Insect counts of DIOPSIS cameras depend on screen properties

Robin Strik
Department of Ecology
Faculty of Science – Radboud University
22-01-2024

Abstract

Insect numbers and diversity have been declining rapidly on a global scale, which is problematic due to the various important roles of insects in ecosystems. Insect monitoring projects can help quantify this decline, and to improve their accuracy, new sampling methods are constantly being developed. DIOPSIS uses an automated, non-destructive sampling method; namely the automatic photographing of insects combined with the identification of these insects by a machine learning algorithm. DIOPSIS requires insects to land on a screen in front of a camera, but whether an insect actually lands depends on the screen's properties. Therefore, I wanted to know whether the observed species count on the DIOPSIS setup differs between screens with different properties, namely screen version (v1 and v2), color (yellow, white, and blue), orientation (vertical and horizontal), and texture (smooth and rough). I expected that there would be count patterns visible in the preference of insects for different treatments, and that these patterns would differ between groups of insect species. I performed a field experiment following a fractional factorial design in which I tested multiple screen treatments in two distinct localities: an agricultural landscape located near the Dutch city of Nijmegen, and the tropical greenhouse of the Hortus Botanicus in Leiden. The photos that were taken were analyzed using a machine learning identification algorithm, and the resulting data was fitted to several zero-inflated generalized linear mixed models in order to estimate hourly insect counts. These counts differed significantly between treatments, and also showed a different response pattern to treatments for different insect groups. Furthermore, the patterns in estimated counts differed between localities, highlighting the effect of environmental factors on the sampling process. In order to make optimal use of DIOPSIS – and similar automated sampling systems – it is important to consider the properties of the screen you deploy in the field.

Keywords: biomonitoring, automation, computer vision, identification, machine learning

Introduction

In 2017, Hallmann et al. found that the total biomass of flying insects had decreased by more than 75% in protect areas in western Germany over the past 27 years. This conclusion reflects those of other studies, such as that by Kühn et al. (2013), who report that butterfly populations have declined by almost 50% from 1990 until 2011 across Europe. Another example comes from a meta-analysis by Dirzo et al. (2014), which shows that 67% of the monitored invertebrate populations had a mean decline in abundance of 45%. Since 2017, more research has been published, strengthening the view

that insect populations are not doing well. For example, in the Netherlands, a study has shown that from 1890 to 2017, the occurrence of butterflies in the Netherlands has decreased by 84%, and likely even more (van Strien et al. 2019). A study from the United States shows similar results for beetles: comparing trapped beetles in the period of 1973-1977 to that of 2015-2017, beetle numbers declined by 83%, and the total number of taxa decreased by 39% (Harris et al. 2019).

Insects perform important roles in ecosystems. Such roles include pollination, decomposition, consumption of organisms, and also acting as a food source for others (Losey and Vaughan 2006; Allsopp et al. 2008; Seibold et al. 2021). Over 80% of all flowering plants are estimated to depend on insects for their survival. In fact, all these ecosystem services combined have been estimated in 2006 to have an annual value of US$ 57 billion in the United States alone (Losey and Vaughan 2006). Furthermore, insects species are estimated to encompass over 75% of all plant and animal life on earth (Chapman 2009). As a result, 90% of the insect species have not been named yet, and we also do not know their exact roles in their respective ecosystems (van der Sluijs 2020). Given the problematic nature of the insect declines that have been described, it is not surprising that there has been a large increase in publications regarding the decline in arthropods and insects over the past years. Using a simple search query of “insect decline” OR “arthropod decline” on Web of Science shows that the number of yearly published papers went from 305 in 2012 to 761 in 2022 for these search terms alone.

While the increased attention to this problem is positive, much research and many monitoring schemes are still only focused on certain insect groups, such as butterflies and beetles (Hoye et al. 2021). Other groups are not that easily monitored due to high costs or other difficulties such as the act of collecting the insects being too taxing, or a lack of knowledge required to correctly identify species of certain taxa. This all leads to biases in knowledge about insects (Pawar 2003). Furthermore, such traditional methods often require destructive sampling. In order to make insect data collection more accessible and more evenly distributed across taxa, new detection and identification methods can be employed. The most prominent of these methods are acoustic monitoring, radar, molecular methods, and computer vision (van Klink et al. 2022). Computer vision in particular has the useful property of being able to estimate insect abundances. By using video material of trapped insects and a reference library, deep learning methods can be used to determine individuals up to the species level (van Klink et al. 2022). In order to obtain video material, insects still require to be trapped, be it destructive or non-destructive. Trapping methods for computer vision purposes that have already been used include sticky traps, pheromone traps, and light traps (Yalcin 2015; Hogeweg et al. 2019; Gerovichev et al. 2021).

DIOPSIS is a smart insect camera system that combines computer vision and deep learning with color/light traps, without the need of destructive sampling. Its most recent version is v2 (Figure 1). DIOPSIS is an acronym of Digital Identification of Photographically Sampled Insect Species. The general principle of the DIOPSIS setup is that it takes a picture of its screen after set time intervals. Only when it detects movement does it take a picture independent from the interval. The images that are taken can be analyzed using a deep learning pipeline to generate species (or, when not possible, other taxonomic order levels) identifications and an estimation of the individuals’ masses (van Lieshout 2020). The pipeline is currently being developed as part of the ARISE Biodiversity project. The entire capture to identification process happens without directly killing any insects, as would have been the case with e.g. sticky traps or pan traps.


Figure 1: DIOPSIS v2

In the setup of DIOPSIS, the screen plays an important role by acting as the attractant. In the current DIOPSIS version (DIOPSIS v2), the screen is fluorescent yellow, and it has a smooth texture. It can also be oriented either horizontally or vertically. However, different insect species are known to have different preferences for these three factors. As a result, species counts may be biased towards those species that are especially attracted to the properties of the current screen.

The screen's color is one of the first factors detected by an approaching insect. It is therefore of importance that it attracts a sufficient and diverse number of insects in order generate representative data. The screen of the current DIOPSIS version is yellow, but there are other colors that may be preferred by certain insect groups. Insects can be attracted to colors for various reasons. For example, many insect species do not only have innate color preferences to find food sources, but they are also capable of learning which colors (i.e. which flowers) give the highest reward (e.g. nectar) (van der Kooi et al. 2019). A color can also be attractive to a species because it can help camouflage an insect if it is the same color (Ide 2022). On the other hand, color may also repel certain species, as is the case for horseflies (Tabanidae), who are repelled by green colors in order to increase attraction towards host animals for blood meals (Meglic et al. 2019).

Screen orientation is important for getting an insect to land and stay on the screen. Insect species can require specific orientations of the flowers they visit. For example, a deviation from the normal vertical placement of the flowers of the Asiatic dayflower (*Commelina communis*) results in less successful landings of pollinators (Ushimaru and Hyodo 2005). Whether an insect is able to, and decides to, stay on the screen also depends on its ability to hold on to its surface. For surface attachment, insects can use two distinct types of body parts: claws and adhesive devices (Scholz et al. 2010). Claws can be used to latch onto larger structures, but adhesive devices work on a smaller scale. The adhesive devices can be further divided into hairy adhesives and smooth adhesives (pads), which both use a thin layer of liquid secretion to facilitate adhesion (Beutel and Gorb 2001; Bullock et al. 2008). Individual species only have one of these types of adhesive devices. In general, the ability to adhere to a surface with adhesive devices decreases when roughness increases, although hairy adhesives are better optimized to attached to rougher surfaces (Bullock et al. 2008; Zhou et al. 2014). An example of surface roughness influencing insect attraction is that of bark beetles (Scolytinae), who prefer rough bark over smooth bark, as their grip on smooth bark is low (Ferrenberg et al. 2014).

DIOPSIS is under constant development, and as a crucial part of the setup, the screen may potentially be further improved. Therefore, I want to know the effects that the different screen properties have on the observed species counts. In order to come to an informative conclusion, I aim to obtain an answer to the following research questions:
1. Do observed species counts differ between the version 1 and 2 screens of the DIOPSIS setup? Since both screen versions have different properties, I expect that this leads to the attractance of different species, resulting in different counts for different screen versions.
2. Do observed species counts differ between yellow, white, and blue screens in the DIOPSIS setup? I expect species to follow their inherent or learned preference for color, resulting in different counts across differently colored screens.
3. Do observed species counts differ between vertical and horizontal oriented screens in the DIOPSIS setup? Here, I expect species to follow their inherent preference for surface orientation, resulting in different counts across differently oriented screens.
4. Do observed species counts differ between smooth and rough textured screens in the DIOPSIS setup? Because species have different capabilities of attaching to differently textured screens, I expect this to result in different counts across differently textured screens.

Materials and methods

In order to answer my research questions, I performed a field experiment with DIOPSIS in which I tested the ability of 10 different screen-treatments to attract insects. The photographs were identified using a machine learning algorithm, after which I created several models to predict hourly insect counts.

To test for the choice for a particular screen (e.g. that of the current DIOPSIS v2 or that of the v1 camera system) on caught insect counts, a range of screen types were used as treatments in the experiment. First, the screen that is part of the current standard DIOPSIS (v2) setup was selected as a treatment. During previous field runs of the DIOPSIS cameras, it was observed that insects seemed to have more difficulties landing on the screen of the v2 camera than that of the v1 camera (E. Jongejans, personal communication). Hence, the v1 screen was also selected as a treatment. The v1 screen is made from yellow (RAL 1023) foamed PVC. The product page of this product and other products that I used for the construction of the screens can be found in appendix 1.

In order to test the effects of screen color, orientation, and texture on insect counts, I prepared multiple screens in addition to the v1 and v2 screens. From now on, I will refer to these additional screens as new screens. The new screens are made out of white (RAL 9003) foamed PVC with the same dimensions (height x width) as the v2 screen. The reason I chose foamed PVC is the fact that this material was also used for the v1 screens. I spray painted these new screens in three distinct colors: fluorescent yellow, blue, and white. This was done using Sparvar spray paint with color codes 3104, 3107, and 3108 respectively. My reason for choosing these three colors is that they are often used for pan traps (Leong and Thorp 1999; Toler et al. 2005; Campbell and Hanula 2007). Spray paint covers a surface with particles that have diameters in the micrometer range (Bremmer and van Engelen 2006). Because the screens were covered in particles that function on the same scale as the adhesive pads of insects, I described the spray-painted screens' texture as being rough.

In order to see the effects of screen texture, half of the new screens were covered with a plate of clear, UV permeable plexiglass. The thickness of these plates was 3mm. The purpose of the plexiglass, also known as PMMA (poly (methyl methacrylate)), was to introduce a smooth texture while keeping other factors, mainly color and reflectance, as constant as possible. Plexiglass was

chosen for this purpose due to its resistance to shattering, its high visible light and UV light permittance (≥90% of wavelengths 380 – 780 nm and ≥80% of wavelength 315 nm respectively at 3mm thickness), and its low light reflection (~3.9%) (PLEXIGLAS ; Szczurowski 2013; Ali et al. 2015). Instead of plexiglass I could also have chosen panes of soda-lime glass, the most commonly used glass type for windows (Ashby 2013). However, the properties of plexiglass are overall more favorable for this application than glass, which shatters more easily, has a lower visible and ultraviolet light permittance (~89% of wavelength 580 nm and ~36% of wavelength 315 nm respectively at 3mm thickness), and a higher light reflection (~4.3%) (Rubin 1985).

Before deciding on the aforementioned method for screen texture-differentiation, I experimented with another approach. Instead of applying the texture after the painting process (i.e. covering the screen with plexiglass), I spray painted small pieces of objects that already had different textures. Pictures of the results can be found in appendix 2. A problem that arose was that the underlying material visibly changed the screen's colors. Additionally, this method focusses on differences in the macrotexture – i.e. textures visible to the naked eye – instead of the microtexture. After all, the microtexture is determined by the spray paint cover, but the difference in macrotexture of a smooth plastic plate and a cardboard plate stays the same after painting.

The new screens were all attached to the screen of the DIOPSIS frame using clamps made of galvanized steel (Figure 2). To prevent a build-up of moisture under the plexiglass, I wrapped waterproof tape (3M Scotch Tape 389) along the sides of the foamed PVC and plexiglass plates (Figure 2). The tape was also applied to the new screens with a rough texture (i.e. without a plexiglass cover) in order to prevent any potential bias caused by the tape.



Figure 2: Two of the new screens placed in front of the DIOPSIS V2 screen | Two clamps are used to secure the new screen to the V2 screen in the top left and top right corners. Waterproof tape is wrapped around the sides of the new screen.

Lastly, the orientation of the screen was another factor in my experiment: I placed the DIOPSIS frame in either its vertical or horizontal position. This was only applied to the new rough and smooth yellow screens.
In total, I prepared eight different screen types: one v1 screen, one v2 screen, three rough new screens (colored yellow, blue, and white respectively), and three smooth new screens (colored yellow, blue, and white respectively). The total number of treatments was ten, as the new yellow screens were placed both horizontally and vertically for both textures. Table 1 shows an overview of the treatments that were used in the experiment.

Table 1: Treatments used in the experiment

Treatment	Version	Color*	Texture**	Orientation
v2 vertical	1	Fluorescent yellow	Smooth	Vertical
v1 vertical	2	Yellow	Rough	Vertical
Blue rough vertical	New	Fluorescent blue	Rough	Vertical
Blue smooth vertical	New	Fluorescent blue	Smooth	Vertical
White rough vertical	New	Fluorescent white	Rough	Vertical
White smooth vertical	New	Fluorescent white	Smooth	Vertical
Yellow rough vertical	New	Fluorescent yellow	Rough	Vertical
Yellow smooth vertical	New	Fluorescent yellow	Smooth	Vertical
Yellow rough horizontal	New	Fluorescent yellow	Rough	Horizontal
Yellow smooth horizontal	New	Fluorescent yellow	Smooth	Horizontal

* The fluorescent yellow color of the v2 and the yellow new screens are not exactly the same: the v2 screens use yellow stickers, while the yellow new screens use spray paint
** The v2, v1, and new screens are made of different materials, and therefore their microtextures may differ: the v2 screens use smooth stickers, the v1 screens are made of unpainted foamed PVC, and the new screens use spray painted foamed PVC for a rough texture and Plexiglas for a smooth texture

The experiment itself was performed in two distinct localities. Locality 1 was the tropical greenhouse of the Hortus Botanicus in Leiden. This locality was chosen because it allowed me to begin earlier in the season; the assumption being that insects are active earlier in spring in warm greenhouses than outside. Locality 2 was the agricultural area in the former floodplains of the river Waal, near the villages Ooij and Leuth, located north-east of the city of Nijmegen, the Netherlands. I chose to include this locality since the DIOPSIS cameras have been deployed there in previous years as part of the Living Lab Ooijpolder-Groesbeek research project (Radboud University 2023). For each locality, six locations were selected to place the cameras. Figure 3A shows the locations in the Hortus Botanicus, while figure 3B shows the locations near Nijmegen. From now on, I will refer to the latter locality as “Ooijpolder” due to the cameras’ general proximity to this area.

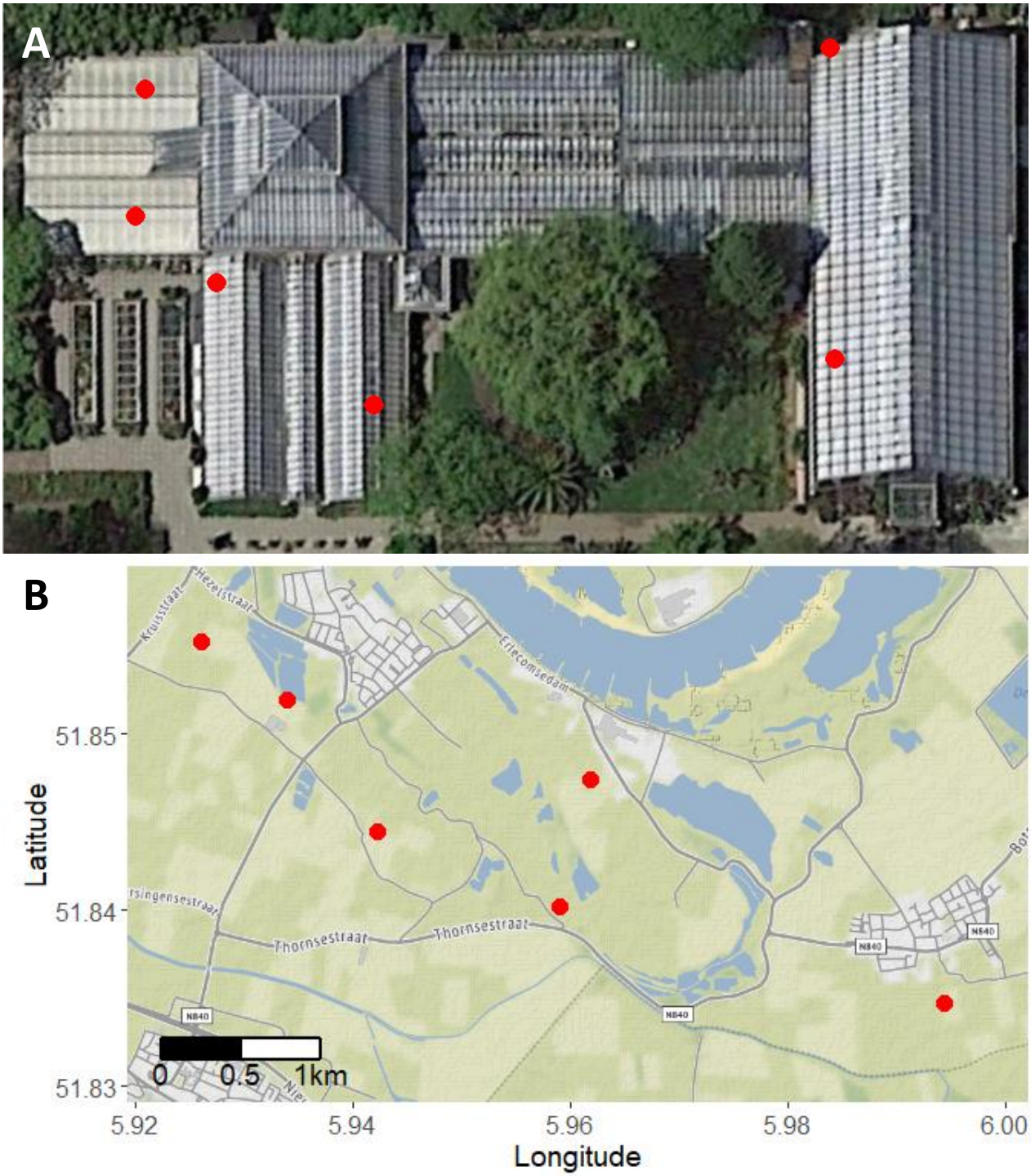


Figure 3: Locations of the DIOPSIS insect cameras | Figure 3A and 3B respectively show the locations in the Hortus Botanicus in Leiden and in the agricultural landscape near Ooij and Leuth.

The screens were placed following a fractional factorial design and according to a rotation schedule generated in the programming language R (R Core Team 2023). The rotation schedule was created according to some rules I set. First of all, it had to be completely randomly generated to prevent bias. Secondly, my experiment took place together with a second student's experiment. This experiment looked for the effect of an aromatic substance on the three new colored screens' attractance. Therefore, three additional treatments had to be included in the rotation schedule that

are not part of my experiment. This brought the total number of treatments to thirteen. Thirdly, it had to take into account the limited number of screens that was available. Therefore, the script assumed that one screen of each screen type was available for each timeslot. Fourthly, each camera would have to have received each treatment exactly once. Lastly, the treatments had to be rotated after two days when possible. Rotations were not possible during weekends, so certain treatment slots lasted three days. Appendix 5 shows the final rotation schedules. The R scripts to generate both locations' rotation schedules are presented in Appendix 6; Script 1 (Hortus Botanicus) and Script 2 (Ooijpolder).

There was a deviation from the rules I set for the Hortus Botanicus' schedule, as the treatments with the attractant could only start from the sixth timeslot onwards due to supply issues of the attractant. Furthermore, additional timeslots were added to both schedules because multiple camera errors occurred, causing some treatments to be either heavily underrepresented or missing for certain locations. More about these issues can be read in the Results section of this report.

After the experiment, the photos were sent to the ARISE Biodiversity project for the machine learning based identification process (ARISE 2022). This process utilized the same algorithm as was used in previous years.

