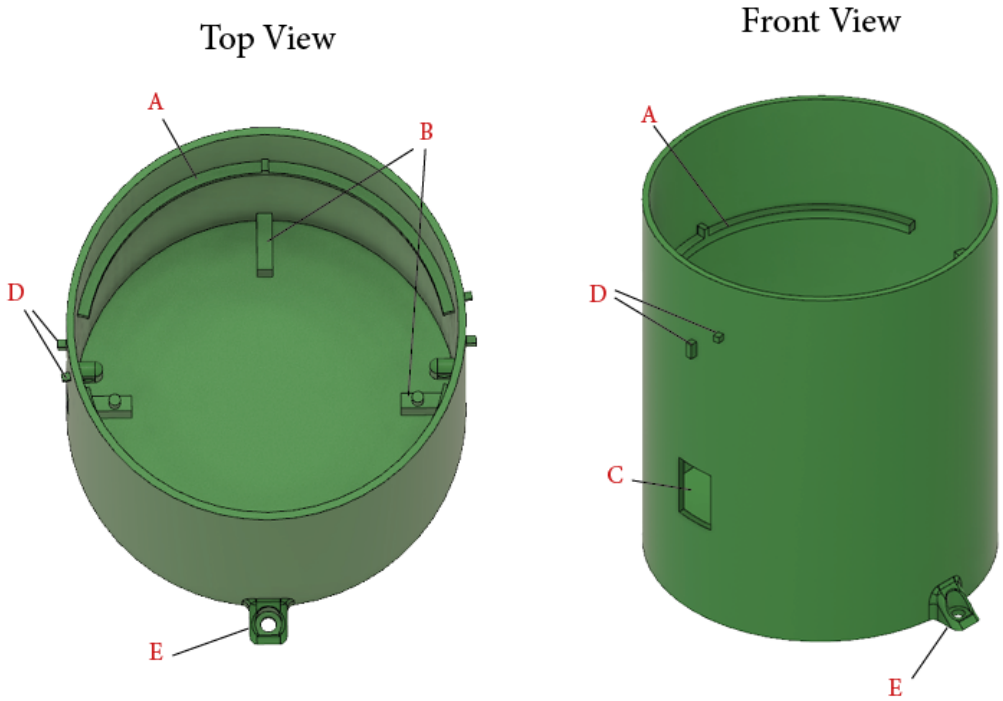


***Figure 3**: Top view and front view of the 3D-design of the stem with **A)** support beams to carry the nectar reservoir case with protrusion to fit the notches so movement of the parts relative to each other is prevented, **B)** PCB supports of which two have cylindrical protrusion, **C)** hole for on/off switch, **D)** twist-and-lock protrusions and **E)** mounting feet.*

PCB (see section 3.3.13 Printed circuit board for more information). On top of the PCB, there is space left to place the battery module. This design with a broad stem, in which the battery is not put upright, ensures a low centre of gravity which helps to avoid tipping over of the robotic flower. The stem also has two support beams, one at each side to hold up the nectar reservoir case (see further). Also here, two protrusions are provided which fit the notches in the reservoir case to prevent it from moving around. Keeping the right position is important for the well-functioning of the servo refilling system. In the side of the stem there is also a hole to fit in the on/off switch that can be clipped in and sealed tight with silicone (Handson, Leusden; see Addendum 2 for more information). Further, there are also protrusions for the twist-and-lock system to fit the landing platform on top of the stem which keeps the components from disconnecting easily. Finally, on two sides of the flower mounting feet are constructed to fix the robotic flower on the substrate to keep it from tipping over if necessary. For example, the flower could be anchored in the ground using a small picket, or it could be screwed onto a platform.

3.2.2 Landing platform

The landing platform (Figure 4) is constructed so that it fits over the stem and is kept in place with a twist-and-lock system for which notches are present on the side. The platform serves as

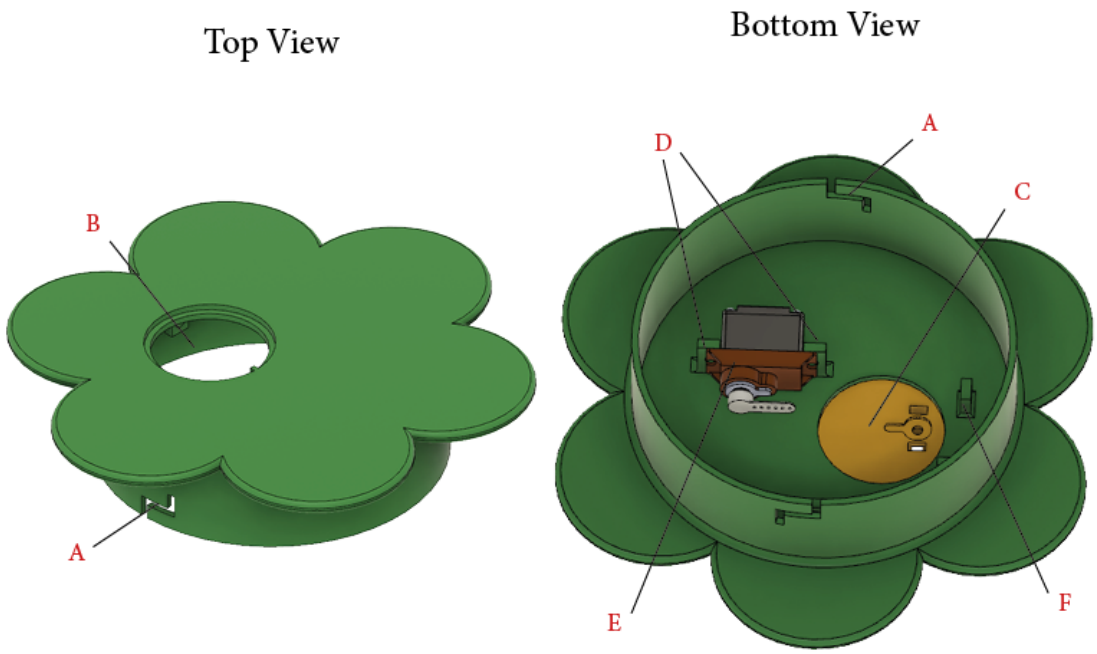


***Figure 4**: Top and bottom (with servomotor and central flower disk attached) view of the landing platform with **A)** twist-and-lock notches, **B)** opening for the central flower disk and **C)** bottom view of central flower disk in place, **D)** protrusions to attach the servomotor to the landing platform, **E)** the servomotor and **F)** protrusion to guide cables.*

a landing spot for foraging pollinators so that they can easily visit the central flower disk (see further) when they spot it. The colour green was chosen to make a cryptic background in contrast with the central flower disk (Chittka et al., 1994). In the place where the central flower disk must fit, an opening with a ledge is made. In this ledge, a protrusion is constructed in such a way that the central flower disk with notch can be placed in the right position before being fixated with transparent silicone (Handson, Leusden; Addendum 2). In this way, the seal is also made waterproof. On the bottom of the landing platform two protrusions with holes are constructed to attach the servomotor with zip ties (2.5 x 200 mm). Other structures are protrusions with holes to guide the cables coming from the infrared emitter and sensor attached to the central flower disk so that they don't hinder the servo-arm during refilling.

3.2.3 Central flower disk

Optimal foraging theory states that search time is very important to determine which prey will be targeted (Pyke et al., 1977). For bumble bees, the flowers are prey and the detectability depends largely on the size and the colour contrast it makes with the background (Spaethe et al., 2001). First, the time needed by a bee to find an artificial flower is affected by the size of this flower: search time is strongly negatively correlated to size of the flower (Spaethe et al.,

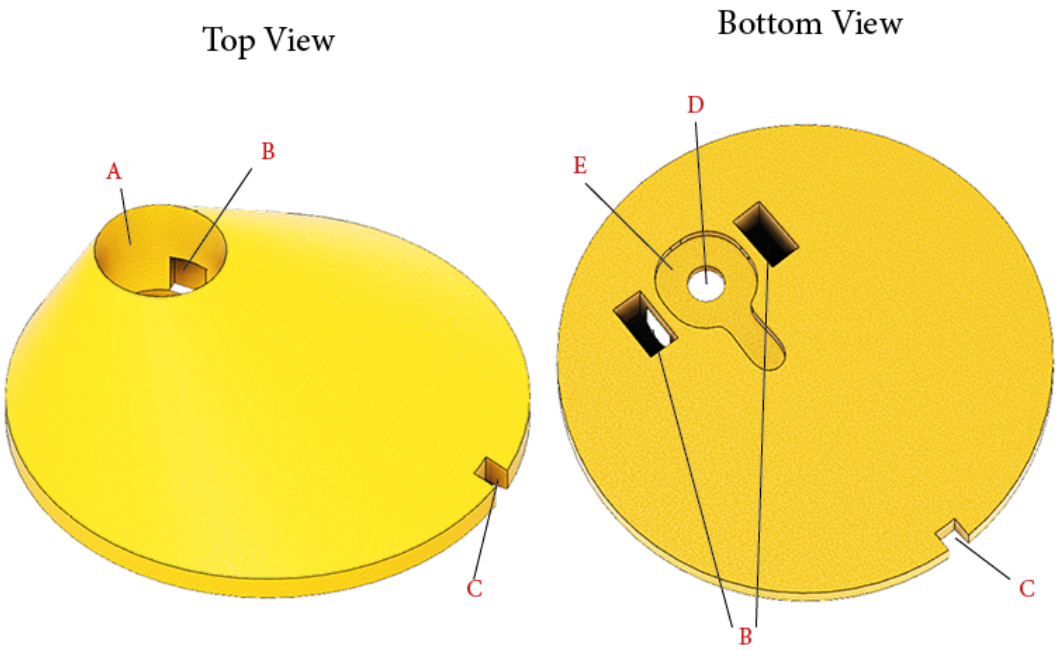


***Figure 5**: Top and bottom view of the central flower disk with **A)** feeding hole, **B)** notch for the infrared emitter/ sensor, **C)** notch to secure the right position in the landing platform, **D)** nectar hole and **E)** a notch shaped like the nectar cup.*

2001). In this robotic flower, the central flower disk (Figure 5) represents the detectable flower and was designed with a wide diameter (39,6 mm) to enhance detectability. Second, the central flower disk is also designed to contrast with the green background of the landing platform. To understand this contrast, we must look at the colour vision from the bee's (or Hymenoptera in general) perspective. Most Hymenoptera have trichromatic vision of colour, i.e., they have blue-, green- and UV-receptors working together to form an image (Chittka et al.,1994). Depending on the size of the flower, two systems of flower detection are used by bumble bees. If the flower subtends between 5° and 15° (α in Figure 6) only the green receptor is used, if an area bigger than 15° is subtended the bumble bee is able to recognize its colour (Spaethe et al., 2001). This means that in large flowers (≤15 mm) the colour contrast predominantly determines the search time, while in small flowers the green contrast will be more important. For example, Spaethe et al. (2001) showed that bumble bees could find large yellow flowers more quickly compared to large white flowers; with small flowers this was the other way around. For the design of this robotic flower field, the central flower disk was chosen to be bright yellow (human colour, described as 'lemon' by Spaethe et al.) as this gives the shortest search periods between two flowers for every diameter of flower tested by Spaethe et al. (2001). For flowers with a diameter equal to 28 mm, the largest tested diameter, the search time was only 2 seconds. Bright yellow results in a good green contrast as well as a clear colour contrast. Following the data from Figure 6 (Spaethe et al., 2001) a bumble bee would be able to start detecting (subtended at least 15°) the bright yellow coloured central flower (diameter of 39.6 mm) from a height of approximately 15.30 cm.

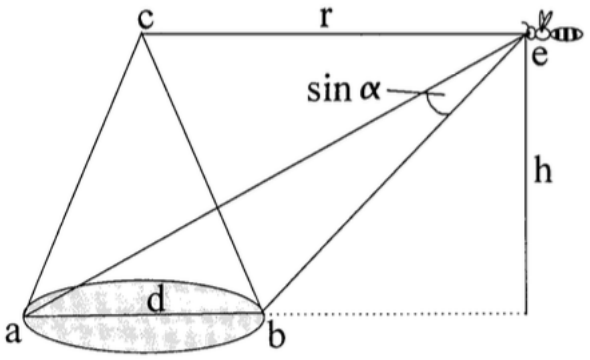


***Figure 6**: At a certain height (h), a bumble bee (e) can detect colour when the diameter (d) of a flower is big enough so that the subtended angle α ≥ 15° (Spaethe et al., 2001)*

The central flower disk also has other functions besides attracting the pollinators. First, a cylindrical hole is constructed in the central flower disk through wich pollinators can enter the flower to reach the nectar. This feeding hole was specifically designed for *B. terrestris,* but there is a large variation in size of their workers. Larger workers are more efficient in gathering nectar, so they are more likely to do the foraging while smaller individuals will often stay inside the nest (Spaethe & Weidenmuller, 2002). The feeding hole can thus not be too small for the large bumble bee workers to be able to enter it. Still, in contrast to the 13 mm in the previous version of Kuusela & Lämsä (2016), we chose a smaller diameter for the feeding hole in this new robotic flower. Based on experience by Pozo et al. (2020) and their supporting information

S4 that showed a mean worker thorax width of 6.97 ± 0.85 mm we chose to make the feeding hole 10 mm in diameter to reduce the interference of external light (e.g., from the sun) with the infrared sensor while still leaving enough space for the workers to enter. The depth of the feeding hole should be deep enough so that bumble bees are forced to crawl inside to reach the nectar and thereby causing a disturbance in the infrared barrier of the detection system. With the average length of bumble bee workers being 14.1 ± 2.4 mm (personal communication with Biobest, Westerlo, Belgium), the depth of the feeding hole was chosen to be 9 mm. This also gives enough space in the feeding hole to place the infrared sensor and emitter at the side and also protect the detection system partially from disturbance of incomming environmental light. At the bottom of the feeding hole, there is a nectar hole with a diameter of 3 mm and depth of 1 mm through which the bumble bees can reach the nectar cup that is pressed against the central flower disk by the servomotor. A notch shaped like the nectar cup is made at the bottom of the central flower disk to ensure a tight fit and a correct position of the nectar cup relative to the central flower disk. The conical shape of the central flower disk is to avoid rain running off from the landing platform in the feeding hole as much as possible, but the shape surrounding the feeding hole can be changed (see section 4.1 Uses).

3.2.4 Nectar cup

The nectar cup (Figure 7) has the function to present the discrete portions of artificial nectar to the visitors of the flower in the central flower disk. The volume of nectar varies substantially depending on the flower and interspecific and intraspecific variances can be large (Real & Rathcke, 1988). Not only the volume of nectar but also the amount of sugar (or concentration) that is offered varies considerably, with some species offering no sugar (e.g., *Papaver somniferum*) and other species offering up to 11312 µg sugar (*Impatiens glandulifera*) each day (Raine & Chittka, 2007). Raine & Chittka (2007) calculated the average amount of sugar production over 24 hours in 75 flowering plant species that are visited by *B. terrestris* in Bavaria, Germany. Measurements were done between 1999 and 2002 for 30 to 60 flowers per species over a period of three hours during which visitation was not allowed. After excluding the plant species that did not produce any sugar, the average sugar production was 735 µg per 24 hours. This value is here considered to be a representative value for other temperate regions outside Bavaria. Assuming the use of an artificial nectar with a concentration of 50% sugar, this would give a total volume of 1470 µL nectar each day. If the robotic flower has a basic refill rate of 10 minutes and is in operational mode (i.e., not in the sleep state) for a period of 14 hours (e.g., between 7 am and 9 pm), the nectar cup would need to have a volume of 17.5 µL to offer this 1470 µL of nectar daily. The nectar standing crop, or the amount of nectar that is present at a given time, in Mediterranean plant species around Cazorla (Spain) was on average 6.62 ± 3.05 µL in a sample of 32 plant species, belonging to five families (unpublished data, Pozo, 2021). For this robotic flower, we chose that the volume of the hole in the nectar cup was to be 25 µL, which might be more than the average production in many regions but evaporation

and crystallisation in the nectar cup needs to be compensated in this volume. By changing the 3D-design of the nectar cup, it is easy to adapt the discrete volume of the nectar presented to visitors of the robotic flower according to specific needs.

Another characteristic to consider here is the length of the proboscis of bumble bee workers, this because they need to be able to reach the bottom part of the nectar cup. The proboscis length varies among species and is an ecological important factor in assessing which species have access to floral rewards (Cariveau et al., 2016). The hole in our nectar cup has a diameter of 3 mm (to match the hole in the central flower disk) and depth of 3.5 mm. The thickness of the central flower disk at the point where it meets the nectar cup is 1 mm. This distance must be added to the depth of the hole in the nectar cup, making a total depth of 4.5 mm that pollinators must bridge to the bottom of the cup. This is easily reachable for *B. terrestris* (and *B. lucorum*) workers with a functional proboscis length of 8.1 ± 0.2 mm, *Apis mellifera* workers with 6.6 ± 032 mm and *B. pascuorum* workers with 8.9 ± 0.2 mm (Balfour et al., 2013).

The proboscis length of Lepidoptera in a study with 15 species by Corbet (2000) also showed to be sufficient to reach the nectar in the robotic flower, ranging from 7.34 ± 0.07 mm for *Lycaena phlaeas* to 16.19 ± 0.41 mm for *Inachis io*. Most Syrphinae (Diptera) have a proboscis that is relatively short related to their size, but some species, including *Eupeodes corollae* and

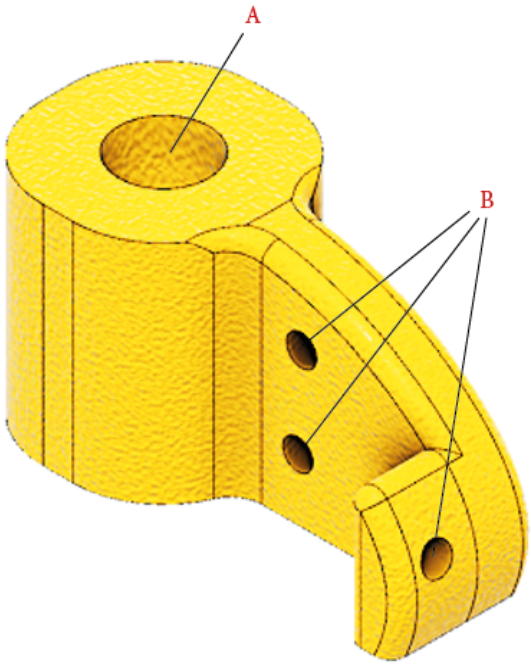


***Figure 7**: The nectar cup with **A)** the hole to present nectar in the central flower disk and **B)** three holes to attach the servo arm.*

Platycheirus manicatus have a longer proboscis (3-5 mm) (Branquart & Hemptinne, 2000). For these syrphids, the nectar cup might be too deep to reach all the way to the bottom, but after research this did not seem to stop them from visiting the robotic flower (see section 3.6 Proof of concept).

The nectar cup has three holes through which the part of the servo-arm, consisting of steel wire (diameter 0.8 mm), can be attached. The other end of the servo-arm is clipped on the spindle of the servomotor. Because the servo-arm partly consists of steel wire, it can be manipulated manually to achieve a close fit between the nectar cup and the central flower disk, this so that both holes are matched.

3.2.5 Nectar reservoir case

The nectar reservoir case (Figure 8) is built to carry the nectar reservoir while also forming a protective barrier between the electronic components at the bottom and the artificial nectar located at the top. The case is supported by beams on the stem and is kept in the right position by notches that fit in protrusions of these beams. Crescent notches at both sides of the nectar reservoir case ensure a passage for the cables running from the servomotor, infrared sensor and emitter attached to the bottom of the landing platform towards the electronics compartment at the bottom of the stem. A rim is provided on the edge of the nectar reservoir case to prevent spill-over nectar from the nectar reservoir to seep into the electronics compartment underneath.

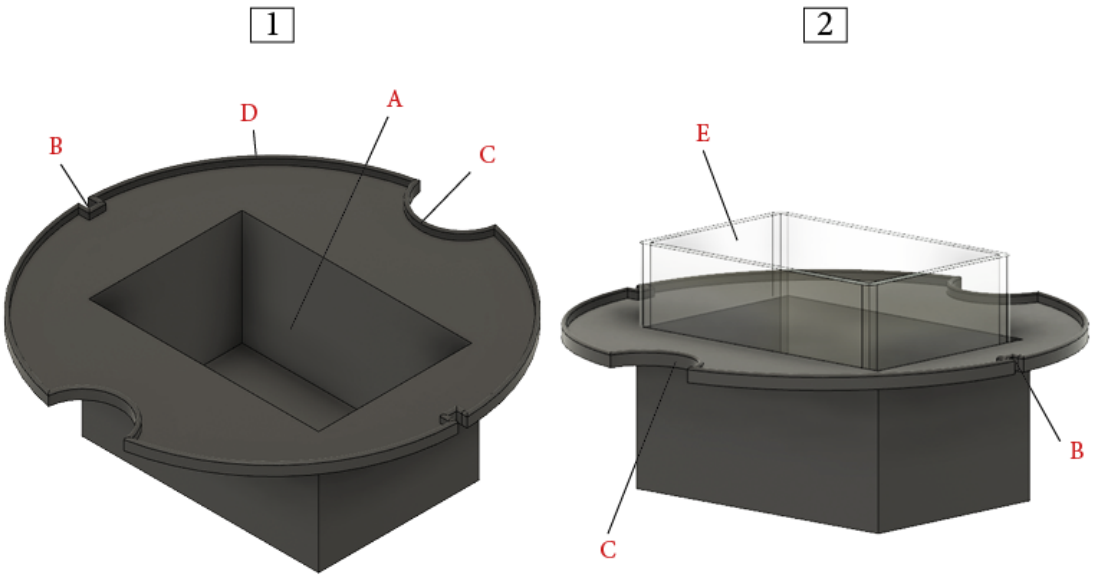


***Figure 8**: Nectar reservoir case without **(1)** and with nectar reservoir **(2)**. The figure shows **A)** the opening for the nectar reservoir, **B)** notches to fit in the support beam protrusions of the stem, **C)** crescent shaped notches to let cables go down from the landing platform, **D)** a rim to prevent nectar seeping to electronics part and **E)** the nectar reservoir.*

3.2.6 Nectar reservoir

The sole purpose of the nectar reservoir (Figure 8.2) is to keep the required amount of artificial nectar for the specific experimental set-up. For this prototype an off-the-shelf reservoir with inside dimensions of 56 × 36 × 44 mm (L × B × H) was used, giving it a volume of 88.7 ml. In a hypothetical set-up with a nectar cup of 25 µL that is depleted every time, a standard refilling rate of 10 minutes and in operational mode for 14 hours every day, the robotic flower would theoretically be capable of offering nectar for more than 40 days. This is when the nectar reservoir is filled up completely to the edge, ignoring spill-over and evaporation and assuming the servo-refilling system can scoop all the nectar down to the last drop. This is not a realistic scenario, but it shows that there is enough margin for an experimental set-up of e.g., two weeks. With long experimental periods other problems will probably arise, such as uncontrolled bacterial growth in the artificial nectar.

The reservoir used here (see Addendum 2 for more information) is made of polystyrene (PS), which means it is not autoclavable or dishwasher safe. The nectar reservoir can be single-use or reusable, in which case it should be washed by hand and can be sterilized using ethylene oxide gas sterilization or gamma irradiation. If another size is wanted, a nectar container could also be custom made, with the restriction that also the nectar reservoir case will have to be modified and keeping in mind that the servo-arm must fit in the reservoir when refilling. If lower volumes are needed, we suggest adjusting the width of the nectar reservoir as this will not drastically affect the design and additionally will lower the evaporation surface area. Another possible addition to minimize evaporation is using a customized lid partly covering the reservoir while leaving a gap for the servo-arm.

3.3 Internal hardware

The internal hardware (Figure 9) of the robotic flower consists of different electronic parts soldered onto a printed circuit board (PCB) and some separate parts: the servomotor, the battery with battery module and infrared sensor and emitter. The first prototypes were made on a breadboard, which allows to easily build and test circuits without soldering by just plugging in jumper wires between the electronics. Step by step new hardware parts were added during software development, until all required functionalities of the robotic flower were fulfilled. The last step was to design a PCB to replace the breadboard and all the jumper wires. Hardware parts were selected in such a way they are quite easy to solder manually for beginners. In the sections below the parts that were used are discussed. Addendum 2 contains an overview of the electrical scheme.

3.3.1 Microcontroller

A microcontroller (or microcontroller unit, MCU) is an integrated circuit (IC), typically with a microprocessor (or central processing unit, CPU), memory, and in- and outputs. The CPU contains the code (firmware) needed to interpret inputs and operate all the functions of the flower. It can be seen as the brains, making each flower able to operate autonomously. Modern computers often have more than one CPU, e.g., dual core or quad core with two respectively four CPU units. The implication is that different tasks can be performed at the same time by the different cores, i.e., multithreading. For this project, we used the open-source Arduino platform, which includes physical microcontroller boards that are very user friendly, but only have one CPU core. The more CPU is used, the more energy is spent. In this project one of the aims was to create an energy-efficient device which justifies the use of a single core. This means multithreading is not possible and while developing the firmware (see section 3.4 Firmware and functions of the robotic flower) this needs to be considered.

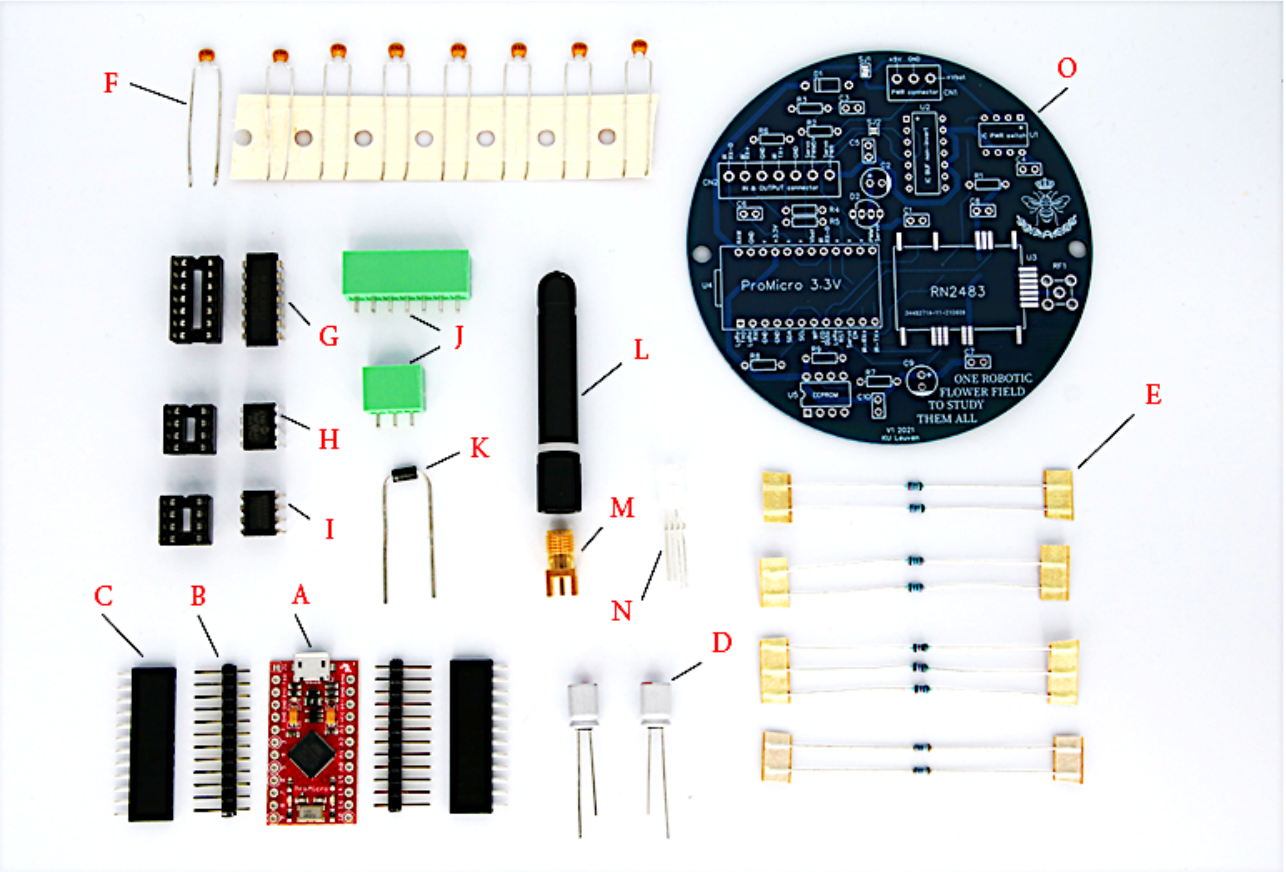


***Figure 9**: Overview of all the hardware parts of the robotic flower that are placed onto the PCB aside from the LoRaWAN module RN2383A-I/RM105 with **A)** Arduino Pro Micro microcontroller with micro-USB connection, **B)** 12 pin header strip with 2.54 mm (1 inch) pitch spacing, **C)** female connection header with 12 pins, **D)** 33 µF condensator, **E)** different value resistors (from top to bottom: 200 Ω, 2 KΩ, 10 KΩ and 100 KΩ), **F)** 100 nF condensators, **G)** level converter with matching plug-in socket block on the left, **H)** external EEPROM with matching plug-in socket block on the left, **I)** power switch with matching plug-in socket block on the left, **J)** connector sockets for in/output (top) and power (down) cables, **K)** diode 30V 1A, **L)** antenna, **M)** antenna connector, **N)** indication LED and **O)** PCB*

Microcontrollers have different kinds of memory: flash memory, random access memory (RAM) and electrically erasable programmable read-only memory (EEPROM). The flash memory and EEPROM are non-volatile, meaning that they are not erased when the power is down. The flash memory holds the firmware and the EEPROM can store some permanent data. In contrast, RAM is volatile, and data stored here will be gone when the microcontroller is shut down. RAM is only used by the CPU to store temporary data while it is computing.
There are many different types of microcontrollers on the market, each with their own specifications. The first prototype was built to get acquainted with the technology using an Arduino uno (Arduino, Italy). This is a microcontroller well-suited for beginners as it is robust and very well documented online. Later we switched to the The Things Uno (The Things Network, China), which consists of the Arduino Leonardo (Arduino, Italy) with an additional microchip LoRaWAN module. This board was specifically made to prototype IoT projects. Finally, the Pro Micro Atmega32U4 3.3V/8MHZ (SparkFun Electronics, United States) was chosen for its convenient small size, easy micro-USB connectivity (in contrast to the Pro Mini) and the present RX/TX hardware serial connections for the LoRaWAN module (Figure 9, A). We chose the Pro Micro 3.3 V with a clock speed of 8 MHz over the Pro Micro 5 V with a clock speed of 16 MHz to lower energy consumption. This implicates that the board will send 3.3 V over the output pins and that, consequently, a level converter is needed for some parts (see below). The voltage regulator on board can deal with voltage inputs up to 12 V, convenient to power the 3.3 V board with a single Lithium-Ion 3.7 V battery with 5 V step-up boost converter module. The Atmega32U4 processor (Microchip, Thailand) features 32 KB Flash memory, 2.5 KB RAM, and 1 KB EEPROM.

The Pro Micro microcontroller has through-hole connections for 12 GPIO (general-purpose input/output) pins on each side, so two header strips (Figure 9, B) are soldered here to allow the board to be simply plugged in and out of the PCB with connection headers (Figure 9, C), e.g., for a firmware update with USB connection. To further improve energy efficiency, the on-board red LED that is standard on the Pro Micro to signal that the board is powered was removed manually with pliers as it cannot be switched of using firmware.

3.3.2 Battery & battery module

As mentioned above, a rechargeable 18650 Li-Ion 3.7 V button-top battery with 3400 mAh (VABO NV, Heusden-Zolder) was used. The number 18650 is based on the size of the battery, which is 18 mm diameter and 65 mm in length. Being a button-top, the battery has a positive pole with a small protrusion in contrary to flat-top batteries that have a flat positive and negative pole. The batteries have a built-in protection circuit that protects the batteries from overcharging and over-discharging; both could indeed damage the cell and possibly even lead to a safety hazard (fire and/or explosion). Li-Ion cells should be handled with care. The 3.7 V gives the

nominal or average voltage, a fully charged battery will be maximum 4.2 V and in use the voltage will drop until the minimum of 3.0 V is reached (Figure 10; Chen & Rincon-Mora, 2006) . Charge time is approximately 5.5 hours according to the manufacturer data sheet.

To connect the battery to the microcontroller, a 5 V step-up boost Converter module for 18650 Lithium batteries with a micro-USB for charging (Aideepen, China) was used. This module is designed as an UPS (uninterruptable power supply) and converts the 3.7 V coming from the battery to a stable 5 V output, while it is also possible to power the microcontroller and charge the battery through the micro-USB connection. The clip-in part of the module was originally designed for flat-top batteries but was here manually modified with pincers to fit the button-top battery we selected. If used with flat-top batteries, no modification would be needed. The module also allows the installation of the rocker switch (E-switch, China) to switch the robotic flower on or off.

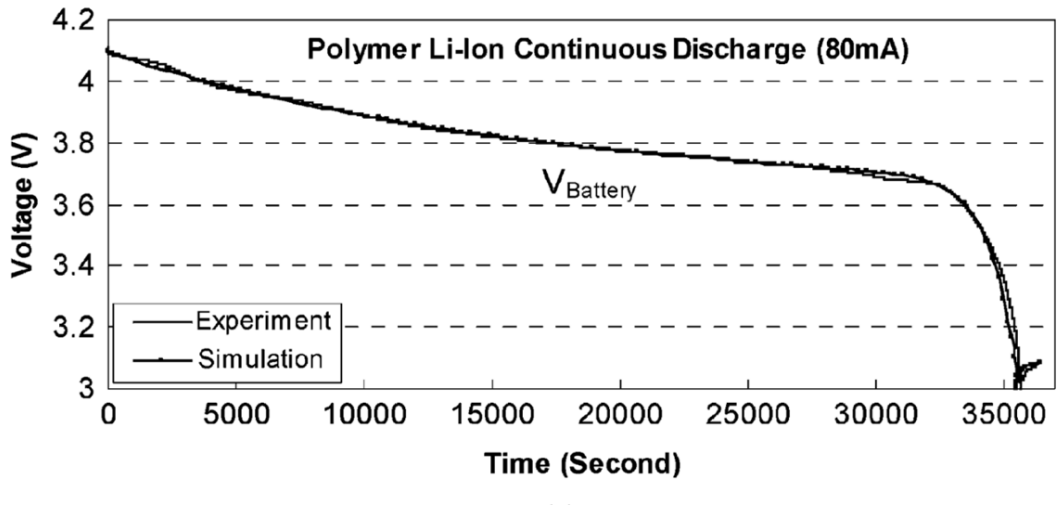


***Figure 10**: A typical voltage response curve, showing the maximum voltage around 4.2, the minimum at 3 and the nominal voltage around 3.7 for lithium-ion batteries (Chen & Rincon-Mora, 2006).*

The 3.7 V Li-Ion batteries that were used here have 3400 mAh (milliampère hour). This value represents the total amount of energy that can be stored in the battery, which combined with the mean energy consumption can give an indication of how long the battery will be able to power the robotic flower. First, we need to take the conversion of 3.7 V to 5 V in the battery module into account. As the wattage ($W = V \times I$, with V in volts and I in ampere) before and after conversion stays the same, the formula $I_2 = V_2/V_1 \times I_1$ can be used to determine the new value for mAh. In this case, that gives 2516 mAh, but there is a loss of energy during the conversion that must be considered. By taking a wide margin of 10% loss, the value used for further calculations of battery life is 2264 mAh. Next, the energy consumption of the robotic flower needs to be known. For this, we performed a 24-hour measurement, containing the work

state and the sleep state, to find the mean value for energy consumption of the robotic flower. This was done using Joulescope™, a precision energy analyser that can be connected to a computer and, among others, show changes over time like an oscilloscope (Figure 11). Joulescope™ was also used during software development to reduce energy consumption. Using parameters 10 pm and 6 am for going to sleep and waking up time respectively with 7 hours of deep sleep and refilling rate as well as sending rate at 10 minutes, the mean value for basal energy consumption (without visitors) over 24 hours was 4.1 mA. This value combined with the battery capacity yields a predicted battery lifetime of 19.56 days according to the Oregon Embedded online battery life calculator[1] which reduces the lifetime with 15% automatically to account for self-discharge of the battery. This value is an estimation, as several factors might influence the battery life, one of them being the ambient temperature in which the robotic flower is used. Li-Ion batteries usually have an optimal operating temperature between 15°C and 35°C, higher or lower temperatures may affect the battery performance negatively (Ma et al., 2018). For example, when using the robotic flowers in a glasshouse, it is good to know that high temperatures could lead to a reduced lifetime of the batteries.

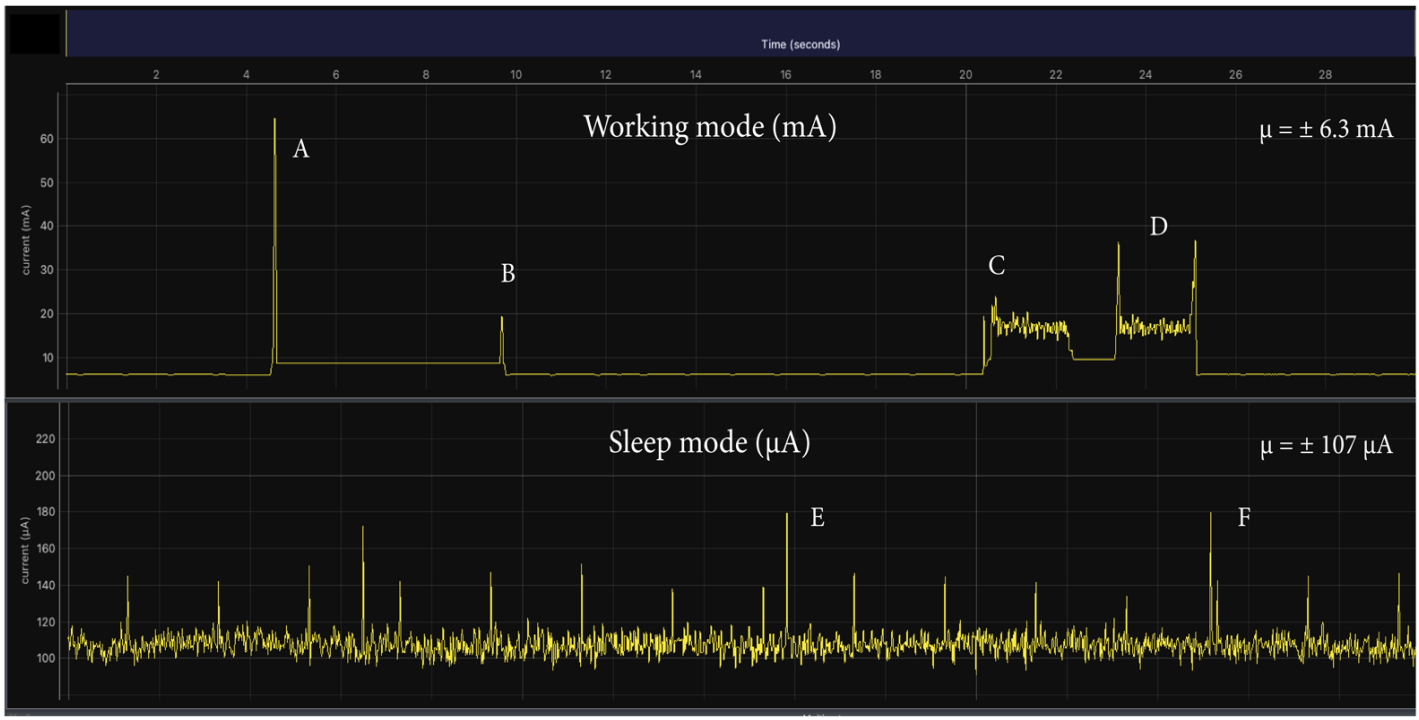


***Figure 11**: Energy consumption of the robotic flower during working mode and sleep mode with **A)** the spike of the LoRaWAN module sending data, **B)** the LoRaWAN module receiving a message after sending, **C)** the servomotor moving to the open position, **D)** the servomotor moving to the close position, **E)** and **F)** showing two energy consumption spikes during sleep mode. Made with Joulescope™.*

[1] https://oregonembedded.com/batterycalc.htm

The mean energy consumption during the work state (see section 3.4.4 Main loop) was around 6.3 mA without floral visitation. This value went up to 6.8 mA when a visit of variable length (but longer than the visit threshold of three seconds, see Table 2) was simulated every minute, which is logical as the servomotor will have to work more frequently and the LoRaWAN module will have to send larger data packages each time. In Figure 11, the two spikes in energy consumption during LoRaWAN communication are shown, A and B for uplink and downlink respectively. In between these two spikes, the LoRaWAN module stays awake, but after receiving the message from the backend application (spike B), the module goes back to sleep, and the energy consumption goes down again. The module is awake for approximately six seconds. The bottom line, for example between B and C, gives the energy consumption of the microcontroller and the IR detection system that is switched on shortly once every second. The periods of higher power consumption C and D are the servomotor that moves the arm to open and closed position with one second of staying still in the nectar reservoir in between these movements.

The sleep state (see section 3.4.4 Main loop) only had an energy consumption around 107 µA (or 0.107 mA), meaning that it effectively reduces power consumption overall. A longer sleep mode could significantly reduce the overall mean energy consumption of the robotic flower. Looking at Figure 11, the relatively larger spikes in power consumption, like indicated by E and F, are caused by the short moments in which the microcontroller wakes up approximately every 8 seconds.

3.3.3 Diode

A diode has the function to make sure the current does only flow one way by conducting very well in one direction and very bad in the other. Here, we used a Schottky diode 30 V, 1 A (STMicroelectronics, China), a necessary component in the electrical circuit to prevent current running from the microcontroller to the battery in the case the microcontroller receives power from the on-board micro-USB and from the battery module at the same time (Figure 9, K).

3.3.4 Servomotor

The servomotor we used here was a 9 g 180° Micro Servo (DFRobot, China) with operating voltage 4.8 - 6 V. The microcontroller, with operating on 3.3 V, cannot supply the required voltage to power the servomotor. Therefore, the power to drive the servomotor comes from the battery module with its 5 V output. The reported stall torque is 1.6 kg*cm at 4.8 V, meaning that the servo has enough torque to withstand a force of 1.6 kg at a 1 cm radius from the shaft. With a servo-arm roughly 5 cm long in the robotic flower, the servo should be able to lift 320 g which is more than enough for its use in the robotic flower. The weight of the servo-arm together with the nectar cup (volume 25 µl) was measured to be only 0.97 g. The servo has three cables, one for the 5 V power supply, one for ground and the last one is for the pulse width

modulation (PWM) signal telling the servomotor which position it must take. By the duration of the pulse signal, the rotor knows the position it must move to. The servomotor works on 5 V PWM signals, which cannot directly come from an output pin of the Pro Micro 3.3 V. Therefore, the 3.3 V signal coming from the microcontroller needs to be converted to a 5 V signal that can be interpreted by the servomotor with a level converter (see below). When the wanted position is reached, the servomotor will keep using some power to hold that position. To avoid this, a power switch (see below) is included so that a signal of the microcontroller can turn off the power supply of the battery to the servomotor. We also want to mention that the low-cost and -weight servomotors built with plastic gears are susceptible to wear and tear. For this application the selected type of servomotor should be sufficient, but for more extensive uses it might be advised to look for a similar servomotor (but more expensive) with metal gears.

3.3.5 Indication LED

The addition of a LED (light emitting diode) to the hardware is mostly useful in developing the firmware, as this LED can be programmed to signal actions in the microcontroller that otherwise are not visible, e.g., when the microcontroller is in sleep state (Figure 9, N). We used an addressable RGB LED (SparkFun Electronics, China), which means the colour of the led can be chosen through a data pin on the LED, not with a PWM signal like the servomotor but with a carrier signal that not only determines the colour but also the brightness. These LEDs need a supply of 4.5 to 6 V, and the data signal is also expected to be 5 V. For this reason, the LED is powered by the 5 V coming from the battery through the power switch just like the servomotor. This means that if we want the LED to work, the power switch needs to be activated first. The 3.3 V carrier signal coming from the Pro Micro, is converted to a 5 V carrier signal by the level converter analogue to the servomotor PWM signal. In the working mode, this LED has not much function and will be switched off most of the time to save energy.

3.3.6 Level converter

The Buffer, Non-Inverting 4 Element level converter (Texas Instruments, Malaysia) needs to be powered by the battery through the power switch because it has an operating voltage range of 4.5 to 5.5 V, and the microcontroller can only deliver 3.3 V (Figure 9, G). Signal voltages coming from the microcontroller are 3.3 V and pass through the level converter that changes them to a 5 V signal to control the servomotor and the indication LED.

3.3.7 Power switch

The power switch, an N-channel MOSFET (Texas Instruments, Mexico), has an input of 5 V from the battery module and, if switched on, gives a 5 V output to power the level converter, LED, and servomotor (Figure 9, H). A 3.3 V signal from the microcontroller can switch the power switch off and on, making it possible to only power these hardware parts when they need

to work. This is an important feature in making the robotic flower as energy efficient as possible.

3.3.8 Infrared emitter & sensor

The infrared (IR) emitter and sensor are both light emitting diodes (LED) placed in the feeding hole (see Figure 2) and are responsible for detecting visitors of the robotic flower. We used an Infrared (IR) Emitter 940 nm, 1.2 V, 50 mA, 60° Radial (Everlight Electronics Co Ltd., Taiwan) with a corresponding Phototransistors 940 nm Sensor, Side View Radial (Everlight Electronics Co Ltd., Taiwan). The emitter sends out a beam of IR-light with a wavelength (940 nm) bigger than that of visible light (380-700 nm). This beam is caught by the emitter on the other side of the feeding hole, and a voltage is produced and passed on to an input pin of the microcontroller to be interpreted.

3.3.9 Resistors

Electricity going through a wire can be compared to water flowing in a hose: the voltage being the pressure on the water, the current being the size of the hose and the resistance being a clod in the hose. All three variables affect each other and together they determine how much water will run through the hose. This is described in Ohm's law: $V = I \times R$, with V for voltage in Volt (V), I for current in Ampère (A) and R for resistance in Ohm (Ω). By influencing the resistance, the voltage and current in the circuit can be changed. In this project we used four different resistors (Figure 9, E): 200 Ω ±1% 0.5W Axial Metal Film (Vishay, South Korea), 2 KΩ ±1% 0.5W Axial Metal Film (Vishay, South Korea), 10 KΩ ±1% 0.5W Axial Metal Film (Vishay, India), 100 KΩ ±1% 0.4W Axial Metal Film (Vishay, Czech Republic). They were used for two reasons, first to reduce the current flow going in hardware parts and second to adjust the voltage of signals coming into the microcontroller.

Several parts in the electrical circuit have a specific maximum current, and will thus need a preliminary resistor to reduce the current going in. An example for this is the IR emitter, having a maximum current rating (I_{max}) of 50 mA. This means that to avoid damage the current going through the IR emitter should not be higher than this value. To reduce the amount of current running through, a resistor is used. We aim to a current of 10 mA, which is well below the I_{max} but still high enough for the IR emitter to function well. To calculate which value of resistor is needed to achieve the wanted current, we need to know the voltage running through the IR emitter. In the datasheet of the IR emitter, we find the forward voltage (Vf). This is the amount of voltage that is lost in the diode, in this case it is 1.2 V. Thus, of the 3,3 V coming from the output pin from the microcontroller and going in the IR emitter, 2.1 V comes out again. To calculate the value of the resistor that would be suited, we then make use of Ohm's law: $(2.1\ V)/(0.010\ A) = R$. The result is that we should use a resistor around the value of 210 Ω. We therefore opted to use a 200 Ω resistor.

An example for the second use of the resistors, reducing the voltage of incoming signals, is in the voltage divider that is needed to allow the microcontroller to read the remaining voltage of the battery. The battery gives a maximum of 4.2 V when fully charged, but the microcontroller can only handle inputs up to 3.3 V. Therefore, the voltage coming from the battery must first be divided by two and later doubled again by using software on the microcontroller. The voltage divider works by putting two resistors in series which gives the following equation derived from Ohm's law for the output voltage in function of the input voltage: $V_{out} = V_{in} \times (R_2/(R_1 + R_2))$, with R_1 and R_2 the value of the serial resistors. Here we made the voltage divider with two 100 KΩ resistors, distributing the input voltage equally over both with as result in an output voltage of 0.5 times the input voltage. For example, if the input voltage is 4.2, the microcontroller will receive a 2.1 output voltage.

3.3.10 Capacitors

Capacitors, also known as condensators, are used to smooth out the current in the electrical circuit (Figure 9, D). By storing electrical charges and releasing them when there is a shortage in the circuit, capacitors act as a buffer. For example, when the servomotor starts to rotate, it will draw a lot of charge creating a peak that can be supported by a capacitor placed closely in the electrical circuit. For this project we used a 33 µF 20 V Aluminium Polymer Capacitor, Radial, Can, 49 mΩ (Nichicon, China) for the servomotor and the LoRaWAN module as these components can create peak energy draws. Multiple 0.1 µF (or 100 nF) ±10% 50 V Ceramic Capacitor X7R Radial (Kemet, Mexico) are placed in the circuit as well to smooth out smaller fluctuations.

3.3.11 LoRaWAN module

The LoRaWAN transceiver module is the hardware part responsible for the low-power and long-range wireless data transmission with radio frequency (RF) technology (see Figure 1). We used the RN2483A-I/RM105 (Microchip Technology, Thailand), but the RN2483A-I/RM104 (Microchip Technology, Thailand) can be used as well as the only difference is a firmware update on the module. The module operates on 3.3 V and can thus be directly powered by the microcontroller. The TX serial connection of the microcontroller sends signals to the LoRaWAN module to be transmitted. The RX serial connection of the microcontroller receives the signals that are picked up by the LoRaWAN module. The transceiver module operates in the 433 MHz and 868 MHz frequency bands, but here the latter is used as this frequency is less likely to get disturbed by other signals during transmission. To send and receive RF signals, an antenna able to detect the right frequency is needed, here the Straight RF Antenna 855MHz ~ 880MHz -1.5dBi SMA Male Connector Mount (Linx Technologies Inc., China) was used (Figure 9, L). To connect the antenna to the PCB, we used an SMA Connector Receptacle, Female Socket 50 Ohm Through Hole Solder (Molex, Taiwan; Figure 9, M).

3.3.12 External EEPROM

As the internal EEPROM on the Pro Micro is very limited (only 1 KB, or 1024 bytes), we provided an external EEPROM (Figure 9, H). The 512Kbit I2C Serial EEPROM (Microchip Technology, United States) has memory space of 64 KB that is available to store data. For this prototype we did not use it, but it could be a practical addition for future research purposes. For example, it could be used to securely save data locally to prevent a permanent loss when the data package fails to be successfully transmitted over LoRaWAN.

3.3.13 Printed circuit board

A printed circuit board (PCB) contains all the electronic connections among the hardware parts (Figure 9, O). The PCB consists of a combination of layers non-conducting material with etched conductive copper tracks. Besides traces the PCB also contains through-hole pads, which connect the copper layers and allow the assembly of through-hole hardware parts by soldering. Multiple layers of copper circuitry are possible for complex designs, but our PCB is relatively simple and has only two copper layers. On top, there is also a silk layer, which contains text to indicate assembly instructions. The design (Figure 12) was made using the online software by EasyEDA, which also provides a direct link to JLCPCB for fabrication of the PCB that was used. While designing the custom PCB for the robotic flower, some aspects had to be considered. First, and most important is the placement of the parts on the PCB. Parts that belong together should also be places close on the PCB. For example, the two 33 µF capacitors are placed close to the LoRaWAN and servomotor. Second, as much traces as possible should be in the first layer. By working with via's, small holes connecting the two layers, a trace can be made to subvert another in the second layer and then come back to the first layer. This reduces the surface of traces on the second layer. The second layer can then be filled completely with copper, with exception of an outline around the through-hole pads, via's and traces in the second layer. In this way the second layer forms an uninterrupted copper surface that is as big as possible (blue part in Figure 12). This surface is used to connect the ground, or negative pole of all the hardware parts so that no traces are needed. This is important to assure the best

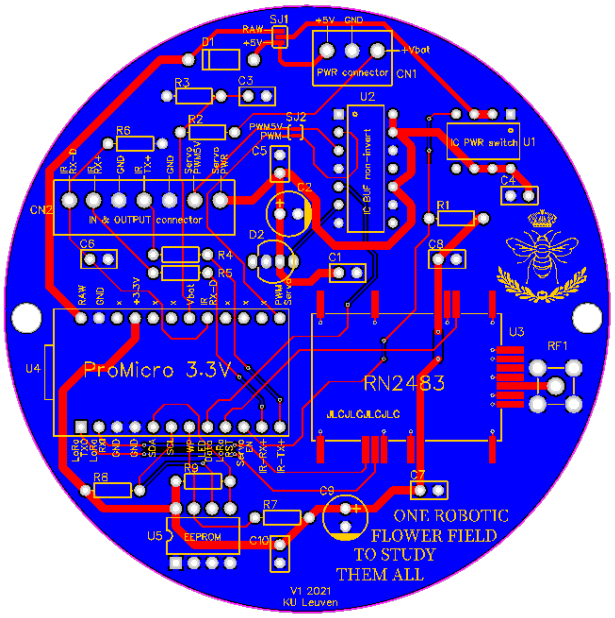


***Figure 12**: Design of the PCB with the top copper layer in red, the bottom copper layer in blue and the silk layer in yellow.*

functionality of the RF antenna that needs a large ground plane. Third, routing angles of traces should be 45°, mostly for conventional reasons. Fourth aspect is the trace width, which is usually denoted with the unit mil, that equals one thousandth of an inch or 0.0254 mm. For signal traces we used 10 mil, which is sufficiently large. Power traces on the other hand should be as big as possible for more efficiency, that's why they were given a 45 mil width. Fifth, most components were chosen to be through-hole for easy manual assembly but the LoRaWAN module is in that respect an exception, as it is a surface mount device (SMD). For this reason, the PCB contains SMD pads on the places where the LoRaWAN module must be connected. Sixth, to fit the cylindrical protrusions of the PCB-supports in the stem of the robotic flower, the PCB must have two holes at the right location. Seventh and last, we provided two solder bridge jumpers (see SJ1 and SJ2 in Addendum 3). A solder bridge jumper is a trace that is interrupted by two or more pads that can be bridged (or not) by soldering the pads together to form the desired confirmation of the PCB. In this way, the same PCB can be used for different uses or with other hardware choices. SJ1 consists of three pads (Figure 13). The power switch (middle pad) needs to be connected to one of the two other pads to receive power. Depending on the use of the PCB with the microcontroller powered by a battery (connect to bottom pad) or by the micro-USB (connect top pad). In our case, we used a battery, meaning the middle pad and bottom pad needed to be connected using solder to get a fully functioning robotic flower. SJ2 is made to use in the case a servomotor was used that works with a 3.3 V PWM-signal, making the level converter irrelevant. If SJ2 is bridged, the level converter should not be installed anyway. That's not the case for this prototype, so SJ2 is left untouched. In short, the solder jumper bridges make the PCB more versatile operable.

The production of the PCB was done by the manufacturer JLCPCB, who offers rapid and highly reliable production for small-batch orders. The production parameters include base material FR-4, two layers, delivery as single PCB with thickness 1.6 mm, surface finish HASL (with lead) and outer copper weight 1 oz. The colour of the solder mask can be selected, here it is blue with white silkscreen.

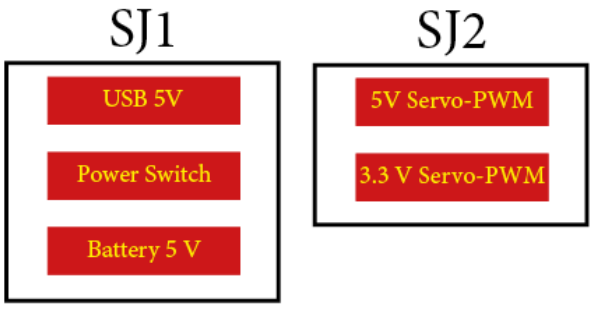


***Figure 13**: Configuration of both solder bridge jumpers on the PCB of the robotic flower.*

3.3.14 Connectors

The last parts that need to be soldered onto the PCB are the connectors. There are two connector sockets, the power connector, and the in/output connector. They allow the connection of cables coming from the battery, servomotor, IR emitter and IR sensor. A 3 Position Terminal Block

Header, Male Pins (3.5 mm), Vertical Through Hole (Molex, China) was used, fitting a 3 Position Terminal Block Plug, Female Sockets (3.5 mm) 90° Free Hanging (In-Line) (Molex, China) for the power connection. The power, ground and battery level line are connected here. The in/output connector has seven inputs: ground, power and PWM-signal of the servomotor, power and ground of the IR emitter, and power ground/data from the IR sensor. We used a 7 Position Terminal Block Header, Male Pins (3.5 mm) Vertical Through Hole (Molex, China) with the corresponding 7 Position Terminal Block Plug, Female Sockets (3.50mm) 270° Free Hanging (In-Line) (Molex, China). To allow the plugging in of the microcontroller with the pin header strips, two rails of the 12 Position Header Connector (2.54 mm) Through Hole Tin (Sullins Connector Solutions, Taiwan) were soldered to the board. To plug in the level converter, external EEPROM and power switch we used IC plug-in sockets: 8 (2 × 4) Positions DIP (7.62 mm) Row Spacing Through Hole (Assmann WSW Components, China) and 14 (2 × 7) Positions DIP (7.62 mm) Row Spacing Through Hole (Assmann WSW Components, China). To connect the rocker switch (E-switch, China) a Connector Header Through Hole 2 position (2.50 mm) (JST sales America Inc., China) was soldered to the battery module and used in combination with Quick Connect Wires 2.54 mm JST-XH connector and Tab Connectors Type 4.8 mm. The result of all the parts soldered to the PCB can be seen in Figure 14.

3.3.15 Complete flower

The cost of a complete flower is around € 116 (Table 1), VAT (Belgium) included, delivery costs excluded. The most expensive parts are the 3D-print (ca. € 30), the LoRaWAN module (€ 15.70) and the microcontroller (€ 18.26). There is a margin to go cheaper but this comes with a trade-off, e.g., by reducing thickness of the 3D-print it is possible to reduce the weight and thus the price, but at the same time the firmness will be reduced. The electronic hardware parts can also be selected paying more attention to their price, but it is possible that the reliability will be lower if cheaper variants or imitations are used. In general, the more parts are bought, the cheaper they get. The prices in Table 1 are approximations for the number of parts to construct one flower. If more robotic flowers were to be constructed, the price could be reduced. The final cost will also fluctuate according to the VAT in different countries.

To fully assemble all the hardware parts and the housing to make a complete robotic flower, some other materials are needed that are not included in Table 1. First, steel wire to make the servo-arm that connects the servomotor with the nectar cup. This wire has a diameter of 0.8 mm to fit the assembly holes in the nectar cup. Second, transparent silicone (Handson, Leusden; Addendum 2) is used to fixate the central flower disk to the landing platform and seal the rocker switch (E-switch, China) in the stem. The same silicone was also used to fixate the IR emitter and IR sensor in the provided cavities in the central flower disk. It is important for the silicone to be transparent to let the IR-light bundle go through. Third, solder tin to connect all the parts to the PCB. Fourth, wires to connect IR sensor and IR emitter to the PCB and to make a connection between the battery module and PCB to read the battery level. Fifth, tie wraps used to tie up the servomotor to the bottom side of the landing platform. Sixth, conformal coating

***Table 1**: Overview of the costs to produce one robotic flower, VAT (Belgium) included, delivery costs excluded.*

Part	Price (€)	Amount/ flower	price/ flower (€)
Pro Micro 3.3 V	€ 18.26	1	€ 18.26
40 pin Header strip	€ 1.00	1	€ 1.00
LoRaWAN module	€ 15.17	1	€ 15.17
Antenna	€ 8.55	1	€ 8.55
Antenna Connector	€ 3.56	1	€ 3.56
Servomotor	€ 3.68	1	€ 3.68
indication LED (per 5)	€ 3.00	0.2	€ 0.60
Rocker switch	€ 0.68	1	€ 0.68
Rocker switch connector JST-XH male	€ 0.14	1	€ 0.14
Quick connect cables (per 10)	€ 5.95	0.1	€ 0.60
IR emitter	€ 0.39	1	€ 0.39
IR sensor	€ 0.35	1	€ 0.35
Power switch IC	€ 2.58	1	€ 2.58
Level converter IC	€ 0.47	1	€ 0.47
External EEPROM IC	€ 2.14	1	€ 2.14
Diode	€ 0.42	1	€ 0.42
Resistor 200 Ω	€ 0.19	2	€ 0.38
Resistor 2 KΩ	€ 0.19	2	€ 0.38
Resistor 10 KΩ	€ 0.19	3	€ 0.57
Resistor 100 KΩ	€ 0.22	2	€ 0.44
Capacitor 100 nF	€ 0.24	8	€ 1.92
Capacitor 33 µF	€ 0.79	2	€ 1.58
Connection headers female (12 pos.)	€ 0.80	2	€ 1.60
Power connector male (3 pos.)	€ 1.09	1	€ 1.09
Power connector female (3 pos.)	€ 1.16	1	€ 1.16
In/Out connector male (7 pos.)	€ 2.12	1	€ 2.12
In/Out connector female (7 pos.)	€ 2.48	1	€ 2.48
Plug-in IC socket (8 pos.)	€ 0.19	2	€ 0.38
Plug-in IC socket (14 pos.)	€ 0.23	1	€ 0.23
Battery	€ 10.00	1	€ 10.00
Battery module	€ 1.84	1	€ 1.84
PCB	€ 1.00	1	€ 1.00
Nectar reservoir (per 10)	€ 8.00	0.1	€ 0.80
3D-print	€ 30.00	1	€ 30.00
		TOTAL:	**€ 116.56**

spray PLASTIK 70 (Kontakt Chemie, Switzerland) to protect the outside of the 3D-printed components to make them more waterproof. The quick drying spray is based on acrylic resins forming a colourless and transparent layer. Three layers of coating were applied on the surfaces that can be in contact with moisture, e.g., rain falling on the landing platform during outdoor use of the robotic flower. Also, the bottom of the PCB, after soldering, is given a coating to protect against moisture, avoid short circuits and prevent electrical leakages as the spray also has insulating properties.

3.4 Firmware and functions of the robotic flower

To build the firmware of the robotic flower, we used the integrated development environment (IDE) Visual Studio Code (VSC) with the PlatformIO extension as an alternative to the Arduino IDE to develop software for the Pro Micro microcontroller. The software is written in the programming language C++, but the Pro Micro physical environment is limited and therefore not all the standard C++ functions can be used. The written source code is compiled first and can then be uploaded to the microcontroller with a micro-USB connection. Compiling is done by a compiler, a program that translates the source code to target code that can be interpreted by the microcontroller and executed. The compiler throws overboard every bit of the source code that is not necessary, for example the comments in the code added as information to support developers (lines starting with '//').