All data processing was done through R, and I have provided a list with the packages I used in appendix 3. The R scripts themselves can be found in Appendix 6; Script 3 (Hortus Botanicus) and Script 4 (Ooijpolder). The output of the identification process consisted of datasets separated by camera number containing an entry for each identified individual. Each entry contained additional variables such as the date and time of the individual's first observation, the taxon level to which it was determined, and the estimated probability of the identification being correct (the identification score). In order to prepare the data for my analyses, I processed and supplemented the data as follows: First of all, I decided to create two datasets split by locality. I made this decision because of the large differences between the two locations, such as species compositions and the absence of weather effects in the Hortus Botanicus. Then, I added the treatment types to the data entries, while removing any entries that did not have a treatment type of interest (including those treated with an attractant). Furthermore, only the name of the lowest determined taxon level was provided in the datasets. This means that e.g. for individuals determined up to the species level, only the species name was provided, while for those individuals determined up to the family level, only the family name was provided. In order to list all the taxon level names of an individual, I used the GBIF (Global Biodiversity Information Facility) API (GBIF 2023). Next, I had to group the individuals per unit of time. In previous years, the individuals were grouped per hour, which made me decide to do the same for my analysis. This also meant that I had to discard observations that were not part of a full hour. For example, if a camera was turned off (either on purpose or due to software/hardware failure) at the 51st minute of an hour (e.g. 14.50), all other observations of that hour (14.00 – 14.59) were not used. Lastly, I decided to add a variable indicating whether the observation took place during the day or the night. During preliminary analysis of the photos, there appeared to be a large difference in the number of individuals during the day and night. Using the Sunrise Sunset API (Sunrise Sunset 2023), I assigned day as being in between sunrise and sunset, both rounded to the nearest hour, and vice versa for night.

Converting these observations to usable count data consisted of multiple steps: 1. Selecting a taxon level and taxa name of interest, e.g. the class Insecta. 2. Selecting a minimum identification score. This score indicates the certainty of the identification, ranging from 0 to 1. I included all individuals with a score higher than 0.50. 3. Counting the number of individuals that conform to these criteria

per hour. 4. Combining these datasets into one. These steps had to be repeated in order to compile a count dataset for other taxa levels and taxa names.

The count data were analyzed using generalized linear mixed-effects models with zero-inflation. This was done using the R package glmmTMB (Brooks et al. 2017)**.** For example, building a model for counts of individuals of the class Insecta in the Ooijpolder was realized by first selecting fitting explanatory variables for the response variable n (counts per hour). I selected dayCount_normalized (the normalized day number counted from the start of the experiment), dayOrNight (a variable indicating whether the hourly count takes place during the day (1) or the night (0)), and fullTreatment (a categorical variable with 10 levels for treatments differing containing screen color, screen orientation, screen texture, and/or screen version). There were six locations in the Ooijpolder: one for each camera ID. Therefore, I included a random part in the model, consisting of a random-intercept model, with cameraID as the grouping factor, and a random-slope model, with cameraID as the grouping factor and dayCount_normalized as the effect. This resulted in the following model (model 1):

Model 1

$$\text{Conditional: } n \sim \text{dayCount_normalized} + \text{dayOrNight*fullTreatment} + (1 + \text{dayCount_normalized} \mid \text{cameraID})$$
$$\text{Zero-inflated: } n \sim \text{dayCount_normalized} + (1 + \text{dayCount_normalized} \mid \text{cameraID})$$

I then used the drop1-function to find the most parsimonious model: the model with the lowest AIC. However, the current model was already the most parsimonious. Lastly, I validated the distribution of the model's residuals using the R package DHARMa (Hartig 2022a). First, I used a Kolmogorov–Smirnov (KS) test to test whether the sample follows the model's distribution correctly. I used DHARMa's built-in dispersion test to test whether the dispersion of the residuals did not deviate significantly. I tested only for overdispersion as this is recommended in the DHARMa documentation (Hartig 2022b). Lastly, I tested for significantly deviating outliers. Since a negative-binomial distribution is integer-valued, I used DHARMa's bootstrapped outlier test as per the documentation's recommendation (Hartig 2022b). These steps were repeated for the other models, and their p-values are reported in Appendix 4. Additionally, adjusted R^2-values were determined with the performance package (Lüdecke et al. 2021). To determine the estimated marginal means of the models and their significance when compared to the baseline, for which I had selected the version 2 screen, I used the R package emmeans (Lenth 2023). Finally, to visualize comparisons to the baseline, I used the R package ggplot2 (Wickham 2016).

Based on the results and visual comparisons between the actual photos that were taken, I found the counts of the blue screen treatments during the night to be unrealistically low. Therefore, I decided to also do some manual counts to compare against the counts determined by the algorithm. To do this, I randomly selected 120 10-minute intervals from the Ooijpolder data for which I would manually count the individuals present on the photos. These 120 samples contained 40 samples per color (excluding the v1 and v2 screens), and 20 samples per day-/nighttime, resulting in a total of 6 * 20 samples.

Results

My research goal was to quantify potential effects of screen version, screen color, screen texture, and screen orientation on the observed insect counts and species in the DIOPSIS setup. To achieve this goal, rotation schedules of different screens were generated in advance for both localities.

There were large deviations between the number of full hours during which the cameras were active, especially for the cameras in the Hortus (Figure 5 and 6). The total amount of observation hours was also considerably lower for the Hortus data when compared to the data of the Ooijpolder. Furthermore, these same deviations can be seen in the percentage of full hours planned in the rotation schedule that were actually fulfilled during the implementation of the experiment (Table 2). Part of the deviations within each locality can be explained by the fact that some of the time intervals were three – instead of two – days long; namely when occurring during weekends. Since the attractant-treatments had an equal chance of being scheduled in the weekend as the other treatments, some difference between the cameras was to be expected. Other factors were less predictable, such as camera downtime, which was especially present in the Hortus. In the end, the treatment with the fewest measurement-hours during the day in the Ooijpolder was white smooth vertical with 81 hours, and during the night also white smooth vertical with 186 hours. For the Hortus, the treatment with the fewest hours during the day was vertical v1 with 62 hours, and during the night white vertical smooth with 128 hours. It should be noted that there were unavoidably more measurement-hours during the day than the night, due to the fact that the amount of time between sunrise and sunset was higher than the amount between sunset and sunrise. Because the experiment in the Hortus took place earlier in the season, the duration of the night was also overall longer than in the Ooijpolder.

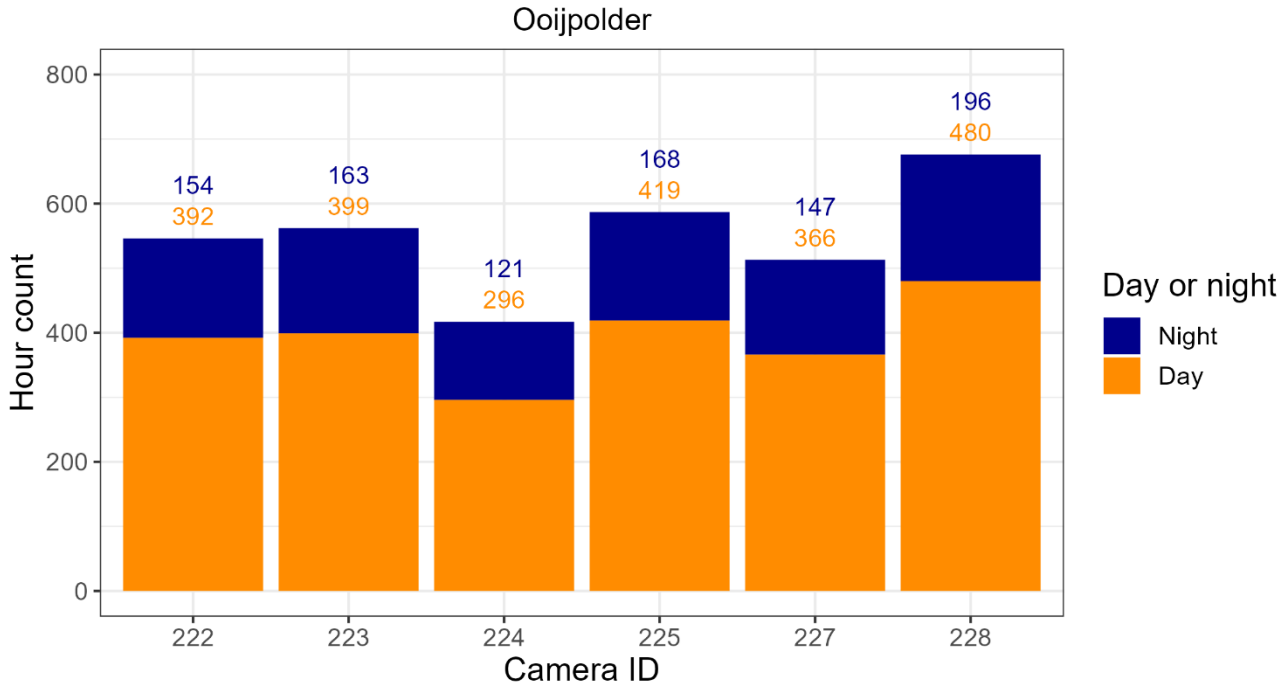


Figure 4: The number of full hours of camera activity in the Ooijpolder | Hours during which the cameras were (partially) not working are excluded.

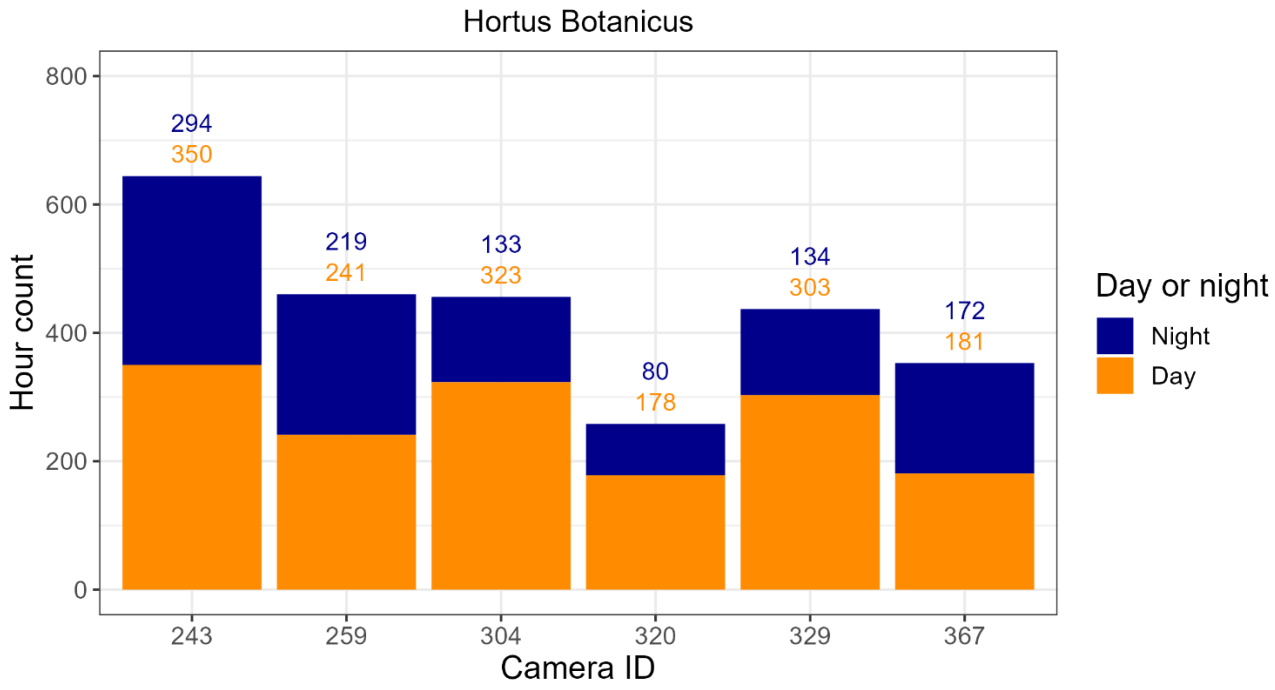


Figure 5: The number of full hours of camera activity in the Hortus. | Hours during which the cameras were (partially) not working are excluded.

Table 2: The percentage of full hours planned in the rotation schedule that was actually realized during the execution of the experiment

Ooijpolder		Hortus	
Camera ID	**% of planned hours realized**	**Camera ID**	**% of planned hours realized**
222	78.4%	243	92.5%
223	90.1%	259	76.7%
224	69.5%	304	65.5%
225	97.8%	320	38.4%
227	89.1%	329	62.8%
228	97.1%	367	52.5%

The species data from the identification algorithm again showed large differences in insect order counts between the Hortus (Figure 6) and the Ooijpolder (Figure 7) (identification certainty ≥ 0.5). It should be noted that these counts are not a direct representation of the total efficiency of the treatments at capturing insects, since they do not take into account differences in the number of treatment-hours. The counts did show that overall, there were far less observations made in the Hortus than in the Ooijpolder: 331 and 46764 individuals (with a certainty of 0.5 or higher) respectively. On the other hand, the same insect orders were detected in both localities. The counts from the Hortus also appeared to have a disproportionately large number of NAs; that is to say that these individuals were identified as insects with a certainty of 0.5 or higher, but that the algorithm was unable to detect the order or any higher taxonomic level.

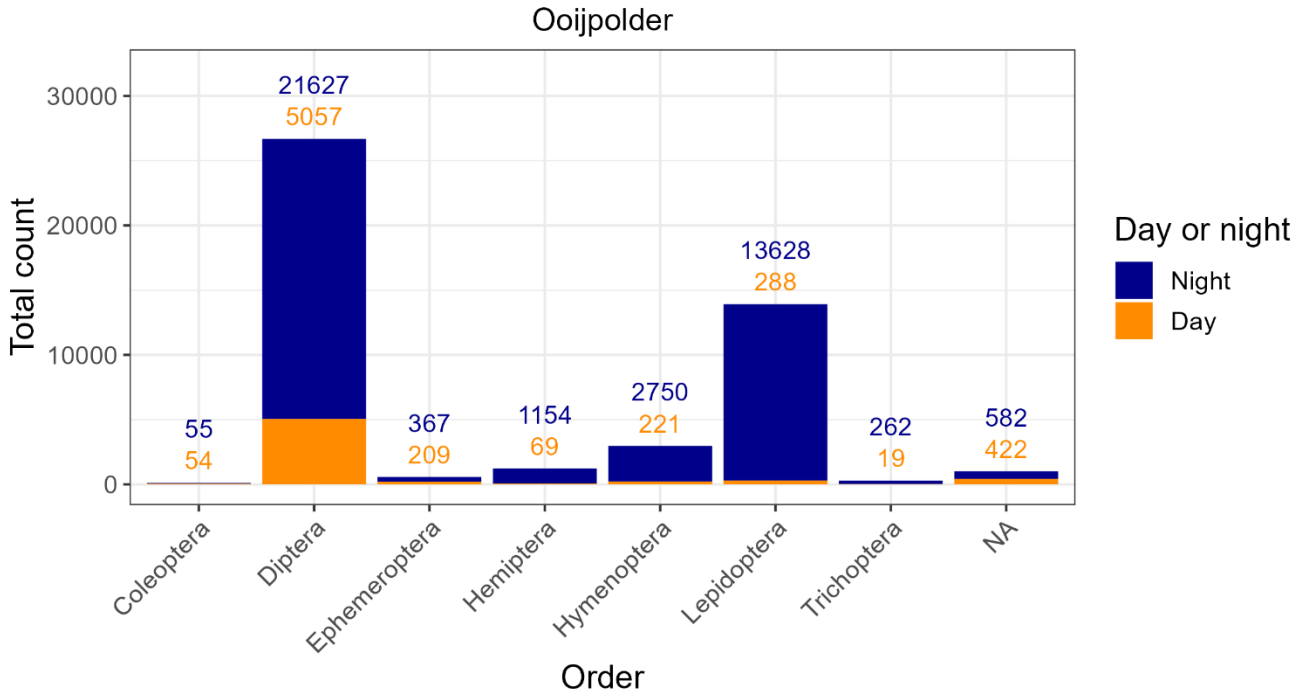


Figure 6: The total number of individuals in the Ooijpolder, aggregated by insect order | The certainty of the observations included is ≥ 0.5

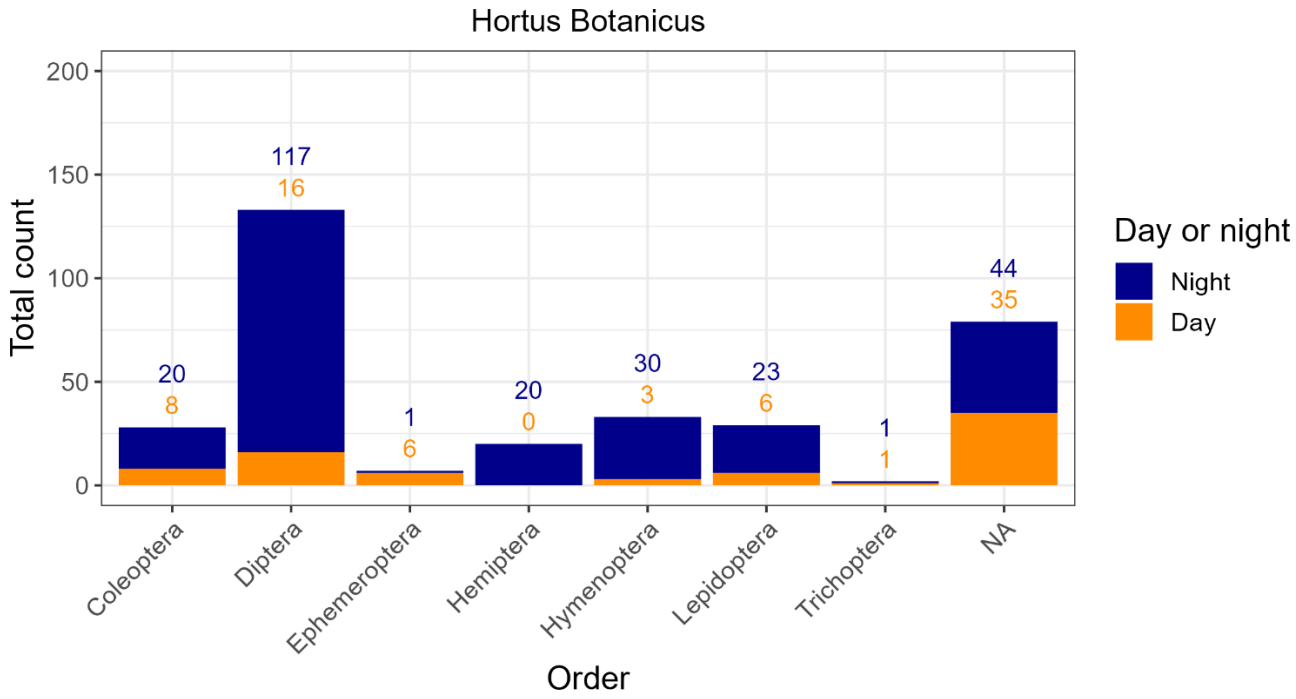


Figure 7: The total number of individuals in the Hortus, aggregated by insect order | The certainty of the observations included is ≥ 0.5

Ooijpolder models

The actual effects of the screens were determined using zero-inflated binomial generalized linear mixed models. These models all used the v2 screen, both during daytime and during nighttime, as a baseline. The first model that I present is the model containing all insects counts in the Ooijpolder. Because this model was used as an example, the creation of this model is already extensively described in the materials and methods-section, where it is shown as Model 1. The adjusted R^2 of this model was 0.998, whose high value was likely caused by overfitting due to the high number of treatment factors included in the model. Certain other models in this results section also show this behavior. This was not deemed to be problematic, since the model was not meant to be directly applied to other datasets but is instead more of an exploratory nature. Model 1 predicts significant differences between counts, depending on treatment and whether it was day- or nighttime (Figure 8). Note that the v1, v2, and yellow screens all had a different yellow hue, as can be seen in the figure. The values represent the overall mean predicted counts per hour, with the values of the zero-inflated part of the model included. The figure shows multiple significant values. Of note were the considerably lower values of both the blue screens during the night, while that of the blue rough vertical screen was significantly higher during daytime. A similar phenomenon could be observed at the yellow horizontal screens: these had much higher count values than the baseline during the day, while the opposite was true during the night. This interaction effect was also visible for the v1 vertical screen and the yellow rough vertical screen. Furthermore, the white smooth vertical screen had a considerably high predicted count during the night. In this model, the baseline odds of not having any counts at any given hour were $1.976*10^{-4}$ (Table 3).