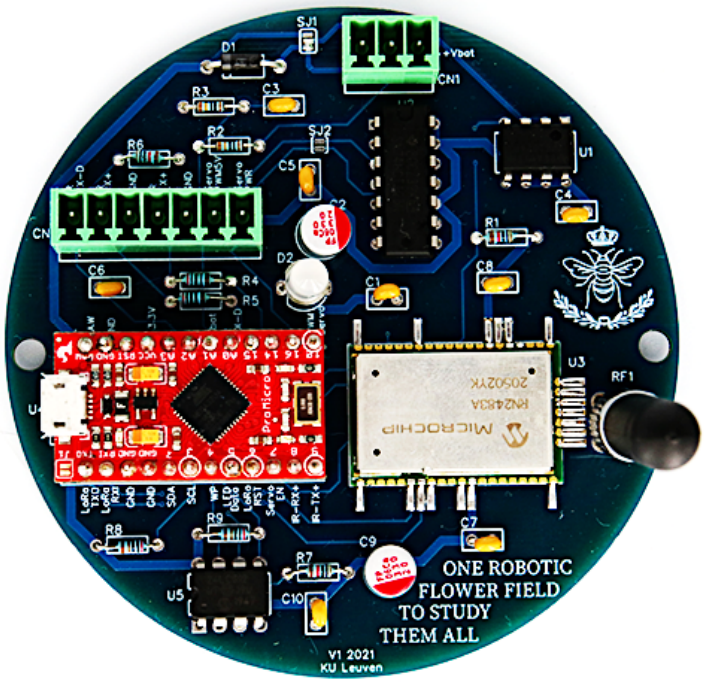


***Figure 14**: The result of soldering and plugging all the hardware parts to the PCB, including the surface mounted LoRaWAN module.*

***Table 2**: The adjustable variables with their prototype value and explanation.*

Variable name	Prototype value	Explanation
SLEEP_TIME	25200	time of deep sleep during sleep state (in ms) → 7h
SENSOR_SENSITIVITY	3	sensitivity of IR sensor
MAX_VISIT_DURATION	250	maximum time a visit is recorded (in sec)
PRECISION	1000	divide by 1000 to go from ms to sec
CYCLE_TIME	1000	one visit detection every cycle possible (in ms)
ON_TIME	1	time during the cycle the IR system is on (in ms)
OBSTR_TRESH	3	determines when to give obstruction alarm
SERVO_OPEN	90	position of servo, nectar cup in nectar reservoir
SERVO_CLOSE	0	position of servo, nectar cup in central flower disk
SERVO_SPEED	10	speed of servomotor movement
REFILL_GAP	30000	time between a visit and a refill (in ms) → 30 sec.
REFILL_TIME	1000	time the nectar cup remains in nectar reservoir (in ms)
VISIT_TRESHOLD	3	minimum time of visit before refill is triggered (in sec)
AUTO_REFILL_TIME	600000	automatic refill interval (in ms) → 10 min.
WORK_SEND_FREQUENCY	600000	send interval, work state (in ms) → 10 min.
SLEEP_SEND_FREQUENCY	900	send interval, sleep state (in sec) → 15 min.
GOING_TO_SLEEP_SEND_FREQUENCY	120000	send interval, going-to-sleep state (in ms)→ 2 min.
RESERVED_VISIT_SPACE	100	maximum number of visits that can be stored in RAM
RETRY_AMOUNT	5	times the LoRaWAN module retries to send
NORMAL_FLASH_INTERVAL	5000	LED flash interval, battery normal (in ms) (DEV only)
WARNING_FLASH_INTERVAL	1000	LED flash interval, battery low (in ms) (DEV only)
FLASH_TIME	100	time LED is on, only in dev. mode (in ms)
WARNING_VOLTAGE	3.2	voltage threshold level for alarm

The firmware consists of three files that need to be compiled together and uploaded to the microcontroller. The 'main' code can be found in the C++ source code file (.cpp), this is the file telling the microcontroller what to do. The two header files (.h) contain information that needs to be included in the main code: the 'variables' and the 'fixed_variables'. In this way, the declarations of the adjustable variables are put in a separate file making it more convenient to adjust them and change the functionality of the flower. Simply change the value for a variable in the header file, compile it with the other files and upload to the microcontroller. The adjustable variables with their prototype value and explanation are listed in Table 2. For example, if a standard refilling rate of 15 minutes is preferred over the 10 minutes in the prototype, the variable 'AUTO_REFILL_TIME' needs to be changed from 600,000 ms to 900,000 ms. The adjustable variables can be recognized by the all-capital format with underscores. The fixed variables could have been included in the main source code but are put in a separate header file to keep it more orderly and readable. There is also a fourth file available with development configuration variables: 'variables_dev'. This file can be included in the main source file instead of the normal variables (for the operational configuration) to get a different setting of the robotic flower function. These setting include full operation of the indication led, shorter intervals (e.g., sending and refilling) and debug printing in the serial

monitor. The serial port enables serial communication during development via USB with the computer so that the software developer can follow what is happening using printed statements. Including the 'variables_dev' makes it is easier to understand the functioning of the robotic flower and can be useful while developing or getting to know the firmware.

When a program is uploaded to the microcontroller after compiling, it is stored on the flash memory of the CPU. Every time the microcontroller is powered, e.g., by switching on the battery power with the rocker switch, the program will start running. When the power is lost, the microcontroller will stop running and all the information in RAM is lost, acting like a reset. The program always starts with a set-up loop, which is only run once after powering up. Every time the robotic flower is switched on, the flower begins to run the set-up once and after that continues with the main loop, which keeps running as long power is supplied.

To get connected to DingNet, an application must be constructed on the DingNet console webpage after registering and logging in. The same procedure applies when using the original The Things Network console instead of using the KU Leuven customized environment DingNet. Once the application is created, the extended unique identifier (EUI) of the application (16 characters) can be chosen and an application key (32 characters) generated. Next, all the devices must be added using their unique device EUI (16 characters). The DingNet backend will need this information to send information received from the devices to the right application.

The main source file is built up in different parts. The first part at the top begins with including all the libraries and header files that are needed for the program to run. Next, functions and structures that are used by the robotic flower are declared. After that, there is code for the set-up that runs only once at the beginning followed by the main loop code. In the following text, we will focus on the logic behind the main source code with references to the lines where corresponding code can be found. The complete main source code can be found in Addendum 4, where there is also a link to a public GitHub repository containing all the files needed to have complete functioning firmware. We recommend using Addendum 4 next to the following text.

3.4.1 Includes

In the beginning of the main source code (Addendum 4, lines 1-15), different libraries are included to expand the functions that can be used specifically for the application that is needed. These libraries were downloaded before and are added to the project by including them here. The 'Arduino' library is the first one to be included and is needed to work with the Arduino IDE in PlatformIO. The 'FastLed' library (Daniel Garcia) offers functions to direct the indication LED, 'VarSpeedServo' (David Garrison) is a library used to direct the movement of the servomotor, 'rn2xx3' (JP Meijers) gives functions to communicate with the LoRaWAN module. The library 'Queue' (SMFSW) is designed to work with queues, a type of container

adapter we will use to save the visit data in the robotic flower. To allow some form of energy management in the microcontroller, we used the library 'Low-Power' (Rocket Scream Electronics). Before including 'uDebugLib' (Naguissa), which allows to block serial debug lines from compiling, first the header files are included because they contain information needed to successfully include the library.

3.4.2 Functions and structures

After the needed libraries and header files are included, the main source code contains declarations of structures and functions that will be used in the set-up and/or main loop. The functions form a loop that can be called inside another loop. For example, the functions needed to make the connection with DingNet will be called during the set-up code when the robotic flower is switched on.

Structuring internal EEPROM

The first time a new robotic flower is connected, the parameters that are needed to make connection with DingNet (or any other IoT network) must be stored in a permanent memory of the microcontroller. The internal EEPROM is used to this purpose. A structure is needed to put this information. The structure was named 'eepromConfig' (Addendum 4, lines 133-139) and it contains four elements: the number of times a configuration has been written to the EEPROM, the device EUI (16 characters), the application EUI (16 characters) and the Application Key (32 characters). Room for an extra character is provided in DevEUI, AppEUI and AppKey. This will be used for the null terminator ('\0'). To write something to EEPROM, another function is needed, for that purpose the 'configWrite' function (Addendum 4, lines 142-146) was created. Every time this function is used, the number of writes will go up by one.

The 'configWrite' function is used in the next function where the EEPROM configuration is initialized: 'configInit' (Addendum 4, lines 149-161). The DevEUI is a unique number and is set by requesting it from the LoRaWAN module directly. To initialize the EEPROM configuration, the AppEUI and AppKey are filled up with a string of corresponding size, 16 and 32 characters respectively, containing only zeros. Because they are null-terminated hexadecimal strings, an extra 0 (or '\0') is added at the end in the extra position that was created before. The microcontroller can use this null termination as a reference to determine the length of the strings.

The 'configRead' function (Addendum 4, lines 164-172) allows to read what was stored in the 'eepromConfig' structure that was constructed before. The 'configInit' function is called here, but only if the EEPROM has not been initialized before. If that is the case, the number of times the EEPROM has been written is set to 0.

Connection with DingNet

In order to make connection with DingNet, the parameters need to be set first. They will be stored in the EEPROM as was mentioned before, but we need functions that will allow us to enter them in the system through a serial connection with USB. The first function, 'menuPrint' (Addendum 4, lines 176-191), will show a menu in the debug window with the possible actions (Figure 15).

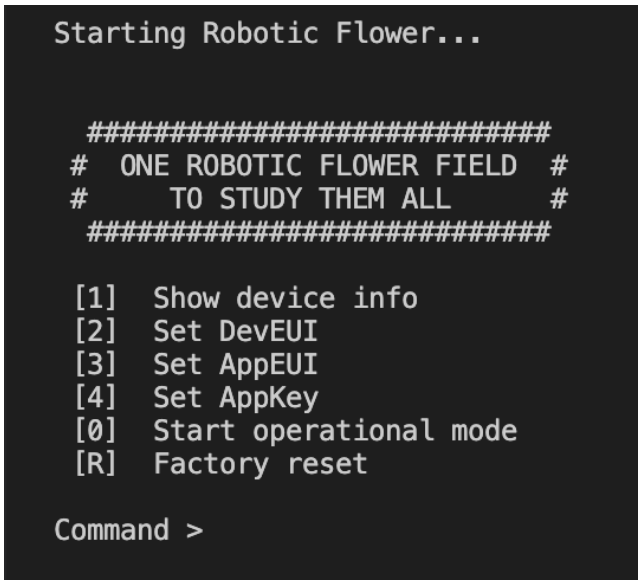


***Figure 15**: The menu that is printed in the debug window by the function 'menuPrint'.*

A function 'inputReadline' (Addendum 4, lines 197-245) is used to read the input in the debug window and another function, 'verifyHexstring' (Addendum 4, lines 230-245) to check whether the input is a hexadecimal string of the expected length. These two functions are both used in the function 'serialHandler' (Addendum 4, lines 249-349) that is used to handle the input of the user and store the parameters in the EEPROM for configuration. The function starts by printing the menu and then looks at the command entered by the user with the 'inputReadline' function. The length of the input is checked to be just one character as expected (see 'menuPrint'). If this is not the case, a message is sent on the debug window to alert the user of an invalid input. If the input is valid, the function continues to the right action asked for by the command. For example, if the command was '1', the debug window will show the parameters that are currently stored on EEPROM. It is possible the DeVEUI remains empty if the microcontroller was not able to communicate with the LoRaWAN module yet. A solution might be to do a factory reset by typing 'R' in the input window. If the command was '2', the user gets the possibility to enter the DevEUI manually, but this should not be necessary as it was automatically requested previously from the LoRaWAN module during the 'configInit' function. The other parameters, being the application EUI and the application, can be set by using the other commands, e.g., with input '3', the user can set the AppEUI and store it on EEPROM. When all the parameters are saved, the user can switch to operational mode by typing '0' in the debug window.

Sending with LoRaWAN

First, structures need to be constructed so that the data that will be sent with LoRaWAN can be stored. Recording of the visits uses the internal clock time to register the start of the visit, which is stored together with the duration of the visit in the structure 'strVisitRec' (Addendum 4, lines

22-26). The internal clock time at the start of a visit will later be used to calculate how many seconds have passed between the start of the visit and the moment a data package is sent. At the application backend, running on Node-Red, this information will then be converted to the actual time the visit started in the form of 'hh:mm:ss'. The data of one visit thus consists of two variables: start time and visit duration. The start time needs 4 bytes, as it is an unsigned long. The visit duration takes only 1 byte of memory because the maximum visit duration (see Table 2) was set at 250 seconds. One byte has 8 bits, and each bit gets appointed the value 0 or 1. Therefore it is possible to store 2^8, or 256 different values in one byte (from 0 to 255). These two variables determining a visit take up 5 bytes of space combined and will be called a 'visit element'.

A container was made to reserve space in the RAM memory so that the visit elements can be put there (Addendum 4, line 29). The container was then converted to a queue with the first in first out (FIFO) principle (Addendum 4, line 32). This means that the visit elements are added at the end of the queue and are taken away at the front. The queue also has no overwrite, so that once the queue is full, nothing will be added anymore. If overwrite was chosen to be true, new visits would be saved at the expense of the oldest visit present in the queue when the reserved visit space is filled entirely with visit elements. The size of this reserved space is determined by the variable RESERVED_VISIT_SPACE, which determines the maximum amount of visit elements that can be stored. The Pro Micro has 2.5 KB of RAM, which equals 2560 bytes. In the prototype the reserved visit space was kept at 100, meaning that 500 bytes of RAM are blocked. The other bytes are kept free to assure good functioning of the microcontroller, as RAM is the workspace of the CPU. More space can be reserved for visit elements, but 100 elements should be sufficient as the elements that are regularly sent over LoRaWAN will be removed from the queue clearing space. If the container is made too large, there is a chance of malfunction in the microcontroller due to a stack overflow: the program tries to use more RAM than is available on the CPU.

Programmed system alarms also need a structure so that they can be stored properly, (Addendum 4, lines 34-41). These alarms all are stored as booleans, meaning the value can only be false/true or 0/1. Because they only need one bit of space, all the alarms are stored together in 1 byte to save space in the data package. The first alarm signals that the queue is full, meaning there were too many visits so that the container is completely full. When this happens, two variables can be changed to fix it: more space on RAM could be reserved, or the data could be sent over LoRaWAN more often. The second alarm is given when a visitor was in the robotic flower at the moment of data transmission, meaning the visit duration was cut off. As there is only one CPU, the robotic flower is not able to do multiple tasks at once. Sending data and simultaneously registering floral visitors is not possible. The third warning is triggered when

the battery is almost empty. This warning gives the user the chance to change the battery during the night and restart the flower if needed. The fourth and last alarm is triggered if the maximum visit time has been reached a certain number of times in a row. This could indicate that a visitor is sleeping or dead inside the flower, or that another object is continuously blocking the IR sensor. These four alarms can easily be extended, as one byte has 8 bits and only 5 bits are currently used. Besides that, existing alarms can also be modified to indicate other factors.

The uplink message structure (Addendum 4, lines 43-47) consists of two variables, the battery voltage, and the alarms. Together these variables are stored in 3 bytes that are sent every time (see further). The data message structure (Addendum 4, lines 49-53) consists of the time a visit started before the data were send and the visit duration, together 5 bytes. These are sent only if the flower registered any visitors in between two data sending events.

The 'SENDdata' function (Addendum 4, lines 56-128; not to be confused with 'SendData', see further) is first assigning values to the right variables of the uplink message structure before converting this message to a hexadecimal string and putting it in a buffer. Next, as long as the buffer still has room, the first visit element is taken from the queue repeatedly and converted to a hexadecimal string before being added to the buffer. When the start time of the visit is assigned to the data message structure, it is converted to the time (in seconds) that has passed since the visit started until the time of sending (Addendum 4, lines 86). The LoRaWAN module has limited data rate, and therefore the message size cannot be too large. We adopted a maximum message size of 53 bytes, being 3 bytes for the uplink message structure (alarms and battery voltage) and 50 bytes for data message structure (equals 10 visit elements from the queue). When the buffer is full or the visit queue is empty, the buffer will be sent with the LoRaWAN module. If the data transmission was successful, the module receives a confirmation and information from the backend application saying whether it is time to sleep or not for the robotic flower. If no confirmation message was received, the data will be sent again. This will be repeated several times, depending on the adjustable variables RETRY_AMOUNT. For the prototype, this was chosen to be five times. The total send function takes about 6 seconds unless multiple transmission attempts are needed. During this time the flower cannot do anything else.

Read Battery

This function (Addendum 4, lines 352-366) is used to obtain the voltage level of the battery, making it possible for the user to stay aware of the lifetime of the robotic flower through the frontend application and intervene if necessary. The voltage coming from the battery goes through the voltage divider (see section 3.3.9 Resistors) to be halved and is read from an analogue pin. The analogue to digital converter has 10 bits and will thus grade the incoming value on a scale from 0 to 1023 (2^{10} possibilities). The analogue pin value that is read will thus be the incoming voltage (theoretically between 0 and 3.3) divided by 1024. The microcontroller

can then multiply the obtained value with 2 and $\frac{3.3}{1024}$ to derive the current voltage level in the battery. By using a multimeter, the voltage output on the output pin mas measured to be 3.27 V in practice rather than 3.3 V. This value was used to read the voltage level of the battery more correctly. Moreover, after further calibration with the multimeter, a correction factor was set at 1.005797. If the battery voltage is lower than a threshold value, e.g., 3.2 V, a warning is given.

Refill and Automatic refill

The first thing this 'refill' function (Addendum 4, lines 368-381) does, is to turn on the power switch, so that the servomotor and level converter are provided with 5 V power coming from the battery. Next, a command is given to move the servo to the open position, rotating the servo arm so that the nectar cup gets dipped in the nectar reservoir. When the servo is at the open position, there is a short delay to allow the nectar cup to be filled before it is moving up again. The position of the servo when closed, the position when open and the delay time can all be changed in the adjustable variables, just like the speed of the servomotor. We chose a slow movement of the servo-arm to make sure the nectar cup is getting fully filled without the formation of air bubbles while also lowering the energy consumption of this action and reducing noise and vibration. After a visit that has passed a certain threshold duration, the refill function is activated (see section Work state in 3.4.4 Main loop).

Nectar depletion in the nectar cup because of evaporation over time is prevented by the 'AutomaticRefill' function (Addendum 4, lines 383-390). This function triggers the 'refill' function on a regular basis. In the prototype we chose an automatic refill interval of 10 minutes, but this can be very easily changed. In a warmer experimental setting, a shorter interval time might be desired, or a longer interval could also be selected to save battery energy. The program is constructed in such a way that the automatic refill is prevented when a visitor is inside the flower. After every refill, also the ones after a visit, the timer for the automatic refilling is reset.

Battey Led flash

The indication LED has the function to show the status of the robotic flower. In this 'Battery_LEDflash' function (Addendum 4, lines 392-435) the LED is programmed to flash during the working state in development mode. The flash will be green when the battery level is normal, but it will change to red and flash at higher frequency when there is a battery warning. This is useful during development, but consumes too much energy and could possibly influence visitor behaviour in the operational mode.

Save to RAM

When the end of a visit is detected, the parameters visit time and duration need to be saved together as a visit element and put at the back in the queue. The function 'saveToRAM'

(Addendum 4, lines 437-469) makes this possible. In the development configuration, the function also shows a blue flash of the LED when the visit is saved.

Send data

The function 'SENDdata' that was discussed above does the configuring of the message and the actual sending by communicating with the LoRaWAN module. The function 'sendData' (Addendum 4, lines 476-528) contains the 'SENDdata' function with some more functionalities. The most important extra functionality is the regular interval at which the data is sent, here chosen to be once every 10 minutes. The LoRaModule needs to be woken up first, as it is put to sleep during the interval to reduce energy use. Next, the 'read_battery' function is executed so the current battery voltage can be sent to the backend application. The function also checks if a visitor is in the flower at the moment of sending, and if so, terminates the visit registration and saves parameters of the visit element. After that, the actual sending of data starts using the 'SENDdata' function. In development configuration, the indication LED will shine yellow during sending. When a confirmation is received, or the transmission has failed after the chosen amount of retries, the LoRaWAN module is put back to sleep until it is woken up again right before a new sending event. The timer is reset at the end of the function, to keep track when the function should be executed again.

3.4.3 Set-up

The set-up (Addendum 4, lines 531-657) is run once at the beginning of the program. Here, variables such as timers and alarms are initialized, and pin modes of the microcontroller are defined as inputs or outputs. The indication LED is attached and immediately shows a red light at the start of the set-up. The code continues with resetting the LoRaWAN module so that it is ready for use. When the microcontroller is connected with an USB cable, the serial monitor is started to have communication with the computer. The pin mode for the power switch is defined and the switch is turned on by setting the output level low. At the end of the set-up, the level on the pin is written high to stop power going to the servomotor, indication LED and level converter. The robotic flower will start in the 'wakeUp' state (see section 3.4.4 Main loop).

The functions to make the DingNet connection in the set-up are essential to the well-functioning of the data transmission. The frequency plan of the LoRaWAN module is set to the Europe 863-870 MHz, other regions have a different frequency plan. If there is a serial connection available with USB, the parameters to make the connection can be inserted using the 'serialHandler' function that was discussed earlier. The indication LED that was turned on at the start of the set-up is turned off again. The robotic flower will wait for 10 seconds to see if there is a serial connection available. If not, the microcontroller continues to try and make a connection with the current stored parameters in EEPROM. The first time, the robotic flower needs to be configured with a serial connection, but after that the rocker switch just needs to be turned on

and the robotic flower will start functioning without any input. Now, the robotic flower will start attempting to make a connection with over the air activation (OTAA) and this will be repeated until DingNet was successfully joined. The indication LED will be shining red during OTAA and will flash green when a connection is established. Now the robotic flower is almost ready to begin. The only thing left is attaching the servomotor. The servo-arm will quickly go into the nectar reservoir and stay there for 30 seconds before slowly returning to the feeding hole when the robotic flower is already running in the main loop.

The set-up is terminated by turning off the USB module on the microcontroller (Addendum 4, lines 648-653) if not in development configuration. This is done to save battery, as the USB connection is only useful for development. The USB module will be turned on again when restarting the flower, so that the user is able to upload firmware updates.

3.4.4 Main Loop

The main loop (Addendum 4, lines 660-897) contains the code that is run repeatedly until the robotic flower is switched off by cutting the power. The principle of looping is simple: the CPU reads through the lines of code, starting from the top and executes everything it is ordered to do. When the bottom of the sketch is reached, the loop starts again from the top. Inside the main loop, small loops can be found, for example a 'for' loop or a 'while' loop. These loops give a condition, and if this condition is met, the loop is executed. In this code, there are many timed events, e.g., refilling and sending over LoRaWan. To realise this, these loops are used in combination with a timer, which allow to let actions happen if a certain time has passed. To construct these timers, we use the internal clock of the microcontroller, which gives the time that has passed in milliseconds (ms) since the power was turned on. This value is stored as an unsigned long, meaning it has 32 bits available with a range from 0 to 4,294,967,295 (2^{32}). When converting milliseconds to days, it can be calculated that the microcontroller can keep track of the internal clock for 49.7 days, and in fact even more because the timer freezes when the robotic flower is in sleep mode during the night (see further). When the maximum is reached, the clock will start again from 0, which could cause some timed systems in the code to fail. In the current conformation, the battery would be empty long before these approximately seven weeks have passed (see section 3.3.2 Battery)

The main loop is split up according to the four states of the robotic flower: wake-up, work, going-to-sleep and sleep. As was defined in the set-up, the robotic flower starts in the wake-up state and then continues to the work state when it is ready. When a message is received from the backend application that it is time to go to sleep, the flower will first go to the going-to-sleep state. Here, the robotic flower prepares to go to sleep before switching to the sleep state which is maintained until a message tells the flower it is time to wake-up again.

Wake-up state

At the start of working, the robotic flower begins to run through the wake-up state (Addendum 4, lines 738-766) to prepare itself for the work state that follows. The refill gap timer is started and will refill the flower 30 seconds later (see work state below). Other timers are also started, and a first IR sensor value is measured so that it can later be used to compare the new IR sensor value. At the end of the wake-up state, the state is changed to work.

Work state

The work state (Addendum 4, lines 769-896) is the state where all the functions come together. During the work state, the functions 'sendData' and 'automaticRefill' are running all the time making sure the visit data is sent over LoRaWAN with a fixed interval and that the nectar does not evaporate or crystalize inside the nectar cup. In the 'sendData' function, a port can be specified. This is some extra information about the sending event that will be received by the backend application. If the port is set to 1, the backend application knows there can be valuable visit data in the message. When the port is 2, as is the case when sending at the end of sleep state, the backend application does not need to look for data in the message. Every time the robotic flower sends a message, it also receives a message containing information on whether it is time to go to sleep or not. If the answer is yes, the state will switch to going-to-sleep.

To make the visit detection with the IR system energy efficient, a cycle of one second is constructed, in which only a small fraction of the time is used to switch on the IR emitter an IR sensor. The IR sensor is switched on once every second and one millisecond later the IR emitter is switched on. A delay of one millisecond later the value of the IR emitter is measured and again one millisecond later the IR emitter and IR sensor are switched off. In this way, the IR sensor is only on for three milliseconds per second, the IR emitter only for two milliseconds. This means the IR detection system only uses energy 0.3% of the time, while it is still able to detect visitors with a precision of one second. This accuracy was deemed sufficient in this case and has been the same in the robotic flower field built by Kuusela & Lämsä (2016), but can be altered if needed. The visit duration is measured by subtracting the internal clock at the end of the visit with the internal clock at the beginning. This is then transferred from milliseconds to seconds and rounded (Addendum 4, line 831), meaning a visit that lasted less than 0.5 seconds is not recorded. If more accurate measures are needed, the cycle time can be shortened, e.g., if a precision of 0.1 second is desired the cycle time can be changed to 100 milliseconds in the adjustable variables. A shorter cycle time will be at the expense of battery lifetime. To further improve the energy efficiency, the microcontroller checks only once every 100 milliseconds if the cycle time has passed (Addendum 4, line 893). When a shorter cycle time was chosen to increase precision, it is recommended to lower this delay value as well for accuracy.

A visit is detected if the new measured value of the IR emitter has dropped a certain amount (the sensor sensitivity) relative to the previous measured value. This system in which the new value is compared to the previous measured value was brought into being to compensate the fluctuations in the surrounding light conditions. The sensor sensitivity can be adjusted so that it is able to notice a visitor, but it should also not be too sensitive (see section 3.6 Proof of concept). The maximum visit duration was set at 250 seconds, after which the visit is terminated, and a new visit starts. Previous experience with a robotic flower field (M. I. Pozo et al., 2020) showed that a bumble bee staying this long is probably sleeping or dead inside the robotic flower. During a visit, the automatic refill function is prevented so that the floral visitor is not disturbed. When a visit that lasted at least the time of the chosen threshold ends, a refill timer is started to refill the nectar cup after a certain time chosen as refill gap. This gap between visit and refilling is necessary to avoid learning behaviour of the bumble bees (Cnaani et al., 2006). If the robotic flower replenished the nectar directly after a visit, the bumble bees could get lazy and just wait on the flower until the next meal is served in the feeding hole. For the prototype, this value was chosen at 30 seconds. The automatic refill timer is also restarted when there is a refill triggered by a visit.

Going-to-sleep state

Before entering the going-to-sleep state (Addendum 4, lines 663-690), the visit queue is checked. If the queue is not empty, it will be emptied by sending messages with LoRaWAN until nothing is left to send. In this way, the visit container in RAM is ready to be filled up again the next day and data cannot get lost during the sleep state of the robotic flower if a problem would arise with power supply. When this is done and all energy consuming parts are switched off, the robotic flower is ready to go to the sleep state.

Sleep state

The main idea of the sleep state (Addendum 4, lines 693-735) is to save energy at night, when the flower is not getting visitors anyway. There is no refilling of nectar, no visit detection, and less communication with LoRaWAN. The sleep state begins with a long, deep sleep. This is done with a looped power down function (Addendum 4, line 708), that puts the microcontroller to sleep each time for 8 seconds. The sleep time may deviate around 10% as the microcontroller cannot accurately keep track of time when sleeping. When the deep sleep is over, the microcontroller will also go in a repeated 8 second sleep loop but will now be sending a LoRaWAN message at regular intervals to be able to receive the command to go to wake-up state when the time has come. The times for going to sleep and wake up are selected by the user in the frontend application and can be changed at any time (see section 3.5 Application software; Figure 20) take deep sleep time into account that must be set before flower is used). For example, the flower can be told to go to sleep at 10 p.m., and wake up at 6 a.m., which gives a total sleep of 8 hours. Of the total sleep of 8 hours, 7 hours can be deep sleep, where the

robotic flower will use the least energy possible (see section 3.3.2 Battery). If the flower is set to send for example every 15 minutes, the latest moment the flower will wake up is a quarter of an hour past the set wake up time.

3.5 Application software

An application was built to monitor all the robotic flowers that are in use and access the gathered data. This application consists of a backend and a frontend, and it is made with Node-Red, a browser-based programming tool that allows to create flows with JavaScript functions. By using the different nodes that can be wired together, a complex flow can be made in a visual organised way. The backend application receives the information and data from the LoRaWAN backend (see Figure 1) and processes it. The backend also generates the frontend application or dashboard, that can be accessed via html and has four tabs: 'Battery', 'Alarms', 'Sleep time' and 'Files'.

The backend application consists of different flows: one flow handling the downlink and uplink data per flower (Figure 16), and one flow for logging the data of all flowers in CSV-files (Figure 18). The uplink and downlink data flow starts with a node that receives the information that is

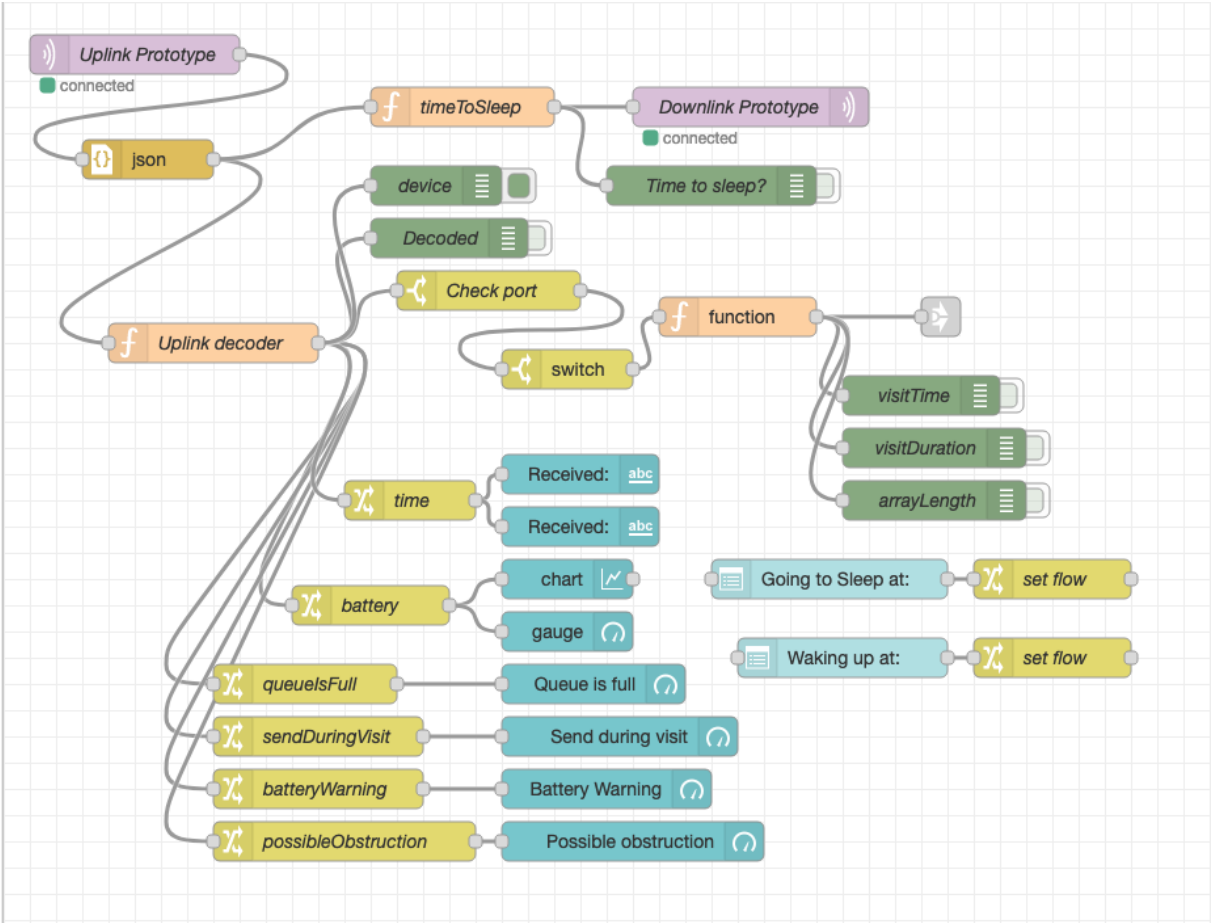


***Figure 16**: The uplink and downlink data flow, here the one for the prototype robotic flower, which receives data, processes it, and sends a message back to the robotic flower. Besides that, it also creates different tabs of the frontend application.*

send by the robotic flower over LoRaWAN through the DingNet backend with data broker. The incoming package is then translated and forwarded to the next nodes in the flow. One line of the flow will prepare a downlink package containing the information to let the robotic flower know it is time to go to sleep or not. The other split-off continues to the function node ‘uplink decoder’, where the information that was received is further interpreted bit by bit.

```
1.  // uplink decoder
2.
3.    // topic: "v3/robotic-flower2/devices/flower-prototype2/up"
4.
5.  var buf = Buffer.from(msg.payload.uplink_message.frm_payload,'base64');
6.  var decoded = {};
7.
8.  decoded.battery = buf.readUint16LE(0)/100;
9.
10. var binary = buf.readUInt8(2).toString(2).split('').map(function(s) { return parseInt(s);
    });
11.
12. decoded.queueIsFull = binary[4]; //bit 4
13. decoded.sendDuringVisit = binary[3]; //bit 3
14. decoded.batteryWarning = binary[2]; //bit 2
15. decoded.possibleObstruction = binary[1]; //bit 1
16.
17. if (buf.length > 3){
18.     var timeBeforeSend = {};
19.     var visitDuration = {};
20.     for (let i = 1;i < ((buf.length-5)/5+1);i++) {
21.         timeBeforeSend[i] = buf.readUint16LE(i*5);
22.         visitDuration[i] = buf.readUint8(i*5+2);
23.     }
24. }
25.
26. var newmsg = {};
27. newmsg.application = msg.payload.end_device_ids.application_ids.application_id;
28. newmsg.device = msg.payload.end_device_ids.device_id;
29. newmsg.port = msg.payload.uplink_message.f_port
30. newmsg.time = msg.payload.received_at;
31. newmsg.payload = decoded;
32. newmsg.dataSize = buf.length;
33. newmsg.timeBeforeSend = timeBeforeSend
34. newmsg.visitDuration = visitDuration;
35.
36. return newmsg;
```

Form this function, the obtained information is split up and divided over the different nodes. The visit data is sent to the file logger flow, the other information about battery voltage and the status of the alarms is passed on to the tab ‘Battery’ and the tab ‘Alarms’ respectively in the frontend application. This flow also creates the tab ‘Sleep time’, in which the hours are set between which the robotic flower must be in the sleep state.

The data logger flow receives the visit data from all the flowers that are in use and combines adds it to a CSV-file stored on the database. Every day, a new CSV file is created that contains a column for time, duration of the visit and the device from which the data was received (Figure

17). If one big CSV-file, containing the data of multiple days is preferred, this can be changed in the ‘filename generator’ function node.

Addendum 5 contains the JSON (JavaScript Object Notation) of both kind of flows, which can be imported in Node-Red just by copy and pasting it. Be aware that some functions must be adjusted before they can function in other systems than the one made here. For example, the topic of the downlink- and uplink-node must be changed to the right application and device name.

RoboticFlowerField_2021

time	duration	device
18:19:37	5	flower1
18:20:09	5	flower1
18:25:30	49	flower2
18:31:01	199	flower2

***Figure 17**: Example of the CSV-file that is created every day by the backend application. For every visit, the data, duration, and which flower was visited is saved here.*

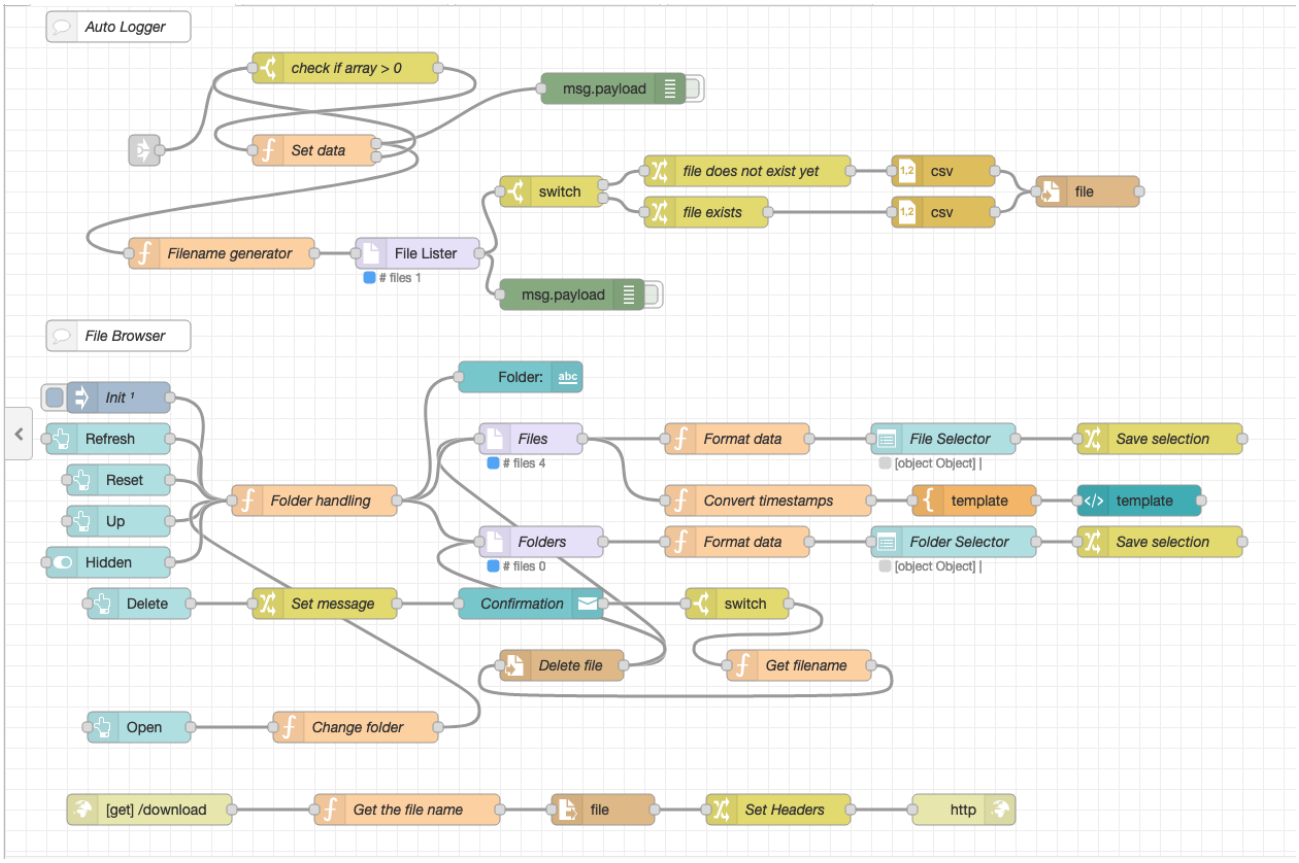


***Figure 18**: The data logger flow. This flow saves the incoming data of all the robotic flowers in a new CSV-file every day and creates a file browser tab in the frontend application.*

The frontend application allows the user to monitor all the robotic flower that are used in one site. In the tab ‘Battery’ (Figure 19), the current voltage of the battery for each device is shown with a gauge together with a graph showing the progression of the voltage over time. The other tabs (Figure 20) show the alarms, give the user the option to change the hours of the sleep mode and download the CSV-files to the local machine.

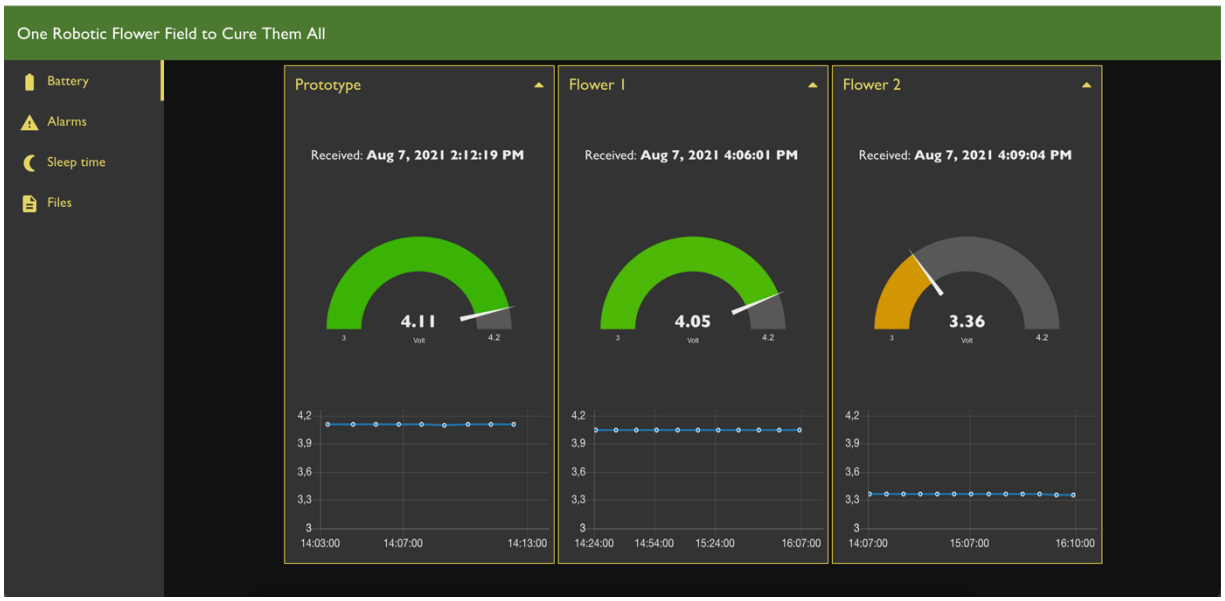


Figure 19: *Frontend application with 'Battery' tab opened.*

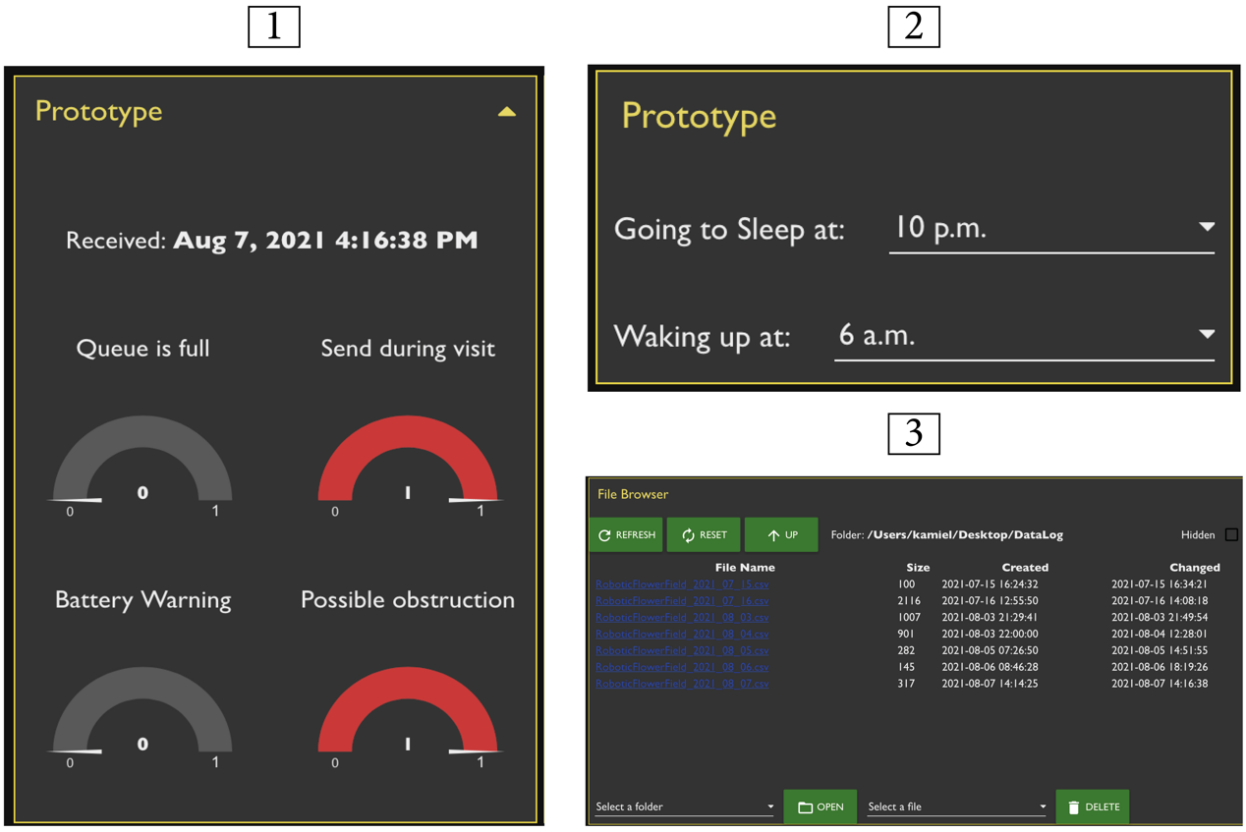


Figure 20: *Views of different frontend application tabs for the prototype with* ***1)*** *'Alarms',* ***2)*** *'Sleep time' and* ***3)*** *'Files'*

3.6 Proof of concept

A proof of concept was performed to test whether the robotic flower does what it was designed to do. Multiple insects in different settings were used on July 16, 2021, at the Botany building (PLTK) of the KU Leuven in Heverlee. The question was if the robotic flowers could attract visitors by offering artificial nectar and if so, correctly register the visit data. Therefore, the CSV-file with visit data of the robotic flower had to be compared to the manually collected visit time and visit duration. Two robotic flowers were used of which the nectar reservoir was filled with Biogluc (50% *w/w*) sugar syrup (composed of 25.0% sucrose, 37.5% fructose, 34.5% dextrose, 2.0% maltose, 1% higher sugars and 0.05% preservatives) made by Biobest (Billiet et al., 2016).

As a first setting, the robotic flower was placed in the garden. Floral visitors were recorded in an open field setting, being a small botanical garden. When the visitor went in the feeding hole, the probing time and duration were noted. The captured individuals were identified and besides *B. terrestris*, also *B. pascuorum*, *Anthidium manicatum*, *Apis mellifera, Psythirus* sp., Syrphidae, Bombyliidae and even Lepidoptera showed interest in the robotic flower at least to a certain extent. The garden setting of the proof of concept was not very successful as many visits were not or only partly recorded by the robotic flower (Addendum 6, Garden). In retrospect, this was probably because of a wrong value for the adjustable variable 'SENSOR_SENSITIVITY' was used. The sensitivity of the IR detection system was set at 10 at the start, as this was a good value in the breadboard prototype but proved to be insufficient in the garden conditions.

The second setting was in the greenhouse with the robotic flower placed inside a small flight cage (60 × 60 × 60 cm; front and back in clear vinyl, right and left in polyester netting mesh, Bugdorm, Megaview Ltd.; Figure 21) together with lab-reared floral visitors from Biobest (Westerlo, Belgium). Two species were tested: bumble bees (*B. terrestris*), and *Eupeodes corollae* (a common European species of Syrphidae). After a training period, both species showed great interest in the robotic flower, making it sometimes hard to visually discriminate multiple visits. That is most likely why the results of the manual data collection often show only one visit where the robotic flower recorded multiple visits (Addendum 6, Flight cage). After some visits that were not satisfyingly recorded by the robotic flower with IR sensitivity 10 (as was the case in the garden set-up), the sensitivity was set at 3, meaning that a visit is detected if the IR sensor value read by the microcontroller is 3 units lower than the previous value that was read. Overall, this led to satisfying results: all visits detected manually detected were also present in the data sent by the robotic flower. The robotic flower has a visit time accurate to the second while the manual registration only had minutes, but the visit times seem to match perfectly. The duration overlap gives a view on the difference between manual and

robotic measurement of the visit duration. This was close to 100% in many cases. In the cases the deviation was larger, this could be because multiple visits were measured by the robotic flower compared to one large visit in the manual censuses. The small gaps in between these visits can explain the difference in visit duration. For example, in visit 9 in the flight cage set-up (Addendum 6, Flight cage), the manual measurement shows one long visit of 77 seconds, while the robotic flower registered five different visits of 9, 5, 4, 12 and 27 seconds. The total of the three separate visits gives a duration of 57 seconds, which means a duration overlap of 74%. When looking closer to the intervals between those visits, we see gaps of 2, 5, 1 and 12 seconds. Taking these gaps into account, the result is a duration of 77 seconds and thus a duration overlap of 100%. Moreover, in the visits with a small duration, the duration overlap can drastically deviate from the target value of 100% because of manual registration bias. Starting or stopping the timer a fraction later or earlier and rounding errors by the observer has a big impact if the visit duration was only a couple of seconds. For example, visit 23 (Addendum 6, Flight cage) had a manual measured duration of only 2 seconds, while the robotic flower registered 3 seconds. Only one second difference in this case leads to a duration overlap percentage of 150%.

Looking critically at the results of the analysis of the robotic flower, it can be concluded that the detection of visits works as it should, provided that the IR sensitivity is set to the right value for the set-up. A calibration is thus recommended before starting an experiment using the robotic flowers. This proof of concept also showed that not only bumble bees but also many other floral visitors can be investigated even with the same hardware parts. In the set-up with the flight cage, we had satisfactory results for the syrphids *E. corollae* although it is advisable to reduce the diameter of the feeding hole specifically to better match their body size.

Figure 21*: Set-up of the proof of concept with the flower put in a small flight cage together with a lab-reared bumble bee colony (Biobest, Westerlo, Belgium) in the greenhouse of the botany building (PLTK) at the KU*

4. Discussion

The aim of this thesis was to develop a robotic flower that was autonomous, wireless, energy efficient and that could provide real-time data to the user during an experiment. Because each robotic flower is equipped with its own control unit, it works completely autonomous. Thanks to the LoRaWAN communication, it is possible to remotely access the data on any device using internet during the experiment, making real-time data available without disturbing the experimental set-up or even being in the vicinity of it. By making use of batteries to power the system completely wireless, it becomes possible to widely deploy the robotic flowers, provided there is a LoRaWAN gateway with internet connection within reach. To enlarge the potential duration of an experiment, the design of the robotic flower also focused on the efficient use of energy to maximise battery life. After a successful proof of concept, it can be concluded that the different aims in building this new robotic flower were met.

4.1 Potential applications

The combination of any number of robotic flowers in a set-up makes it possible to create a robotic flower field. Every flower can be seen as a little lab on its own, in which several factors can be controlled and manipulated for different experimental assumptions. One of the most obvious study areas that could benefit from this robotic flower field is on the effect of nectar composition on the behaviour of pollinators, or floral visitors in general. At the level of nectar, many manipulations can be done while keeping other factors that influence foraging behaviour unchanged. Nectar characteristics are being studied widely and many new insights are discovered (Alvarez-Perez et al., 2012; Lamsa et al., 2018; Nepi et al., 2012; Schneider et al., 2012). Not only sugar composition and concentration can influence pollinator behaviour drastically (e.g., by specific sugar or viscosity preference), but the same also counts for amino acids, nectarines, floral secondary metabolites, and trace elements in nectar. All these factors can be manipulated in experiments using the robotic flower, where it can help to give robust and reliable evidence of the influence of these nectar characteristics and components on pollinator behaviour. The effect of human induced disturbances, such as pesticides or pollution traces, on foraging behaviour can be examined as well. For example, an experiment with robotic flowers containing nectar inoculated with a pesticide versus a control could be used to test the effect of this pesticide on foraging behaviour of pollinators. The same could be done to investigate the potential effects of medication in nectar on pollinator behaviour. Nectar microbial inoculation, by yeasts or bacteria, is also an interesting field of study that might benefit from the robotic flower. In the self-medication theory pollinators such as bumblebees might deliberately seek out flowers offering nectar with secondary compounds that reduce parasitic load or enhance colony development (Pozo et al., 2020; Richardson et al., 2015). In further investigating this theory, the robot flower might be an added value, as all factors can be

controlled in the robotic flower which is not possible in natural flower experiments. The discrete amount of offered nectar can also be manipulated together with other flower characteristics to research associative learning. Moreover, not only single manipulations, but combinations of two or more can be done.

Besides nectar composition, the appearance of the flower can also be changed in many ways. The central flower disk is easily changed while keeping the rest the robotic flower hardware unchanged. Not only the central flower disk, but also other parts like the landing platform can be changed: colour, size and two-dimensional shape can be altered, but also the three-dimensional shape can be changed simply by changing the 3D-design. This could be an interesting option, as bees are able to detect and learn to recognise these shapes which might affect foraging behaviour (Werner et al., 2016). For example, as damaged flower parts, such as petals, could be a sign of smaller nectar award, these defects can invoke avoidance behaviour in pollinators (Goulson et al., 2007). By making robotic flowers that mimic these floral defects, and others that don't, the selectivity by floral visitors could be investigated.

Another major advantage of using the robotic flower for behavioural experiments is the possibility to achieve much larger datasets than is the case with traditional observational experiments. For example, compared to Woodcock et al. (2013), where visual observations were conducted for periods of 7 minutes to gather data about flower visitation, this new robotic flower system is a considerable improvement with observation periods up to one week and possibly even more. This does not only mean more data, but also more reliable data because the stochastic effects of observational snapshots are avoided. Moreover, the use of the robotic flowers could also be combined with other techniques such as Radio Frequency identification (RFID) as was proposed by Ohashi et al. (2010). In this way, the individual movements between the flowers can also be monitored on top of the data provided by the robotic flower on its own.

4.2 Limitations

Needless to say, this robotic flower also has limitations and users must be aware that it lacks many characteristics of real flowers. The robotic flower is made of hard plastic material which does not resemble natural plant tissue in feeling and in appearance. The flowers also lack any UV patterns that are present in some flowers. In this respect it should be taken into account that Chittka et al. (1994) concluded that UV signals are just one of the three components of trichromatic bee colour vision and often a too high importance is attributed to the UV component on its own. The three colour receptors (blue, green and UV) are used together and have the same importance to help the flower stand out from a cryptic background. The absence of UV-patterns only will not keep floral visitors away, but it might slightly reduce the salience of the robotic flower.

There are two other main limitations to the robotic flower. First, it is not possible to discriminate between floral visitor species using the robotic flower in an open field and second, the measured foraging pattern only gives a global view and does not provide information on individual choices. The latter problem may be compensated by using many single individuals together with robotic flowers. For example, in an experiment with pollinators on game theory, in which an individual strategy is influenced by the strategy of the other players (Vandamme, 1994), the robotic flowers could prove to be very useful using this individual testing in combination with colony testing.

The inability of the flower to measure exactly how much artificial nectar has been depleted by the visitor is also a limitation of the robotic flower, although the measured probing time is a good proxy for nectar depletion. That is why the maximum visit duration in the robotic flower is set at 250 seconds, giving the visitors enough time to complete nectar depletion. Visits that take longer time than this are no longer likely to be foraging related. The nectar cup gets replenished with nectar automatically to prevent drying out, disregarding if its empty or not.

A possible disturbing factor is the noise and vibration the servomotor produces when the nectar gets refilled. Hymenoptera have been shown to be able to perceive substrate vibrations as well as airborne sound in the form of air flow or the velocity of particles detected by the Johnston's organ in the antennae (Hrncir et al., 2005). It is possible that the sound created by the robotic flower could influence foraging behaviour, but how or to what extent is not known. To avoid this vibration and noise, a refilling system based on an electromagnet could be used (Cnaani et al., 2006), although such a system has shown to have a risk for misalignment of the nectar cup and the feeding hole (Kuusela & Lämsä, 2016).

During an experiment with artificial nectar, uncontrolled microbial contamination can take place in the nectar reservoir, especially if the temperatures inside the flower are optimal for their growth. These microbes could also lead to unexpected effects on pollinator foraging behaviour but can perfectly be determined and quantified in the nectar reservoir by users.

The price and the size of the robotic flower go hand in hand. Nowadays, much smaller hardware parts that have the same functions as the ones used here are available. This would mean the housing of the robotic flower can be smaller, reducing the 3D-printing cost. Nonetheless, we selected the bigger ones deliberately so that the through-hole assembly is easily managed by entry level users. Smaller electronics come together with other assembly techniques that require more experience. Another factor that will lower the price of the flowers is the use of cheaper

components. Also, the number of flowers that is constructed will determine the price: the more, the cheaper is the general rule here.

Data communication with LoRaWAN is very well suited for this kind of application, but also has its limitations and concerns. Being a low-power and long-range framework, LoRaWAN has a lesser performance when it comes to the amount of data that can be sent. All devices have to share the same infrastructure of gateways. Therefore a fair spectrum division needs to be kept in mind and consumed air time must be kept as small as possible (Adelantado et al., 2017). That is the reason why the robotic flower sends a maximum of 53 bytes (containing alarms, battery level and visit data) every 10 minutes. Packages that are lost during transmission cannot be recovered. This can be compensated to a certain extent by asking the backend for a confirmation if the package was received and if needed the transmission can be retried. Still, there is a chance of losing data, albeit it is not very likely if multiple transmissions are attempted. To create more certainty that no data are lost, the external EEPROM which was already provided in the hardware could be used to also store the visit data locally as back-up. In this case, some extra firmware would be needed to write and retrieve these stored data.

4.3 Adaptations, future perspectives & improvements

The design of the robotic flower can be adapted to fit the requirements for specific research topics. Many parameters in the firmware can be easily changed in the adjustable variables header file that is included in the main sketch on the microcontroller (see Table 2). These variables include the time of deep sleep, the sensor sensitivity, refill and send frequencies and the IR detection system cycle time (or the precision of detection). All these can directly influence the functioning of the robotic flower and the battery life. Besides software changes, also hardware can be adapted. Extra batteries can be added for longer experimental periods and other sensors can also be used. The microcontroller has two analogue and three digital GPIO pins left that are unused (see Addendum 3: electrical scheme). Sensors to measure temperature, humidity, light intensity, or other factors could give useful information. For example, the foraging behaviour of some bees depends on temperature, solar radiation humidity and wind speed (Vicens & Bosch, 2000). One alternative possibility could also be to develop and place one device as measuring hub containing these extra sensors providing centralized data for all surrounding flowers. These data could then be used to adjust parameters in the robotic flower functioning, e.g., going in sleep state when it is getting dark or refilling nectar more frequently in high temperatures to prevent complete evaporation of nectar in the nectar cup.

The design of the 3D-printed parts, such as the landing platform or central flower disk can be changed as well. The size, colour and morphology can be manipulated. In this way, the robotic flower can be adapted to study a range of floral visitors besides bees or to compare between species. On average 38% of floral visitors are non-bee species (Rader et al., 2016), making this

group of species also worthwhile for future research on pollination. In the proof of concept, it was already proven that non-bee species, specifically syrphids and lepidopterans, are attracted to the robotic flower. But not only insects can be studied with the robotic flower, with the right hardware changes also bigger animals such as mammals (e.g., flower visiting rodents) and birds (e.g., hummingbirds) can be targeted. Going one step further in this thought process, the robotic flower does not even need to be a flower but could present any other object. By changing the dimensions of the feeding cup, smaller or larger amounts of nectar can be presented to the visitors, discriminating for example for a tropical or a temperate regime.

A possible update to the robotic flower could be adding UV tags mimicking the ones found on real flowers to improve visitor attraction. In the same category there is another way that plants attract pollinators that has been ignored until now: floral scent. Plant volatiles consist of a complex mix of many substances and attraction of pollinators is only one of many functions (Pichersky & Gershenzon, 2002). The robotic flower could be upgraded with a small pump system able to present small amounts of etheric oil containing the wanted substances to further approximate natural flower systems or to test the influence of different substances on the foraging behaviour.

Besides nectar, also pollen is a floral reward that is subject to much research on pollinator behaviour. The function of presenting pollen in the robotic flower was not yet implemented, but if this would be the case in a future version, this might open a whole new perspective for research with this robotic flower. For example, the study of buzz pollination, where pollinators extract pollen from anthers by vibrating, has been understudied although it is present in some very important crop species such as tomatoes (De Luca & Vallejo-Marin, 2013). Therefore, this field of study could make good use of a robotic flower able to present pollen to visitors, for example by designing a 3D-print of the robotic flower featuring an artificial anther fitted with the necessary electronics to release pollen or pollen-like substances when vibration is registered by a sensor.

In open field use, a limitation of the robotic flower is that the visitor cannot be recognized. To tackle this issue, a camera could be attached taking a picture of the feeding hole at the beginning of a visit. The downside is that pictures are too big to send over LoRaWAN and would thus need to be stored locally for later analysis. Another option is using a form of artificial intelligence where the combination of information from different sensors such as weight and buzz vibration could lead to a postulation of the visitor identity.

As for the software, many improvements are possible here as well. A firmware is never finalised, therefore future updates will be useful to improve the robotic flower functioning. Making the flower completely autonomous using an Arduino-style microcontroller has many

benefits, but it also has the limitation that there is only CPU core, making multithreading impossible. Consequently, it is not possible to do multiple tasks at once. Our robotic flower is for example not able to detect a visit during data transmission. With some more advanced coding making use of interrupts, this problem might be circumvented. The energy efficiency could be pushed up even further through a firmware update too. The microcontroller now runs with a clock speed of 8 MHz, but this application could probably also be run 8 times slower at 1 MHz and still work effectively. To do this, the sketch needs to be compiled for 1 MHz (coded in the initialization file with extension '.ini', see also files on GitHub[2]) and the microcontroller needs to get the command to lower the clock speed as well. The possibility to do this was implemented in the firmware (see Addendum 4, line 535), but some hardware changes would be needed, e.g., the indication LED prevented the sketch to compile at 1 MHz. The frontend application could be extended to improve the user experience. A possible example is adding an interface showing a map where the geolocation of all deployed robotic flowers is displayed, but many other functionalities might be added.