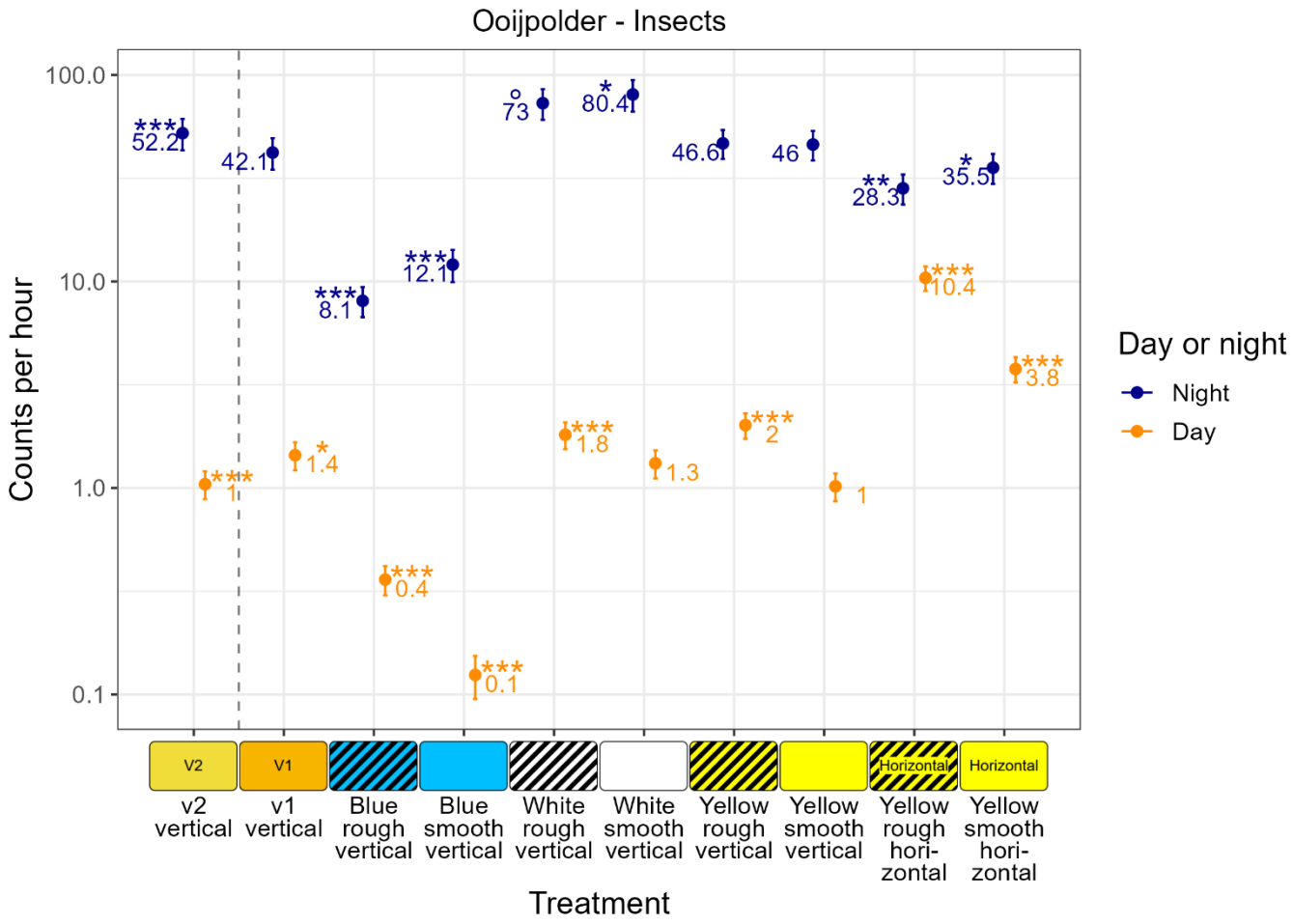


Figure 8: Visualization of the predicted insect counts per hour of the conditional part of Model 1, divided by treatment and whether the count is for day- or nighttime | The values show the mean predicted counts per hour for those hours when the screen has a chance to attract insects. Note that the y-axis is on a log10 scale. The v2 vertical treatment is used as the baseline for comparisons: the night-counts use the v2 vertical treatment during the night as a baseline, and the day-counts use the v2 vertical treatment during the day. Error bars indicate the standard error. The numeric values next to the points give the counts per hour for that specific point. Significance of deviance of the count, as compared to the corresponding baseline, is given using symbols: p = 0: *** p = 0.001: **, p = 0.01: *, and p = 0.05: °.

Table 3: Values of the zero-inflated part of Model 1

	Prediction	**Standard error lower value**	**Standard error upper value**	**p-value**
Intercept	$1.976*10^{-4}$	$8.286*10^{-6}$	$4.715*10^{-3}$	0.00717 (**)
Day count (normalized)	$5.061*10^{-6}$	$6.335*10^{-7}$	$4.043*10^{-5}$	0.0778 (°)

The prediction is the odds of a screen having no chance to attract any insects during any given hour. The symbols next to the p-values indicate significance, with p = 0: *** p = 0.001: **, p = 0.01: *, and p = 0.05: °.

I will present three additional models of the three most commonly counted insect orders in the Ooijpolder. In the Ooijpolder, the three most commonly counted insect orders were those of the Diptera, the Hymenoptera, and the Lepidoptera. Therefore the next model, Model 2, focused on a specific insect order: that of the Diptera. This model had an adjusted R^2 of 0.994. The overall pattern of the predicted counts was comparable to that of the insects (Figure 9). Relatively seen however, the counts of the blue screens were much lower, and the deviation of some treatments from the base treatment was now less, or more, significant. The yellow rough horizontal screens especially appeared to attract many Diptera during the day. Since none of the values of the zero-inflated part were significant, the baseline odds of the model could be assumed to be the null hypothesis of 1 (Table 4). This means that any given hour had a fifty percent chance of not having any chance of attracting Diptera at all.

Model 2

Conditional: $n \sim dayCount_normalized + dayOrNight*fullTreatment + (1 + dayCount_normalized \mid cameraID)$
Zero-inflated: $n \sim dayCount_normalized + dayOrNight + (1 + dayCount_normalized \mid cameraID)$

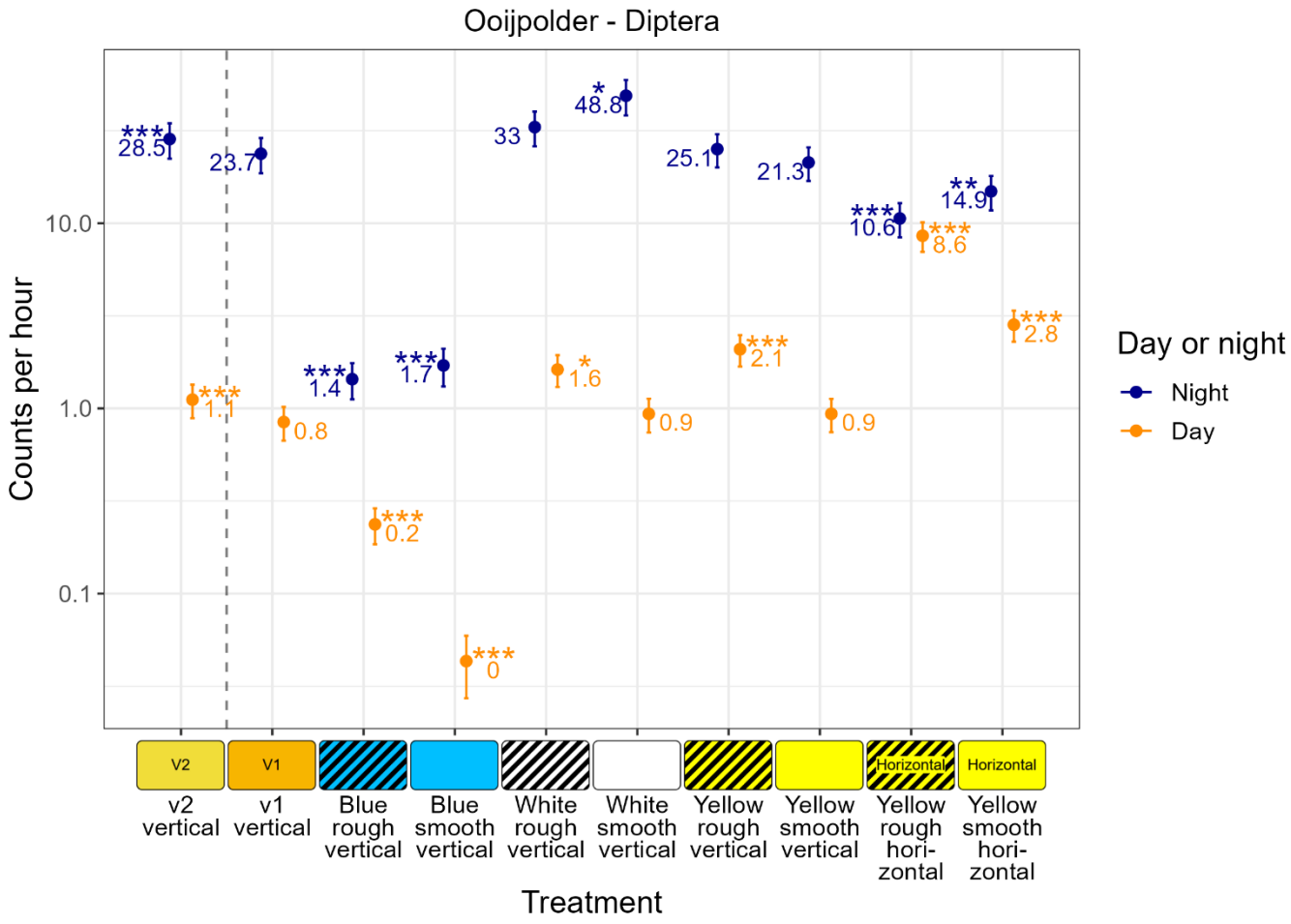


Figure 9: Visualization of the predicted Diptera counts per hour of the conditional part of Model 2, divided by treatment and whether the count is for day- or nighttime | The values show the mean predicted counts per hour for those hours when the screen has a chance to attract Diptera. Note that the y-axis is on a log10 scale. The v2 vertical treatment is used as the baseline for comparisons: the night-counts use the v2 vertical treatment during the night as a baseline, and the day-counts use the v2 vertical treatment during the day. Error bars indicate the standard error. The numeric values next to the points give the counts per hour for that specific point. Significance of deviance of the count, as compared to the corresponding baseline, is given using symbols: p = 0: *** p = 0.001: **, p = 0.01: *, and p = 0.05: °.

Table 4: Values of the zero-inflated part of Model 2

	Prediction	**Standard error lower value**	**Standard error upper value**	**p-value**
Intercept	$1.886*10^{-8}$	$6.650*10^{-83}$	$5.352*10^{66}$	0.917
Day count (normalized)	$6.041*10^{-9}$	$2.496*10^{-9}$	$1.462*10^{-8}$	0.198
Daytime	$3.887*10^{-2}$	$1.375*10^{-76}$	$1.098*10^{73}$	0.932

The prediction is the odds of a screen not attracting any Diptera during any given hour. The symbols next to the p-values indicate significance, with p = 0: *** p = 0.001: **, p = 0.01: *, and p = 0.05: °.

The next model, Model 3, focused on the order of the Hymenoptera. The adjusted R^2 of this model was 0.854. The predicted counts per hour were overall quite low in this model, with significant values reaching close to zero individuals per hour (Figure 10). Again, the yellow horizontal screens appeared to attract substantially more individuals during the day as compared to the rest of the treatments, while the opposite was true during the night. Here too, the blue screens had very low values during the night. The counts of the blue screens during the day were missing due to the predictions having unrealistically high standard errors and very high p-values ($p > 0.99$). Here too, the baseline odds could be assumed to be 1 (Table 5).

Model 3

$$\text{Conditional: } n \sim \text{dayOrNight*fullTreatment} + (1 \mid \text{cameraID})$$
$$\text{Zero-inflated: } n \sim \text{dayOrNight} + (1 \mid \text{cameraID})$$

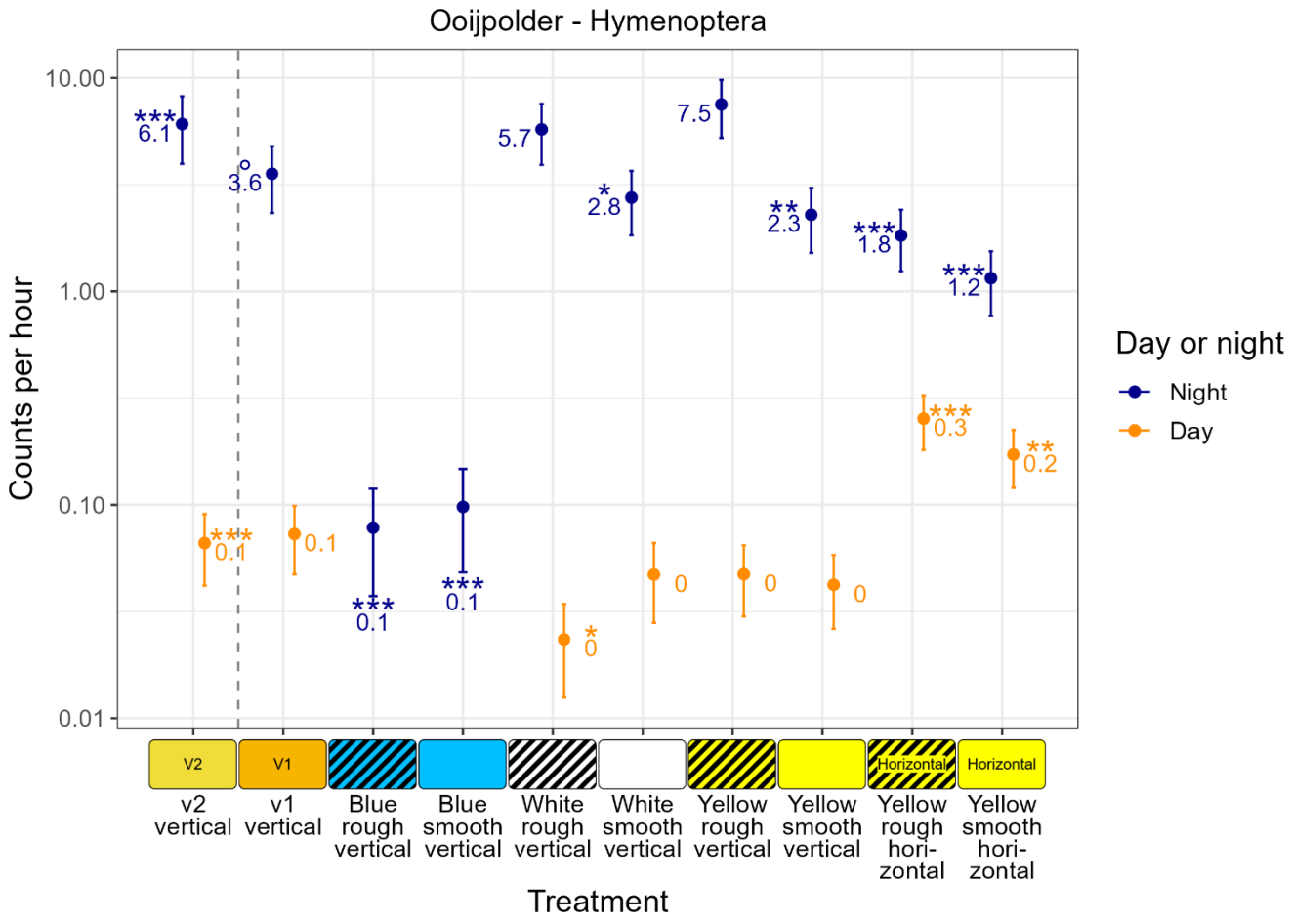


Figure 10: Visualization of the predicted Hymenoptera counts per hour of the conditional part of Model 3, divided by treatment and whether the count is for day- or nighttime | The values show the mean predicted counts per hour for those hours when the screen has a chance to attract Hymenoptera. Note that the y-axis is on a log10 scale. The v2 vertical treatment is used as the baseline for comparisons: the night-counts use the v2 vertical treatment during the night as a baseline, and the day-counts use the v2 vertical treatment during the day. Error bars indicate the standard error. The numeric values next to the points give the counts per hour for that specific point. Significance of deviance of the count, as compared to the corresponding baseline, is given using symbols: p = 0: *** p = 0.001: **, p = 0.01: *, and p = 0.05: °.

Table 5: Values of the zero-inflated part of Model 3

	Prediction	**Standard error lower value**	**Standard error upper value**	**p-value**
Intercept	0.367	0.207	0.653	0.0818 °
Daytime	$4.151*10^{-6}$	$1.096*10^{-95}$	$1.572*10^{84}$	0.9560

The prediction is the odds of a screen having no chance to attract any Hymenoptera during any given hour. The symbols next to the p-values indicate significance, with p = 0: *** p = 0.001: **, p = 0.01: *, and p = 0.05: °.

For the last model, Model 4, of the Ooijpolder, I focused on the Lepidoptera order. The adjusted R^2 of this model was 0.994. The predictions of Model 4 showed a clear divide between day and night (Figure 11). Furthermore, the yellow horizontal screens were the only treatments during the day with a significant deviation from the v2 vertical screen. For the night counts there were more significant deviations. The white screens especially seemed to function better during the night. The baseline odds could again be set to 1 (Table 6).

Model 4

Conditional: n ~ dayOrNight*fullTreatment + (1 + dayCount_normalized | cameraID)
Zero-inflated: n ~ dayOrNight + (1 + dayCount_normalized | cameraID)

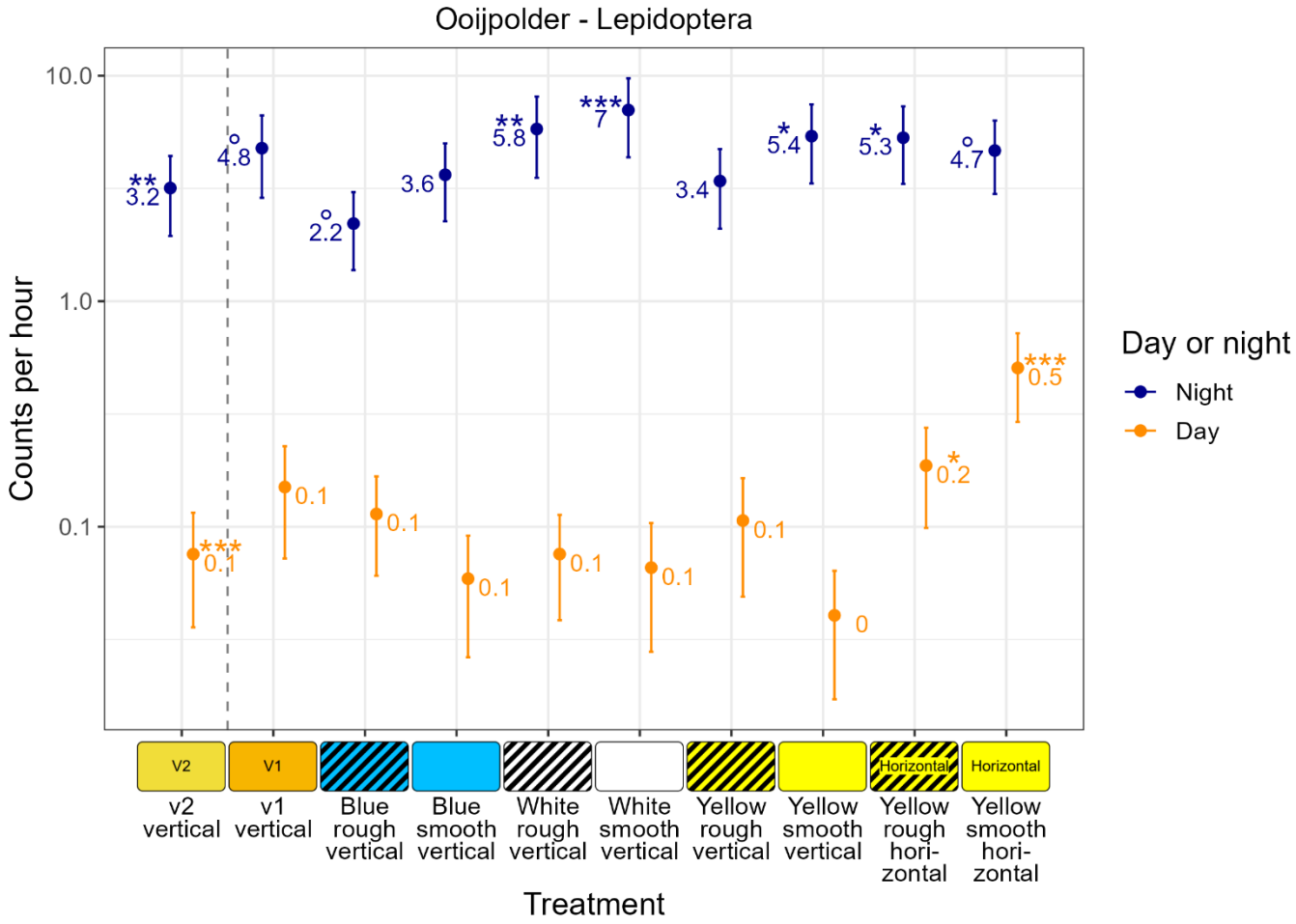


Figure 11: Visualization of the predicted Lepidoptera counts per hour of the conditional part of Model 4, divided by treatment and whether the count is for day- or nighttime | The values show the mean predicted counts per hour for those hours when the screen has a chance to attract Lepidoptera. Note that the y-axis is on a log10 scale. The v2 vertical treatment is used as the baseline for comparisons: the night-counts use the v2 vertical treatment during the night as a baseline, and the day-counts use the v2 vertical treatment during the day. Error bars indicate the standard error. The numeric values next to the points give the counts per hour for that specific point. Significance of deviance of the count, as compared to the corresponding baseline, is given using symbols: p = 0: *** p = 0.001: **, p = 0.01: *, and p = 0.05: °.