The above-mentioned adaptations are beyond the scope of this thesis but could be an interesting addition to robotic flowers in future research. This prototype robotic flower should be seen as a foundation to adapt specifically for an array of research purposes.

5. Conclusion

With this new robotic flower, we have been able to improve existing methodologies to perform behavioural studies on pollinators. The proof of concept was successful in lab conditions, but also for outdoor use the robotic flower could be used, given the right sensor sensitivity is used. Future research on many aspects of foraging behaviour could profit from making use of this system and we hope the gained knowledge can help in averting and rewinding the ongoing pollinating insect biodiversity loss and – as a result of their symbiotic relationships with other animal and plant species – of overall biodiversity. We hope new insights in pollinator behaviour can also help in agricultural intensification, as this will be essential in save-guarding agricultural yields for the rising human population and increasing well-fare.

[2] https://github.com/Kamiel-debeuckelaere/One-Robotic-Flower-Field-To-Study-Them-All.git

6. References

Adelantado, F., Vilajosana, X., Tuset-Peiro, P., Martinez, B., Melia-Segui, J., & Watteyne, T. (2017). Understanding the Limits of LoRaWAN. *IEEE COMMUNICATIONS MAGAZINE*, *55*(9), 34–40. https://doi.org/10.1109/MCOM.2017.1600613

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE COMMUNICATIONS SURVEYS AND TUTORIALS*, *17*(4), 2347–2376. https://doi.org/10.1109/COMST.2015.2444095

Alvarez-Perez, S., Herrera, C. M., & de Vega, C. (2012). Zooming-in on floral nectar: a first exploration of nectar-associated bacteria in wild plant communities. *FEMS MICROBIOLOGY ECOLOGY*, *80*(3), 591–602. https://doi.org/10.1111/j.1574-6941.2012.01329.x

Baker, H. G., & Baker, I. (1973). Amino-Acids in Nectar and Their Evolutionary Significance. *NATURE*, *241*(5391), 543–545. https://doi.org/10.1038/241543b0

Balfour, N. J., Garbuzov, M., & Ratnieks, F. L. W. (2013). Longer tongues and swifter handling: why do more bumble bees (Bombus spp.) than honey bees (Apis mellifera) forage on lavender (Lavandula spp.)? *ECOLOGICAL ENTOMOLOGY*, *38*(4), 323–329. https://doi.org/10.1111/een.12019

Biesmeijer, J. C., Roberts, S. P. M., Reemer, M., Ohlemueller, R., Edwards, M., Peeters, T., … Kunin, W. E. (2006). Parallel declines in pollinators and insect-pollinated plants in Britain and the Netherlands. *SCIENCE*, *313*(5785), 351–354. https://doi.org/10.1126/science.1127863

Billiet, A., Meeus, I., Van Nieuwerburgh, F., Deforce, D., Wäckers, F., & Smagghe, G. (2016). Impact of sugar syrup and pollen diet on the bacterial diversity in the gut of indoor-reared bumblebees (Bombus terrestris). *Apidologie*, *47*(4), 548–560. https://doi.org/10.1007/s13592-015-0399-1

Branquart, E., & Hemptinne, J. L. (2000). Selectivity in the exploitation of floral resources by hoverflies (Diptera : Syrphinae). *ECOGRAPHY*, *23*(6), 732–742. https://doi.org/10.1034/j.1600-0587.2000.230610.x

Briggs, H. M., Graham, S., Switzer, C. M., & Hopkins, R. (2018). Variation in context-dependent foraging behavior across pollinators. *ECOLOGY AND EVOLUTION*, *8*(16), 7964–7973. https://doi.org/10.1002/ece3.4303

Bronstein, J. L., Alarcon, R., & Geber, M. (2006). The evolution of plant-insect mutualisms. *NEW PHYTOLOGIST*, *172*(3), 412–428. https://doi.org/10.1111/j.1469-8137.2006.01864.x

Burkle, L. A., Marlin, J. C., & Knight, T. M. (2013). Plant-Pollinator Interactions over 120 Years: Loss of Species, Co-Occurrence, and Function. *SCIENCE*, *339*(6127), 1611–1615. https://doi.org/10.1126/science.1232728

Calisi, R. M., & Bentley, G. E. (2009). Lab and field experiments: Are they the same animal? *Hormones and Behavior*, *56*(1), 1–10. https://doi.org/https://doi.org/10.1016/j.yhbeh.2009.02.010

Cameron, S. A., Lozier, J. D., Strange, J. P., Koch, J. B., Cordes, N., Solter, L. F., & Griswold, T. L. (2011). Patterns of widespread decline in North American bumble bees. *PROCEEDINGS OF THE NATIONAL ACADEMY OF SCIENCES OF THE UNITED STATES OF AMERICA*, *108*(2), 662–667. https://doi.org/10.1073/pnas.1014743108

Cariveau, D. P., Nayak, G. K., Bartomeus, I., Zientek, J., Ascher, J. S., Gibbs, J., & Winfree, R. (2016). The Allometry of Bee Proboscis Length and Its Uses in Ecology. *PLOS ONE*, *11*(3). https://doi.org/10.1371/journal.pone.0151482

Charlesworth, D., Morgan, M. T., & Charlesworth, B. (1990). Inbreeding Depression, Genetic Load,

and the Evolution of Outcrossing Rates in a Multilocus System with No Linkage. *Evolution*, *44*(6), 1469–1489. https://doi.org/10.2307/2409330

Chen, M., & Rincon-Mora, G. A. (2006). Accurate electrical battery model capable of predicting, runtime and I-V performance. *IEEE TRANSACTIONS ON ENERGY CONVERSION*, *21*(2), 504–511. https://doi.org/10.1109/TEC.2006.874229

Chittka, L., Shmida, A., Troje, N., & Menzel, R. (1994). Ultraviolet as a component of flower reflections, and the color-perception of Hymenoptera. *VISION RESEARCH*, *34*(11), 1489–1508. https://doi.org/10.1016/0042-6989(94)90151-1

Chittka, L., Thomson, J. D., & Waser, N. M. (1999). Flower constancy, insect psychology, and plant evolution. *NATURWISSENSCHAFTEN*, *86*(8), 361–377. https://doi.org/10.1007/s001140050636

Cnaani, J., Thomson, J. D., & Papaj, D. R. (2006). Flower choice and learning in foraging bumblebees: Effects of variation in nectar volume and concentration. *ETHOLOGY*, *112*(3), 278–285. https://doi.org/10.1111/j.1439-0310.2006.01174.x

Corbet, S. A. (2000). Butterfly nectaring flowers: butterfly morphology and flower form. *ENTOMOLOGIA EXPERIMENTALIS ET APPLICATA*, *96*(3), 289–298. https://doi.org/10.1046/j.1570-7458.2000.00708.x

Costanza, R., dArge, R., deGroot, R., Farber, S., Grasso, M., Hannon, B., … vandenBelt, M. (1997). The value of the world's ecosystem services and natural capital. *NATURE*, *387*(6630), 253–260. https://doi.org/10.1038/387253a0

Crall, J. D., Gravish, N., Mountcastle, A. M., & Combes, S. A. (2015). BEEtag: A Low-Cost, Image-Based Tracking System for the Study of Animal Behavior and Locomotion. *PLOS ONE*, *10*(9). https://doi.org/10.1371/journal.pone.0136487

De Luca, P. A., & Vallejo-Marin, M. (2013). What's the `buzz' about? The ecology and evolutionary significance of buzz-pollination. *CURRENT OPINION IN PLANT BIOLOGY*, *16*(4), 429–435. https://doi.org/10.1016/j.pbi.2013.05.002

Dyer, A. G., & Garcia, J. E. (2014). Color Difference and Memory Recall in Free-Flying Honeybees: Forget the Hard Problem. *Insects*, *5*(3), 629–638. https://doi.org/10.3390/insects5030629

Essenberg, C. J. (2015). FLOBOTS: ROBOTIC FLOWERS FOR BEE BEHAVIOUR EXPERIMENTS. *Journal of Pollination Ecology*, *15*(1), 1–5. Retrieved from https://pubmed.ncbi.nlm.nih.gov/25722755

Fenster, C. B., Armbruster, W. S., Wilson, P., Dudash, M. R., & Thomson, J. D. (2004). Pollination syndromes and floral specialization. *ANNUAL REVIEW OF ECOLOGY EVOLUTION AND SYSTEMATICS*, *35*, 375–403. https://doi.org/10.1146/annurev.ecolsys.34.011802.132347

Gallai, N., Salles, J.-M., Settele, J., & Vaissiere, B. E. (2009). Economic valuation of the vulnerability of world agriculture confronted with pollinator decline. *ECOLOGICAL ECONOMICS*, *68*(3), 810–821. https://doi.org/10.1016/j.ecolecon.2008.06.014

Garibaldi, L. A., Saez, A., Aizen, M. A., Fijen, T., & Bartomeus, I. (2020). Crop pollination management needs flower-visitor monitoring and target values. *JOURNAL OF APPLIED ECOLOGY*, *57*(4), 664–670. https://doi.org/10.1111/1365-2664.13574

Garibaldi, L. A., Steffan-Dewenter, I., Winfree, R., Aizen, M. A., Bommarco, R., Cunningham, S. A., … Klein, A. M. (2013). Wild Pollinators Enhance Fruit Set of Crops Regardless of Honey Bee Abundance. *SCIENCE*, *339*(6127), 1608+. https://doi.org/10.1126/science.1230200

Gegear, R. J., & Laverty, T. M. (2005). Flower constancy in bumblebees: a test of the trait variability hypothesis. *ANIMAL BEHAVIOUR*, *69*(4), 939–949.

https://doi.org/10.1016/j.anbehav.2004.06.029

Giurfa, M. (2004). Conditioning procedure and color discrimination in the honeybee Apis mellifera. *NATURWISSENSCHAFTEN*, *91*(5), 228–231. https://doi.org/10.1007/s00114-004-0530-z

Givnish, T. J., Spalink, D., Ames, M., Lyon, S. P., Hunter, S. J., Zuluaga, A., … Cameron, K. M. (2015). Orchid phylogenomics and multiple drivers of their extraordinary diversification. *PROCEEDINGS OF THE ROYAL SOCIETY B-BIOLOGICAL SCIENCES*, *282*(1814), 171–180. https://doi.org/10.1098/rspb.2015.1553

Godfray, H. C. J., Beddington, J. R., Crute, I. R., Haddad, L., Lawrence, D., Muir, J. F., … Toulmin, C. (2010). Food Security: The Challenge of Feeding 9 Billion People. *SCIENCE*, *327*(5967), 812–818. https://doi.org/10.1126/science.1185383

Goulson, D, Cruise, J. L., Sparrow, K. R., Harris, A. J., Park, K. J., Tinsley, M. C., & Gilburn, A. S. (2007). Choosing rewarding flowers; perceptual limitations and innate preferences influence decision making in bumblebees and honeybees. *BEHAVIORAL ECOLOGY AND SOCIOBIOLOGY*, *61*(10), 1523–1529. https://doi.org/10.1007/s00265-007-0384-4

Goulson, D, Lye, G. C., & Darvill, B. (2008). Decline and conservation of bumble bees. *ANNUAL REVIEW OF ENTOMOLOGY*, *53*, 191–208. https://doi.org/10.1146/annurev.ento.53.103106.093454

Goulson, D, Stout, J. C., Hawson, S. A., & Allen, J. A. (1998). Floral display size in comfrey, Symphytum officinale L. (Boraginaceae): relationships with visitation by three bumblebee species and subsequent seed set. *OECOLOGIA*, *113*(4), 502–508. https://doi.org/10.1007/s004420050402

Goulson, Dave. (2010). *Bumblebees: behaviour, ecology, and conservation.* (Second Edi). Oxford: OXFORD UNIV PRESS INC.

Goulson, Dave, Nicholls, E., Botias, C., & Rotheray, E. L. (2015). Bee declines driven by combined stress from parasites, pesticides, and lack of flowers. *SCIENCE*, *347*(6229). https://doi.org/10.1126/science.1255957

Granero, A. M., Sanz, J. M. G., Gonzalez, F. J. E., Vidal, J. L. M., Dornhaus, A., Ghani, J., … Chittka, L. (2005). Chemical compounds of the foraging recruitment pheromone in bumblebees. *NATURWISSENSCHAFTEN*, *92*(8), 371–374. https://doi.org/10.1007/s00114-005-0002-0

Greenleaf, S. S., & Kremen, C. (2006). Wild bee species increase tomato production and respond differently to surrounding land use in Northern California. *BIOLOGICAL CONSERVATION*, *133*(1), 81–87. https://doi.org/10.1016/j.biocon.2006.05.025

Grüter, C., & Ratnieks, F. L. W. (2011). Flower constancy in insect pollinators: Adaptive foraging behaviour or cognitive limitation? *Communicative & Integrative Biology*, *4*(6), 633–636. https://doi.org/10.4161/cib.16972

Hallmann, C. A., Sorg, M., Jongejans, E., Siepel, H., Hofland, N., Schwan, H., … de Kroon, H. (2017). More than 75 percent decline over 27 years in total flying insect biomass in protected areas. *PLOS ONE*, *12*(10). https://doi.org/10.1371/journal.pone.0185809

Harvey, J. A., Heinen, R., Armbrecht, I., Basset, Y., Baxter-Gilbert, J. H., Bezemer, T. M., … de Kroon, H. (2020). International scientists formulate a roadmap for insect conservation and recovery. *NATURE ECOLOGY \& EVOLUTION*, *4*(2), 174–176. https://doi.org/10.1038/s41559-019-1079-8

Heil, M. (2011). Nectar: generation, regulation, and ecological functions. *TRENDS IN PLANT SCIENCE*, *16*(4), 191–200. https://doi.org/10.1016/j.tplants.2011.01.003

Heinrich, B. (1974). Thermoregulation in Endothermic Insects. *SCIENCE*, *185*(4153), 747–756. https://doi.org/10.1126/science.185.4153.747

Hrncir, M., Barth, F., & Tautz, J. (2005). 32 Vibratory and Airborne-Sound Signals in Bee Communication (Hymenoptera). *Insect Sounds and Communication: Physiology, Behaviour, Ecology, and Evolution*. https://doi.org/10.1201/9781420039337.ch32

IPCC. (2021). *Climate Change 2021: The Physical Science Basis. Contribution of Working Group I to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change [Masson-Delmotte, V., P. Zhai, A. Pirani, S. L. Connors, C. Péan, S. Berger, N. Caud, Y. Chen,*. Retrieved from Cambridge University Press

Keasar, T. (2000). The spatial distribution of nonrewarding artificial flowers affects pollinator attraction. *ANIMAL BEHAVIOUR*, *60*, 639–646. https://doi.org/10.1006/anbe.2000.1484

Kim, W., Gilet, T., & Bush, J. W. M. (2011). Optimal concentrations in nectar feeding. *PROCEEDINGS OF THE NATIONAL ACADEMY OF SCIENCES OF THE UNITED STATES OF AMERICA*, *108*(40), 16618–16621. https://doi.org/10.1073/pnas.1108642108

Klein, A.-M., Vaissiere, B. E., Cane, J. H., Steffan-Dewenter, I., Cunningham, S. A., Kremen, C., & Tscharntke, T. (2007). Importance of pollinators in changing landscapes for world crops. *PROCEEDINGS OF THE ROYAL SOCIETY B-BIOLOGICAL SCIENCES*, *274*(1608), 303–313. https://doi.org/10.1098/rspb.2006.3721

Kosior, A., Celary, W., Olejniczak, P., Fijal, J., Krol, W., Solarz, W., & Plonka, P. (2007). The decline of the bumble bees and cuckoo bees (Hymenoptera : Apidae : Bombini) of Western and Central Europe. *ORYX*, *41*(1), 79–88. https://doi.org/10.1017/S0030605307001597

Kremen, C., Williams, N. M., Aizen, M. A., Gemmill-Herren, B., LeBuhn, G., Minckley, R., … Ricketts, T. H. (2007). Pollination and other ecosystem services produced by mobile organisms: a conceptual framework for the effects of land-use change. *ECOLOGY LETTERS*, *10*(4), 299–314. https://doi.org/10.1111/j.1461-0248.2007.01018.x

Kuusela, E., & Lämsä, J. (2016). A low-cost, computer-controlled robotic flower system for behavioral experiments. *ECOLOGY AND EVOLUTION*, *6*(8), 2594–2600. https://doi.org/10.1002/ece3.2062

Labandeira, C. C. (1997). Insect mouthparts: Ascertaining the paleobiology of insect feeding strategies. *ANNUAL REVIEW OF ECOLOGY AND SYSTEMATICS*, *28*, 153–193. https://doi.org/10.1146/annurev.ecolsys.28.1.153

Lämsä, J., Kuusela, E., Tuomi, J., Juntunen, S., & Watts, P. C. (2018). Low dose of neonicotinoid insecticide reduces foraging motivation of bumblebees. *PROCEEDINGS OF THE ROYAL SOCIETY B-BIOLOGICAL SCIENCES*, *285*(1883). https://doi.org/10.1098/rspb.2018.0506

Lunau, K. (2000). The ecology and evolution of visual pollen signals. *PLANT SYSTEMATICS AND EVOLUTION*, *222*(1–4), 89–111. https://doi.org/10.1007/BF00984097

Ma, S., Jiang, M., Tao, P., Song, C., Wu, J., Wang, J., … Shang, W. (2018). Temperature effect and thermal impact in lithium-ion batteries: A review. *Progress in Natural Science: Materials International*, *28*(6), 653–666. https://doi.org/https://doi.org/10.1016/j.pnsc.2018.11.002

Makino, T. T., & Sakai, S. (2007). Experience changes pollinator responses to floral display size: from size-based to reward-based foraging. *FUNCTIONAL ECOLOGY*, *21*(5), 854–863. https://doi.org/10.1111/j.1365-2435.2007.01293.x

Memmott, J., Craze, P. G., Waser, N. M., & Price, M. V. (2007). Global warming and the disruption of plant-pollinator interactions. *ECOLOGY LETTERS*, *10*(8), 710–717. https://doi.org/10.1111/j.1461-0248.2007.01061.x

Michener, C. D. (2000). *The bees of the world*. Baltimore: John Hopkins University Press.

Ne'eman, G., Shavit, O., Shaltiel, L., & Shmida, A. (2006). Foraging by male and female solitary bees with implications for pollination. *JOURNAL OF INSECT BEHAVIOR*, *19*(3), 383–401. https://doi.org/10.1007/s10905-006-9030-7

Nepi, M., Soligo, C., Nocentini, D., Abate, M., Guarnieri, M., Cai, G., … Pacini, E. (2012). Amino acids and protein profile in floral nectar: Much more than a simple reward. *FLORA*, *207*(7), 475–481. https://doi.org/10.1016/j.flora.2012.06.002

Nicolson, S. W. (2011). Bee food: the chemistry and nutritional value of nectar, pollen and mixtures of the two. *AFRICAN ZOOLOGY*, *46*(2), 197–204. https://doi.org/10.3377/004.046.0201

Ohashi, K., D'Souza, D., & Thomson, J. D. (2010). An automated system for tracking and identifying individual nectar foragers at multiple feeders. *BEHAVIORAL ECOLOGY AND SOCIOBIOLOGY*, *64*(5), 891–897. https://doi.org/10.1007/s00265-010-0907-2

Ollerton, J., Winfree, R., & Tarrant, S. (2011). How many flowering plants are pollinated by animals? *OIKOS*, *120*(3), 321–326. https://doi.org/10.1111/j.1600-0706.2010.18644.x

Pacini, E., Nepi, M., & Vesprini, J. L. (2003). Nectar biodiversity: a short review. *PLANT SYSTEMATICS AND EVOLUTION*, *238*(1–4), 7–21. https://doi.org/10.1007/s00606-002-0277-y

Paldi, N., Zilber, S., & Shafir, S. (2003). Associative olfactory learning of honeybees to differential rewards in multiple contexts - Effect of odor component and mixture similarity. *JOURNAL OF CHEMICAL ECOLOGY*, *29*(11), 2515–2538. https://doi.org/10.1023/A:1026362018796

Partap, U., Partap, T., & He, Y. H. (2001). Pollination failure in apple crop and farmers' management strategies in Hengduan Mountains, China. In Benedek, P and Richards, KW (Ed.), *PROCEEDINGS OF THE EIGHT INTERNATIONAL POLLINATION SYMPOSIUM POLLINATION: INTEGRATOR OF CROPS AND NATIVE PLANT SYSTEMS* (pp. 225–230). https://doi.org/10.17660/ActaHortic.2001.561.32

Phalan, B., Onial, M., Balmford, A., & Green, R. E. (2011). Reconciling Food Production and Biodiversity Conservation: Land Sharing and Land Sparing Compared. *SCIENCE*, *333*(6047), 1289–1291. https://doi.org/10.1126/science.1208742

Pichersky, E., & Gershenzon, J. (2002). The formation and function of plant volatiles: perfumes for pollinator attraction and defense. *CURRENT OPINION IN PLANT BIOLOGY*, *5*(3), 237–243. https://doi.org/10.1016/S1369-5266(02)00251-0

Poinar, G. (2016). Orchid pollinaria (Orchidaceae) attached to stingless bees (Hymenoptera: Apidae) in Dominican amber. *Neues Jahrbuch Für Geologie Und Paläontologie - Abhandlungen*, *279*, 287–293. https://doi.org/10.1127/njgpa/2016/0556

Potts, S. G., Biesmeijer, J. C., Kremen, C., Neumann, P., Schweiger, O., & Kunin, W. E. (2010). Global pollinator declines: trends, impacts and drivers. *TRENDS IN ECOLOGY & EVOLUTION*, *25*(6), 345–353. https://doi.org/10.1016/j.tree.2010.01.007

Potts, S. G., Imperatriz-Fonseca, V., Ngo, H. T., Aizen, M. A., Biesmeijer, J. C., Breeze, T. D., … Vanbergen, A. J. (2016). Safeguarding pollinators and their values to human well-being. *NATURE*, *540*(7632), 220–229. https://doi.org/10.1038/nature20588

Pozo, M. I., van Kemenade, G., van Oystaeyen, A., Aledon-Catala, T., Benavente, A., den Ende, W., … Jacquemyn, H. (2020). The impact of yeast presence in nectar on bumble bee behavior and fitness. *ECOLOGICAL MONOGRAPHS*, *90*(1). https://doi.org/10.1002/ecm.1393

Pozo, M., Lievens, B., & Jacquemyn, H. (2014). *Impact of Microorganisms on Nectar Chemistry, Pollinator Attraction and Plant Fitness.*

Provoost, M., Weyns, D., & IEEE. (2019). DingNet: A Self-Adaptive Internet-of-Things Exemplar. In *2019 IEEE/ACM 14TH INTERNATIONAL SYMPOSIUM ON SOFTWARE ENGINEERING FOR ADAPTIVE AND SELF-MANAGING SYSTEMS (SEAMS 2019)*. https://doi.org/10.1109/SEAMS.2019.00033

Pyke, G. H., Pulliam, H. R., & Charnov, E. L. (1977). Optimal Foraging - Selective Review of Theory and Tests. *QUARTERLY REVIEW OF BIOLOGY*, *52*(2), 137–154. https://doi.org/10.1086/409852

Rader, R. A., Bartomeus, I. B., Garibaldi, L. A., Garratt, M. P. D., Howlett, B. G., Winfree, R. G., … Woyciechowski, M. (2016). Non-bee insects are important contributors to global crop pollination. *PROCEEDINGS OF THE NATIONAL ACADEMY OF SCIENCES OF THE UNITED STATES OF AMERICA*, *113*(1), 146–151. https://doi.org/10.1073/pnas.1517092112

Raguso, R. A. (2004). Why are some floral nectars scented? *ECOLOGY*, *85*(6), 1486–1494. https://doi.org/10.1890/03-0410

Raine, N. E., & Chittka, L. (2007). Nectar production rates of 75 bumblebee-visited flower species in a German flora (Hymenoptera : Apidae : Bombus terrestris). *ENTOMOLOGIA GENERALIS*, *30*(2), 191–192.

Ratnayake, M. N., Dyer, A. G., & Dorin, A. (2021). Tracking individual honeybees among wildflower clusters with computer vision-facilitated pollinator monitoring. *PLOS ONE*, *16*(2). https://doi.org/10.1371/journal.pone.0239504

Real, L., & Rathcke, B. J. (1988). Patterns of individual variability in floral resources. *ECOLOGY*, *69*(3), 728–735. https://doi.org/10.2307/1941021

Richardson, L. L., Adler, L. S., Leonard, A. S., Andicoechea, J., Regan, K. H., Anthony, W. E., … Irwin, R. E. (2015). Secondary metabolites in floral nectar reduce parasite infections in bumblebees. *PROCEEDINGS OF THE ROYAL SOCIETY B-BIOLOGICAL SCIENCES*, *282*(1803). https://doi.org/10.1098/rspb.2014.2471

Sanchez-Bayo, F., & Wyckhuys, K. A. G. (2019). Worldwide decline of the entomofauna: A review of its drivers. *BIOLOGICAL CONSERVATION*, *232*, 8–27. https://doi.org/10.1016/j.biocon.2019.01.020

Schiestl, F. P., & Johnson, S. D. (2013). Pollinator-mediated evolution of floral signals. *TRENDS IN ECOLOGY \& EVOLUTION*, *28*(5), 307–315. https://doi.org/10.1016/j.tree.2013.01.019

Schneider, C. W., Tautz, J., Gruenewald, B., & Fuchs, S. (2012). RFID Tracking of Sublethal Effects of Two Neonicotinoid Insecticides on the Foraging Behavior of Apis mellifera. *PLOS ONE*, *7*(1). https://doi.org/10.1371/journal.pone.0030023

Simons, D. J., & Chabris, C. F. (1999). Gorillas in our midst: sustained inattentional blindness for dynamic events. *PERCEPTION*, *28*(9), 1059–1074. https://doi.org/10.1068/p2952

Simpson, B., & Neff, J. (1981). Floral Rewards: Alternatives to Pollen and Nectar. *Annals of the Missouri Botanical Garden*, *68*, 301. https://doi.org/10.2307/2398800

Sokolowski, M. B. C., & Abramson, C. I. (2010). From foraging to operant conditioning: A new computer-controlled Skinner box to study free-flying nectar gathering behavior in bees. *JOURNAL OF NEUROSCIENCE METHODS*, *188*(2), 235–242. https://doi.org/10.1016/j.jneumeth.2010.02.013

Spaethe, J., Tautz, J., & Chittka, L. (2001). Visual constraints in foraging bumblebees: Flower size and color affect search time and flight behavior. *PROCEEDINGS OF THE NATIONAL ACADEMY OF SCIENCES OF THE UNITED STATES OF AMERICA*, *98*(7), 3898–3903. https://doi.org/10.1073/pnas.071053098

Spaethe, J., & Weidenmuller, A. (2002). Size variation and foraging rate in bumblebees (Bombus terrestris). *INSECTES SOCIAUX*, *49*(2), 142–146. https://doi.org/10.1007/s00040-002-8293-z

Steffen, W., Broadgate, W., Deutsch, L., Gaffney, O., & Ludwig, C. (2015). The trajectory of the Anthropocene: The Great Acceleration. *ANTHROPOCENE REVIEW*, *2*(1), 81–98. https://doi.org/10.1177/2053019614564785

Stevenson, P. C., Nicolson, S. W., & Wright, G. A. (2017). Plant secondary metabolites in nectar: impacts on pollinators and ecological functions. *FUNCTIONAL ECOLOGY*, *31*(1), 65–75. https://doi.org/10.1111/1365-2435.12761

Stoekl, J., Brodmann, J., Dafni, A., Ayasse, M., & Hansson, B. S. (2011). Smells like aphids: orchid flowers mimic aphid alarm pheromones to attract hoverflies for pollination. *PROCEEDINGS OF THE ROYAL SOCIETY B-BIOLOGICAL SCIENCES*, *278*(1709), 1216–1222. https://doi.org/10.1098/rspb.2010.1770

Sun, S., Leshowitz, M. I., & Rychtář, J. (2018). The signalling game between plants and pollinators. *Scientific Reports*, *8*(1), 6686. https://doi.org/10.1038/s41598-018-24779-0

Tscharntke, T., Clough, Y., Wanger, T. C., Jackson, L., Motzke, I., Perfecto, I., … Whitbread, A. (2012). Global food security, biodiversity conservation and the future of agricultural intensification. *BIOLOGICAL CONSERVATION*, *151*(1), 53–59. https://doi.org/10.1016/j.biocon.2012.01.068

Tylianakis, J. M., Didham, R. K., Bascompte, J., & Wardle, D. A. (2008). Global change and species interactions in terrestrial ecosystems. *ECOLOGY LETTERS*, *11*(12), 1351–1363. https://doi.org/10.1111/j.1461-0248.2008.01250.x

Vandamme, E. (1994). Game Theory - Evolutionary Game Theory. *EUROPEAN ECONOMIC REVIEW*, *38*(3–4), 847–858. https://doi.org/10.1016/0014-2921(94)90121-X

Vasquez, A., & Olofsson, T. C. (2009). The lactic acid bacteria involved in the production of bee pollen and bee bread. *JOURNAL OF APICULTURAL RESEARCH*, *48*(3), 189–195. https://doi.org/10.3896/IBRA.1.48.3.07

Verheyen, J., Delnat, V., & Stoks, R. (2019). Increased Daily Temperature Fluctuations Overrule the Ability of Gradual Thermal Evolution to Offset the Increased Pesticide Toxicity under Global Warming. *ENVIRONMENTAL SCIENCE \& TECHNOLOGY*, *53*(8), 4600–4608. https://doi.org/10.1021/acs.est.8b07166

Vicens, N., & Bosch, J. (2000). Weather-dependent pollinator activity in an apple orchard, with special reference to Osmia cornuta and Apis mellifera (Hymenoptera : Megachilidae and Apidae). *ENVIRONMENTAL ENTOMOLOGY*, *29*(3), 413–420. https://doi.org/10.1603/0046-225X-29.3.413

Waser, N. M. (1986). Flower Constancy - Definition, Cause, and Measurement. *AMERICAN NATURALIST*, *127*(5), 593–603. https://doi.org/10.1086/284507

Waser, N. M., Chittka, L., Price, M. V, Williams, N. M., & Ollerton, J. (1996). Generalization in pollination systems, and why it matters. *ECOLOGY*, *77*(4), 1043–1060. https://doi.org/10.2307/2265575

Werner, A., Stürzl, W., & Zanker, J. (2016). Object Recognition in Flight: How Do Bees Distinguish between 3D Shapes? *PLOS ONE*, *11*(2), 1–13. https://doi.org/10.1371/journal.pone.0147106

Winfree, R., Aguilar, R., Vazquez, D. P., LeBuhn, G., & Aizen, M. A. (2009). A meta-analysis of bees' responses to anthropogenic disturbance. *ECOLOGY*, *90*(8), 2068–2076. https://doi.org/10.1890/08-1245.1

Wolf, T. J., Ellington, C. P., & Begley, I. S. (1999). Foraging costs in bumblebees: field conditions cause large individual differences. *INSECTES SOCIAUX*, *46*(3), 291–295. https://doi.org/10.1007/s000400050148

Woodcock, B. A., Edwards, M., Redhead, J., Meek, W. R., Nuttall, P., Falk, S., … Pywell, R. F. (2013). Crop flower visitation by honeybees, bumblebees and solitary bees: Behavioural differences and diversity responses to landscape. *AGRICULTURE ECOSYSTEMS \& ENVIRONMENT*, *171*, 1–8. https://doi.org/10.1016/j.agee.2013.03.005

Wright, G. A., & Schiestl, F. P. (2009). The evolution of floral scent: the influence of olfactory learning by insect pollinators on the honest signalling of floral rewards. *FUNCTIONAL ECOLOGY*, *23*(5), 841–851. https://doi.org/10.1111/j.1365-2435.2009.01627.x

Zattara, E. E., & Aizen, M. A. (2021). Worldwide occurrence records suggest a global decline in bee species richness. *ONE EARTH*, *4*(1), 114–123. https://doi.org/10.1016/j.oneear.2020.12.005

7. Addendum

Addendum 1: Risk analysis.

This master thesis was completed in a rather special year, given the corona-pandemic restrictions. I started working in the lab, but continued the work from home when the infection rates and other corona-related numbers got worse. To do that, I installed a second desk and a second computer screen so that I could work ergonomically.

During the making of this thesis, the practical work mainly consisted of electronical design and assembly of the robotic flower. Making use of a bread board for prototyping reduced hazardous situations a lot as this does not require any soldering. When the prototype was ready, soldering was necessary to complete the next step of the design: assembly of the PCB. As I do not have the right equipment at home to work safely, this was done at the KU Leuven facility FabLab. There, a soldering station was available with the right infrastructure. The solder that was used was lead-free to avoid inhalation of toxic fumes during heating with the soldering iron. A small fan was used to blow away any other smoke that was produced. I was aware of the implied risks of getting burned when the soldering iron is not handled correctly, therefore I got advice from someone with more experience before continuing.

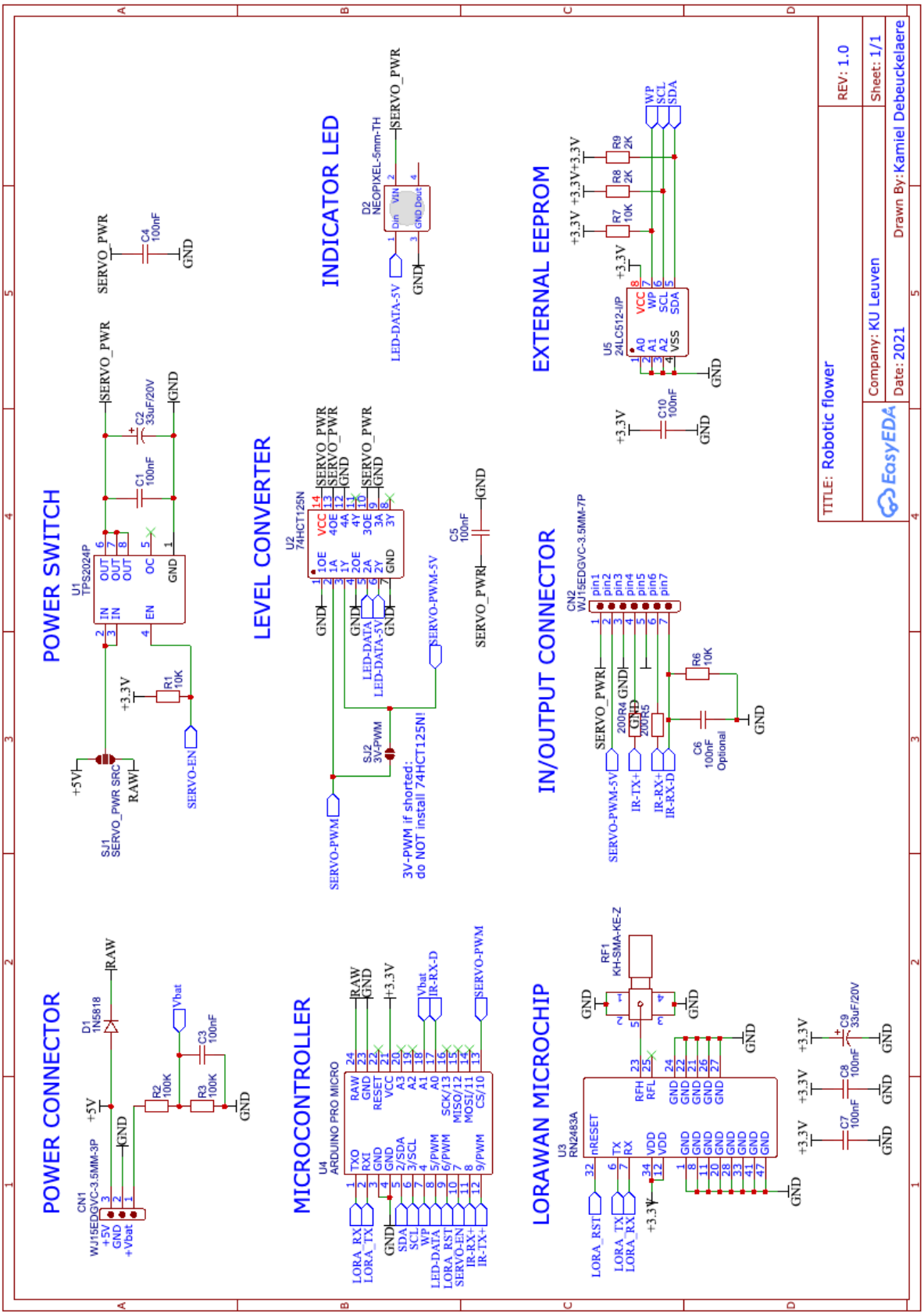
While working with different pollinator species for the proof of concept, we were aware of the risk of getting stung. Especially when handling the lab-reared *B. terrestris* we proceeded carefully. In this way, we did not get close to danger. Again, experienced helping hands assured a smooth process.

Addendum 2: List of materials used to produce the robotic flowers.

Part	Specifications	Artikelnummer fabrikant	#/flower	Company	URL
Microcontroller Pro Micro	ATMEGA32U4 3.3V/8MHZ	DEV-12587	1	**DigiKey**	https://www.digikey.be/product-detail/en/sparkfun-electronics/DEV-12587/1568-1061-ND/5140826
Micro Servo	SERVOMOTOR RC 4.8V	SER0006	1	**DigiKey**	https://www.digikey.be/product-detail/en/dfrobot/SER0006/1738-1385-ND/7597224
LED (per 5)	ADDRESS LED DISCR SERIAL RGB 5PC	COM-12986	0.2	**DigiKey**	https://www.digikey.be/product-detail/en/sparkfun-electronics/COM-12986/1568-1213-ND/5673799
ON/OFF Switch	SWITCH ROCKER SPST 10A 125V	RA1113112R	1	**DigiKey**	https://www.digikey.be/product-detail/en/e-switch/RA1113112R/EG5619-ND/3778055
IR emitter	EMITTER IR 940NM 50MA RADIAL	IR908-7C-F	1	**DigiKey**	https://www.digikey.be/product-detail/en/everlight-electronics-co-ltd/IR908-7C-F/1080-1089-ND/2675580
IR receiver	SENSOR PHOTO 940NM SIDE VIEW RAD	PT908-7C-F	1	**DigiKey**	https://www.digikey.be/product-detail/en/everlight-electronics-co-ltd/PT908-7C-F/1080-1164-ND/2675655
Power Switch	IC PWR SWITCH N-CHANNEL 1:1 8DIP	TPS2024P	1	**DigiKey**	https://www.digikey.be/product-detail/en/texas-instruments/TPS2024P/296-12234-5-ND/413526
Level Converter	IC BUF NON-INVERT 5.5V 14DIP	SN74HCT125N	1	**DigiKey**	https://www.digikey.be/product-detail/en/texas-instruments/SN74HCT125N/296-8386-5-ND/376860
Diode	DIODE SCHOTTKY 30V 1A DO41	1N5818	1	**DigiKey**	https://www.digikey.be/product-detail/en/stmicroelectronics/1N5818/497-4548-1-ND/770973
Res 200	RES 200 OHM 1/2W 1% AXIAL	SFR16S0002000FR500	2	**DigiKey**	https://www.digikey.be/product-detail/en/vishay-beyschlag-draloric-bc-components/SFR16S0002000FR500/PPC200XCT-ND/594751
Res 2k	RES 2K OHM 1/2W 1% AXIAL	SFR16S0002001FR500	2	**DigiKey**	https://www.digikey.be/product-detail/en/vishay-beyschlag-draloric-bc-components/SFR16S0002001FR500/PPC2.00KXCT-ND/594777
Res 10 k	RES 10K OHM 1/2W 1% AXIAL	SFR16S0001002FR500	3	**DigiKey**	https://www.digikey.be/product-detail/en/vishay-beyschlag-draloric-bc-components/SFR16S0001002FR500/PPC10.0KXCT-ND/594695
Res 100k	RES 100K OHM 0.4W 1% AXIAL	MBA02040C1003FRP00	2	**DigiKey**	https://www.digikey.be/product-detail/en/vishay-beyschlag-draloric-bc-components/MBA02040C1003FRP00/BC100KXCT-ND/336963
Capacitor 100nF	CAP CER 0.1UF 50V X7R RADIAL	C322C104K5R5TA7301	8	**DigiKey**	https://www.digikey.be/product-detail/en/kemet/C322C104K5R5TA7301/399-9877-1-ND/3726183
Capacitor 33uF	CAP ALUM POLY 33UF 20% 20V T/H	RNS1D330MDS1	2	**DigiKey**	https://www.digikey.be/product-detail/en/nichicon/RNS1D330MDS1/493-14237-ND/4991537
Antenna	RF ANT 868MHZ WHIP STR SMA MALE	ANT-868-CW-RH-SMA	1	**DigiKey**	https://www.digikey.be/product-detail/en/linx-technologies-inc/ANT-868-CW-RH-SMA/ANT-868-CW-RH-SMA-ND/1962848
Antenna connector	CONN SMA RCPT STR 50 OHM PCB	733910060	1	**DigiKey**	https://www.digikey.be/product-detail/en/molex/0733910060/WM5543-ND/1465165
Cable conn. on/off JST-XH male	CONN HEADER VERT 2POS 2.5MM	B2B-XH-A-M(LF)(SN)	1	**DigiKey**	https://www.digikey.be/product-detail/en/jst-sales-america-inc/B2B-XH-A-M-LF-SN/455-2879-ND/3926507
power connector 3p male (pcb)	TERM BLOCK HDR 3POS VERT 3.5MM	395016003	1	**Digikey**	https://www.digikey.be/product-detail/en/molex/0395016003/WM25698-ND/2735251

power connector 3p female	TERM BLOCK PLUG 3POS 3.5MM	395038003	1	**Digikey**	https://www.digikey.be/product-detail/en/molex/0395038003/WM25709-ND/4480336
in/output connector 7p male	TERM BLOCK HDR 7POS VERT 3.5MM	395016007	1	**DigiKey**	https://www.digikey.be/product-detail/en/molex/0395016007/WM25702-ND/2735255
in/output connector 7p female	TERM BLOCK PLUG 7POS 3.5MM	395037007	1	**Digikey**	https://www.digikey.be/product-detail/en/molex/0395037007/WM25519-ND/4480323
conn headers female 12p	CONN HDR 12POS 0.1 TIN PCB	PPTC121LFBN-RC	2	**DigiKey**	https://www.digikey.be/product-detail/en/sullins-connector-solutions/PPTC121LFBN-RC/S6100-ND/807231
plug-in IC sockets 4x2p	CONN IC DIP SOCKET 8POS TIN	A 08-LC-TT	2	**DigiKey**	https://www.digikey.be/product-detail/en/assmann-wsw-components/A-08-LC-TT/AE9986-ND/821740
plug-in IC sockets 7x2p	CONN IC DIP SOCKET 14POS TIN	A 14-LC-TT	1	**DigiKey**	https://www.digikey.be/product-detail/en/assmann-wsw-components/A-14-LC-TT/AE9989-ND/821743
EEPROM	512 Kbit, 64K x 8bit, Serial I2C (2-Wire), 400 kHz, DIP, 8 Pins	24LC512-I/P	1	**Farnell**	https://be.farnell.com/microchip/24lc512-i-p/serial-eeprom-512kbit-400khz-dip/dp/9758020?CMP=GRHB-OCTOPART
LoRaWAN module	RN2483A-I/RM104 or RN2483A-I/RM105	RN2483A-I/RM104	1	**Farnell**	https://be.farnell.com/microchip/rn2483a-i-rm104/transceiver-module-300kbps-870mhz/dp/2920841?st=rn2483
Coating spray	PLASTIK 70 conformal coating	2646050	/	**Farnell**	https://be.farnell.com/kontakt-chemie/plastik-70/conformal-coating-aerosol-200ml/dp/2646050
Battery	Li-ion 3500mah, 18650, flat top	12429	1	**VaBo**	http://www.vabo.be/ned/ned_main.htm
Silicone	Transparant sanitary Kit Handson 310 ml	604449	/	**Gamma**	https://www.gamma.nl/assortiment/handson-sanitairkit-transparant-310-ml/p/B604449
Tie wraps (100 pcs.)	200x2,5 mm 100p, Handson	110695	/	**Gamma**	https://www.gamma.nl/assortiment/handson-bindband-zwart-2-5-x100-mm-100-stuks/p/B110695
Solder tin	0.7 mm tin/copper 50 gram, Griffon	607125	/	**Gamma**	https://www.gamma.nl/assortiment/griffon-printsoldeer-0-7-mm-tin-koper-50-gram/p/B607125
PCB	Printed Circuit Board	/	1	**JLCPCB**	https://jlcpcb.com/
3D Pint	PLA print using	/	1	**FabLab**	https://fablab-leuven.be/?lang=en
wire	Silicon flexible wire, 7.5m 26AWG	KW-2371	/	**KIWI electronics**	https://www.kiwi-electronics.nl/kw-2371?search=Siliconen%20Draad%20Soepel&description=true
12 pin header	40 pin header strip	KW-2424	0.66	**KIWI electronics**	https://www.kiwi-electronics.nl/40-pin-header-strip?search=header%20pins&description=true
quick conn. on/off tab (10 pcs)	Tabs till 4.8mm, 2.54mm JST-XH - male	KW-2310	0.1	**KIWI electronics**	https://www.kiwi-electronics.nl/kw-2310
Plastic containers	IN: 56x36x50 mm, OUT: 60x40x52(LxBxH), with lid	31185	0.1	**Bodemschat**	https://www.bodemschat.nl/en/plastic-box-60-x-40-x-52-mm.html
Battery Module	5V Micro USB 18650 Lithium Battery Charger,UPS module Step up	/	1	**AliExpress**	https://nl.aliexpress.com/store/4706110?spm=a2g0o.detail.1000007.1.165664d57AGZiY

Addendum 3: Electrical scheme outlining the connections between the hardware parts.



Addendum 4: Firmware sketch outlining the code running on the microcontroller.

Link to the public GitHub repository containing all the firmware files that are needed to construct a functioning robotic flower:
https://github.com/Kamiel-debeuckelaere/One-Robotic-Flower-Field-To-Study-Them-All.git

```
#include <Arduino.h>
#include <FastLED.h>
#include <VarSpeedServo.h>
#include <rn2xx3.h>
//#include <EEPROM.h>
#include <cppQueue.h>
#include <LowPower.h>
#include <avr/power.h>

// include "variables" for normal function/ "variables_DEV" for developmental function
#include "variables"

#include "fixed_variables"

#include <uDebugLib.h>

//FUNCTIONS

// 1) LoRaWAN module (RN2483)

  // structuring
typedef struct strVisitRec
{
  unsigned long startVisit; // internal clock time start visit (sec.) --> 4 bytes
  uint8_t visit;            // visit duration (sec.) --> 1 byte
} visitRec;

// reserve space in SRAM memory for this max. amount of visitElements
visitRec visitContainer[RESERVED_VISIT_SPACE];

//make queue from container, FIFO: first in first out, overwrite = false
cppQueue visitQueue(sizeof(visitRec), RESERVED_VISIT_SPACE, FIFO, false);

struct Alarms
{
  bool queueIsFull : 1;     // 0= false, 1= true, too many visits for container
  bool sendDuringVisit : 1; // 0 = false, 1 = true, send during visit
  bool batteryWarning : 1;  // 0= false, 1 = true, battery under warning voltage
  bool obstruction : 1;     // 0= false, 1 = true, IR sensor obstructed longer than threshold
```

```
  bool startBit : 1;        // always 1, for decoding purposes in NodeRed
};

struct uplinkmsg
{
  uint16_t battery;
  Alarms alarms;
} uplinkmsg;

struct datamsg
{
  unsigned long startTimeBeforeSend;
  uint8_t visitDuration;
} datamsg;

    // sending message
TX_RETURN_TYPE SENDdata(uint8_t port, bool cnf = false)
{
  uplinkmsg.battery = Battery_voltage * 100;
  uplinkmsg.alarms.queueIsFull = queueIsFull;
  uplinkmsg.alarms.sendDuringVisit = sendDuringVisit;
  uplinkmsg.alarms.batteryWarning = batteryWarning;
  uplinkmsg.alarms.obstruction = obstruction;
  uplinkmsg.alarms.startBit = 1;

  String cmd = "mac tx ";
  cmd += (cnf) ? "cnf " : "uncnf ";
  cmd += port;
  cmd += " ";

  // Convert binary data to hexadecimal string
  String data;
  char buffer[3];
  for (uint8_t i = 0; i < sizeof(uplinkmsg); i++)
  {
    sprintf(buffer, "%02X", ((unsigned char *)&uplinkmsg)[i]);
    data += buffer;
    //memcpy(&msgBuffer[i * 2], &buffer, sizeof(buffer));
  }

  int sizeOfBuffer = sizeof(uplinkmsg);

  while ((sizeOfBuffer + 5 <= 53) && (!visitQueue.isEmpty()))
  { //checking if max. sending 53 bytes reached and if there is still data to
  send
    visitRec visitRec2send;
    visitQueue.pop(&visitRec2send);
    datamsg.startTimeBeforeSend = (millis() / PRECISION) -
  visitRec2send.startVisit;
    datamsg.visitDuration = visitRec2send.visit;