Table 6: Values of the zero-inflated part of Model 4

	Prediction	**Standard error lower value**	**Standard error upper value**	**p-value**
Intercept	$1.148*10^{-7}$	$5.071*10^{-71}$	$2.599*10^{56}$	0.913
Daytime	2.535	$1.119*10^{-63}$	$5.738*10^{63}$	0.908

The prediction is the odds of a screen having no chance to attract any Lepidoptera during any given hour. The symbols next to the p-values indicate significance, with p = 0: *** p = 0.001: **, p = 0.01: *, and p = 0.05: °.

Hortus Botanicus models

The first Hortus model, Model 5, predicted the counts of all insects. This model, and the following model, did not contain an interaction effect between the treatment and whether the count was recorded during the night or during the day since it was not possible to create such models with the available data. The adjusted R^2 of this model was 0.354. Overall, counts appeared to be very low (Figure 12). Despite this, there were still significant deviations. The white rough vertical and yellow rough horizontal screens seemed to perform best, while the blue rough vertical screen showed the fewest counts. Still, many of the screens performed above the baseline. The baseline odds of the zero-inflated part were non-significant for the intercept, but they were 3.678 for counts during daytime as compared to nighttime (Table 7).

Model 5

$$\text{Conditional: } n \sim \text{dayOrNight} + \text{fullTreatment} + (1 \mid \text{cameraID})$$
$$\text{Zero-inflated: } n \sim \text{dayOrNight} + (1 \mid \text{cameraID})$$

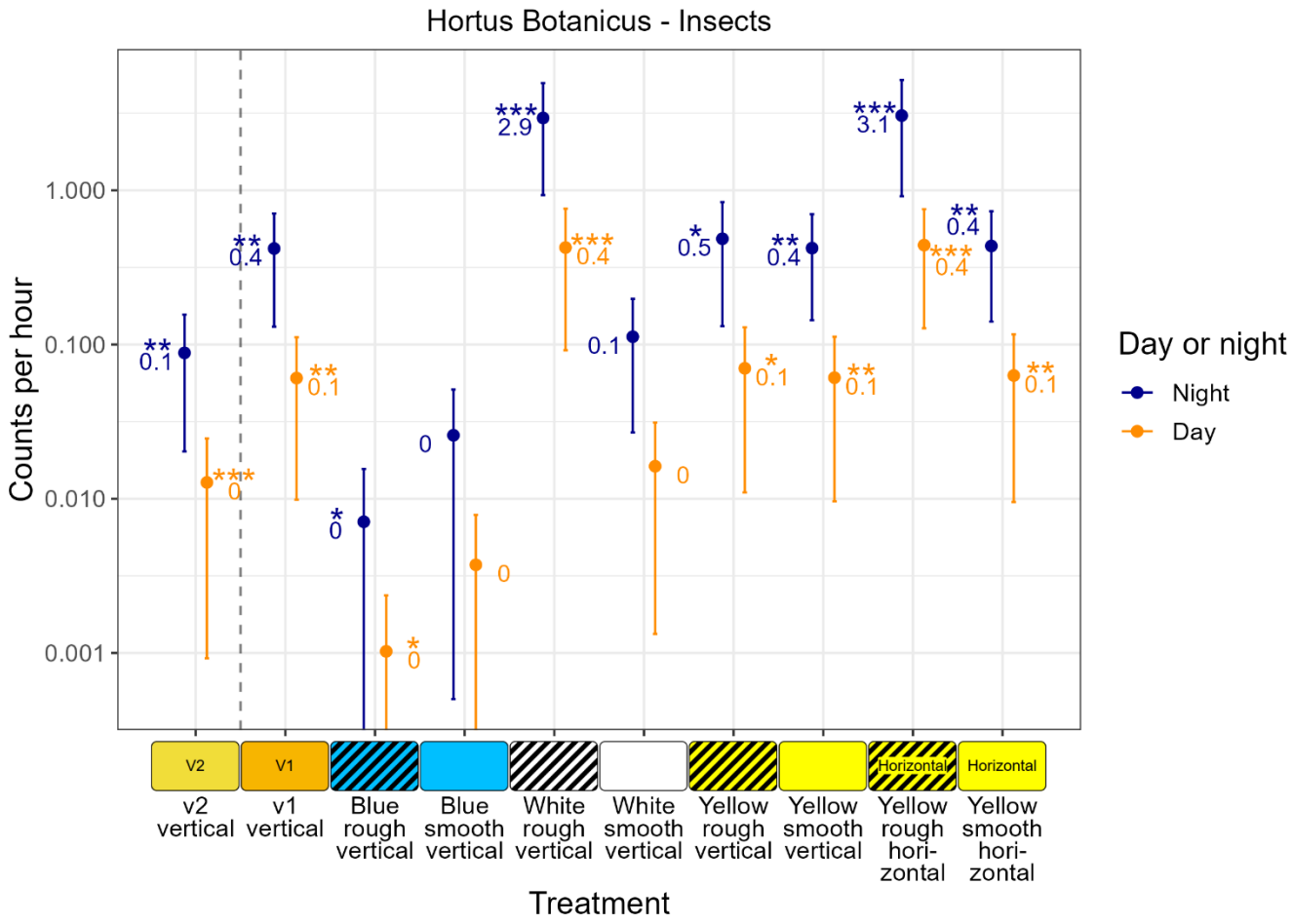


Figure 12: Visualization of the predicted insect counts per hour of the conditional part of Model 5, divided by treatment and whether the count is for day- or nighttime | The values show the mean predicted counts per hour for those hours when the screen has a chance to attract insects. Note that the y-axis is on a log10 scale. The v2 vertical treatment is used as the baseline for comparisons: the night-counts use the v2 vertical treatment during the night as a baseline, and the day-counts use the v2 vertical treatment during the day. Error bars indicate the standard error. The numeric values next to the points give the counts per hour for that specific point. Significance of deviance of the count, as compared to the corresponding baseline, is given using symbols: p = 0: *** p = 0.001: **, p = 0.01: *, and p = 0.05: °.

Table 7: Values of the zero-inflated part of Model 5

	Prediction	**Standard error lower value**	**Standard error upper value**	**p-value**
Intercept	0.638	0.219	1.861	0.675
Daytime	3.678	2.165	6.249	0.000953 ***

The prediction is the odds of a screen having no chance to attract any insects during any given hour. The symbols next to the p-values indicate significance, with p = 0: *** p = 0.001: **, p = 0.01: *, and p = 0.05: °.

Unlike for the Ooijpolder models, I was only able to create a functioning model containing the treatments for the Diptera order. The Hymenoptera and Lepidoptera counts proved to be insufficient to create a model from. Therefore, the second Hortus model, Model 6, looked at the predictions for Diptera, and it had an adjusted R^2 of 0.168. The base counts of the model are non-significant, like many of the other count values in the model (Figure 13). Only the v1 vertical and white rough vertical screens had significantly deviating predicted counts, which were all larger than the other screens' count values. The baseline odds of the zero-inflated part were 14.711 for the intercept and increased by 5.159 for each passing day (Table 8). Furthermore, the odds of a screen having no chance of attracting any Diptera in one hour during the day were much higher than during the night: namely 127.256.

Model 6

$$\text{Conditional: } n \sim \text{dayCount_normalized} + \text{dayOrNight} + \text{fullTreatment} + (1 \mid \text{cameraID})$$
$$\text{Zero-inflated: } n \sim \text{dayCount_normalized} + \text{dayOrNight} + (1 \mid \text{cameraID})$$

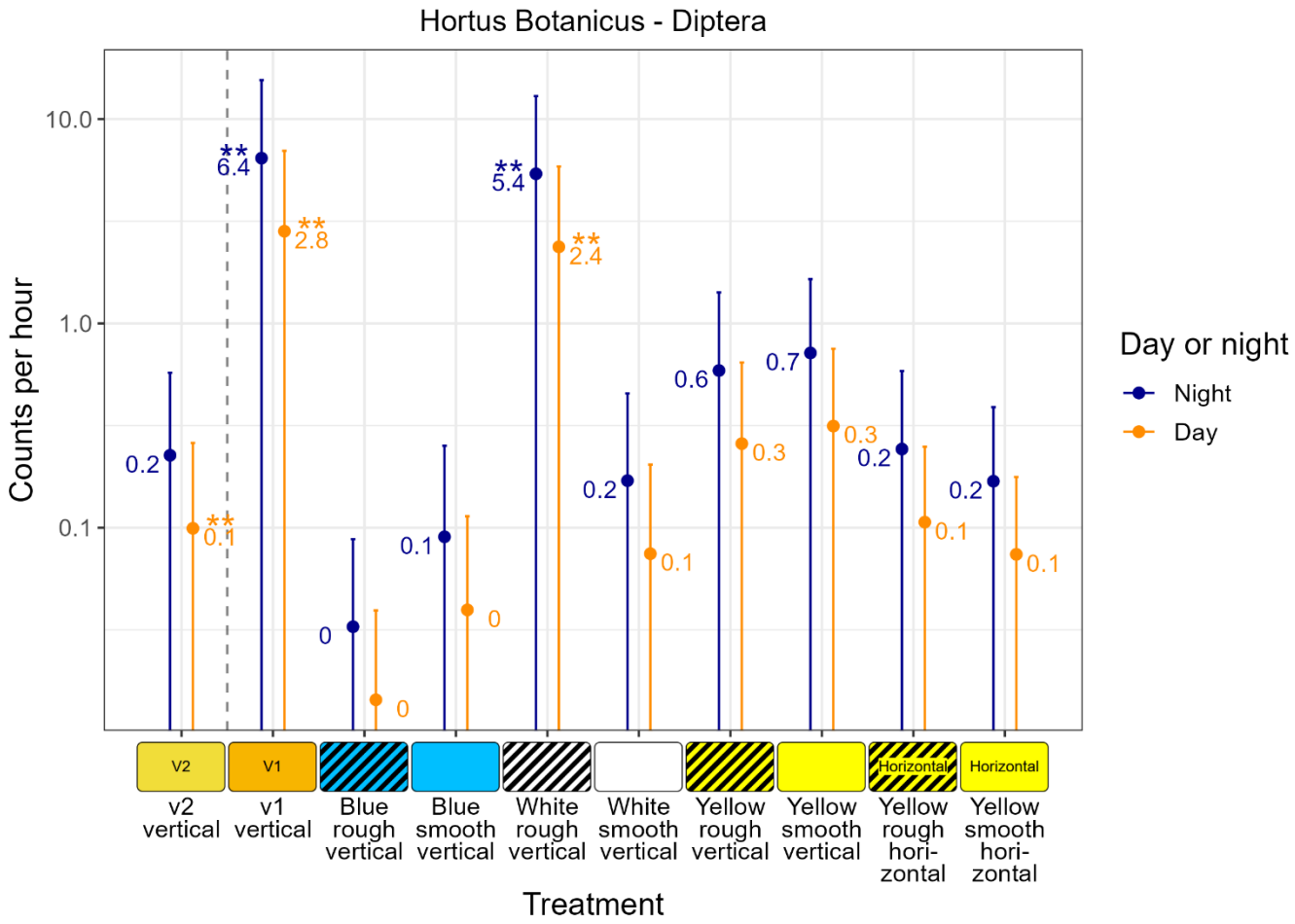


Figure 13: Visualization of the predicted Diptera counts per hour of the conditional part of Model 6, divided by treatment and whether the count is for day- or nighttime | The values show the mean predicted counts per hour for those hours when the screen has a chance to attract Diptera. Note that the y-axis is on a log10 scale. The v2 vertical treatment is used as the baseline for comparisons: the night-counts use the v2 vertical treatment during the night as a baseline, and the day-counts use the v2 vertical treatment during the day. Error bars indicate the standard error. The numeric values next to the points give the counts per hour for that

Table 8: Values of the zero-inflated part of Model 6

	Prediction	**Standard error lower value**	**Standard error upper value**	**p-value**
Intercept	14.711	4.914	44.040	0.0142 *
Day count (normalized)	5.159	4.010	6.637	$3.18*10^{-5}$ ***
Daytime	127.256	76.662	211.241	$2.07*10^{-5}$ ***

The prediction is the odds of a screen not attracting any Diptera during any given hour. The symbols next to the p-values indicate significance, with p = 0: *** p = 0.001: **, p = 0.01: *, and p = 0.05: °.

Manual count comparisons

In most cases, the algorithm predicted lower values than manual counting did (Figure 14). Any values below the black calibration line indicate a lower count by the algorithm as compared to the manual counts, and vice versa for values above the calibration line. The exception to the algorithm predicting lower counts were the counts on the white screens during the day, where there were instances where the counts by the algorithm are much higher than the manual counts. For the counts during the night, the manual counts appeared to deviate most from the algorithm counts for the blue screens. The same was true for the day counts, where the algorithm did not detect any insects on the blue screens during the sample intervals.

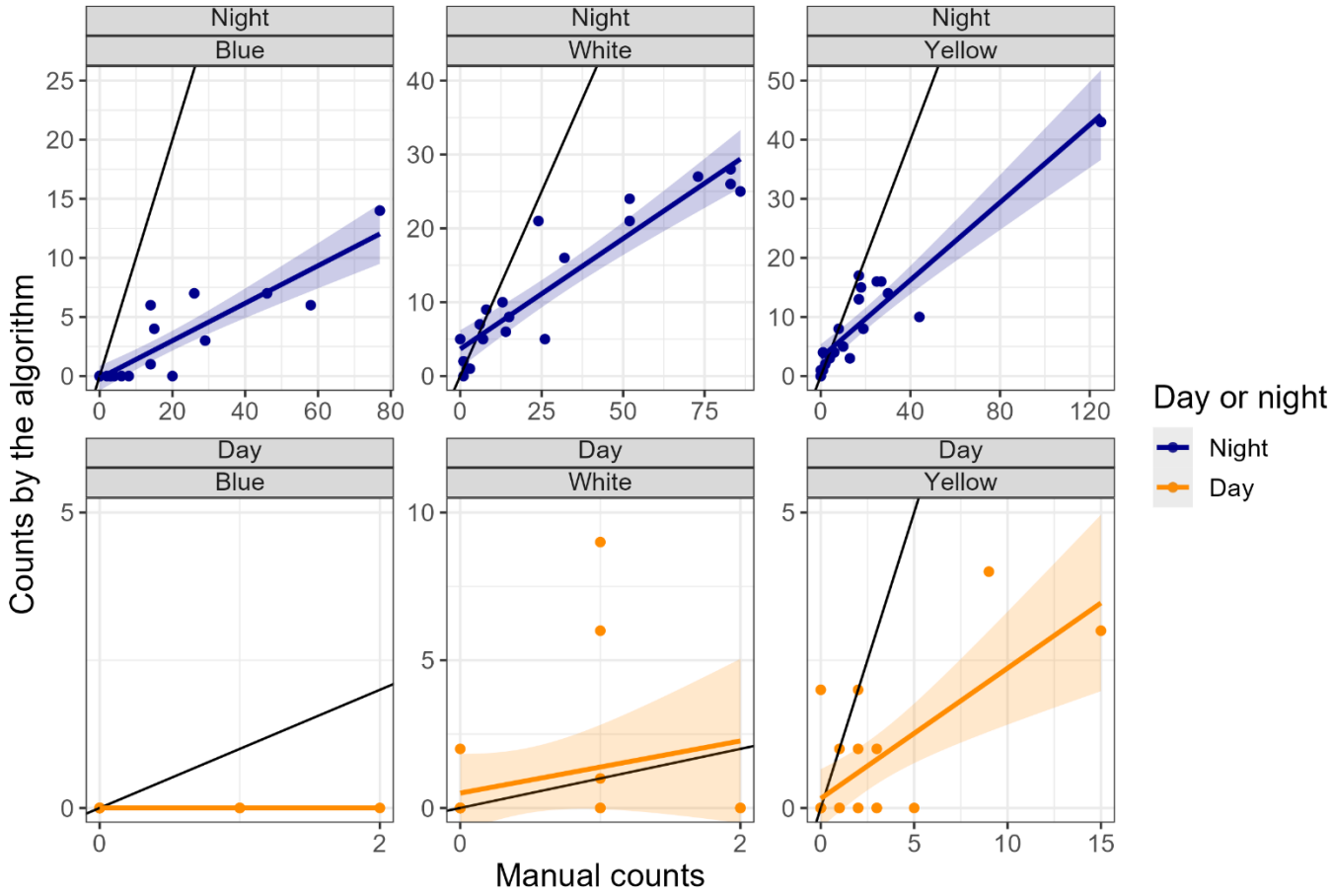


Figure 14: Manual insect counts compared to counts by the algorithm of 10 minute intervals | The black line is a calibration line with the formula y = x. The colored lines show simple linear regression models through the data, and the semi-transparent bounds show the standard error.

Discussion

The goal of DIOPSIS is monitoring insect biodiversity by means of color/light traps and an automated identification process. It is therefore important that we know how screen properties, like color, orientation, and texture, affect the taxonomic composition and number of photographed insects. I expected there to be differences in insect counts between screens that have different properties, such as a certain color, texture, and orientation comparably attracting more individuals, and that these patterns differ between taxonomic groups. Furthermore, I expected that counts and patterns would also differ between day and night, and between different environments.

The results of the experiment showed some clear differences between the trapping capabilities of different treatments. Based on the modelled performance of the treatments, one might for example choose white, smooth, vertical screens when trying to catch as many nocturnal insects as possible in an area comparable to the Ooijpolder. On the other hand, one might prefer a yellow, rough, horizontal screen for day-active insects in that same area. Based on the situation and the insect groups of interest, these results can help make a decision for the appropriate screen to choose.

The results showed some patterns, and deviations from said patterns, in the predicted insects counts. First of all, in the Ooijpolder, the v1 screen had a significantly slightly higher predicted count during the day than the v2 screen when taking into account all insects, but not for Diptera, Hymenoptera, or Lepidoptera. In the Hortus, where insect numbers were lower and included tropical species, the v1 screen appeared to function significantly better than the v2 screen.

Looking at the effect of just the colors during the day in the Ooijpolder (so only the blue, white, and yellow rough vertical screens), blue and yellow performed equally well for all insects together. During the night, none of the colors performed above the baseline. Focusing on just one insect order, the Diptera, the colors yellow and white had the highest counts. During the night, no color performed better than the other colors as compared to the v2 screen. The same was true for the Hymenoptera during the day and during the night. Lepidoptera also appeared to have no single preferred color during the day, but during the night, white screens outperformed the other two colors. Nocturnal Lepidoptera – mostly moths – are known to be attracted to UV light, and as such, the white screens may reflect UV light the strongest (Brehm 2017; Brehm et al. 2021). The counts from the Hortus showed a different color preference for all insects together; white got the highest counts during both night and day. The same goes for Diptera in the Hortus.

Screen texture also appears to have an effect on the number of captured insects. In general, the predicted number of insects per hour on smooth surfaces is higher than or equal to those on rough surfaces during the night, regardless of screen orientation. However, the opposite is true during the day. This pattern generally seems to hold true for Diptera and Lepidoptera, but for Hymenoptera this response appears to be reversed: rough surfaces are preferred during the night, while during the day, there does not seem to be a real preference. In the Hortus, responses to texture are a bit more varying, and no general pattern becomes evident.

Lastly, the screen orientation showed some clear patterns. In the Ooijpolder, during the day, yellow horizontal screens seemed to function much better for all insects than their yellow vertical counterparts. However, this effect was not present during the night, when often the opposite was true: the yellow horizontal screens often performed worse. This pattern could also be observed for Diptera and Hymenoptera. This is in line with the observations made by Olivier (2021), who also found a great preference of Diptera for horizontal screens in the DIOPSIS setup. Lepidoptera counts showed a comparable pattern, but the effect was smaller. The Hortus counts only showed a higher insect count for the rough, yellow, horizontal screen, but no other patterns.

However, caution should be taken in the interpretation of these results. The fact that these experiments took place in only 2 different environments makes it difficult to extrapolate the effects of the screens to other environments (Colwell and Coddington 1994). Insect compositions are very different in the agricultural part of the Ooijpolder than insect community compositions in e.g. a forest. The result could be that local species prefer a completely different color. There is also the inherent bias of the identification algorithm, since some species may be more difficult to identify just by a top-down photo. Different species compositions will experience this bias differently. For example, a similar experiment comparing the effect of screen colors in an early DIOPSIS version by Barendrecht et al. (2019) yielded results that deviated from the results presented in this report. As a result, comparing temporal data from the same locality will likely result in less bias-related problems than comparing two spatially different localities. In an ideal situation, the preferences for each individual species would be known, so that the treatment can be tailor-made for an environment in order to capture as many individuals as possible, or instead to focus on species groups of interest.

Since DIOPSIS is still under development, some additional challenges were posed by hardware and software failure. As has become evident from the results, the cameras sometimes experienced downtime, especially in the Hortus. These outages were not connected to any environmental factors but did appear more often right after the daily reset, which automatically occurred at 03.00h every night. Therefore, some systematic downtime did occur.

Compared to the Ooijpolder, the Hortus data contains a disproportionally large number of NAs. This can be explained by the presence of exotic species in the greenhouse (Figure 15). The ARISE project only focuses on insect species occurring in the Netherlands, and is therefore unable to identify any foreign species, especially those normally occurring in a tropical environment (ARISE 2022). The overall predicted counts are also much lower for the Hortus. Since the cameras in the Hortus were all placed in a closed greenhouse, the effective range of each camera was much smaller than in the Ooijpolder. The flowering plant biodiversity also did not appear to be very high in the Hortus, and I spotted less insects walking and flying around. Another factor contributing to the low numbers may be the various carnivorous plants that were present in some sections of the greenhouse.



Figure 15: A Phasmatodea species (left) and a Blattaria species (right) captured by DIOPSIS in the Hortus Botanicus

A challenge that DIOPSIS inherently experiences is its susceptibility to environmental factors combined with its ability to function autonomously. For example, a strong gust of wind can flip down the screen of the frame, and if the camera is placed in a remote area, it may take several days before someone manually puts the screen back up. Another difficulty that I experienced in my research period pertains to the horizontal screens: one of the cameras was placed close to a group of trees and surrounded by tall grass, both of which shed a lot of pollen during the time of the experiment. These pollen landed on the screen and got stuck there due to the sticky foam excreted by the many spittlebugs (Aphrophoridae) who inhabited the surrounding willows. Figure 16 shows the result of this process. While this is a very specific case, it is an example of how the environment may significantly influence the appearance of the screen, which may also have a negative effect on the ability of the identification algorithm to identify individual insects. Due to the strict rotation schedule in this experiment, I was able to keep the screen relatively clean, but this is more difficult during more autonomous operation of DIOPSIS. Other examples of the environment interfering with the DIOPSIS setup that I experienced are a spider that built its web in front of the camera to catch attracted insects, and birds that consumed insects that were attracted to the screen (Figure 17).



Figure 16: A yellow, smooth, horizontal screen covered in pollen and other plant materials.

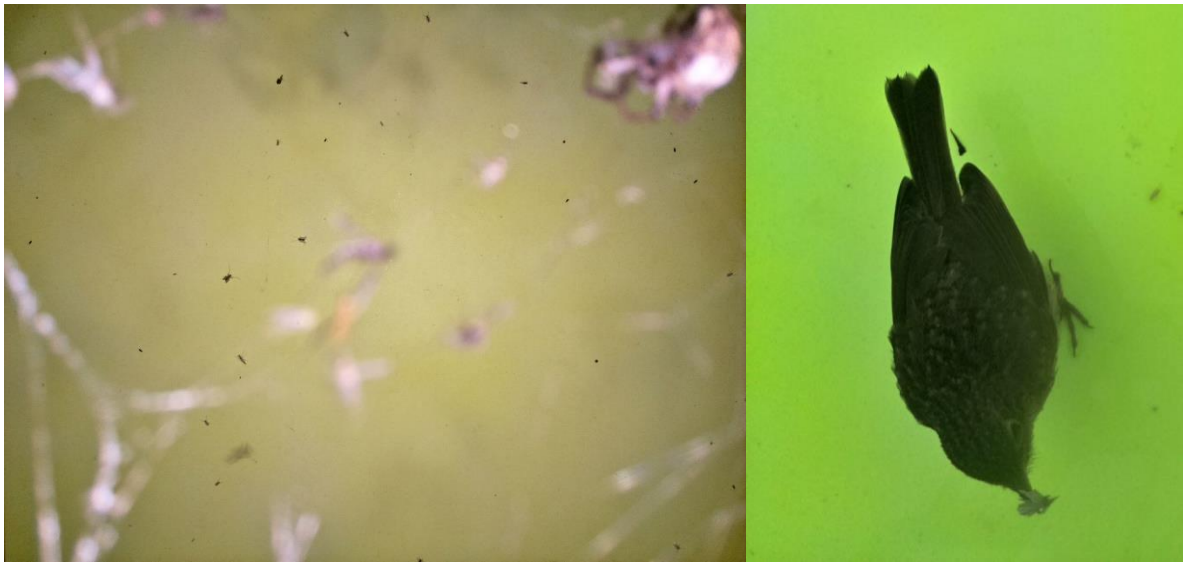


Figure 17: A spider and a bird making use of the attractive properties of DIOPSIS

The setup itself can also prove to be detrimental to the photo quality, and thus the certainty with which the algorithm can detect insects (Bergstrom and Messinger 2023). As becomes evident from the results, the underestimation of counts by the algorithm appears to be greater for blue screens. Looking at the actual photos that were taken, the blue screen appears visually distinct in multiple ways, especially during the night (Figure 18). First of all, the blue screen has a very bright, overexposed spot in the middle of the screen, which is largely absent in the yellow and white screens. Additionally, the insects appear to be much brighter on the blue screen than they do on the other screens. This can influence the ability of the algorithm to recognize the insects if the algorithm is not properly trained on color deviations from the standard, dark-colored insects (Dvornik et al. 2018).



Figure 18: Photos of (left to right, top to bottom) rough blue, yellow, and white screens taken around midnight 22-06-2023 in the Ooijpolder, including a close up of one of the screen's insects

Follow-up research to properly investigate the effect of all these factors on not only DIOPSIS' ability to catch insects, but also the algorithm's ability to identify individuals could greatly increase the reliability and decrease the bias of DIOPSIS automated way of identifying insects.

Over the past few years, several machine learning-based in situ insect sampling/identification methods have been implemented, many of which use computer vision like DIOPSIS (Hoye et al. 2021). For example, Bjerge et al. (2021b) have developed a setup with a screen that functions very similar to DIOPSIS, but with a focus on monitoring moths. They also focus in weather conditions, something not explicitly included in my models. Quantifying the effects of such environmental factors could also be of use for DIOPSIS in a follow-up study. Another computer vision-based setup by Bjerge et al. (2021a) focuses more on insect behavior instead of bulk identification, and, unlike DIOPSIS, does not utilize a screen for the backdrop, but it points the camera directly at the vegetation that the insects visit. This eliminates the potential bias caused by screen properties completely, but instead introduces a greater challenge regarding the identification algorithm, since the background of the photos is not standardized. Computer vision can also be combined with other techniques, like radar and acoustic monitoring (van Klink et al. 2022). These methods do have their own limitations, like radar only being able to detect flying insects, or acoustic monitoring requiring insects to produce sounds. However, the world's largest acoustic sensor network – The Australian Acoustic Observatory – has already managed to successfully capture insect sounds, illustrating that this can be a feasible method on a large scale (Roe et al. 2021; van Klink et al. 2022). Furthermore,

radar combined with a camera pointing at the sky could make it possible to better identify those species that are less inclined to land (Noskov et al. 2021). These are but several of the many automated insect sampling methods that have emerged over the past few years, and such methods will likely become more standard in entomology (Rhodes et al. 2022).

Returning to my research goal, it is clear that there are differences in predicted count patterns when looking at screen properties and their ability to attract certain insect groups. Knowing these patterns, one can more easily decide on the right screen type to employ when trying to catch as many insects as possible in general, or from a certain taxon. This can contribute to the process of standardizing monitoring schemes, which allows for more interoperability between data from different projects, and ultimately results in more substantiated findings to guide conservation and management (Montgomery et al. 2021). My findings can also help with increasing the chance of sampling less abundant species, unveiling them by optimizing the screen based on their apparent preference (Preston 1948).

The development of machine learning and machine vision is progressing quickly, so more specific identification of individuals up to lower taxonomic levels may be possible in the future, even from photos that have already been collected such as in my experiment. Furthermore, any bias caused by the quality of the photos may become less severe in the future. Still, investigating the bias of the current algorithm can help to substantiate any conclusions drawn from insect data provided by the DIOPSIS pipeline, further solidifying DIOPSIS' role of complementing conventional insect trapping and identification methods.

Acknowledgements

I thank dr. ir. Eelke Jongejans for supervising me during this project, Jeff van Eekelen for assisting me with carrying out the experiment in both the Hortus Botanicus and the Ooijpolder, Robin Lexmond MSc and Lewis Jansen for assisting me with the placement of the DIOPSIS cameras in the Ooijpolder, Stephan Peterse for providing DIOPSIS cameras to use in the Hortus Botanicus, and dr. Chantal Huijbers for making arrangements to get the photos through the identification pipeline.

References

Ali U, Karim KJBA, Buang NA (2015) A Review of the Properties and Applications of Poly (Methyl Methacrylate) (PMMA). Polymer Reviews 55:678-705. doi: 10.1080/15583724.2015.1031377

Allsopp MH, de Lange WJ, Veldtman R (2008) Valuing insect pollination services with cost of replacement. PLoS One 3:e3128. doi: 10.1371/journal.pone.0003128

ARISE (2022) Biodiversity. https://www.arise-biodiversity.nl/

Ashby M (2013) Materials and the Environment: Eco-informed Material Choice, 2 edn. Butterworth-Heinemann, Oxford, United Kingdom

Barendrecht R, Bloem G, Krijns R (2019) Automatische beeldherkenning van insecten met cameravallen. HAS Hogeschool, 's-Hertogenbosch

Bergstrom AC, Messinger DW (2023) Image quality and computer vision performance: assessing the effects of image distortions and modeling performance relationships using the general image quality equation. Journal of Electronic Imaging 32. doi: 10.1117/1.Jei.32.2.023018

Beutel RG, Gorb SN (2001) Ultrastructure of attachment specializations of hexapods (Arthropoda): evolutionary patterns inferred from a revised ordinal phylogeny. Journal of Zoological Systematics and Evolutionary Research 39:177-207. doi: 10.1046/j.1439-0469.2001.00155.x

Bjerge K, Mann HMR, Høye TT, Sankey T, Ahumada J (2021a) Real-time insect tracking and monitoring with computer vision and deep learning. Remote Sensing in Ecology and Conservation 8:315-327. doi: 10.1002/rse2.245

Bjerge K, Nielsen JB, Sepstrup MV, Helsing-Nielsen F, Hoye TT (2021b) An Automated Light Trap to Monitor Moths (Lepidoptera) Using Computer Vision-Based Tracking and Deep Learning. Sensors (Basel) 21. doi: 10.3390/s21020343

Brehm G (2017) A new LED lamp for the collection of nocturnal Lepidoptera and a spectral comparison of light-trapping lamps. Nota Lepidopterologica 40. doi: 10.3897/nl.40.11887

Brehm G et al. (2021) Moths are strongly attracted to ultraviolet and blue radiation. Insect Conservation and Diversity 14:188-198. doi: 10.1111/icad.12476

Bremmer J, van Engelen J (2006) Spray Painting. Paint Products Fact Sheet: To assess the risks for the consumer: Updated version for ConsExpo 4. National Institute for Public Health and the Environment, Bilthoven, The Netherlands

Brooks ME et al. (2017) glmmTMB Balances Speed and Flexibility Among Packages for Zero-inflated Generalized Linear Mixed Modeling. The R Journal 9. doi: 10.32614/rj-2017-066

Bullock JM, Drechsler P, Federle W (2008) Comparison of smooth and hairy attachment pads in insects: friction, adhesion and mechanisms for direction-dependence. J Exp Biol 211:3333-3343. doi: 10.1242/jeb.020941

Campbell JW, Hanula JL (2007) Efficiency of Malaise traps and colored pan traps for collecting flower visiting insects from three forested ecosystems. Journal of Insect Conservation 11:399-408. doi: 10.1007/s10841-006-9055-4

Chapman A (2009) Numbers of Living Species in Australia and the World. In: Department of the Environment, Water, Heritage and the Arts (ed), 2 edn. Australian Biodiversity Information Services, Toowoomba, Australia

Colwell RK, Coddington JA (1994) Estimating terrestrial biodiversity through extrapolation. Philos Trans R Soc Lond B Biol Sci 345:101-118. doi: 10.1098/rstb.1994.0091

Dirzo R, Young HS, Galetti M, Ceballos G, Isaac NJ, Collen B (2014) Defaunation in the Anthropocene. Science 345:401-406. doi: 10.1126/science.1251817

Dvornik N, Mairal J, Schmid C (2018) Modeling Visual Context is Key to Augmenting Object Detection Datasets. CoRR abs/1807.07428

Ferrenberg S, Mitton JB, Jones H (2014) Smooth bark surfaces can defend trees against insect attack: resurrecting a 'slippery' hypothesis. Functional Ecology 28:837-845. doi: 10.1111/1365-2435.12228

GBIF (2023) GBIF Home Page. GBIF.org

Gerovichev A, Sadeh A, Winter V, Bar-Massada A, Keasar T, Keasar C (2021) High Throughput Data Acquisition and Deep Learning for Insect Ecoinformatics. Frontiers in Ecology and Evolution 9. doi: 10.3389/fevo.2021.600931
Hallmann CA et al. (2017) More than 75 percent decline over 27 years in total flying insect biomass in protected areas. PLoS One 12:e0185809. doi: 10.1371/journal.pone.0185809
Harris JE, Rodenhouse NL, Holmes RT (2019) Decline in beetle abundance and diversity in an intact temperate forest linked to climate warming. Biological Conservation 240. doi: 10.1016/j.biocon.2019.108219
Hartig J (2022a) DHARMa: Residual Diagnostics for Hierarchical (Multi-Level / Mixed) Regression Models, R package version 0.4.6 edn
Hartig J (2022b) Package 'DHARMa'. https://cran.r-project.org/web/packages/DHARMa/DHARMa.pdf
Hogeweg L, Zeegers T, Katramados I, Jongejans E (2019) Smart Insect Cameras. Biodiversity Information Science and Standards 3. doi: 10.3897/biss.3.39241
Hoye TT et al. (2021) Deep learning and computer vision will transform entomology. Proc Natl Acad Sci U S A 118. doi: 10.1073/pnas.2002545117
Ide J-Y (2022) Why do red/purple young leaves suffer less insect herbivory: tests of the warning signal hypothesis and the undermining of insect camouflage hypothesis. Arthropod-Plant Interactions 16:567-581. doi: 10.1007/s11829-022-09924-x
Kühn E et al. (2013) The European grassland butterfly indicator : 1990–2011. Publications Office
Lenth R (2023) emmeans: Estimated Marginal Means, aka Least-Squares Means. https://CRAN.R-project.org/package=emmeans
Leong JM, Thorp RW (1999) Colour-coded sampling: the pan trap colour preferences of oligolectic and nonoligolectic bees associated with a vernal pool plant. Ecological Entomology 24:329-335. doi: 10.1046/j.1365-2311.1999.00196.x
Losey JE, Vaughan M (2006) The Economic Value of Ecological Services Provided by Insects. BioScience 56. doi: 10.1641/0006-3568(2006)56[311:Tevoes]2.0.Co;2
Lüdecke D, Ben-Shachar M, Patil I, Waggoner P, Makowski D (2021) performance: An R Package for Assessment, Comparison and Testing of Statistical Models. Journal of Open Source Software 6. doi: 10.21105/joss.03139
Meglic A et al. (2019) Horsefly object-directed polarotaxis is mediated by a stochastically distributed ommatidial subtype in the ventral retina. Proc Natl Acad Sci U S A 116:21843-21853. doi: 10.1073/pnas.1910807116
Montgomery GA, Belitz MW, Guralnick RP, Tingley MW (2021) Standards and Best Practices for Monitoring and Benchmarking Insects. Frontiers in Ecology and Evolution 8. doi: 10.3389/fevo.2020.579193
Noskov A, Bendix J, Friess N (2021) A Review of Insect Monitoring Approaches with Special Reference to Radar Techniques. Sensors (Basel) 21. doi: 10.3390/s21041474
Olivier L (2021) DIOPSIS: Testing a new smart insect camera system. Radboud University, Nijmegen
Pawar S (2003) Taxonomic Chauvinism and the Methodologically Challenged. BioScience 53. doi: 10.1641/0006-3568(2003)053[0861:Tcatmc]2.0.Co;2
PLEXIGLAS Technical Information: PLEXIGLAS ® GS, UV transmitting
Preston FW (1948) The Commonness, And Rarity, of Species. Ecology 29:254-283. doi: 10.2307/1930989
R Core Team (2023) R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria
Radboud University (2023) Living Lab Ooijpolder. https://www.ru.nl/onderzoek/onderzoeksprojecten/living-lab-ooijpolder
Rhodes MW, Bennie JJ, Spalding A, Ffrench-Constant RH, Maclean IMD (2022) Recent advances in the remote sensing of insects. Biol Rev Camb Philos Soc 97:343-360. doi: 10.1111/brv.12802

Roe P et al. (2021) The Australian Acoustic Observatory. Methods in Ecology and Evolution 12:1802-1808. doi: 10.1111/2041-210x.13660
Rubin M (1985) Refractive index of Soda lime glass. https://refractiveindex.info/?shelf=glass&book=soda-lime&page=Rubin-clear
Scholz I et al. (2010) Slippery surfaces of pitcher plants: Nepenthes wax crystals minimize insect attachment via microscopic surface roughness. J Exp Biol 213:1115-1125. doi: 10.1242/jeb.035618
Seibold S et al. (2021) The contribution of insects to global forest deadwood decomposition. Nature 597:77-81. doi: 10.1038/s41586-021-03740-8
Sunrise Sunset (2023) Sunrise Sunset API. https://sunrise-sunset.org/api
Szczurowski M (2013) Refractive index of (C5H8O2)n (Poly(methyl methacrylate), PMMA). https://refractiveindex.info/?shelf=organic&book=poly%28methyl_methacrylate%29&page=Szczurowski
Toler TR, Evans EW, Tepedino VJ (2005) Pan-trapping for bees (Hymenoptera : Apiformes) in Utah's West Desert: the importance of color diversity. Pan-Pacific Entomol. 81:103-113
Ushimaru A, Hyodo F (2005) Why do bilateral symmetrical flowers orient vertically? Flower orientation influences pollinator landing behavior. Evolutionary Ecology Research 7:151–160
van der Kooi CJ, Dyer AG, Kevan PG, Lunau K (2019) Functional significance of the optical properties of flowers for visual signalling. Ann Bot 123:263-276. doi: 10.1093/aob/mcy119
van der Sluijs JP (2020) Insect decline, an emerging global environmental risk. Current Opinion in Environmental Sustainability 46:39-42. doi: 10.1016/j.cosust.2020.08.012
van Klink R et al. (2022) Emerging technologies revolutionise insect ecology and monitoring. Trends Ecol Evol 37:872-885. doi: 10.1016/j.tree.2022.06.001
van Lieshout T (2020) Automation of the analysis of insect camera data, Hogeschool van Arnhem en Nijmegen
van Strien AJ, van Swaay CAM, van Strien-van Liempt WTFH, Poot MJM, Wallis de Vries MF (2019) Over a century of data reveal more than 80% decline in butterflies in the Netherlands. Biological Conservation 234:116-122. doi: 10.1016/j.biocon.2019.03.023
Wickham H (2016) ggplot2: Elegant Graphics for Data Analysis. https://ggplot2.tidyverse.org/
Yalcin H (2015) Vision based automatic inspection of insects in pheromone traps 2015 Fourth International Conference on Agro-Geoinformatics (Agro-geoinformatics), pp 333-338
Zhou Y, Robinson A, Steiner U, Federle W (2014) Insect adhesion on rough surfaces: analysis of adhesive contact of smooth and hairy pads on transparent microstructured substrates. J R Soc Interface 11:20140499. doi: 10.1098/rsif.2014.0499

Appendix 1: Products used for the construction of the new screens and their corresponding webpages

Product	Product page
Yellow (RAL 1023) foamed PVC plates (v1 screen)	https://kunststofplatenshop.nl/product/geschuimd-pvc-geel-3-mm/
White (RAL 9003) foamed PVC plates (new screen base)	https://kunststofplatenshop.nl/product/geschuimd-pvc-wit-3-mm/
UV permeable plexiglass plates	https://www.plexiglas.nl/product/plexiglas-uv-doorlatend-3-mm
Sparvar fluorescent spray paint	https://euro-industry.com/main.php?index=details&cat=326&prod=23196965
Waterproof tape (3M Scotch Tape 389)	https://www.3mnederland.nl/3M/nl_NL/p/d/b40065986/

Appendix 2: Objects with different textures spray painted with red paint





When possible, an unpainted piece of the same object is placed above the painted object for a comparison. From left to right, top to bottom, the objects are:
A piece of white plastic
A piece of photo paper
A piece of white paper
A piece of white cardboard
A piece of carboard covered with clear tape
A piece of cardboard
A screen protector for a mobile phone (no unpainted version available)

Not only was the microstructure unlikely to change, since all objects were covered with the same paint particles, but there were also color differences visible between objects that appeared to be the same color before painting (e.g. object 2 and 3). Furthermore, painting an extremely smooth surface, such as that of the screen protector, clearly shows that the paint itself reacts differently to different surfaces. Instead of using the method described here, I chose to differentiate between a rough and a smooth surface by placing a plexiglass plate over screens made out of foamed PVC.