```

```
    for (uint8_t i = 0; i < sizeof(datamsg); i++)
    {
      sprintf(buffer, "%02X", ((unsigned char *)&datamsg)[i]);
      data += buffer;
    }

    sizeOfBuffer += sizeof(datamsg);
  }

  DEBUG_PRINT(F("   Size of data message: ")); //dev.
  DEBUG_PRINT(sizeOfBuffer);
        DEBUG_PRINT(F(" bytes")); //dev.

        retryCount = 0;
        TX_RETURN_TYPE resultOfSend;
        resultOfSend = myLora.txCommand(cmd, data, false);

        while ((TX_WITH_RX != resultOfSend) && retryCount < RETRY_AMOUNT)
        { //send data and receive downlink message from backend
          //DEBUG_PRINT(F("resultOfSend: ")); DEBUG_PRINTLN(resultOfSend);
          resultOfSend = myLora.txCommand(cmd, data, false);
          retryCount++;
        }

        if (retryCount == RETRY_AMOUNT)
        {
          DEBUG_PRINT(F(" / Failed TX and RX after "));
          DEBUG_PRINT(retryCount);
          DEBUG_PRINTLN(F(" retries"));
        }
        else
        {
          DEBUG_PRINTLN(F(" --> SEND"));
          timeToSleepString = myLora.getRx(); // get downlink message
          timeToSleep = timeToSleepString.toInt(); //change string to
integer
          DEBUG_PRINT(F("Sleep message: "));
          DEBUG_PRINT(timeToSleep); DEBUG_PRINT(F(" - "));
          DEBUG_PRINTLN(timeToSleepString);
        }
      }

      // 2) EEPROM configuration

      // Config that is stored and read from EEPROM
      struct eepromConfig
      {
        unsigned long writes; // Number of times a config is written to
EEPROM
        char DevEUI[16 + 1];  // Device EUI as a \0 terminated HEX string.
Example "0011223344556677"
```

```
137.        char AppEUI[16 + 1];  // Application EUI as a \0 terminated HEX
    string. Example "70B3D57ED00001A6"
138.        char AppKey[32 + 1];  // Application key as a \0 terminated HEX
    string. Example "A23C96EE13804963F8C2BD6285448198"
139.      } config;
140.
141.        // Write config data to EEPROM
142.      void configWrite(void)
143.      {
144.        config.writes++; // Increase
145.        eeprom_write_block((const void *)&config, (void *)0,
    sizeof(config));
146.      }
147.
148.        // Initialise config data and write to EEPROM (set everything to 0)
149.      void configInit(void)
150.      {
151.        //DEBUG_PRINTLN(F("Initializing configuration..."));
152.        //String hwEUI = myLora.hweui();
153.        myLora.hweui().toCharArray(config.DevEUI, sizeof(config.DevEUI));
154.        config.DevEUI[16] = 0; //add zero to terminate input
155.        memcpy(config.AppEUI, "0000000000000000", sizeof(config.AppEUI));
156.        config.AppEUI[16] = 0; //add zero to terminate input
157.        memcpy(config.AppKey, "00000000000000000000000000000000",
    sizeof(config.AppKey));
158.        config.AppKey[32] = 0; //add zero to terminate input
159.        configWrite();
160.        //DEBUG_PRINTLN(F("Initializing finished"));
161.      }
162.
163.        // Read config data from EEPROM (empty when uninitialised)
164.      void configRead(void)
165.      {
166.        eeprom_read_block((void *)&config, (void *)0, sizeof(config));
167.        if (config.writes == 0xFFFFFFFF)
168.        { // EEPROM has never been written... start with a clear config
169.          config.writes = 0;
170.          configInit();
171.        }
172.      }
173.
174.      // 3) DingNet connection
175.        // Menu print
176.      void menuPrint(void)
177.      {
178.        DEBUG_PRINTLN(F(""));
179.        DEBUG_PRINTLN(F("  ############################"));
180.        DEBUG_PRINTLN(F(" #  ONE ROBOTIC FLOWER FIELD  #"));
181.        DEBUG_PRINTLN(F(" #     TO STUDY THEM ALL      #"));
182.        DEBUG_PRINTLN(F("  ############################"));
183.        DEBUG_PRINTLN(F(""));
```

```
184.       DEBUG_PRINTLN(F(" [1]  Show device info"));
185.       DEBUG_PRINTLN(F(" [2]  Set DevEUI"));
186.       DEBUG_PRINTLN(F(" [3]  Set AppEUI"));
187.       DEBUG_PRINTLN(F(" [4]  Set AppKey"));
188.       DEBUG_PRINTLN(F(" [0]  Start operational mode"));
189.       DEBUG_PRINTLN(F(" [R]  Factory reset"));
190.       DEBUG_PRINTLN(F(""));
191.     }
192.
193.       // Read serial input (choice of menu, typing by user) until \r
    encounter or buffer is full (buflen-1 chars).
194.         // buffer is always 0 terminated!
195.         // Remove \r and \n from input
196.         // Return number of characters read
197.     int inputReadline(char *buf, uint8_t buflen)
198.     {
199.       uint8_t cnt = 0;
200.
201.       memset(buf, 0, buflen); // Clear current buffer
202.
203.       while (cnt < (buflen - 1))
204.       {
205.         while (Serial.available() > 0)
206.         {
207.           int data = Serial.read();
208.           Serial.write(data); // echo input back
209.           if (data == '\n')
210.           {
211.             continue;
212.           }
213.           if (data == '\r')
214.           {
215.             goto done;
216.           }
217.           buf[cnt] = data;
218.           cnt++;
219.         }
220.         delay(1); // Wait a little before checking serial again
221.       }
222.     done:
223.       DEBUG_PRINTLN(F("")); // Print a newline
224.       return cnt;
225.     }
226.
227.       // Check if the first 'len' characters of 'buf' are a hexadecimal
    string
228.         // Also modify buf and uppercase all characters
229.         // returns true if all checks ok, false otherwise
230.     bool verifyHexstring(char *buf, uint8_t len)
231.     {
232.       for (uint8_t i = 0; i < len; i++)
```

```
233.       {
234.         if (isHexadecimalDigit(buf[i])) // test if hexadecimal
235.         {
236.           buf[i] = toupper(buf[i]); // to uppercase
237.         }
238.         else
239.         {
240.           // Invalid input
241.           return false;
242.         }
243.       }
244.       return true;
245.     }
246.     #define serialBuflen 64 // Length of serial inputbuffer
247.
248.     // Handle user input in Serial Monitor
249.     void serialHandler(void)
250.     {
251.       char buf[serialBuflen];
252.       int len;
253.       char cmd;
254.
255.       menuPrint();
256.
257.       while (1)
258.       {
259.         DEBUG_PRINT(F("Command > "));
260.         len = inputReadline(buf, serialBuflen);
261.
262.         if (len != 1)
263.         {
264.           DEBUG_PRINTLN(F("Invalid input"));
265.           continue;
266.         }
267.
268.         cmd = buf[0]; // store 1st character in different variable (cmd)
   since we reuse buf
269.         switch (cmd)
270.         {
271.         case '1':
272.           DEBUG_PRINT(F("Config writes: "));
273.           DEBUG_PRINTLN(config.writes);
274.           DEBUG_PRINT(F("Radio version: "));
275.           DEBUG_PRINTLN(myLora.sysver());
276.           DEBUG_PRINT(F("HwEUI        : "));
277.           DEBUG_PRINTLN(myLora.hweui());
278.           DEBUG_PRINT(F("DevEUI       : "));
279.           DEBUG_PRINTLN(config.DevEUI);
280.           DEBUG_PRINT(F("AppEUI       : "));
281.           DEBUG_PRINTLN(config.AppEUI);
282.           DEBUG_PRINT(F("AppKey       : "));
```

```
283.           DEBUG_PRINTLN(config.AppKey);
284.           break;
285.         case '2':
286.           DEBUG_PRINT(F("Enter DevEUI: "));
287.           len = inputReadline(buf, serialBuflen);
288.           if (verifyHexstring(buf, 16))
289.           {
290.             memcpy(config.DevEUI, buf, sizeof(config.DevEUI));
291.             configWrite();
292.             DEBUG_PRINTLN(F("DevEUI saved"));
293.           }
294.           else
295.           {
296.             DEBUG_PRINTLN(F("ERROR: Invalid key!"));
297.           }
298.           break;
299.         case '3':
300.           DEBUG_PRINT(F("Enter AppEUI: "));
301.           len = inputReadline(buf, serialBuflen);
302.           if (verifyHexstring(buf, 16))
303.           {
304.             memcpy(config.AppEUI, buf, sizeof(config.AppEUI));
305.             configWrite();
306.             DEBUG_PRINTLN(F("AppEUI saved"));
307.           }
308.           else
309.           {
310.             DEBUG_PRINTLN(F("ERROR: Invalid key!"));
311.           }
312.           break;
313.         case '4':
314.           DEBUG_PRINT(F("Enter AppKey: "));
315.           len = inputReadline(buf, serialBuflen);
316.           if (verifyHexstring(buf, 32))
317.           {
318.             memcpy(config.AppKey, buf, sizeof(config.AppKey));
319.             configWrite();
320.             DEBUG_PRINTLN(F("AppKey saved"));
321.           }
322.           else
323.           {
324.             DEBUG_PRINTLN(F("ERROR: Invalid key!"));
325.           }
326.           break;
327.         case '0':
328.           DEBUG_PRINTLN(F("Switching to operational mode..."));
329.           return; // jump out of loop, start software
330.           break;
331.         case 'r':
332.         case 'R':
333.           DEBUG_PRINT(F("Are you sure? (y/n): "));
```

```
334.           len = inputReadline(buf, serialBuflen);
335.           if (len == 1 && (buf[0] == 'Y' || buf[0] == 'y'))
336.           {
337.             configInit(); // everything to 0, DevEUI reset
338.           }
339.           else
340.           {
341.             DEBUG_PRINTLN(F("Invalid input... aborting"));
342.           }
343.           break;
344.         default:
345.           DEBUG_PRINTLN(F("Unknown option!"));
346.           menuPrint();
347.         }
348.       }
349.     }
350.
351.     // 4) Flower functioning
352.     void Read_battery()
353.     {
354.       //voltage, *2 because of voltage devider, calibration value set at
   1.005797 with multimeter
355.       //measured voltage at VCC pin 3.27 V (not exactly 3.3 V)
356.       Battery_voltage = analogRead(Battery_pin) * 2 * (3.27 / 1024) *
   1.005797;
357.
358.       if (Battery_voltage >= WARNING_VOLTAGE)
359.       {
360.         batteryWarning = 0; //normal function
361.       }
362.       if (Battery_voltage < WARNING_VOLTAGE)
363.       {
364.         batteryWarning = 1; //warning: battery low
365.       }
366.     }
367.
368.     void refill()
369.     {
370.       myServo.attach(servoControl);
371.       digitalWrite(PowerSwitch, LOW); //power switch ON
372.       DEBUG_PRINTLN(F("°Refill°"));
373.       myServo.write(SERVO_OPEN, SERVO_SPEED, true);  //position = open,
   slow speed, true = wait until position reached
374.       delay(REFILL_TIME);                            // keep servo arm in
   nectar reservoir
375.       myServo.write(SERVO_CLOSE, SERVO_SPEED, true); //position = close
376.       digitalWrite(PowerSwitch, HIGH);               //power switch OFF
377.       myServo.detach();                              //to save energy
378.       refillNeed = 0;                                //reset refill need
379.
380.       startTimeAutoRefill = millis();
```

```
381.      }
382.
383.      void automaticRefill()
384.      {
385.        if (millis() >= startTimeAutoRefill + AUTO_REFILL_TIME)
386.        {
387.          DEBUG_PRINT(F("Auto refill...")); //dev.
388.          refill();
389.        }
390.      }
391.
392.      void Battery_LEDflash(unsigned long normalInterval, unsigned long
    warningInterval)
393.      {
394.        currentBatteryTime = millis();
395.        if (DEV_MODE == true) //development
396.        {
397.          if (batteryWarning == 0)
398.          {
399.            if (currentBatteryTime >= LightTimer + normalInterval and
    currentBatteryTime <= LightTimer + normalInterval + FLASH_TIME)
400.            {
401.              digitalWrite(PowerSwitch, LOW); // ON
402.              FastLED.setBrightness(20);
403.              leds[0] = CRGB::Green;
404.              FastLED.show();
405.            }
406.
407.            if (currentBatteryTime > LightTimer + normalInterval +
    FLASH_TIME)
408.            {
409.              leds[0] = CRGB::Black;
410.              FastLED.show();
411.              digitalWrite(PowerSwitch, HIGH); // OFF
412.              LightTimer = millis();
413.            }
414.          }
415.
416.          if (batteryWarning == 1) // flashing to show battery low
417.          {
418.            if (currentBatteryTime >= LightTimer + warningInterval)
419.            {
420.              digitalWrite(PowerSwitch, LOW); // ON
421.              FastLED.setBrightness(20);
422.              leds[0] = CRGB::Red;
423.              FastLED.show();
424.            }
425.
426.            if (currentBatteryTime >= LightTimer + warningInterval +
    FLASH_TIME)
427.            {
```

```
428.              leds[0] = CRGB::Black;
429.              FastLED.show();
430.              digitalWrite(PowerSwitch, HIGH); // OFF
431.              LightTimer = millis();
432.            }
433.          }
434.        }
435.      }
436.
437.      void savetoRAM(unsigned long startTime, uint8_t visitDuration)
438.      {
439.        DEBUG_PRINTLN(F(""));
440.        DEBUG_PRINT(F(" --> SAVING..."));
441.
442.        if (DEV_MODE == true) //development
443.        {
444.          digitalWrite(PowerSwitch, LOW); // ON
445.          FastLED.setBrightness(50);
446.          leds[0] = CRGB::SkyBlue;
447.          FastLED.show();
448.          delay(50);
449.        }
450.
451.        if (visitDuration > 0)
452.        { //only save when visit duration at least 1 sec.
453.          visitRec element;
454.          element.startVisit = startTime; //in sec.
455.          element.visit = visitDuration;  //in sec.
456.          if (!visitQueue.push(&element)) // add visit to queue -> 'first in
   first out' principle
457.          {
458.            //if false: queue is full, too many visits
459.            queueIsFull = 1;
460.            DEBUG_PRINT(F("ERROR: RESERVED_VISIT_SPACE full"));
461.          }
462.
463.          else
464.          {
465.            DEBUG_PRINTLN(F(" SAVED"));
466.            DEBUG_PRINTLN(F(""));
467.            queueIsFull = 0;
468.          }
469.        }
470.
471.        leds[0] = CRGB::Black;           //development
472.        FastLED.show();                  //development
473.        digitalWrite(PowerSwitch, HIGH); // OFF //development
474.      }
475.
476.      void sendData(unsigned long frequency, uint8_t port) // frequency in
   ms
```

```
477.     {
478.       if (millis() >= sendStartTime + frequency)
479.       {
480.         startSending = millis(); //development
481.
482.         myLora.autobaud(); //wake up Lora module
483.         delay(15);
484.
485.         DEBUG_PRINT(F("SENDING..."));
486.
487.         if (DEV_MODE == true) //development
488.         {
489.           digitalWrite(PowerSwitch, LOW); // ON
490.           FastLED.setBrightness(50);
491.           leds[0] = CRGB::Yellow;
492.           FastLED.show();
493.         }
494.
495.         Read_battery(); //to send battery level to backend application
496.
497.         //sending during visit
498.         if (visitState == 1)
499.         {
500.           DEBUG_PRINT(F(" --> during visit"));
501.           visitCounter = round((millis() - startVisitTimer) / PRECISION);
502.           savetoRAM(startVisit, visitCounter);
503.           visitState = 0;
504.           sendDuringVisit = 1;
505.         }
506.
507.         SENDdata(port); //sending, port can be specified
508.
509.         queueIsFull = 0;     // reset
510.         sendDuringVisit = 0; //reset
511.
512.         //DEBUG_PRINT(F("Battery: "));DEBUG_PRINT(Battery_voltage);
    DEBUG_PRINTLN(F("v"));
513.         //DEBUG_PRINT(F("Warning: ")); DEBUG_PRINTLN(batteryWarning);
    DEBUG_PRINTLN(F(""));
514.
515.         //shut down LED when it was switched on in development mode
516.         leds[0] = CRGB::Black;           //development
517.         FastLED.show();                  //development
518.         digitalWrite(PowerSwitch, HIGH); // OFF //development
519.
520.         sendStartTime = millis(); //timer for message sending reset
521.
522.         myLora.sleep(43200000); //save energy in between sending, sleeping
    12 hrs or untill waked up
523.
524.         //show how long microcontroller occupied with sending function
```

```
525.          DEBUG_PRINT(F("send time: ")); //development
526.          DEBUG_PRINTLN((millis() - startSending) / PRECISION);
   //development
527.        }
528.      }
529.
530.      // SET-UP
531.      void setup()
532.      {
533.        //extra energy saving possibile with the right hardware:
534.          //delay(40000);
535.          //clock_prescale_set(clock_div_8); // change clock speed to 1 MHz
   (divide 8 MHz by 8) (also compile sketch with this clock speed)
536.
537.        // 1 Flower function
538.
539.        pinMode(LoRaReset, OUTPUT);
540.        digitalWrite(LoRaReset, LOW); //reset LoRa module at start of flower
541.        delay(10);
542.        digitalWrite(LoRaReset, HIGH);
543.
544.        debugSerial.begin(9600); // link to serial monitor (cable)
545.        delay(5000);             //give 5sec. time to open Serial Monitor
546.
547.        pinMode(PowerSwitch, OUTPUT);
548.        digitalWrite(PowerSwitch, LOW); // power switch ON
549.
550.        FlowerState = wakeUp; //begin state
551.
552.        // 2 InfraRed System
553.        pinMode(IRreceiverPower, OUTPUT);
554.        pinMode(IRemitterPower, OUTPUT);
555.        pinMode(IRsensorRead, INPUT);
556.
557.        obstruction = 0;
558.
559.        // 3 Battery
560.        pinMode(Battery_pin, INPUT);
561.        batteryWarning = 0;
562.
563.        // 4 LED
564.        FastLED.addLeds<WS2812, LEDdataPin, RGB>(leds, 1);
565.        FastLED.setBrightness(40);
566.        digitalWrite(PowerSwitch, LOW); // ON
567.        leds[0] = CRGB::Red;
568.        FastLED.show();
569.
570.        // 5 LoRaWAN initialisation
571.        loraSerial.begin(57600); //send data via Lora radio
572.        delay(3000);
573.        leds[0] = CRGB::Black;
```

```
574.        FastLED.show();
575.
576.        sendDuringVisit = 0;
577.        queueIsFull = 0;
578.        DEBUG_PRINTLN(F(""));
579.        DEBUG_PRINTLN(F("Starting Robotic Flower..."));
580.        DEBUG_PRINTLN(F(""));
581.
582.        delay(500); // Allow some time for the RN2483 to boot and do a
    factory reset
583.        //loraSerial.println("sys factoryRESET");
584.        delay(2000);
585.        myLora.setFrequencyPlan(TTN_EU);
586.        myLora.autobaud();
587.        //DEBUG_PRINTLN(F("Radio module reset"));
588.
589.        // Read config from EEPROM
590.        configRead();
591.        //DEBUG_PRINTLN(F("Read config from EEPROM"));
592.
593.        // Initialise the hardware
594.        //DEBUG_PRINTLN(F("Initialise hardware"));
595.
596.        unsigned long begin = millis();
597.        while (millis() - begin < 10000)
598.        { // when not connected to serial monitor, try to join network
    automatically after 10 sec. with current config
599.          delay(200);
600.          if (debugSerial || debugSerial.available())
601.          {
602.            serialHandler(); //open menu
603.            break;
604.          }
605.        }
606.
607.        // Try to join the network
608.        leds[0] = CRGB::Red;
609.        FastLED.show();
610.        DEBUG_PRINT(F("OTAA join DingNet... "));
611.        while (!myLora.initOTAA(config.AppEUI, config.AppKey,
    config.DevEUI))
612.        { //try to make connection, if not successful try again
613.          DEBUG_PRINTLN(F("FAIL (retry in 10 sec.)"));
614.          delay(10000);
615.          DEBUG_PRINT(F("OTAA join DingNet... ")); //over the air activation
616.          delay(100);
617.        }
618.        DEBUG_PRINTLN(F("OK"));
619.        leds[0] = CRGB::Green;
620.        FastLED.show();
621.        delay(2000);
```

```
622.        leds[0] = CRGB::Black;
623.        FastLED.show();
624.
625.        // 6 Initialize Servo system
626.
627.        myServo.attach(servoControl); //open servo for first fill of nectar
    cup
628.
629.        // 7 start flower functioning
630.        leds[0] = CRGB::Green;
631.        FastLED.show();
632.        delay(100);
633.        leds[0].fadeLightBy(255);
634.        FastLED.show();
635.        delay(100);
636.        leds[0] = CRGB::Green;
637.        FastLED.show();
638.        delay(100);
639.        leds[0] = CRGB::Black;
640.        FastLED.show();
641.
642.        digitalWrite(PowerSwitch, HIGH); // OFF
643.
644.        LightTimer = millis(); // start timer for flashing
645.
646.        Read_battery();
647.
648.        if (DEV_MODE == false)
649.        { // shut of USB module to save energy
650.          USBCON |= (1 << FRZCLK); // Freeze the USB Clock
651.          PLLCSR &= ~(1 << PLLE);  // Disable the USB Clock (PPL)
652.          USBCON &= ~(1 << USBE);  // Disable the USB
653.        }
654.
655.        SENDdata(2); //first send for set-up
656.        myLora.sleep(43200000); //save energy in between sending
657.      }
658.
659.      // MAIN LOOP
660.      void loop()
661.      {
662.        //1. Going-To-Sleep State
663.        if (FlowerState == goingToSleep)
664.        {
665.          DEBUG_PRINTLN(F(""));
666.          DEBUG_PRINTLN(F("GOING TO SLEEP"));
667.
668.          if (visitCounter != 0)
669.          { // bee on flower at start sleep-mode
670.            DEBUG_PRINTLN(F("--> during visit"));
```

```
671.            savetoRAM(startVisit, visitCounter); // save start time & visit
    duration
672.          }
673.
674.          visitState = 0;
675.
676.          digitalWrite(PowerSwitch, HIGH); //power switch OFF
677.          digitalWrite(IRemitterPower, LOW);
678.          digitalWrite(IRreceiverPower, LOW);
679.
680.          while (!visitQueue.isEmpty())
681.          { //send data untill queue is empty
682.            sendData(GOING_TO_SLEEP_SEND_FREQUENCY, 1); // send data over
    LoRaWAN, port 1
683.          }
684.
685.          DEBUG_PRINTLN(F(""));
686.          DEBUG_PRINTLN(F("SLEEP"));
687.
688.          FlowerState = sleep;
689.
690.        } // goingToSleep state end
691.
692.        //2. Sleep State
693.        if (FlowerState == sleep)
694.        {
695.          while (sleepCounter < SLEEP_TIME)
696.          { // deep sleep for some time during night to save battery
697.            sleepCounter += 8;
698.            if (DEV_MODE == true)
699.            { //keep serial monitor open for development
700.              delay(8000);
701.              DEBUG_PRINT(F("Deep Sleep:"));
702.              DEBUG_PRINT(sleepCounter);
703.              DEBUG_PRINT(F("/"));
704.              DEBUG_PRINTLN(SLEEP_TIME);
705.            }
706.            else
707.            { //.powerDown shuts down serial monitor (energy saving)
708.              LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF); //max. sleep
    time 8 sec. (+- 10%)
709.            }
710.          }
711.
712.          //interupt sleep to send message now and then to be able to
    receive message with LoRaWAN
713.          if (DEV_MODE == true)
714.          { //keep serial monitor open for development
715.            sendData(SLEEP_SEND_FREQUENCY_DEV, 2); // send data over
    LoRaWAN, port 2
716.            delay(8000);
```

```
717.          }
718.
719.          else
720.          {
721.            while (wakeUpSendCounter < SLEEP_SEND_FREQUENCY)
722.            {
723.              wakeUpSendCounter += 8;
724.              LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF); //max. sleep
    time 8 sec. (+- 10%)
725.            }
726.            sendData(0,2); //send to be able to receive message for waking
    up
727.            wakeUpSendCounter = 0; // reset counter
728.          }
729.
730.          if (timeToSleep == false) //Go to wake-up when message received
    through LoRaWAN tells it's time to do so
731.          {
732.            FlowerState = wakeUp;
733.          }
734.
735.        } // sleep state end
736.
737.        //3. Wake-Up State
738.        if (FlowerState == wakeUp)
739.        {
740.          sleepCounter = 0; // reset
741.
742.          DEBUG_PRINTLN(F(""));
743.          DEBUG_PRINTLN(F("WAKING UP"));
744.
745.          refillGapTimer = millis(); //first refill in work state when timer
    ends
746.          refillNeed = 1;
747.
748.          startTimeAutoRefill = millis(); //start automatic refill timer
749.          sendStartTime = millis();       //start timer for message sending
750.          LightTimer = millis();          // start timer for flashing
751.
752.          //first IR-sensot read to start
753.          digitalWrite(IRreceiverPower, HIGH);
754.          delay(1);
755.          digitalWrite(IRemitterPower, HIGH);
756.          delay(1);
757.          previousSensorValue = analogRead(IRsensorRead);
758.          digitalWrite(IRreceiverPower, LOW);
759.          digitalWrite(IRemitterPower, LOW);
760.
761.          DEBUG_PRINTLN(F(""));
762.          DEBUG_PRINTLN(F("WORKING"));
763.
```

```
764.          FlowerState = work;
765.
766.        } // wake-up state end
767.
768.        //4. Work State
769.        if (FlowerState == work)
770.        {
771.
772.          if (timeToSleep == true)
773.          { //Go to sleep when message received through LoRaWAN tells it's
    time to do so
774.            FlowerState = goingToSleep;
775.          }
776.
777.          Battery_LEDflash(NORMAL_FLASH_INTERVAL, WARNING_FLASH_INTERVAL);
    // flash led to show functioning (development)
778.          sendData(WORK_SEND_FREQUENCY, 1);   // send data over LoRaWAN
    (e.g. every 10min.), port 1
779.          automaticRefill();                  // automatic servo movement to
    prevent drying out of nectar
780.
781.          // detection system timer reset
782.          if (millis() > startTimeCycle + CYCLE_TIME)
783.          {
784.            startTimeCycle = millis();
785.          }
786.
787.          // refill interval after visit
788.          if (millis() - refillGapTimer >= REFILL_GAP and millis() -
    refillGapTimer < REFILL_GAP + 10 and refillNeed == 1)
789.          {
790.            refill();
791.          }
792.
793.          // detection on
794.          if (millis() <= startTimeCycle + ON_TIME)
795.          {
796.            //IR-sensor system turned on once every TimeCycle
797.            digitalWrite(IRreceiverPower, HIGH);
798.            delay(1);
799.            digitalWrite(IRemitterPower, HIGH);
800.            delay(1);
801.            sensorValue = analogRead(IRsensorRead); //read voltage of IR
    sensor
802.            delay(1);
803.            digitalWrite(IRemitterPower, LOW);
804.            digitalWrite(IRreceiverPower, LOW);
805.
806.            if (SHOW_IR_VALUE == true) //development
807.            {
808.              DEBUG_PRINT(F("IR: "));
```

```
809.              DEBUG_PRINTLN(sensorValue);
810.              //DEBUG_PRINT(F("Previous value: "));
   DEBUG_PRINTLN(previousSensorValue);
811.            }
812.
813.            // visit
814.            if (sensorValue <= previousSensorValue - SENSOR_SENSITIVITY)
815.            {
816.              if (DEV_MODE == true)
817.              {
818.                digitalWrite(PowerSwitch, LOW); // ON
819.                FastLED.setBrightness(50);
820.                leds[0] = CRGB::Red;
821.                FastLED.show();
822.              }
823.
824.              if (visitState == 0)
825.              { //new visit starting
826.                startVisit = round(millis() / PRECISION);
827.                startVisitTimer = millis();
828.              }
829.
830.              // max duration of 250 sec; because of rounding, visits less
   then 0,5 sec. are not registered (=0)
831.              visitCounter = round((millis() - startVisitTimer) /
   PRECISION);
832.
833.              visitState = 1;
834.              previousVisitState = 1;
835.
836.              // max visit time reached
837.              if (visitCounter >= MAX_VISIT_DURATION) //maxCounter not over
   255 (to be able to send visitDuration in 1 byte)
838.              { // visitor might be sleeping (or dead) in flower
839.
840.                DEBUG_PRINTLN(F("Max. visit")); //dev.
841.                //DEBUG_PRINT(visitCounter); DEBUG_PRINTLN(F("sec."));
842.
843.                savetoRAM(startVisit, visitCounter); //save visit start time
   & visit duration
844.
845.                visitState = 0;
846.                previousVisitState = 0;
847.
848.                obstrCounter++;
849.                if (obstrCounter >= OBSTR_TRESH)
850.                {
851.                  obstruction = 1;
852.                }
853.              }
854.
```

```
855.              startTimeAutoRefill = millis(); //avoid automatic refill
   during visit, restart timer
856.              refillGapTimer = millis();      //refill after visit to renew
   'nectar'
857.              LightTimer = millis();          //development, avoid LED
   flashing during visit
858.            }
859.
860.            // no visit
861.            else
862.            {
863.              visitState = 0;
864.              previousSensorValue = sensorValue;
865.              obstruction = 0; //reset
866.
867.              //shut down LED when it was switched on in development mode
868.              leds[0] = CRGB::Black;
869.              FastLED.show();
870.              digitalWrite(PowerSwitch, HIGH); // OFF
871.            }
872.
873.            // end of visit
874.            if (previousVisitState != visitState)
875.            {
876.              DEBUG_PRINT(F("Visit: "));
877.              DEBUG_PRINT(visitCounter);
878.              DEBUG_PRINTLN(F("sec."));
879.
880.              savetoRAM(startVisit, visitCounter); //save start time & visit
   duration to queue
881.
882.              if (visitCounter > VISIT_TRESHOLD)
883.              {
884.                DEBUG_PRINTLN(F("Refill gap"));
885.                DEBUG_PRINTLN(F(""));
886.                refillNeed = 1;       //need to refill after visit
887.              }
888.              visitCounter = 0;
889.            }
890.
891.            previousVisitState = visitState;
892.
893.            delay(100); // check if cycle has passed every 10 times per
   second in stead of constant, save energy
894.
895.          }
896.        } // work state end
897.      }
```

Addendum 5: Node-RED flow (JSON).

These flows in JSON (JavaScript) can be copied and exported directly into Node-RED to construct the backend application. The flower flow gives the code that is needed for each flower in the robotic flower field, the File logger/browser flow contains the code only needed once per robotic flower field and is responsible for the creation of the CSV-files.

Flower flow


```
[{"id":"5fe64185.d640d","type":"tab","label":"Prototype","disabled":false,"info":""},{"id":"b2e1
34c1.57fbd8","type":"mqtt in","z":"5fe64185.d640d","name":"Uplink
Prototype","topic":"v3/robotic-flower2/devices/flower-
prototype2/up","qos":"1","datatype":"auto","broker":"c0072397.aa5ca","x":100,"y":40,"wires":[["c
0ec4484.7f51d"]]},{"id":"ebda4a9e.2344d","type":"function","z":"5fe64185.d640d","name":"Uplink
decoder","func":"// \"v3/robotic-flower2/devices/flower-prototype2/up\"\n\nvar buf =
Buffer.from(msg.payload.uplink_message.frm_payload,'base64');\nvar decoded =
{};\n\ndecoded.battery = buf.readUint16LE(0)/100;\n\nvar binary =
buf.readUInt8(2).toString(2).split('').map(function(s) { return parseInt(s);
});\n\ndecoded.queueIsFull = binary[4]; //bit 4\ndecoded.sendDuringVisit = binary[3]; //bit
3\ndecoded.batteryWarning = binary[2]; //bit 2\ndecoded.possibleObstruction = binary[1]; //bit
1\n\nif (buf.length > 3){\n    var timeBeforeSend = {};\n    var visitDuration = {};\n    for
(let i = 1;i < ((buf.length-5)/5+1);i++) {\n        timeBeforeSend[i] = buf.readUint16LE(i*5);\n
visitDuration[i] = buf.readUint8(i*5+2);\n    }\n}\n\nvar newmsg = {};\nnewmsg.application =
msg.payload.end_device_ids.application_ids.application_id;\nnewmsg.device =
msg.payload.end_device_ids.device_id;\nnewmsg.port =
msg.payload.uplink_message.f_port\nnewmsg.time = msg.payload.received_at;\nnewmsg.payload =
decoded;\nnewmsg.dataSize = buf.length; \nnewmsg.timeBeforeSend =
timeBeforeSend\nnewmsg.visitDuration = visitDuration;\n\n\nreturn
newmsg;","outputs":1,"noerr":0,"initialize":"// Code added here will be run
once\n","finalize":"","x":160,"y":260,"wires":[["1cd3efcd.4e5dd","bf21c776.9f4f58","aa9a1cba.d5a
788","ca8e13e6.bf70c8","746b9af5.2fdcac","bb09e30d.de7c7","dd2e8361.f364b","4b6e1a13.183214","a9
4c2463.570248"]]},{"id":"1cd3efcd.4e5dd","type":"debug","z":"5fe64185.d640d","name":"Decoded","a
ctive":false,"tosidebar":true,"console":false,"tostatus":false,"complete":"true","targetType":"f
ull","statusVal":"","statusType":"auto","x":340,"y":180,"wires":[]},{"id":"c0ec4484.7f51d","type
":"json","z":"5fe64185.d640d","name":"","property":"payload","action":"","pretty":false,"x":110,
"y":120,"wires":[["ebda4a9e.2344d","7a0e152a.1877ec"]]},{"id":"bf21c776.9f4f58","type":"change",
"z":"5fe64185.d640d","name":"battery","rules":[{"t":"set","p":"payload","pt":"msg","to":"payload
.battery","tot":"msg"}],"action":"","property":"","from":"","to":"","reg":false,"x":280,"y":460,
"wires":[["3c27ca49.4b5236","c39a53db.03f93"]]},{"id":"3c27ca49.4b5236","type":"ui_chart","z":"5
fe64185.d640d","name":"","group":"b855ee7d.1894d8","order":3,"width":0,"height":0,"label":"","ch
artType":"line","legend":"false","xformat":"HH:mm:ss","interpolate":"linear","nodata":"","dot":t
rue,"ymin":"3","ymax":"4.2","removeOlder":"1","removeOlderPoints":"","removeOlderUnit":"604800",
"cutout":0,"useOneColor":false,"useUTC":false,"colors":["#1f77b4","#aec7e8","#ff7f0e","#2ca02c",
"#98df8a","#d62728","#ff9896","#9467bd","#c5b0d5"],"outputs":1,"useDifferentColor":false,"x":430
,"y":440,"wires":[[]]},{"id":"c39a53db.03f93","type":"ui_gauge","z":"5fe64185.d640d","name":"","
group":"b855ee7d.1894d8","order":2,"width":0,"height":0,"gtype":"gage","title":"","label":"Volt"
,"format":"{{value}}","min":"3","max":"4.2","colors":["#b21f05","#e6e600","#1baa07"],"seg1":"","
seg2":"","x":430,"y":480,"wires":[]},{"id":"aa9a1cba.d5a788","type":"change","z":"5fe64185.d640d
","name":"queueIsFull","rules":[{"t":"set","p":"payload","pt":"msg","to":"payload.queueIsFull","
tot":"msg"}],"action":"","property":"","from":"","to":"","reg":false,"x":230,"y":520,"wires":[["
cfb6a9ad.437f18"]]},{"id":"ca8e13e6.bf70c8","type":"change","z":"5fe64185.d640d","name":"sendDur
ingVisit","rules":[{"t":"set","p":"payload","pt":"msg","to":"payload.sendDuringVisit","tot":"msg
"}],"action":"","property":"","from":"","to":"","reg":false,"x":240,"y":560,"wires":[["8ac692c7.
6180a8"]]},{"id":"746b9af5.2fdcac","type":"change","z":"5fe64185.d640d","name":"batteryWarning",
"rules":[{"t":"set","p":"payload","pt":"msg","to":"payload.batteryWarning","tot":"msg"}],"action
":"","property":"","from":"","to":"","reg":false,"x":240,"y":600,"wires":[["13101b4b.71c3a5"]]},
{"id":"bb09e30d.de7c7","type":"change","z":"5fe64185.d640d","name":"possibleObstruction","rules"
:[{"t":"set","p":"payload","pt":"msg","to":"payload.possibleObstruction","tot":"msg"}],"action":
"","property":"","from":"","to":"","reg":false,"x":260,"y":640,"wires":[["6a274429.9b93a4"]]},{"
id":"cfb6a9ad.437f18","type":"ui_gauge","z":"5fe64185.d640d","name":"","group":"5bdb97c6.ccc958"
,"order":2,"width":"3","height":"3","gtype":"gage","title":"Queue is
full","label":"","format":"{{value}}","min":0,"max":"1","colors":["#00b500","#e6e600","#ca3838"]
```

```
,"seg1":"","seg2":"","x":450,"y":520,"wires":[]},{"id":"8ac692c7.6180a8","type":"ui_gauge","z":"
5fe64185.d640d","name":"","group":"5bdb97c6.ccc958","order":3,"width":"3","height":"3","gtype":"
gage","title":"Send during
visit","label":"","format":"{{value}}","min":0,"max":"1","colors":["#00b500","#e6e600","#ca3838"
],"seg1":"","seg2":"","x":470,"y":560,"wires":[]},{"id":"13101b4b.71c3a5","type":"ui_gauge","z":
"5fe64185.d640d","name":"","group":"5bdb97c6.ccc958","order":4,"width":"3","height":"3","gtype":
"gage","title":"Battery
Warning","label":"","format":"{{value}}","min":0,"max":"1","colors":["#00b500","#e6e600","#ca383
8"],"seg1":"","seg2":"","x":460,"y":600,"wires":[]},{"id":"6a274429.9b93a4","type":"ui_gauge","z
":"5fe64185.d640d","name":"","group":"5bdb97c6.ccc958","order":5,"width":"3","height":"3","gtype
":"gage","title":"Possible
obstruction","label":"","format":"{{value}}","min":"0","max":"1","colors":["#00b500","#e6e600","
#ca3838"],"seg1":"","seg2":"","x":480,"y":640,"wires":[]},{"id":"dd2e8361.f364b","type":"change"
,"z":"5fe64185.d640d","name":"time","rules":[{"t":"set","p":"payload","pt":"msg","to":"","tot":"
date"}],"action":"","property":"","from":"","to":"","reg":false,"x":310,"y":380,"wires":[["9ab9c
a01.d6617","117dc472.87787c"]]},{"id":"9ab9ca01.d6617","type":"ui_text","z":"5fe64185.d640d","gr
oup":"5bdb97c6.ccc958","order":1,"width":6,"height":2,"name":"","label":"Received:","format":"{{
msg.payload | date:'medium'}}","layout":"row-
center","x":440,"y":360,"wires":[]},{"id":"117dc472.87787c","type":"ui_text","z":"5fe64185.d640d
","group":"b855ee7d.1894d8","order":1,"width":6,"height":2,"name":"","label":"Received:","format
":"{{msg.payload | date:'medium'}}","layout":"row-
center","x":440,"y":400,"wires":[]},{"id":"702e6189.34ea5","type":"link
out","z":"5fe64185.d640d","name":"Auto
Logger","links":["f4879cf0.698a28","f390bbd3.d7e4d8"],"x":715,"y":240,"wires":[]},{"id":"23abae1
6.7e1592","type":"mqtt out","z":"5fe64185.d640d","name":"Downlink
Prototype","topic":"v3/robotic-flower2/devices/flower-
prototype2/down/push","qos":"1","retain":"","broker":"c0072397.aa5ca","x":570,"y":80,"wires":[],
"inputLabels":["msg.payload"]},{"id":"7a0e152a.1877ec","type":"function","z":"5fe64185.d640d","n
ame":"timeToSleep","func":"var currentTime = new Date(msg.payload.received_at);\n\nvar
sleep_time = flow.get(\"sleep_time\");\nvar waking_up = flow.get(\"waking_up\");\n\nvar hours =
currentTime.getHours();\nvar timeToSleep;\n\nif ((hours >= sleep_time) || (hours < waking_up)){
//start & end of sleep mode (e.g. 21h-6h)\n    timeToSleep = \"AQ==\"; //base64: 1\n}\nelse{\n
timeToSleep = \"AA==\"; //base64: 0\n}\n\n\nmsg.payload = {\n  \"downlinks\": [{\n
\"f_port\": 15,\n    \"frm_payload\": timeToSleep,\n    \"priority\": \"NORMAL\"\n
}]\n}\n\nreturn
msg;","outputs":1,"noerr":0,"initialize":"","finalize":"","x":350,"y":80,"wires":[["23abae16.7e1
592","5b252a34.d1191c"]]},{"id":"5b252a34.d1191c","type":"debug","z":"5fe64185.d640d","name":"Ti
me to
sleep?","active":false,"tosidebar":true,"console":false,"tostatus":false,"complete":"payload","t
argetType":"msg","statusVal":"","statusType":"auto","x":540,"y":140,"wires":[]},{"id":"f174c710.
a4d96","type":"function","z":"5fe64185.d640d","name":"","func":"var device = msg.device\n//put
visitDuration in array\nvar visitDurationObject = msg.visitDuration;\nvar visitDuration =
Object.values(visitDurationObject);\n\n//put timeBeforeSend in array\nvar timeBeforeSendObject =
msg.timeBeforeSend;\nvar timeBeforeSend = Object.values(timeBeforeSendObject);\n\n//change
timeBeforeSend to visitTime\nvar sendTime = new Date(msg.time);\nvar h =
sendTime.getHours();\nvar min = sendTime.getMinutes();\nvar sec = sendTime.getSeconds();\nvar
sendTimeInSec = h*3600+min*60+sec;\nvar visitTime
=[];\n\n\ntimeBeforeSend.forEach(function(e){\n    var receivedTimeInSec = sendTimeInSec - e;\n
var date = new Date(0);\n    date.setSeconds(receivedTimeInSec);\n
visitTime.push(date.toISOString().substr(11, 8));\n})\n\nmsg.visitTime =
visitTime;\nmsg.visitDuration = visitDuration;\nmsg.device = device;\nmsg.arrayLength =
visitTime.length;\n\nreturn
msg;","outputs":1,"noerr":0,"initialize":"","finalize":"","x":560,"y":240,"wires":[["702e6189.34
ea5","2cf36923.8f7dae","3b44f125.be520e","45abc684.5aa9c"]]},{"id":"4b6e1a13.183214","type":"swi
tch","z":"5fe64185.d640d","name":"Check
port","property":"port","propertyType":"msg","rules":[{"t":"eq","v":"1","vt":"str"}],"checkall":
"true","repair":false,"outputs":1,"x":370,"y":220,"wires":[["2dd1d211.e9b48e"]],"info":"No use
to continue this of flower in sleep mode (port
2)"},{"id":"2cf36923.8f7dae","type":"debug","z":"5fe64185.d640d","name":"visitTime","active":fal
se,"tosidebar":true,"console":false,"tostatus":false,"complete":"visitTime","targetType":"msg","
statusVal":"","statusType":"auto","x":700,"y":300,"wires":[]},{"id":"3b44f125.be520e","type":"de
bug","z":"5fe64185.d640d","name":"visitDuration","active":false,"tosidebar":true,"console":false
,"tostatus":false,"complete":"visitDuration","targetType":"msg","statusVal":"","statusType":"aut
o","x":710,"y":340,"wires":[]},{"id":"a94c2463.570248","type":"debug","z":"5fe64185.d640d","name
":"device","active":true,"tosidebar":true,"console":false,"tostatus":false,"complete":"device","
targetType":"msg","statusVal":"","statusType":"auto","x":330,"y":140,"wires":[]},{"id":"45abc684
.5aa9c","type":"debug","z":"5fe64185.d640d","name":"arrayLength","active":false,"tosidebar":true
,"console":false,"tostatus":false,"complete":"arrayLength","targetType":"msg","statusVal":"","st
atusType":"auto","x":710,"y":380,"wires":[]},{"id":"2dd1d211.e9b48e","type":"switch","z":"5fe641
85.d640d","name":"","property":"visitDuration","propertyType":"msg","rules":[{"t":"nempty"}],"ch
```

```
eckall":"true","repair":false,"outputs":1,"x":430,"y":280,"wires":[["f174c710.a4d96"]]},{"id":"3
9b1143c.be08d4","type":"ui_dropdown","z":"5fe64185.d640d","name":"","label":"Going to Sleep
at:","tooltip":"","place":"Select
time","group":"8361306c.306d98","order":1,"width":0,"height":0,"passthru":false,"multiple":false
,"options":[{"label":"1 a.m.","value":1,"type":"num"},{"label":"2
a.m.","value":2,"type":"num"},{"label":"3 a.m.","value":3,"type":"num"},{"label":"4
a.m.","value":4,"type":"num"},{"label":"5 a.m.","value":5,"type":"num"},{"label":"6
a.m.","value":6,"type":"num"},{"label":"7 a.m.","value":7,"type":"num"},{"label":"8
a.m.","value":8,"type":"num"},{"label":"9 a.m.","value":9,"type":"num"},{"label":"10
a.m.","value":10,"type":"num"},{"label":"11 a.m.","value":11,"type":"num"},{"label":"12
noon","value":12,"type":"num"},{"label":"1 p.m.","value":13,"type":"num"},{"label":"2
p.m.","value":14,"type":"num"},{"label":"3p.m.","value":15,"type":"num"},{"label":"4
p.m.","value":16,"type":"num"},{"label":"5 p.m.","value":17,"type":"num"},{"label":"6
p.m.","value":18,"type":"num"},{"label":"7 p.m.","value":19,"type":"num"},{"label":"8
p.m.","value":20,"type":"num"},{"label":"9 p.m.","value":21,"type":"num"},{"label":"10
p.m.","value":22,"type":"num"},{"label":"11 p.m.","value":23,"type":"num"},{"label":"12
midnight","value":0,"type":"num"}],"payload":"","topic":"sleep_time","topicType":"msg","x":630,"
y":440,"wires":[["a853166d.07942"]]},{"id":"9e47881a.fc9df8","type":"ui_dropdown","z":"5fe64185.
d640d","name":"","label":"Waking up at:","tooltip":"","place":"Select
time","group":"8361306c.306d98","order":1,"width":0,"height":0,"passthru":true,"multiple":false,
"options":[{"label":"1 a.m.","value":1,"type":"num"},{"label":"2
a.m.","value":2,"type":"num"},{"label":"3 a.m.","value":3,"type":"num"},{"label":"4
a.m.","value":4,"type":"num"},{"label":"5 a.m.","value":5,"type":"num"},{"label":"6
a.m.","value":6,"type":"num"},{"label":"7 a.m.","value":7,"type":"num"},{"label":"8
a.m.","value":8,"type":"num"},{"label":"9 a.m.","value":9,"type":"num"},{"label":"10
a.m.","value":10,"type":"num"},{"label":"11 a.m.","value":11,"type":"num"},{"label":"12
noon","value":12,"type":"num"},{"label":"1 p.m.","value":13,"type":"num"},{"label":"2
p.m.","value":14,"type":"num"},{"label":"3p.m.","value":15,"type":"num"},{"label":"4
p.m.","value":16,"type":"num"},{"label":"5 p.m.","value":17,"type":"num"},{"label":"6
p.m.","value":18,"type":"num"},{"label":"7 p.m.","value":19,"type":"num"},{"label":"8
p.m.","value":20,"type":"num"},{"label":"9 p.m.","value":21,"type":"num"},{"label":"10
p.m.","value":22,"type":"num"},{"label":"11 p.m.","value":23,"type":"num"},{"label":"12
midnight","value":0,"type":"num"}],"payload":"","topic":"waking_up","topicType":"flow","x":640,"
y":500,"wires":[["a783c35c.fe7ae"]]},{"id":"a853166d.07942","type":"change","z":"5fe64185.d640d"
,"name":"set
flow","rules":[{"t":"set","p":"sleep_time","pt":"flow","to":"payload","tot":"msg"}],"action":"",
"property":"","from":"","to":"","reg":false,"x":800,"y":440,"wires":[[]]},{"id":"a783c35c.fe7ae"
,"type":"change","z":"5fe64185.d640d","name":"set
flow","rules":[{"t":"set","p":"waking_up","pt":"flow","to":"payload","tot":"msg"}],"action":"","
property":"","from":"","to":"","reg":false,"x":800,"y":500,"wires":[[]]},{"id":"c0072397.aa5ca",
"type":"mqtt-broker","name":"","broker":"dingnet-
v3.icts.kuleuven.be","port":"1883","clientid":"","usetls":false,"compatmode":false,"keepalive":"
60","cleansession":true,"birthTopic":"","birthQos":"0","birthPayload":"","closeTopic":"","closeQ
os":"0","closePayload":"","willTopic":"","willQos":"0","willPayload":""},{"id":"b855ee7d.1894d8"
,"type":"ui_group","name":"Prototype","tab":"b7a0e06e.3ad428","order":2,"disp":true,"width":"6",
"collapse":true},{"id":"5bdb97c6.ccc958","type":"ui_group","name":"Prototype","tab":"ca7d446e.be
1ff8","order":1,"disp":true,"width":"6","collapse":true},{"id":"8361306c.306d98","type":"ui_grou
p","name":"Prototype","tab":"5ba3322.175bd4c","order":1,"disp":true,"width":"6","collapse":false
},{"id":"b7a0e06e.3ad428","type":"ui_tab","name":"Battery","icon":"mi-
battery_std","order":1,"disabled":false,"hidden":false},{"id":"ca7d446e.be1ff8","type":"ui_tab",
"name":"Alarms","icon":"mi-
report_problem","order":2,"disabled":false,"hidden":false},{"id":"5ba3322.175bd4c","type":"ui_ta
b","name":"Sleep time","icon":"mi-nightlight_round","order":3,"disabled":false,"hidden":false}]
```