Appendix 3: R packages used in this research

Package name	Used for	Reference
cowplot	Stitching graphs together into one figure	Wilke C (2020). *cowplot: Streamlined Plot Theme and Plot Annotations for 'ggplot2'*. R package version 1.1.1, https://CRAN.R-project.org/package=cowplot.
DHARMa	Performing validation checks on my models	Hartig F (2022). *DHARMa: Residual Diagnostics for Hierarchical (Multi-Level / Mixed) Regression Models*. R package version 0.4.6, https://CRAN.R-project.org/package=DHARMa.
emmeans	Calculating the estimated marginal mean values of the models	Lenth R (2023). *emmeans: Estimated Marginal Means, aka Least-Squares Means*. R package version 1.9.0, https://CRAN.R-project.org/package=emmeans.
ggh4x	Controlling the axes of facetted graphs more precisely	van den Brand T (2023). *ggh4x: Hacks for 'ggplot2'*. R package version 0.2.6, https://CRAN.R-project.org/package=ggh4x.
ggmap	Requesting map tiles in order to create a map of the Ooijpolder	D. Kahle and H. Wickham. ggmap: Spatial Visualization with ggplot2. The R Journal, 5(1), 144-161. URL http://journal.r-project.org/archive/2013-1/kahle-wickham.pdf
ggplot2	Creating figures	H. Wickham. ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York, 2016.
ggsn	Adding a "north" symbol to the Ooijpolder map	Santos Baquero O (2019). *ggsn: North Symbols and Scale Bars for Maps Created with 'ggplot2' or 'ggmap'*. R package version 0.5.0, https://CRAN.R-project.org/package=ggsn.
ggtext	Adding HTML elements to my figures	Wilke C, Wiernik B (2022). *ggtext: Improved Text Rendering Support for 'ggplot2'*. R package version 0.1.2, https://CRAN.R-project.org/package=ggtext.
glmmTMB	Creating my models	Mollie E. Brooks, Kasper Kristensen, Koen J. van Benthem, Arni Magnusson, Casper W. Berg, Anders Nielsen, Hans J. Skaug, Martin Maechler and Benjamin M. Bolker (2017). glmmTMB Balances Speed and Flexibility Among Packages for Zero-inflated Generalized Linear Mixed Modeling. The R Journal, 9(2), 378-400. doi: 10.32614/RJ-2017-066.
htmlTable	Creating a table of the screen rotation schedules in HTML	Gordon M, Gragg S, Konings P (2022). *htmlTable: Advanced Tables for Markdown/HTML*. R package version 2.4.1, https://CRAN.R-project.org/package=htmlTable.
htmltools	Saving the HTML table	Cheng J, Sievert C, Schloerke B, Chang W, Xie Y, Allen J (2023). *htmltools: Tools for HTML*. R

		package version 0.5.6.1, https://CRAN.R-project.org/package=htmltools.
httr	Making API requests in order to receive species data and sunrise/sunset data	Wickham H (2023). *httr: Tools for Working with URLs and HTTP*. R package version 1.4.7, https://CRAN.R-project.org/package=httr.
jsonlite	Handling the output of the API requests	Ooms J (2014). "The jsonlite Package: A Practical and Consistent Mapping Between JSON Data and R Objects." https://arxiv.org/abs/1403.2805.
lubridate	Handling dates	Garrett Grolemund, Hadley Wickham (2011). Dates and Times Made Easy with lubridate. Journal of Statistical Software, 40(3), 1-25. URL https://www.jstatsoft.org/v40/i03/.
plyr	General use within my code	Hadley Wickham (2011). The Split-Apply-Combine Strategy for Data Analysis. Journal of Statistical Software, 40(1), 1-29. URL https://www.jstatsoft.org/v40/i01/.
png	Loading in a PNG of an aerial view of the Hortus Botanicus	Urbanek S (2022). *png: Read and write PNG images*. R package version 0.1-8, https://CRAN.R-project.org/package=png.
tidyverse	General use within my code	Wickham H, Averick M, Bryan J, Chang W, McGowan LD, François R, Grolemund G, Hayes A, Henry L, Hester J, Kuhn M, Pedersen TL, Miller E, Bache SM, Müller K, Ooms J, Robinson D, Seidel DP, Spinu V, Takahashi K, Vaughan D, Wilke C, Woo K, Yutani H (2019). "Welcome to the tidyverse." *Journal of Open Source Software*, 4(43), 1686. doi:10.21105/joss.01686.
performance	Calculating the R^2-values of my models	Lüdecke et al., (2021). performance: An R Package for Assessment, Comparison and Testing of Statistical Models. Journal of Open Source Software, 6(60), 3139. https://doi.org/10.21105/joss.03139

Appendix 4: Residual tests and corresponding p-values of the models

Model	**Kolmogorov-Smirnov**	**Dispersion of residuals**	**Residuals outliers**
Ooijpolder - Insects	1	0.244	0.16
Ooijpolder - Diptera	0.788	0.024	0.5
Ooijpolder - Hymenoptera	0.128	0.331	0.96
Ooijpolder - Lepidoptera	0.424	0	0.64
Hortus - Insects	0.976	0.414	0.44
Hortus - Diptera	0.48	0.414	0.44
Hortus - Hymenoptera	0.524	0.101	1
Hortus - Lepidoptera	0.192	0.325	0.36

Appendix 5: Rotation schedules | The upper schedule shows that of the Ooijpolder and the bottom schedule that of the Hortus Botanicus



Appendix 6: R code

Script 1: Hortus schedule generation

```
library(htmlTable)
library(lubridate)
library(htmltools)

# Add screen colors of new screens
screenColors <- c("white", "blue", "yellow")

# Add possible textures of new screens
screenTextures <- c("smooth", "rough")

# Add orientations
screenOrientations <- c("vertical", "horizontal")

# Add presence of attractant
attractantPresence <- c("attractant", "no attractant")

# Add screen versions
screenVersion <- c("v1 screen", "v2 screen")

# Provide the number of cameras
nCameras <- 6

# Add the start date. The start date can be in a weekend, but subsequent dates will
automatically be scheduled to take place during weekdays
startDate <- as.Date("2023-04-24", "%Y-%m-%d")
# Add the length of the interval between visits in days
interval_days <- 2

set.seed(20230508)



# Create a vector to hold all treatments:
# Create vectors with treatments for new screens
combinationsNewScreens1 <- c()
for (i in c(1:length(screenColors))) {
  combinationsNewScreens1 <- append(combinationsNewScreens1, paste(screenColors[i],
screenTextures[2], attractantPresence[2], screenOrientations[1]))
}

combinationsNewScreens2 <- c()
for (i in c(1:length(screenColors))) {
  combinationsNewScreens2 <- append(combinationsNewScreens2, paste(screenColors[i],
screenTextures[1], attractantPresence[2], screenOrientations[1]))
}
```

```
combinationsNewScreens3 <- c()
for (i in c(1:length(screenColors))) {
  combinationsNewScreens3 <- append(combinationsNewScreens3, paste(screenColors[i],
screenTextures[2], attractantPresence[1], screenOrientations[1]))
}

combinationsNewScreens4 <- c()
for (i in c(1:length(screenTextures))) {
  combinationsNewScreens4 <- append(combinationsNewScreens4, paste(screenColors[3],
screenTextures[i], attractantPresence[2], screenOrientations[2]))
}

# Create vector with treatments for the old (v1 and v2) screens
combinationsOldScreens <- c()
for (i in c(1:length(screenVersion))) {
  combinationsOldScreens <- append(combinationsOldScreens, paste(screenVersion[i],
attractantPresence[2], screenOrientations[1]))
}


# Merge the vectors into one vector that holds all combinations
possibleCombinations <- c(combinationsNewScreens1, combinationsNewScreens2,
combinationsNewScreens3, combinationsNewScreens4, combinationsOldScreens)


# Create a vector to hold the individual screens
individualScreens <- c()
# 3 times 3 colors:
# - 1 set rough surface without attractant
# - 1 set smooth surface without attractant
# - 1 set rough surface with attractant
# For consistency 1 time v1 screen and 1 time v2 screen
for (i in c(1:length(screenColors))) {
  individualScreens <- append(individualScreens, paste(screenColors[i],
screenTextures[2], attractantPresence[2]))
}
for (i in c(1:length(screenColors))) {
  individualScreens <- append(individualScreens, paste(screenColors[i],
screenTextures[1], attractantPresence[2]))
}
for (i in c(1:length(screenColors))) {
  individualScreens <- append(individualScreens, paste(screenColors[i],
screenTextures[2], attractantPresence[1]))
}
for (i in c(1:length(screenVersion))) {
  individualScreens <- append(individualScreens, paste(screenVersion[i],
attractantPresence[2]))
}
# The screens in this vector can be used once per timeslot
```

```
# Create a data frame to hold the schedule
schedule <- data.frame(matrix(NA, nCameras, length(possibleCombinations)))
colnames(schedule) <- paste("Timeslot", c(1:ncol(schedule)))
rownames(schedule) <- paste("Camera", c(1:nrow(schedule)))


# Since the attractant was not yet available for use at the beginning of the
experiment, timeslot 1 to 5 did not yet include any treatments with the attractant.
Instead, the treatments were randomly chosen in using the method that is employed
in this R script, but without the attractant-treatments as an option. This was
repeated for all 5 timeslots, until the attractant became available. This is why
these timeslots need to be added manually. Additionally, the treatments were
randomly selected for the sixth timeslot; these included treatments with the
attractant. However, since a seed was not yet set during that process, the result
is not replicable. Therefore, timeslot 6 also needed to be added manually:
# Create a data frame to hold the treatments used in timeslot 1 to 6
timeslot1to6_treatments <- data.frame(matrix(NA, nCameras, 6))
colnames(timeslot1to6_treatments) <- c("Timeslot1", "Timeslot2", "Timeslot3",
"Timeslot4", "Timeslot5", "Timeslot6")
rownames(timeslot1to6_treatments) <- paste("camera",
c(1:nrow(timeslot1to6_treatments)))

# Create a vector to hold the treatments used in timeslot 1
timeslot1to6_treatments$Timeslot1 <- c("blue smooth no attractant", "yellow rough
no attractant", "v1 screen no attractant", "yellow smooth no attractant", "v2
screen no attractant", "blue rough no attractant")
# Create a vector to hold the treatments used in timeslot 2
timeslot1to6_treatments$Timeslot2 <- c("blue rough no attractant", "yellow smooth
no attractant", "yellow rough no attractant", "white rough no attractant", "blue
smooth no attractant", "v2 screen no attractant")
# Create a vector to hold the treatments used in timeslot 3
timeslot1to6_treatments$Timeslot3 <- c("v2 screen no attractant", "blue smooth no
attractant", "white smooth no attractant", "yellow smooth no attractant", "yellow
rough no attractant", "white rough no attractant")
# Create a vector to hold the treatments used in timeslot 4
timeslot1to6_treatments$Timeslot4 <- c("yellow smooth no attractant", "white smooth
no attractant", "blue rough no attractant", "v1 screen no attractant", "white rough
no attractant", "yellow rough no attractant")
# Create a vector to hold the treatments used in timeslot 5
timeslot1to6_treatments$Timeslot5 <- c("yellow rough no attractant", "v1 screen no
attractant", "blue smooth no attractant", "v2 screen no attractant", "yellow smooth
no attractant", "white smooth no attractant")
# Create a vector to hold the treatments used in timeslot 6
timeslot1to6_treatments$Timeslot6 <- c("yellow rough no attractant", "blue rough no
attractant", "white rough attractant", "yellow rough attractant", "v1 screen no
attractant", "yellow smooth no attractant")
```

```
# Fill in the empty schedule with the screen combinations
# As long as redo is TRUE, the code reruns itself
redo <- TRUE
while (redo) {
  # Create pools for every slot and camera (row and column) in the form of vectors.
  for (i in c(7:ncol(schedule))) {
    assign(paste0("slotPool", i), individualScreens)
  }


  for (i in c(1:nCameras)) {
    for(j in c(1:6)) {
      if (j == 1) {
      assign(paste0("cameraPool", i), c(individualScreens, paste(screenColors[3],
screenTextures[1], attractantPresence[2]), paste(screenColors[3],
screenTextures[2], attractantPresence[2]))[-match(timeslot1to6_treatments[i, j],
c(individualScreens, paste(screenColors[3], screenTextures[1],
attractantPresence[2]), paste(screenColors[3], screenTextures[2],
attractantPresence[2])))])
      } else {
        assign(paste0("cameraPool", i), eval(parse(text = paste0("cameraPool",
i)))[-match(timeslot1to6_treatments[i, j], eval(parse(text = paste0("cameraPool",
i))))])
    }
  }
}

  schedule[, 1] <- timeslot1to6_treatments[, 1]

  schedule[, 2] <- timeslot1to6_treatments[, 2]

  schedule[, 3] <- timeslot1to6_treatments[, 3]

  schedule[, 4] <- timeslot1to6_treatments[, 4]

  schedule[, 5] <- timeslot1to6_treatments[, 5]

  schedule[, 6] <- timeslot1to6_treatments[, 6]

  # Set proceed to TRUE, as this will determine whether the sampling can proceed,
or if the run has already failed
  proceed <- TRUE
  for (i in c(1:nCameras)) {
    for (j in c(7:ncol(schedule))) {
      # For every cell, check whether the two pools (slot and camera) both still
have combinations in common, and whether proceed is still TRUE.
```

```
      if (sum(eval(parse(text = paste0("cameraPool", i))) %in% eval(parse(text =
paste0("slotPool", j)))) > 0 && proceed) {
        # Create a temporary pool which contains those combinations that both pools
have in common
        pool <- intersect(eval(parse(text = paste0("cameraPool", i))),
eval(parse(text = paste0("slotPool", j))))
        # Sample a random combination from this temporary pool
        setupCombo <- sample(pool, 1)
        # Place this combination in the current cell
        schedule[i, j] <- setupCombo
        # Remove this combination from the original two pools, so that it cannot be
used again as an option
        assign(paste0("slotPool", j), eval(parse(text = paste0("slotPool", j)))[-
match(setupCombo, eval(parse(text = paste0("slotPool", j))))])
        assign(paste0("cameraPool", i), eval(parse(text = paste0("cameraPool",
i")))[-match(setupCombo, eval(parse(text = paste0("cameraPool", i))))])

        # If the two pools of a cell no longer have a combination in common, the
run has failed
      } else {
        # mark every cell as "failed", and redirect every cycle of the for-loop
towards this section of the if loop by marking proceed as FALSE
        schedule[i, j] <- "failed"
        proceed <- FALSE
      }
    }
  }
  # Check whether the run has failed by looking at the last cell
  if (schedule[nCameras, ncol(schedule)] == "failed") {
    # Redo the entire run if the run has failed
    redo <- TRUE
  } else {
    # Keep the current schedule if the run was successful
    redo <- FALSE
    # Remove the pools that you created from the environment
    for (i in c(1:nCameras)) {
      rm(list = paste0("cameraPool", i))
    }
    for (i in c(7:ncol(schedule))) {
      rm(list = paste0("slotPool", i))
    }
  }
}



# # This code is used for checking whether the previous code worked correctly
(which it does, so I left this in as a comment)
```

```
# # This code can only be run correctly at this point in the script
# for (i in c(1:nCameras)) {
#   if (sum(sort(as.character(as.vector(schedule[i,])))) ==
sort(c(individualScreens, paste(screenVersion[1], attractantPresence[2]),
paste(screenVersion[2], attractantPresence[2])))) == ncol(schedule)) {
#     print(paste("Camera", i, "receives all treatments"))
#   } else {
#     print(paste("Camera", i, "does not receive all treatments"))
#   }
# }
#
# maxOccurences <- aggregate(individualScreens, list(num=individualScreens),
length)
#
# for (i in c(1:ncol(schedule))) {
#   k <- 0
#   for (j in maxOccurences$num) {
#     countsColumn <- aggregate(schedule[, 1], list(num=schedule[, 1]), length)
#     if (j %in% countsColumn$num) {
#       if (countsColumn[countsColumn$num == j, ][, 2] <=
maxOccurences[maxOccurences$num == j, ][, 2]) {
#       k <- k + countsColumn[countsColumn$num == j, ][, 2]
#       }
#     }
#   }
#   if (k == nCameras) {
#     print(paste("This treatment is possible for timeslot", i))
#   }
# }

# Create a data frame to hold the orientations employed in the treatments of
timeslot 1 to 6. This needs to be done seperately from the rest of the treatment
(i.e. color, texture, attractant):
timeslot1to6_orientations <- data.frame(matrix(NA, nCameras, 6))
colnames(timeslot1to6_orientations) <- c("Timeslot1", "Timeslot2", "Timeslot3",
"Timeslot4", "Timeslot5", "Timeslot6")
timeslot1to6_orientations$Timeslot1 <- rep("vertical", 6)
timeslot1to6_orientations$Timeslot2 <- c("vertical", "horizontal", rep("vertical",
4))
timeslot1to6_orientations$Timeslot3 <- c(rep("vertical", 3), "horizontal",
rep("vertical", 2))
timeslot1to6_orientations$Timeslot4 <- rep("vertical", 6)
timeslot1to6_orientations$Timeslot5 <- rep("vertical", 6)
timeslot1to6_orientations$Timeslot6 <- c("horizontal", rep("vertical", 5))



# Add the screen orientations to the schedule. This process takes into account the
orientation used in timeslot 1 to 6
```

```
for (i in c(1:nrow(schedule))) {
  # Create two pools: one for adding orientation variations to the yellow rough no
attractant-screen, and one for the yellow smooth no attractant-screen
  orientationPool1 <- screenOrientations
  orientationPool2 <- screenOrientations
  for (j in c(1:ncol(schedule))) {
    # Add a randomly sampled orientation from pool 1 if the screen is yellow rough
no attractant:
    if (schedule[i, j] == "yellow rough no attractant") {
      if (j == 1 || j == 2 || j == 3 || j == 4 || j == 5 || j == 6) {
        schedule[i, j] <- paste(schedule[i, j], timeslot1to6_orientations[i, j])
        orientationPool1 <- orientationPool1[!orientationPool1 %in%
timeslot1to6_orientations[i, j]]
      } else {
      # Sample an orientation randomly and temporarily hold this sample
      currentSample <- sample(orientationPool1, 1)
      # Add the value of the sample to the value of the current cell
      schedule[i, j] <- paste(schedule[i, j], currentSample)
      # Remove the value of the sample from the pool so that it cannot be used
again
      orientationPool1 <- orientationPool1[!orientationPool1 %in% currentSample]
      }
      # Do the same as described above for the yellow smooth no attractant screens
    } else if (schedule[i, j] == "yellow smooth no attractant") {
      if (j == 1 || j == 2 || j == 3 || j == 4 || j == 5 || j == 6) {
        schedule[i, j] <- paste(schedule[i, j], timeslot1to6_orientations[i, j])
        orientationPool2 <- orientationPool2[!orientationPool2 %in%
timeslot1to6_orientations[i, j]]
        } else {
      currentSample <- sample(orientationPool2, 1)
      schedule[i, j] <- paste(schedule[i, j], currentSample)
      orientationPool2 <- orientationPool2[!orientationPool2 %in% currentSample]
      }
      # If the value of the current cell is anything else, add "vertical" to the
value
    } else {
      schedule[i, j] <- paste(schedule[i, j], "vertical")
    }
  }
}
# Remove the pools that you created from the environment
for (i in c(1:length(screenOrientations))) {
  rm(list = paste0("orientationPool", i))
}


# Add the treatments that were repeated due to camera failure
schedule$'Timeslot 14' <- c("blue rough attractant vertical",
                            "yellow rough no attractant vertical",
```

```
                              "yellow smooth no attractant horizontal",
                              "v2 screen no attractant vertical",
                              "v1 screen no attractant vertical",
                              "yellow rough attractant vertical")
schedule$'Timeslot 15' <- c("white rough no attractant vertical",
                              "yellow rough attractant vertical",
                              "yellow rough no attractant horizontal",
                              "white rough no attractant vertical",
                              "yellow smooth no attractant vertical",
                              "v1 screen no attractant vertical")



# Create a new data frame which will be used for adding dates to the column names
scheduleFinal <- schedule


# Change the first column name to the start date defined at the beginning of this
script
names(scheduleFinal)[1] <- paste(wday(startDate, week_start = 1, label = TRUE, abbr
= FALSE), format(startDate, "%d-%m-%Y"))
# At the start, assign a value of 0 the elapsedDays
elapsedDays <- 0
# One by one, change each column name into a date based on the start date, the
interval length, and the fact that no activities will take place during the weekend
(Saturday and Sunday):
for (i in c(2:ncol(scheduleFinal))) {
  # First check whether adding the number of elapsed days and the interval length
results in a date that takes place in the weekend
  if (wday(startDate + elapsedDays + interval_days, week_start = 1) <= 5) {
    # If the date will take place on a weekday, sum the interval length with the
number of elapsed days in order to update the number of elapsed days
    elapsedDays <- elapsedDays + interval_days
    # Sum the elapsed days with the start date and change the column name into this
newly calculated date
    names(scheduleFinal)[i] <- paste(wday(startDate + elapsedDays, week_start = 1,
label = TRUE, abbr = FALSE), format(startDate + elapsedDays, "%d-%m-%Y"))
    # If the date would not be on a weekday, check on which day in the weekend it
will take place:
  } else if (wday(startDate + elapsedDays + interval_days, week_start = 1) >= 6) {
    # Repeat the same steps as above with the addition of extra days based on
whether the date would have been a Saturday (+2 days) or a Sunday (+1 day)
    elapsedDays <- elapsedDays + interval_days +
      if (wday(startDate + elapsedDays + interval_days, week_start = 1) == 6) {
        2
      } else {
        1
      }
```

```
    names(scheduleFinal)[i] <- paste(wday(startDate + elapsedDays, week_start = 1,
label = TRUE, abbr = FALSE), format(startDate + elapsedDays, "%d-%m-%Y"))
  }
}

rownames(scheduleFinal) <- c("Camera 243", "Camera 259", "Camera 304", "Camera
320", "Camera 329", "Camera 367")

# A mistake was made during the execution of the experiment which led some
treatments being flipped. This has to be corrected in the "scheduleFinal"-data
frame
scheduleFinal[1, 9] <- "blue rough no attractant vertical"
scheduleFinal[5, 9] <- "blue rough attractant vertical"
scheduleFinal[5, 10] <- "blue rough no attractant vertical"

write.csv2(scheduleFinal, "output/tables/scheduleFinal_Hortus.csv")



scheduleFinalNW <- scheduleFinal

names(scheduleFinalNW)[1] <- format(startDate, "%d-%m-%y")
# At the start, assign a value of 0 the elapsedDays
elapsedDays <- 0
# One by one, change each column name into a date based on the start date, the
interval length, and the fact that no activities will take place during the weekend
(Saturday and Sunday):
for (i in c(2:ncol(scheduleFinalNW))) {
  # First check whether adding the number of elapsed days and the interval length
results in a date that takes place in the weekend
  if (wday(startDate + elapsedDays + interval_days, week_start = 1) <= 5) {
    # If the date will take place on a weekday, sum the interval length with the
number of elapsed days in order to update the number of elapsed days
    elapsedDays <- elapsedDays + interval_days
    # Sum the elapsed days with the start date and change the column name into this
newly calculated date
    names(scheduleFinalNW)[i] <- format(startDate + elapsedDays, "%d-%m-%y")
    # If the date would not be on a weekday, check on which day in the weekend it
will take place:
  } else if (wday(startDate + elapsedDays + interval_days, week_start = 1) >= 6) {
    # Repeat the same steps as above with the addition of extra days based on
whether the date would have been a Saturday (+2 days) or a Sunday (+1 day)
    elapsedDays <- elapsedDays + interval_days +
      if (wday(startDate + elapsedDays + interval_days, week_start = 1) == 6) {
        2
      } else {
        1
      }
```

```
    names(scheduleFinalNW)[i] <- format(startDate + elapsedDays, "%d-%m-%y")
  }
}


# Create an empty matrix that will be used to hold all the cell locations (row and
column) of the schedule
cells <- matrix('', nrow(scheduleFinalNW) * ncol(scheduleFinalNW), 2)
# The following for-loop will add the cell locations to the "cells"-matrix
counter <- 1
for(i in c(1:ncol(scheduleFinalNW))) {
  for (j in c(1:nrow(scheduleFinalNW))) {
    cells[counter, ] <- c(j, i)
    counter <- counter + 1
  }
}
# Change the storage mode of the matrix to numeric instead of character
storage.mode(cells) <- "numeric"


# Create an empty matrix that will be used to hold the CSS code for the different
background colors/patterns
cellStyle <- matrix('', nrow(scheduleFinalNW) * ncol(scheduleFinalNW), 2)
# The following for-loop will fill the "cellStyle"-matrix based on the contents of
each corresponding "scheduleFinalNW"-cell. The second column of the "cellStyle"-
matrix is used to hold CSS code that will make the text legible if the cell has a
background pattern
counter <- 1
for(i in c(1:ncol(scheduleFinalNW))) {
  for (j in c(1:nrow(scheduleFinalNW))) {
    if (scheduleFinalNW[j, i] == "v1 screen no attractant vertical") {
      cellStyle[counter, 1] <- "background-color: #F7B500"
    } else if (scheduleFinalNW[j, i] == "v2 screen no attractant vertical") {
      cellStyle[counter, 1] <- "background-color: #F1DD39"
    } else if (scheduleFinalNW[j, i] == "yellow rough no attractant vertical") {
      cellStyle[counter, 1] <- "background-color: yellow; background-image:
repeating-linear-gradient(135deg, transparent,  transparent 5px, black 5px, black
10px)"
    } else if (scheduleFinalNW[j, i] == "yellow rough no attractant horizontal") {
      cellStyle[counter, 1] <- "background-color: yellow; background-image:
repeating-linear-gradient(135deg, transparent,  transparent 5px, black 5px, black
10px)"
      cellStyle[counter, 2] <- "background-color: yellow; border-radius: 2px"
    } else if (scheduleFinalNW[j, i] == "yellow smooth no attractant vertical") {
      cellStyle[counter, 1] <- "background-color: yellow"
    } else if (scheduleFinalNW[j, i] == "yellow smooth no attractant horizontal") {
      cellStyle[counter, 1] <- "background-color: yellow"
    } else if (scheduleFinalNW[j, i] == "blue rough no attractant vertical") {
```

```
      cellStyle[counter, 1] <- "background-color: DeepSkyBlue; background-image:
repeating-linear-gradient(135deg, transparent, transparent 5px, black 5px, black
10px)"
    } else if (scheduleFinalNW[j, i] == "blue smooth no attractant vertical") {
      cellStyle[counter, 1] <- "background-color: DeepSkyBlue"
    } else if (scheduleFinalNW[j, i] == "white rough no attractant vertical") {
      cellStyle[counter, 1] <- "background-color: white; background-image:
repeating-linear-gradient(135deg, transparent,  transparent 5px, black 5px, black
10px)"
    } else if (scheduleFinalNW[j, i] == "white smooth no attractant vertical") {
      cellStyle[counter, 1] <- "background-color: white"
    } else if (scheduleFinalNW[j, i] == "yellow rough attractant vertical") {
      cellStyle[counter, 1] <- "background-color: yellow; background-
image:  repeating-linear-gradient(45deg, black 25%, transparent 25%, transparent
75%, black 75%, black), repeating-linear-gradient(45deg, black 25%, yellow 25%,
yellow 75%, black 75%, black); background-position: 0 0, 10px 10px;background-size:
20px 20px;"
    } else if (scheduleFinalNW[j, i] == "blue rough attractant vertical") {
      cellStyle[counter, 1] <- "background-color: DeepSkyBlue; background-
image:  repeating-linear-gradient(45deg, black 25%, transparent 25%, transparent
75%, black 75%, black), repeating-linear-gradient(45deg, black 25%, DeepSkyBlue
25%, DeepSkyBlue 75%, black 75%, black); background-position: 0 0, 10px
10px;background-size: 20px 20px;"
    } else if (scheduleFinalNW[j, i] == "white rough attractant vertical") {
      cellStyle[counter, 1] <- "background-color: white; background-
image:  repeating-linear-gradient(45deg, black 25%, transparent 25%, transparent
75%, black 75%, black), repeating-linear-gradient(45deg, black 25%, white 25%,
white 75%, black 75%, black); background-position: 0 0, 10px 10px;background-size:
20px 20px;"
    }
    counter <- counter + 1
  }
}



# Create a empty data frame that will be used as the base of the HTML-table
emptySchedule <- data.frame(matrix("", nrow = nrow(scheduleFinalNW), ncol =
ncol(scheduleFinalNW)))
rownames(emptySchedule) <- rownames(scheduleFinalNW)
colnames(emptySchedule) <- colnames(scheduleFinalNW)

# Fill the "emptySchedule" data frame with the text that will appear in HTML-table
for (i in c(1:nrow(scheduleFinalNW))) {
  for (j in c(1:ncol(scheduleFinalNW))) {
    if (grepl("horizontal", scheduleFinalNW[i, j], fixed = TRUE)) {
      emptySchedule[i, j] <- "Horizontal"
    } else if (grepl("v1", scheduleFinalNW[i, j], fixed = TRUE)) {
      emptySchedule[i, j] <- "V1"
```

```
    } else if (grepl("v2", scheduleFinalNW[i, j], fixed = TRUE)) {
      emptySchedule[i, j] <- "V2"
    }
  }
}


# Create a function that can be used the place HTML code in a data frame based on
cell locations, and implements CSS code into the HTML code
inject_div <- function(x, where, style) {
  x[where] <- sprintf('<div style = \'%s\'> <span style = \'%s\' > %s </span>
</div>',
                      gsub(';*$', ';', style[, 1]),
                      gsub(";*$", ";", style[, 2]),
                      x[where])
  x
}


# Some CSS code is applicable to all the cells of the final HTML-table, and can
thus be added to the "cellStyle"-matrix with a single line of code:
cellStyle[, 1] <- paste(cellStyle[, 1], "; width: 90px; height: 35px; border-
radius: 6px; font-family: Sans-serif; name: Arial; padding-top: 15px; border: solid
1px", sep = "")

# Execute the inject_div-function, thereby filling the "emptySchedule"-data frame
with the HTML/CSS code
cellStyle_final <- inject_div(emptySchedule,
                              where = cells,
                              style = cellStyle)


# Using the htmlTable-command, create the HTML-table
htmlTable_Ooijpolder <- htmlTable(cellStyle_final,
                                  css.cell = append("font-family: Sans-serif; name:
Arial; max-width: 60px; min-width: 60px", rep("", ncol(scheduleFinalNW))),
                                  css.header = "font-family: Sans-serif; name:
Arial; max-width: 100px")


save_html(htmlTable_Ooijpolder, "output/tables/htmlTable_Hortus.html")
```