File logger/browser flow


```
[{"id":"abe6ee11.6d2e","type":"tab","label":"File Logger/
Browser","disabled":false,"info":""},{"id":"f390bbd3.d7e4d8","type":"link
in","z":"abe6ee11.6d2e","name":"Auto
Logger","links":["2d351ade.b48eee","2e8d0393.34576c","702e6189.34ea5","e99cfe12.5bf3c8","b088020
7.5bcc6"],"x":135,"y":140,"wires":[["d9841209.1e6fc8"]]},{"id":"98e69319.aa7b3","type":"comment"
,"z":"abe6ee11.6d2e","name":"Auto
Logger","info":"","x":110,"y":20,"wires":[]},{"id":"85958157.8a8bb8","type":"function","z":"abe6
ee11.6d2e","name":"Set data","func":"var now = new Date();\n\nvar msg1 = {};\nmsg1.payload = {\n
\"time\" : msg.visitTime.shift(),\n    \"duration\" : msg.visitDuration.shift(),\n    \"device\"
: msg.device\n}\n\nmsg.payload = {\n    \"arrayLength\" : msg.visitTime.length\n}\n\n\nreturn
[msg1,
msg];","outputs":2,"noerr":0,"initialize":"","finalize":"","x":300,"y":140,"wires":[["624d314d.5
5f17","b33c9f2f.92447"],["d9841209.1e6fc8"]],"outputLabels":["msg1","msg2"]},{"id":"624d314d.55f
17","type":"function","z":"abe6ee11.6d2e","name":"Filename generator","func":"// Get the current
time and convert it to text\nvar now = new Date();\nvar yyyy = now.getFullYear();\nvar mm =
now.getMonth() < 9 ? \"0\" + (now.getMonth() + 1) : (now.getMonth() + 1); // getMonth() is zero-
based\nvar dd  = now.getDate() < 10 ? \"0\" + now.getDate() : now.getDate();\nvar hh =
now.getHours() < 10 ? \"0\" + now.getHours() : now.getHours();\nvar mmm  = now.getMinutes() < 10
? \"0\" + now.getMinutes() : now.getMinutes();\nvar ss  = now.getSeconds() < 10 ? \"0\" +
now.getSeconds() : now.getSeconds();\n\n// Generate out file name pattern\nmsg.fname =
\"RoboticFlowerField_\"+ yyyy + '_' + mm + '_' + dd + \".csv\";\n// Full filename with path for
the file node later\nmsg.filename = \"/Users/kamiel/Desktop/DataLog/\"+ msg.fname;\n\n// We save
the current payload into a different place on the msg object\nmsg.filecontent =
msg.payload;\n\n// We are passing the file name search pattern to fs node to tell us if the file
exists or not\nmsg.payload =
{\"pattern\":msg.fname};\n\nnode.status({fill:\"blue\",shape:\"ring\",text:msg.fname});\nreturn
msg;","outputs":1,"noerr":0,"initialize":"","finalize":"","x":210,"y":240,"wires":[["2699598a.56
dbf6"]]},{"id":"2699598a.56dbf6","type":"fs-file-
lister","z":"abe6ee11.6d2e","name":"","start":"/Users/kamiel/Desktop/DataLog","pattern":"","fold
ers":"*","hidden":true,"lstype":"files","path":true,"single":true,"depth":0,"stat":true,"showWar
nings":false,"x":400,"y":240,"wires":[["6c420ea2.89a9e8","8a98ceff.114098"]]},{"id":"6c420ea2.89
a9e8","type":"debug","z":"abe6ee11.6d2e","name":"","active":false,"tosidebar":true,"console":fal
se,"tostatus":false,"complete":"payload","targetType":"msg","statusVal":"","statusType":"auto","
x":550,"y":280,"wires":[]},{"id":"8a98ceff.114098","type":"switch","z":"abe6ee11.6d2e","name":""
,"property":"$count(msg.payload)","propertyType":"jsonata","rules":[{"t":"eq","v":"0","vt":"num"
},{"t":"else"}],"checkall":"true","repair":false,"outputs":2,"x":530,"y":180,"wires":[["7b9b25b3
.fd1784"],["571eff76.6395c8"]]},{"id":"cdec6034.b0f6f8","type":"csv","z":"abe6ee11.6d2e","name":
"","sep":",","hdrin":"","hdrout":"once","multi":"mult","ret":"\\n","temp":"time,duration,device"
,"skip":"0","strings":true,"include_empty_strings":false,"include_null_values":false,"x":910,"y"
:160,"wires":[["c6b5d123.1747"]]},{"id":"c6b5d123.1747","type":"file","z":"abe6ee11.6d2e","name"
:"","filename":"","appendNewline":false,"createDir":true,"overwriteFile":"false","encoding":"non
e","x":1050,"y":180,"wires":[[]]},{"id":"571eff76.6395c8","type":"change","z":"abe6ee11.6d2e","n
ame":"file
exists","rules":[{"t":"set","p":"payload","pt":"msg","to":"filecontent","tot":"msg"}],"action":"
","property":"","from":"","to":"","reg":false,"x":680,"y":200,"wires":[["6736e3cb.c179b4"]]},{"i
d":"7b9b25b3.fd1784","type":"change","z":"abe6ee11.6d2e","name":"file does not exist
yet","rules":[{"t":"set","p":"payload","pt":"msg","to":"filecontent","tot":"msg"}],"action":"","
property":"","from":"","to":"","reg":false,"x":720,"y":160,"wires":[["cdec6034.b0f6f8"]]},{"id":
"4fa73dd9.83cca4","type":"comment","z":"abe6ee11.6d2e","name":"File Browser","info":"1) Update
the default folder in the Init node\n2) Update the default folder in the Reset node as well\n3)
The Graph button can read any CSV file which have any number of values, but the first column
always contains a timestamp column with javascript timestamp of the data
point.","x":110,"y":320,"wires":[]},{"id":"993d7272.843ae","type":"fs-file-
lister","z":"abe6ee11.6d2e","name":"Files","start":"/home/pi","pattern":"*.*","folders":"*","hid
den":false,"lstype":"files","path":true,"single":true,"depth":0,"stat":true,"showWarnings":false
,"x":510,"y":420,"wires":[["dd88bd23.37cde","3e094f82.15d6e"]]},{"id":"d185a45.2327a58","type":"
inject","z":"abe6ee11.6d2e","name":"Init","props":[{"p":"payload"},{"p":"topic","vt":"str"}],"re
peat":"","crontab":"","once":true,"onceDelay":0.1,"topic":"change","payload":"/Users/kamiel/Desk
top/DataLog","payloadType":"str","x":110,"y":380,"wires":[["3d4e5e51.bdf952"]]},{"id":"a715f7a4.
a398a8","type":"ui_button","z":"abe6ee11.6d2e","name":"","group":"160e81fb.f1c86e","order":1,"wi
dth":"2","height":"1","passthru":false,"label":"Refresh","tooltip":"","color":"","bgcolor":"","i
con":"refresh","payload":"","payloadType":"str","topic":"refresh","topicType":"str","x":100,"y":
420,"wires":[["3d4e5e51.bdf952"]]},{"id":"eb17c7ee.e61988","type":"ui_dropdown","z":"abe6ee11.6d
2e","name":"File Selector","label":"","tooltip":"","place":"Select a
file","group":"160e81fb.f1c86e","order":9,"width":"5","height":"1","passthru":false,"options":[{
"label":"","value":"","type":"str"}],"payload":"","topic":"","x":910,"y":420,"wires":[["be4830df
```

```
.5775d"]]},{"id":"dd88bd23.37cde","type":"function","z":"abe6ee11.6d2e","name":"Format
data","func":"// format the data for the dropdown\nmsg.options = [];\nfor (var i=0;
i<msg.payload.length; i++) {\n    // This is a file\n    obj = {};\n    obj
[msg.payload[i].name.replace(/^.*(\\\\|\\/|\\:)/, '')]=msg.payload[i].name;\n
msg.options.push(obj);\n}\nmsg.payload={};\nreturn
msg;","outputs":1,"noerr":0,"x":710,"y":420,"wires":[["eb17c7ee.e61988"]]},{"id":"638728aa.cd0b0
8","type":"ui_template","z":"abe6ee11.6d2e","group":"160e81fb.f1c86e","name":"","order":6,"width
":"18","height":"6","format":"<div ng-bind-html=\"msg.payload\" height=\"400\" style=\"height:
400px;\"><br/>\n</div>\n\n","storeOutMessages":true,"fwdInMessages":true,"templateScope":"local"
,"x":1100,"y":480,"wires":[[]]},{"id":"9d27e846.b31db8","type":"template","z":"abe6ee11.6d2e","n
ame":"","field":"payload","fieldType":"msg","format":"handlebars","syntax":"mustache","template"
:"<table width=\"100%\">\n    <tr><th>File
Name</th><th>Size</th><th>Created</th><th>Changed</th></tr>\n    {{#payload}}\n        <tr>\n
<td><a href=\"/download?filename={{name}}\" target=\"blank\">{{fname}}</a></td>\n
<td>{{stat.size}}</td>\n            <td>{{stat.created}}</td>\n
<td>{{stat.changed}}</td>\n        </tr>\n
{{/payload}}\n</table>\n","output":"str","x":940,"y":480,"wires":[["638728aa.cd0b08"]]},{"id":"3
e094f82.15d6e","type":"function","z":"abe6ee11.6d2e","name":"Convert timestamps","func":"for
(var i=0; i<msg.payload.length; i++) {\n    msg.payload[i].stat.created =
msg.payload[i].stat.created.toISOString().slice(0, 19).replace('T', ' ');\n
msg.payload[i].stat.changed = msg.payload[i].stat.changed.toISOString().slice(0,
19).replace('T', ' ');\n    msg.payload[i].stat.accessed =
msg.payload[i].stat.accessed.toISOString().slice(0, 19).replace('T', ' ');\n
msg.payload[i].stat.statusChanged = msg.payload[i].stat.statusChanged.toISOString().slice(0,
19).replace('T', ' ');\n    msg.payload[i].fname =
msg.payload[i].name.replace(/^.*(\\\\|\\/|\\:)/, '');\n}\nreturn
msg;","outputs":1,"noerr":0,"x":740,"y":480,"wires":[["9d27e846.b31db8"]]},{"id":"7b7e0d58.05e24
4","type":"ui_button","z":"abe6ee11.6d2e","name":"","group":"160e81fb.f1c86e","order":10,"width"
:"2","height":"1","passthru":false,"label":"Delete","tooltip":"","color":"","bgcolor":"","icon":
"delete","payload":"","payloadType":"str","topic":"","x":130,"y":580,"wires":[["8da8147a.999af8"
]]},{"id":"7d490dd1.8458b4","type":"function","z":"abe6ee11.6d2e","name":"Get
filename","func":"// Get the filename from the flow context\nlet filename =
flow.get(\"fileselected\");\n\n// check, if the filename is undefined that means it does not
exist yet, nothing is selected yet\n// return: do not output anything\nif (filename===undefined)
{\n    return;\n}\n\n// return the filename to the file-in node to delete\nmsg.filename =
filename;\n\nif (msg.filename.replace(/^.*(\\\\|\\/|\\:)/, '')[0]!==\".\") {\n    // Only do
this if this is a file, we don't delete folders\n    // and delete the context/selection as we
are deleting the file as well\n    flow.set(\"fileselected\");\n    return
msg;\n}","outputs":1,"noerr":0,"x":770,"y":640,"wires":[["372ee262.fc94de"]]},{"id":"372ee262.fc
94de","type":"file","z":"abe6ee11.6d2e","name":"Delete
file","filename":"","appendNewline":true,"createDir":false,"overwriteFile":"delete","encoding":"
none","x":540,"y":640,"wires":[["993d7272.843ae","5219875b.c070d8"]]},{"id":"aa8c482.93734b8","t
ype":"http
in","z":"abe6ee11.6d2e","name":"","url":"/download","method":"get","upload":false,"swaggerDoc":"
","x":140,"y":780,"wires":[["d969ba04.e24028"]]},{"id":"d19cc7d8.646328","type":"http
response","z":"abe6ee11.6d2e","name":"","statusCode":"","headers":{},"x":930,"y":780,"wires":[]}
,{"id":"d969ba04.e24028","type":"function","z":"abe6ee11.6d2e","name":"Get the file
name","func":"msg.filename = msg.req.query.filename;\nmsg.contentdisposition = \"attachment;
filename=\\\"\" + msg.req.query.filename.replace(/^.*(\\\\|\\/|\\:)/, '') + \"\\\"\";\nreturn
msg;\n","outputs":1,"noerr":0,"x":390,"y":780,"wires":[["e92381c3.c4cd2"]],"outputLabels":["Fold
er selected","File selected"]},{"id":"e92381c3.c4cd2","type":"file
in","z":"abe6ee11.6d2e","name":"","filename":"","format":"","chunk":false,"sendError":false,"enc
oding":"none","x":580,"y":780,"wires":[["99ff4953.d0d5c8"]]},{"id":"99ff4953.d0d5c8","type":"cha
nge","z":"abe6ee11.6d2e","name":"Set
Headers","rules":[{"t":"set","p":"headers","pt":"msg","to":"{}","tot":"json"},{"t":"set","p":"he
aders.content-type","pt":"msg","to":"text/csv","tot":"str"},{"t":"set","p":"headers.Content-
Disposition","pt":"msg","to":"contentdisposition","tot":"msg"}],"action":"","property":"","from"
:"","to":"","reg":false,"x":750,"y":780,"wires":[["d19cc7d8.646328"]]},{"id":"554f0095.ac1b2","t
ype":"ui_toast","z":"abe6ee11.6d2e","position":"dialog","displayTime":"3","highlight":"","sendal
l":false,"outputs":1,"ok":"Yes","cancel":"No","raw":false,"topic":"","name":"Confirmation","x":5
10,"y":580,"wires":[["24df84fe.ebf45c"]]},{"id":"8da8147a.999af8","type":"change","z":"abe6ee11.
6d2e","name":"Set message","rules":[{"t":"set","p":"topic","pt":"msg","to":"Delete
confirmation","tot":"str"},{"t":"set","p":"payload","pt":"msg","to":"Are you sure you want to
delete this
file?","tot":"str"}],"action":"","property":"","from":"","to":"","reg":false,"x":310,"y":580,"wi
res":[["554f0095.ac1b2"]]},{"id":"24df84fe.ebf45c","type":"switch","z":"abe6ee11.6d2e","name":""
,"property":"payload","propertyType":"msg","rules":[{"t":"eq","v":"Yes","vt":"str"}],"checkall":
"true","repair":false,"outputs":1,"x":710,"y":580,"wires":[["7d490dd1.8458b4"]]},{"id":"3d4e5e51
.bdf952","type":"function","z":"abe6ee11.6d2e","name":"Folder handling","func":"let folder =
context.get(\"folder\");\nif (folder===undefined) {\n    folder=\"/\";\n
```

```
context.set(\"folder\", folder);\n}\n\nlet hidden = context.get(\"hidden\");\nif
(hidden===undefined) {\n    hidden=false;\n    context.set(\"hidden\", hidden);\n}\n\nif
(msg.topic===\"up\") {\n    var the_arr = folder.split('/');\n    the_arr.pop();\n
folder=the_arr.join('/'); \n    context.set(\"folder\", folder);\n}\nif (msg.topic===\"change\")
{\n    folder=msg.payload;    \n    context.set(\"folder\", folder);\n}\nif
(msg.topic===\"hidden\") {\n    hidden=msg.payload;    \n    context.set(\"hidden\",
hidden);\n}\n\nmsg.payload = {\"start\":folder, \"hidden\": hidden};\nreturn
msg;","outputs":1,"noerr":0,"initialize":"","finalize":"","x":300,"y":480,"wires":[["993d7272.84
3ae","399ac341.7d43bc","5219875b.c070d8"]]},{"id":"399ac341.7d43bc","type":"ui_text","z":"abe6ee
11.6d2e","group":"160e81fb.f1c86e","order":4,"width":"9","height":"1","name":"","label":"Folder:
","format":"{{msg.payload.start}}","layout":"row-
left","x":500,"y":360,"wires":[]},{"id":"b40ea1d8.c700a","type":"ui_button","z":"abe6ee11.6d2e",
"name":"","group":"160e81fb.f1c86e","order":3,"width":"2","height":"1","passthru":false,"label":
"Up","tooltip":"","color":"","bgcolor":"","icon":"arrow_upwards","payload":"","payloadType":"str
","topic":"up","x":110,"y":500,"wires":[["3d4e5e51.bdf952"]]},{"id":"80940e39.5035b","type":"ui_
button","z":"abe6ee11.6d2e","name":"","group":"160e81fb.f1c86e","order":8,"width":"2","height":"
1","passthru":false,"label":"Open","tooltip":"","color":"","bgcolor":"","icon":"folder_open","pa
yload":"","payloadType":"str","topic":"","x":130,"y":700,"wires":[["8c1dfaac.979588"]]},{"id":"8
c1dfaac.979588","type":"function","z":"abe6ee11.6d2e","name":"Change folder","func":"// Get the
filename from the flow context\nlet folderselected = flow.get(\"folderselected\");\n\n// check,
if the filename is undefined that means it does not exist yet, nothing is selected yet\n//
return: do not output anything\nif (folderselected===undefined) {\n    return;\n}\n\nmsg.topic =
\"change\";\nmsg.payload = folderselected;\n\nreturn
msg;","outputs":1,"noerr":0,"x":340,"y":700,"wires":[["3d4e5e51.bdf952"]]},{"id":"58414ec4.c716e
","type":"ui_button","z":"abe6ee11.6d2e","name":"","group":"160e81fb.f1c86e","order":2,"width":"
2","height":"1","passthru":false,"label":"Reset","tooltip":"","color":"","bgcolor":"","icon":"au
torenew","payload":"/Users/kamiel/Desktop/DataLog","payloadType":"str","topic":"change","topicTy
pe":"str","x":110,"y":460,"wires":[["3d4e5e51.bdf952"]]},{"id":"5219875b.c070d8","type":"fs-
file-
lister","z":"abe6ee11.6d2e","name":"Folders","start":"/home/pi","pattern":"*.*","folders":"*","h
idden":false,"lstype":"directories","path":true,"single":true,"depth":0,"stat":true,"showWarning
s":false,"x":520,"y":520,"wires":[["ebc234be.d53fe8"]]},{"id":"37485c24.212054","type":"ui_dropd
own","z":"abe6ee11.6d2e","name":"Folder Selector","label":"","tooltip":"","place":"Select a
folder","group":"160e81fb.f1c86e","order":7,"width":"5","height":"1","passthru":false,"options":
[{"label":"","value":"","type":"str"}],"payload":"","topic":"","x":920,"y":520,"wires":[["2875c7
9f.531558"]]},{"id":"ebc234be.d53fe8","type":"function","z":"abe6ee11.6d2e","name":"Format
data","func":"// format the data for the dropdown\nmsg.options = [];\nfor (var i=0;
i<msg.payload.length; i++) {\n    // This is a foler\n    obj = {};\n    obj
[\"[\"+msg.payload[i].name.replace(/^.*(\\\\|\\/|\\:)/, '')+\"]\"]=msg.payload[i].name;\n
msg.options.push(obj);\n}\nmsg.payload={};\nreturn
msg;","outputs":1,"noerr":0,"x":710,"y":520,"wires":[["37485c24.212054"]]},{"id":"be4830df.5775d
","type":"change","z":"abe6ee11.6d2e","name":"Save
selection","rules":[{"t":"set","p":"fileselected","pt":"flow","to":"payload","tot":"msg"}],"acti
on":"","property":"","from":"","to":"","reg":false,"x":1120,"y":420,"wires":[[]]},{"id":"2875c79
f.531558","type":"change","z":"abe6ee11.6d2e","name":"Save
selection","rules":[{"t":"set","p":"folderselected","pt":"flow","to":"payload","tot":"msg"}],"ac
tion":"","property":"","from":"","to":"","reg":false,"x":1120,"y":520,"wires":[[]]},{"id":"928e6
0a7.994be","type":"ui_switch","z":"abe6ee11.6d2e","name":"","label":"Hidden","tooltip":"Show
hidden files or
not","group":"160e81fb.f1c86e","order":5,"width":"2","height":"1","passthru":false,"decouple":"f
alse","topic":"hidden","topicType":"str","style":"","onvalue":"true","onvalueType":"bool","onico
n":"check_box","oncolor":"#000000","offvalue":"false","offvalueType":"bool","officon":"check_box
_outline_blank","offcolor":"#000000","animate":true,"x":100,"y":540,"wires":[["3d4e5e51.bdf952"]
]},{"id":"6736e3cb.c179b4","type":"csv","z":"abe6ee11.6d2e","name":"","sep":",","hdrin":"","hdro
ut":"none","multi":"mult","ret":"\\n","temp":"time,duration,device","skip":"0","strings":true,"i
nclude_empty_strings":false,"include_null_values":false,"x":910,"y":200,"wires":[["c6b5d123.1747
"]]},{"id":"b33c9f2f.92447","type":"debug","z":"abe6ee11.6d2e","name":"","active":false,"tosideb
ar":true,"console":false,"tostatus":false,"complete":"payload","targetType":"msg","statusVal":""
,"statusType":"auto","x":590,"y":80,"wires":[]},{"id":"d9841209.1e6fc8","type":"switch","z":"abe
6ee11.6d2e","name":"check if array >
0","property":"payload.arrayLength","propertyType":"msg","rules":[{"t":"neq","v":"0","vt":"str"}
],"checkall":"true","repair":false,"outputs":1,"x":330,"y":60,"wires":[["85958157.8a8bb8"]]},{"i
d":"160e81fb.f1c86e","type":"ui_group","name":"File
Browser","tab":"b63d1f91.68095","order":1,"disp":true,"width":"18","collapse":false},{"id":"b63d
1f91.68095","type":"ui_tab","name":"Files","icon":"mi-
description","order":4,"disabled":false,"hidden":false}]
```

Addendum 6: Proof of concept tables

Part 1: Garden

	Manual measurement					Robotic Flower			Analysis Robotic Flower	
	Species	**Class**	**Remarks**	**Visit Time (hh:mm)**	**Visit Duration (s)**	**Visit Time (hh:mm:ss)**	**Visit Duration (s)**	**Sensitivity**	**Visit registration**	**Percentage time**
1	Syrphidae	Fly		13:33	8	X	X	10	No	X
2	*Bombus pascuorum*	Bee	male	13:45	5	X	X	10	No	X
3	Bombilidae	Fly		14:08	240	14:08:49	78	10	Yes	33%
4	Lepidoptera	Butterfly		14:13	240	X	X	10	No	X
5	*Bombus pascuorum*	Bee		14:19	3	X	X	10	No	X
6	Syrphidae	Fly	didn't drink	14:22	?	X	X	10	No	X
7	*Psythirus*	Bee		14:35	225	15:35:23	26	10	Yes	12%
8	*Anthidium manicatum*	Bee	head in, didn't drink?	14:59	?	14:59:17	2	10	Yes	?
9	Bombyliidae	Fly	didn't drink	15:11	?	X	X	10	No	X
10	*Apis mellifera*	Bee	didn't drink	15:23	?	X	X	10	No	X
11	*Apis mellifera*	Bee	2 visits by same bee	15:39	35	15:39:23	26	10	Yes	74%
12	*Apis mellifera*	Bee	3 visits by same bee	15:39	10	15:40:01	14	10	Yes	140%
13	*Apis mellifera*	Bee		15:49	8	15:49:20	3	10	Yes	38%
14	*Bombus terrestris*	Bee		15:51	148	15:51:26	52	10	Yes	35%
15	*Bombus terrestris*	Bee	male	15:56	9	15:57:20	9	10	Yes	100%

Part 2: Flight cage

	Manual Measurement			Robotic Flower			Analysis Robotic Flower	
	Species	Visit Time (hh:mm)	Visit Duration (s)	Visit Time (hh:mm:ss)	Visit Duration (s)	Sensitivity	Visit registation	Duration overlap (%)
1	*B. terrestris*	14:51	30	X	X	10	No	X
2	*B. terrestris*	14:52	20	14:52:07	2	10	Yes	15%
				14:52:11	1	10		
3	*B. terrestris*	14:53	87	X	X	10	No	X
4	*B. terrestris*	15:01	9	X	X	10	No	X
5	*B. terrestris*	15:18	128	15:18:16	51	3	Yes	98%
				15:19:09	52	3		
				15:20:02	23	3		
6	*B. terrestris*	15:21	27	15:21:06	20	3	Yes	96%
				15:21:38	6	3		
7	*B. terrestris*	15:22	67	15:22:04	3	3	Yes	93%
				15:22:09	1	3		
				15:22:12	58	3		
8	*B. terrestris*	15:23	56	15:23:17	10	3	Yes	66%
				15:23:30	12	3		
				15:23:51	15	3		
9	*B. terrestris*	15:24	77	15:24:09	9	3	Yes	74%
				15:24:20	5	3		
				15:24:30	4	3		
				15:24:35	12	3		
				15:24:59	27	3		
10	*B. terrestris*	15:28	39	15:28:03	27	3	Yes	82%
				15:28:39	5	3		

11	*B. terrestris*	15:33	26	15:33:54	22	3	Yes	104%
				15:34:18	5	3		
12	*B. terrestris*	15:35	23	15:35:09	25	3	Yes	109%
13	*B. terrestris*	15:35	9	15:35:48	3	3	Yes	33%
14	*B. terrestris*	15:35	29	15:35:56	21	3	Yes	72%
15	*B. terrestris*	15:37	13	15:37:03	9	3	Yes	69%
16	*B. terrestris*	15:37	5	15:37:22	14	3	Yes	280%
17	*B. terrestris*	15:37	14	15:37:44	12	3	Yes	100%
				15:37:58	2	3		
18	*B. terrestris*	15:42	46	15:42:11	39	3	Yes	98%
				15:42:52	6	3		
19	*B. terrestris*	15:43	7		30	3	Yes	125%
20	*B. terrestris*	15:43	17	15:43:05				
21	*B. terrestris*	15:43	31	15:43:38	8	3	Yes	90%
				15:43:48	2	3		
				15:43:56	18	3		
22	*B. terrestris*	15:44	47	15:44:49	37	3	Yes	79%
23	*B. terrestris*	15:45	2	15:45:30	3	3	Yes	150%
24	*B. terrestris*	15:46	5	15:46:00	1	3	Yes	140%
				15:46:15	6	3		
25	*B. terrestris*	15:47	10		38	3	Yes	127%
26	*B. terrestris*	15:47	20	15:47:04				
27	*E. corollae*	15:56	75	15:56:15	3	3	Yes	85%
				15:56:53	13	3		
				15:57:41	30	3		
				15:58:17	18	3		
28	*E. corollae*	15:59	175	15:58:43	190	3	Yes	109%

PLANT CONSERVATION AND POPULATION BIOLOGY
Kasteelpark Arenberg 31 box 2435
3000 LEUVEN, BELGIË
tel. + 32 16 32 15 20
fax + 32 16 19 68
www.kuleuven.be