Script 2: Ooijpolder schedule generation

```
library(htmlTable)
library(lubridate)
library(htmltools)

# Add screen colors of new screens
screenColors <- c("white", "blue", "yellow")

# Add possible textures of new screens
screenTextures <- c("smooth", "rough")

# Add orientations
screenOrientations <- c("vertical", "horizontal")

# Add presence of attractant
attractantPresence <- c("attractant", "no attractant")

# Add screen versions
screenVersion <- c("v1 screen", "v2 screen")

# Provide the number of cameras
nCameras <- 6

# Add the start date. The start date can be in a weekend, but subsequent dates will
automatically be scheduled to take place during weekdays
startDate <- as.Date("2023-05-31", "%Y-%m-%d")
# Add the length of the interval between visits in days
interval_days <- 2

set.seed(20230530)



# Create a vector to hold all treatments:
# Create vectors with treatments for new screens
combinationsNewScreens1 <- c()
for (i in c(1:length(screenColors))) {
  combinationsNewScreens1 <- append(combinationsNewScreens1, paste(screenColors[i],
screenTextures[2], attractantPresence[2], screenOrientations[1]))
}

combinationsNewScreens2 <- c()
for (i in c(1:length(screenColors))) {
  combinationsNewScreens2 <- append(combinationsNewScreens2, paste(screenColors[i],
screenTextures[1], attractantPresence[2], screenOrientations[1]))
}

combinationsNewScreens3 <- c()
for (i in c(1:length(screenColors))) {
```

```
  combinationsNewScreens3 <- append(combinationsNewScreens3, paste(screenColors[i],
screenTextures[2], attractantPresence[1], screenOrientations[1]))
}

combinationsNewScreens4 <- c()
for (i in c(1:length(screenTextures))) {
  combinationsNewScreens4 <- append(combinationsNewScreens4, paste(screenColors[3],
screenTextures[i], attractantPresence[2], screenOrientations[2]))
}

# Create vector with treatments for the old (v1 and v2) screens
combinationsOldScreens <- c()
for (i in c(1:length(screenVersion))) {
  combinationsOldScreens <- append(combinationsOldScreens, paste(screenVersion[i],
attractantPresence[2], screenOrientations[1]))
}


# Merge the vectors into one vector that holds all combinations
possibleCombinations <- c(combinationsNewScreens1, combinationsNewScreens2,
combinationsNewScreens3, combinationsNewScreens4, combinationsOldScreens)


# Create a vector to hold the individual screens
individualScreens <- c()
# 3 times 3 colors:
# - 1 set rough surface without attractant
# - 1 set smooth surface without attractant
# - 1 set rough surface with attractant
# For consistency 1 time v1 screen and 1 time v2 screen
for (i in c(1:length(screenColors))) {
  individualScreens <- append(individualScreens, paste(screenColors[i],
screenTextures[2], attractantPresence[2]))
}
for (i in c(1:length(screenColors))) {
  individualScreens <- append(individualScreens, paste(screenColors[i],
screenTextures[1], attractantPresence[2]))
}
for (i in c(1:length(screenColors))) {
  individualScreens <- append(individualScreens, paste(screenColors[i],
screenTextures[2], attractantPresence[1]))
}
for (i in c(1:length(screenVersion))) {
  individualScreens <- append(individualScreens, paste(screenVersion[i],
attractantPresence[2]))
}
# The screens in this vector can be used once per timeslot
```

```
# Create a data frame to hold the schedule
schedule <- data.frame(matrix(NA, nCameras, length(possibleCombinations)))
colnames(schedule) <- paste("Timeslot", c(1:ncol(schedule)))
rownames(schedule) <- paste("Camera", c(1:nrow(schedule)))


# Fill in the empty schedule with the screen combinations
# As long as redo is TRUE, the code reruns itself
redo <- TRUE
while (redo) {
  # Create pools for every slot and camera (row and column) in the form of vectors.
  for (i in c(1:ncol(schedule))) {
    assign(paste0("slotPool", i), individualScreens)
  }

  for (i in c(1:nCameras)) {
    assign(paste0("cameraPool", i), c(individualScreens, paste(screenColors[3],
screenTextures[1], attractantPresence[2]), paste(screenColors[3],
screenTextures[2], attractantPresence[2])))
  }

  # Set proceed to TRUE, as this will determine whether the sampling can proceed,
or if the run has already failed
  proceed <- TRUE
  for (i in c(1:nCameras)) {
    for (j in c(1:ncol(schedule))) {
      # For every cell, check whether the two pools (slot and camera) both still
have combinations in common, and whether proceed is still TRUE.
      if (sum(eval(parse(text = paste0("cameraPool", i))) %in% eval(parse(text =
paste0("slotPool", j)))) > 0 && proceed) {
        # Create a temporary pool which contains those combinations that both pools
have in common
        pool <- intersect(eval(parse(text = paste0("cameraPool", i))),
eval(parse(text = paste0("slotPool", j))))
        # Sample a random combination from this temporary pool
        setupCombo <- sample(pool, 1)
        # Place this combination in the current cell
        schedule[i, j] <- setupCombo
        # Remove this combination from the original two pools, so that it cannot be
used again as an option
        assign(paste0("slotPool", j), eval(parse(text = paste0("slotPool", j)))[-
match(setupCombo, eval(parse(text = paste0("slotPool", j))))])
        assign(paste0("cameraPool", i), eval(parse(text = paste0("cameraPool",
i)))[-match(setupCombo, eval(parse(text = paste0("cameraPool", i))))])

        # If the two pools of a cell no longer have a combination in common, the
run has failed
      } else {
```

```
        # mark every cell as "failed", and redirect every cycle of the for-loop
towards this section of the if loop by marking proceed as FALSE
        schedule[i, j] <- "failed"
        proceed <- FALSE
      }
    }
  }
  # Check whether the run has failed by looking at the last cell
  if (schedule[nCameras, ncol(schedule)] == "failed") {
    # Redo the entire run if the run has failed
    redo <- TRUE
  } else {
    # Keep the current schedule if the run was successful
    redo <- FALSE
    # Remove the pools that you created from the environment
    for (i in c(1:nCameras)) {
      rm(list = paste0("cameraPool", i))
    }
    for (i in c(1:ncol(schedule))) {
      rm(list = paste0("slotPool", i))
    }
  }
}




# # This code is used for checking whether the previous code worked correctly
(which it does, so I left this in as a comment)
# # This code can only be run correctly at this point in the script
# for (i in c(1:nCameras)) {
#   if (sum(sort(as.character(as.vector(schedule[i,])))) ==
sort(c(individualScreens, paste(screenColors[3], screenTextures[1],
attractantPresence[2]), paste(screenColors[3], screenTextures[2],
attractantPresence[2])))) == ncol(schedule)) {
#     print(paste("Camera", i, "receives all treatments"))
#   } else {
#     print(paste("Camera", i, "does not receive all treatments"))
#   }
# }
#
# maxOccurences <- aggregate(individualScreens, list(num=individualScreens),
length)
#
# for (i in c(1:ncol(schedule))) {
#   k <- 0
#   for (j in maxOccurences$num) {
#     countsColumn <- aggregate(schedule[, 1], list(num=schedule[, 1]), length)
#     if (j %in% countsColumn$num) {
```

```
#       if (countsColumn[countsColumn$num == j, ][, 2] <=
maxOccurences[maxOccurences$num == j, ][, 2]) {
#       k <- k + countsColumn[countsColumn$num == j, ][, 2]
#       }
#     }
#   }
#   if (k == nCameras) {
#     print(paste("This treatment is possible for timeslot", i))
#   }
# }



# Add the screen orientations to the schedule
for (i in c(1:nrow(schedule))) {
  # Create two pools: one for adding orientation variations to the yellow rough no
attractant-screen, and one for the yellow smooth no attractant-screen
  orientationPool1 <- screenOrientations
  orientationPool2 <- screenOrientations
  for (j in c(1:ncol(schedule))) {
    # Add a randomly sampled orientation from pool 1 if the screen is yellow rough
no attractant:
    if (schedule[i, j] == "yellow rough no attractant") {
      # Sample an orientation randomly and temporarily hold this sample
      currentSample <- sample(orientationPool1, 1)
      # Add the value of the sample to the value of the current cell
      schedule[i, j] <- paste(schedule[i, j], currentSample)
      # Remove the value of the sample from the pool so that it cannot be used
again
      orientationPool1 <- orientationPool1[!orientationPool1 %in% currentSample]
      # Do the same as described above for the yellow smooth no attractant screens
    } else if (schedule[i, j] == "yellow smooth no attractant") {
      currentSample <- sample(orientationPool2, 1)
      schedule[i, j] <- paste(schedule[i, j], currentSample)
      orientationPool2 <- orientationPool2[!orientationPool2 %in% currentSample]
      # If the value of the current cell is anything else, add "vertical" to the
value
    } else {
      schedule[i, j] <- paste(schedule[i, j], "vertical")
    }
  }
}
# Remove the pools that you created from the environment
for (i in c(1:length(screenOrientations))) {
  rm(list = paste0("orientationPool", i))
}

# Add the treatments that were repeated due to camera failure
schedule$'Timeslot 14' <- c("v2 screen no attractant vertical",
```

```
                              "yellow rough no attractant vertical",
                              "blue rough attractant vertical",
                              "yellow smooth no attractant vertical",
                              "blue rough attractant vertical",
                              "blue rough no attractant vertical")
schedule$'Timeslot 15' <- c("blue rough no attractant vertical",
                              "white rough attractant vertical",
                              "v1 screen no attractant vertical",
                              "white rough attractant vertical",
                              "yellow smooth no attractant horizontal",
                              "yellow rough no attractant vertical")


# Create a new data frame which will be used for adding dates to the column names
scheduleFinal <- schedule


# Change the first column name to the start date defined at the beginning of this
script
names(scheduleFinal)[1] <- paste(wday(startDate, week_start = 1, label = TRUE, abbr
= FALSE), format(startDate, "%d-%m-%Y"))
# At the start, assign a value of 0 the elapsedDays
elapsedDays <- 0
# One by one, change each column name into a date based on the start date, the
interval length, and the fact that no activities will take place during the weekend
(Saturday and Sunday):
for (i in c(2:ncol(scheduleFinal))) {
  # First check whether adding the number of elapsed days and the interval length
results in a date that takes place in the weekend
  if (wday(startDate + elapsedDays + interval_days, week_start = 1) <= 5) {
    # If the date will take place on a weekday, sum the interval length with the
number of elapsed days in order to update the number of elapsed days
    elapsedDays <- elapsedDays + interval_days
    # Sum the elapsed days with the start date and change the column name into this
newly calculated date
    names(scheduleFinal)[i] <- paste(wday(startDate + elapsedDays, week_start = 1,
label = TRUE, abbr = FALSE), format(startDate + elapsedDays, "%d-%m-%Y"))
    # If the date would not be on a weekday, check on which day in the weekend it
will take place:
  } else if (wday(startDate + elapsedDays + interval_days, week_start = 1) >= 6) {
    # Repeat the same steps as above with the addition of extra days based on
whether the date would have been a Saturday (+2 days) or a Sunday (+1 day)
    elapsedDays <- elapsedDays + interval_days +
      if (wday(startDate + elapsedDays + interval_days, week_start = 1) == 6) {
        2
      } else {
        1
      }
```

```
    names(scheduleFinal)[i] <- paste(wday(startDate + elapsedDays, week_start = 1,
label = TRUE, abbr = FALSE), format(startDate + elapsedDays, "%d-%m-%Y"))
  }
}

rownames(scheduleFinal) <- c("Camera 222", "Camera 223", "Camera 224", "Camera
225", "Camera 227", "Camera 228")


# Swap the treatment of camera 225, 31-05-2023 with that of camera 225, 09-06-2023
temp1 <- scheduleFinal[4, 1]
temp2 <- scheduleFinal[4, 5]
scheduleFinal[4, 1] <- temp2
scheduleFinal[4, 5] <- temp1

# Swap the treatment of camera 228, 31-05-2023 with that of camera 228, 23-06-2023
temp1 <- scheduleFinal[6, 1]
temp2 <- scheduleFinal[6, 11]
scheduleFinal[6, 1] <- temp2
scheduleFinal[6, 11] <- temp1


write.csv2(scheduleFinal, "output/tables/scheduleFinal_Ooijpolder.csv")




scheduleFinalNW <- scheduleFinal

names(scheduleFinalNW)[1] <- format(startDate, "%d-%m-%y")
# At the start, assign a value of 0 the elapsedDays
elapsedDays <- 0
# One by one, change each column name into a date based on the start date, the
interval length, and the fact that no activities will take place during the weekend
(Saturday and Sunday):
for (i in c(2:ncol(scheduleFinalNW))) {
  # First check whether adding the number of elapsed days and the interval length
results in a date that takes place in the weekend
  if (wday(startDate + elapsedDays + interval_days, week_start = 1) <= 5) {
    # If the date will take place on a weekday, sum the interval length with the
number of elapsed days in order to update the number of elapsed days
    elapsedDays <- elapsedDays + interval_days
    # Sum the elapsed days with the start date and change the column name into this
newly calculated date
    names(scheduleFinalNW)[i] <- format(startDate + elapsedDays, "%d-%m-%y")
    # If the date would not be on a weekday, check on which day in the weekend it
will take place:
  } else if (wday(startDate + elapsedDays + interval_days, week_start = 1) >= 6) {
```

```
    # Repeat the same steps as above with the addition of extra days based on
whether the date would have been a Saturday (+2 days) or a Sunday (+1 day)
    elapsedDays <- elapsedDays + interval_days +
      if (wday(startDate + elapsedDays + interval_days, week_start = 1) == 6) {
        2
      } else {
        1
      }
    names(scheduleFinalNW)[i] <- format(startDate + elapsedDays, "%d-%m-%y")
  }
}


# Create an empty matrix that will be used to hold all the cell locations (row and
column) of the schedule
cells <- matrix('', nrow(scheduleFinalNW) * ncol(scheduleFinalNW), 2)
# The following for-loop will add the cell locations to the "cells"-matrix
counter <- 1
for(i in c(1:ncol(scheduleFinalNW))) {
  for (j in c(1:nrow(scheduleFinalNW))) {
    cells[counter, ] <- c(j, i)
    counter <- counter + 1
  }
}
# Change the storage mode of the matrix to numeric instead of character
storage.mode(cells) <- "numeric"


# Create an empty matrix that will be used to hold the CSS code for the different
background colors/patterns
cellStyle <- matrix('', nrow(scheduleFinalNW) * ncol(scheduleFinalNW), 2)
# The following for-loop will fill the "cellStyle"-matrix based on the contents of
each corresponding "scheduleFinalNW"-cell. The second column of the "cellStyle"-
matrix is used to hold CSS code that will make the text legible if the cell has a
background pattern
counter <- 1
for(i in c(1:ncol(scheduleFinalNW))) {
  for (j in c(1:nrow(scheduleFinalNW))) {
    if (scheduleFinalNW[j, i] == "v1 screen no attractant vertical") {
      cellStyle[counter, 1] <- "background-color: #F7B500"
    } else if (scheduleFinalNW[j, i] == "v2 screen no attractant vertical") {
      cellStyle[counter, 1] <- "background-color: #F1DD39"
    } else if (scheduleFinalNW[j, i] == "yellow rough no attractant vertical") {
      cellStyle[counter, 1] <- "background-color: yellow; background-image:
repeating-linear-gradient(135deg, transparent,  transparent 5px, black 5px, black
10px)"
    } else if (scheduleFinalNW[j, i] == "yellow rough no attractant horizontal") {
```

```
      cellStyle[counter, 1] <- "background-color: yellow; background-image:
repeating-linear-gradient(135deg, transparent,  transparent 5px, black 5px, black
10px)"
      cellStyle[counter, 2] <- "background-color: yellow; border-radius: 2px"
    } else if (scheduleFinalNW[j, i] == "yellow smooth no attractant vertical") {
      cellStyle[counter, 1] <- "background-color: yellow"
    } else if (scheduleFinalNW[j, i] == "yellow smooth no attractant horizontal") {
      cellStyle[counter, 1] <- "background-color: yellow"
    } else if (scheduleFinalNW[j, i] == "blue rough no attractant vertical") {
      cellStyle[counter, 1] <- "background-color: DeepSkyBlue; background-image:
repeating-linear-gradient(135deg, transparent, transparent 5px, black 5px, black
10px)"
    } else if (scheduleFinalNW[j, i] == "blue smooth no attractant vertical") {
      cellStyle[counter, 1] <- "background-color: DeepSkyBlue"
    } else if (scheduleFinalNW[j, i] == "white rough no attractant vertical") {
      cellStyle[counter, 1] <- "background-color: white; background-image:
repeating-linear-gradient(135deg, transparent,  transparent 5px, black 5px, black
10px)"
    } else if (scheduleFinalNW[j, i] == "white smooth no attractant vertical") {
      cellStyle[counter, 1] <- "background-color: white"
    } else if (scheduleFinalNW[j, i] == "yellow rough attractant vertical") {
      cellStyle[counter, 1] <- "background-color: yellow; background-
image:  repeating-linear-gradient(45deg, black 25%, transparent 25%, transparent
75%, black 75%, black), repeating-linear-gradient(45deg, black 25%, yellow 25%,
yellow 75%, black 75%, black); background-position: 0 0, 10px 10px;background-size:
20px 20px;"
    } else if (scheduleFinalNW[j, i] == "blue rough attractant vertical") {
      cellStyle[counter, 1] <- "background-color: DeepSkyBlue; background-
image:  repeating-linear-gradient(45deg, black 25%, transparent 25%, transparent
75%, black 75%, black), repeating-linear-gradient(45deg, black 25%, DeepSkyBlue
25%, DeepSkyBlue 75%, black 75%, black); background-position: 0 0, 10px
10px;background-size: 20px 20px;"
    } else if (scheduleFinalNW[j, i] == "white rough attractant vertical") {
      cellStyle[counter, 1] <- "background-color: white; background-
image:  repeating-linear-gradient(45deg, black 25%, transparent 25%, transparent
75%, black 75%, black), repeating-linear-gradient(45deg, black 25%, white 25%,
white 75%, black 75%, black); background-position: 0 0, 10px 10px;background-size:
20px 20px;"
    }
    counter <- counter + 1
  }
}


# Create a empty data frame that will be used as the base of the HTML-table
emptySchedule <- data.frame(matrix("", nrow = nrow(scheduleFinalNW), ncol =
ncol(scheduleFinalNW)))
rownames(emptySchedule) <- rownames(scheduleFinalNW)
colnames(emptySchedule) <- colnames(scheduleFinalNW)
```

```
# Fill the "emptySchedule" data frame with the text that will appear in HTML-table
for (i in c(1:nrow(scheduleFinalNW))) {
  for (j in c(1:ncol(scheduleFinalNW))) {
    if (grepl("horizontal", scheduleFinalNW[i, j], fixed = TRUE)) {
      emptySchedule[i, j] <- "Horizontal"
    } else if (grepl("v1", scheduleFinalNW[i, j], fixed = TRUE)) {
      emptySchedule[i, j] <- "V1"
    } else if (grepl("v2", scheduleFinalNW[i, j], fixed = TRUE)) {
      emptySchedule[i, j] <- "V2"
    }
  }
}


# Create a function that can be used the place HTML code in a data frame based on
cell locations, and implements CSS code into the HTML code
inject_div <- function(x, where, style) {
  x[where] <- sprintf('<div style = \'%s\'> <span style = \'%s\' > %s </span>
</div>',
                      gsub(';*$', ';', style[, 1]),
                      gsub(";*$", ";", style[, 2]),
                      x[where])
  x
}


# Some CSS code is applicable to all the cells of the final HTML-table, and can
thus be added to the "cellStyle"-matrix with a single line of code:
cellStyle[, 1] <- paste(cellStyle[, 1], "; width: 90px; height: 35px; border-
radius: 6px; font-family: Sans-serif; name: Arial; padding-top: 15px; border: solid
1px", sep = "")

# Execute the inject_div-function, thereby filling the "emptySchedule"-data frame
with the HTML/CSS code
cellStyle_final <- inject_div(emptySchedule,
                              where = cells,
                              style = cellStyle)

# During one timeslot, the screen was swapped (V1 -> yellow smooth no attractant
vertical) halfway into the 2-day duration. The HTML/CSS code of this cell has to be
changed to reflect this deviation
cellStyle_final[3, 15] <- "<div style = 'border-radius: 6px; font-family: Sans-
serif; name: Arial; height: 50px; width: 90px; border: solid 1px; background-color:
yellow'> <div style = 'float: left; height: 35px; width: 50%; margin-right: 5px;
background-color: #F7B500; border-top-left-radius: 6px; border-bottom-left-radius:
6px; padding-top: 15px'> V1 </div> </div>"
```

```
# Using the htmlTable-command, create the HTML-table
htmlTable_Ooijpolder <- htmlTable(cellStyle_final,
                                  css.cell = append("font-family: Sans-serif; name:
Arial; max-width: 60px; min-width: 60px", rep("", ncol(scheduleFinalNW))),
                                  css.header = "font-family: Sans-serif; name:
Arial; max-width: 100px")


save_html(htmlTable_Ooijpolder, "output/tables/htmlTable_Ooijpolder.html")
```

Script 3: Statistics Hortus model

```
library(plyr)
library(tidyverse)
library(lubridate)
library(httr)
library(jsonlite)
library(cowplot)
library(glmmTMB)
library(DHARMa)
library(ggtext)
library(ggh4x)
library(performance)
library(emmeans)

##### Hortus #####

# Start by defining the camera numbers of interest
cameraNumbers <- c("243", "259", "304", "320", "329", "367")

# The following code individually loads each camera's dataset, removes all rows
from before the starting date of the experiment, assigns the corresponding camera
number (cameraID) to each row, and finally combines all datasets into one
hortusData table
hortusData <- as.data.frame(c())
for (i in cameraNumbers) {
  currentDataset <- paste0("DIOPSIS", i, "_data")
  assign(currentDataset, read.csv(paste0("input/rawData/RUN-Hortus-2023-
20230727T110008Z-001/RUN-Hortus-2023/Camera ", i, "_2023-", i, "_data_dedup.csv")))
  assign(currentDataset, cbind(eval(parse(text = currentDataset)), "cameraID" = i))
  hortusData <- rbind(hortusData, eval(parse(text = currentDataset)))
}

# import a file containing the treatment intervals of each camera
treatmentDates_Hortus <- read.csv2("input/rawData/treatmentDates_Hortus.csv")

# The following steps will assign the correct treatment to each individual row in
the hortusData dataset
# First, create a table that has all treatments with their time intervals nested by
cameraID
nestedIDs <- treatmentDates_Hortus %>%
  mutate(across(c("startDateTime", "endDateTime"), ymd_hms)) %>%
  group_nest(cameraID)
# Change the class of the cameraIDs to integer so that it is compatible in the
following steps
hortusData$cameraID <- as.integer(hortusData$cameraID)

# Change uid to a usable date format in a new dateTime column
hortusData <- hortusData %>%
```

```
  mutate(dateTime = uid) %>%
  mutate(across("dateTime", ymd_hms))

# Add a new column to hortusData that contains each row's treatment, indicated by
the treatment's row number in the nestedIDs table
hortusData <- left_join(x = hortusData, y = nestedIDs, by = "cameraID") %>%
  as_tibble() %>%
  mutate(treatmentNumber = map2(dateTime, data, ~which(.x %within%
interval(.y$startDateTime, .y$endDateTime)))) %>%
  select(-data)
# Change the class of the treatment number to integer make it compatible in the
next steps
hortusData$treatmentNumber <- as.integer(hortusData$treatmentNumber)


# Remove the rows containing NAs in the treatmentNumber column from hortusData,
since this indicates that there was no treatment or no treatment of interest during
those time intervals
hortusData <- subset(hortusData, !is.na(treatmentNumber))

# Add four new columns to hortusData to hold the actual treatments of each row
addition <- data.frame(matrix(nrow = nrow(hortusData), ncol = 4))
hortusData <- cbind(hortusData, addition)
colnames(hortusData)[(ncol(hortusData) - 3):ncol(hortusData)] <- c("screenColor",
"screenTexture", "screenOrientation", "screenVersion")

# Add the treatments to these new columns. The treatment is picked from
treatmentDates_Hortus
for (i in c(1:nrow(hortusData))) {
  hortusData[i, c((ncol(hortusData) - 3):ncol(hortusData))] <-
subset(treatmentDates_Hortus, cameraID == hortusData[i, ]$cameraID)[hortusData[i,
]$treatmentNumber, ][2:5]
}



# Import a file containing the uptime intervals of each camera
uptimeDates_Hortus <- read.csv2("input/rawData/uptimeDates_Hortus.csv")

# The following steps will remove any rows that contain data outside of the uptime
intervals of the cameras. This is done in order to remove any data gathered from
photos that were taken during repairs/troubleshooting of the camera.
# First, create a table that has all uptime intervals nested by cameraID
nestedIDs <- uptimeDates_Hortus %>%
  mutate(across(c("startDateTime", "endDateTime"), ymd_hms)) %>%
  group_nest(cameraID)

# Add a new column to hortusData that contains the status of the camera (1 = up, 0
= down) for each row
```

```
hortusData <- left_join(x = hortusData, y = nestedIDs, by = "cameraID") %>%
  as_tibble() %>%
  mutate(cameraStatus = map2_int(dateTime, data, ~+any(.x %within%
interval(.y$startDateTime, .y$endDateTime)))) %>%
  select(-data)

# Select only those rows containing data when the cameras were up
hortusData <- subset(hortusData, cameraStatus == 1)



# To implement date in the linear model, we will use several proxies for time
# First, create a column containing the day-number of the experiment. The count
starts with 24-04-2023 as day 1.
hortusData$dayCount <- as.integer(as.Date(hortusData$dateTime) - as.Date("2023-04-
24")) + 1
# Then, create a column that contains the hour-number of that day, with hour 1
starting at 00:00 of each day.
hortusData$hourCount <- as.integer(substr(format(hortusData$dateTime, "%Y-%m-%d
%H:%M:%OS"), 12, 13)) + 1


# In the uptimeDates_Hortus dataset, add new columns that contain new dates rounded
up and down to the nearest hour. The following steps will use this data to remove
any data that is not part of a 1 hour time interval during which the camera was
online.
uptimeDates_Hortus$startDateTime_oneHour <-
ceiling_date(ymd_hms(uptimeDates_Hortus$startDateTime), "1 hour")
uptimeDates_Hortus$endDateTime_oneHour <-
floor_date(ymd_hms(uptimeDates_Hortus$endDateTime), "1 hour")

# Update nestedIDs to include the newly created columns
nestedIDs <- uptimeDates_Hortus %>%
  group_nest(cameraID)

# For each row, add an indicator whether the row should be kept or not (0 =
discard, 1 = keep) based on whether the row is part of a complete 1 hour interval.
hortusData <- left_join(x = hortusData, y = nestedIDs, by = "cameraID") %>%
  as_tibble() %>%
  mutate(cameraKeepUptime = map2_int(dateTime, data, ~+any(.x %within%
interval(.y$startDateTime_oneHour, .y$endDateTime_oneHour)))) %>%
  select(-data)


# Repeat the process of rounding up and down to the nearest full hour for the
treatment dates.
treatmentDates_Hortus$startDateTime_oneHour <-
ceiling_date(ymd_hms(treatmentDates_Hortus$startDateTime), "1 hour")
```

```
treatmentDates_Hortus$endDateTime_oneHour <-
floor_date(ymd_hms(treatmentDates_Hortus$endDateTime), "1 hour")

nestedIDs <- treatmentDates_Hortus %>%
  group_nest(cameraID)

hortusData <- left_join(x = hortusData, y = nestedIDs, by = "cameraID") %>%
  as_tibble() %>%
  mutate(cameraKeepTreatment = map2_int(dateTime, data, ~+any(.x %within%
interval(.y$startDateTime_oneHour, .y$endDateTime_oneHour)))) %>%
  select(-data)

# Remove the the data points that are not part of a complete 1 hour cycle
hortusData <- subset(hortusData, cameraKeepUptime == 1 & cameraKeepTreatment == 1)

# The following code is commented because it uses an API and will therefore not
function without an internet connection or when the API is down. An Rda file of
taxaTable_hortus created on 18-09-2023 will be loaded instead. It is also possible
to un-comment this code in order to create the taxaTable_hortus dataframe yourself.
# # I will now proceed to list the correct names of the taxa levels of each
individual. The original dataset contains an object_order column that contains only
the lowest taxon level name (e.g. Chironomidae), which is not very useful for data
analysis.
# # First, create a list that contains all the different taxon names in the
object_order column.
# object_orderFixed <- levels(as.factor(hortusData$object_order))
# # Change the taxa names that deviate from the standard format
# object_orderFixed <- sub("Auchenorrhyncha indet.", "Auchenorrhyncha",
object_orderFixed)
#
# # Create a table for holding all the correct taxa levels of each taxon name
# taxa <- c("kingdom", "phylum", "class", "order", "family", "genus", "species")
# taxaTable_hortus <- data.frame(matrix(NA, nrow = length(object_orderFixed), ncol
= length(taxa)))
# colnames(taxaTable_hortus) <- taxa
#
# # Fill taxaTable with the correct taxa levels/names of each taxon name given in
object_order. This is done using the GBIF (Global Biodiversity Information
Facility) API.
# for (i in object_orderFixed) {
#   answer <-  GET("https://api.gbif.org/v1/species",
#                  query = list(name = i))
#   answer_json <- fromJSON(rawToChar(answer$content))
#   tempTaxa <- c()
#   for (j in taxa) {
#     if (is.null(answer_json[["results"]][[j]][1])) {
#       tempTaxa <- append(tempTaxa, NA)
#     } else {
```

```
#       tempTaxa <- append(tempTaxa, answer_json[["results"]][[j]][1])
#     }
#   }
#   taxaTable_hortus[match(i, object_orderFixed), ] <- tempTaxa
# }
#
# taxaTable_hortus$object_order <- levels(as.factor(hortusData$object_order))
#
# # Save taxaTable_hortus in order to use it without having to access the api
# save(taxaTable_hortus, file = "taxaTable_hortus.Rda")

load("input/RDAFiles/taxaTable_hortus.Rda")

# Add the columns from taxaTable_hortus to hortusData
hortusData <- hortusData %>%
  left_join(taxaTable_hortus, by = "object_order")


# The following code is commented because it uses an API and will therefore not
function without an internet connection or when the API is down. An Rda file of
hortusSunData created on 18-09-2023 will be loaded instead. It is also possible to
un-comment this code in order to create the hortusSunData dataframe yourself.
# # Add a variable that indicates whether the individual was observed during the
day (0) or the night (1), based on sunrise and sunset
# hortusLat <- c(52.156817)
# hortusLon <- c(4.483137)
#
# hortusSunData <- data.frame(cameraID = rep(cameraNumbers, each =
length(levels(as.factor(format(hortusData$dateTime, "%Y-%m-%d"))))),
#                                 date =
rep(levels(as.factor(format(hortusData$dateTime, "%Y-%m-%d"))),
length(cameraNumbers)),
#                                 lat = rep(hortusLat, each =
length(levels(as.factor(format(hortusData$dateTime, "%Y-%m-%d"))))),
#                                 lon = rep(hortusLon, each =
length(levels(as.factor(format(hortusData$dateTime, "%Y-%m-%d"))))),
#                                 sunrise = NA,
#                                 sunset = NA)
#
#
# for (i in c(1:nrow(hortusSunData))) {
#   answer <- GET("https://api.sunrise-sunset.org/json",
#                 query = list(lat = hortusSunData$lat[i],
#                              lng = hortusSunData$lon[i],
#                              date = hortusSunData$date[i]))
#   answer_json <- fromJSON(rawToChar(answer$content))
#   hortusSunData$sunrise[i] <- paste(hortusSunData$date[i],
format(strptime(answer_json[["results"]][["sunrise"]], "%I:%M:%S %p") + hours(2),
"%H:%M:%S"))
```

```
#   hortusSunData$sunset[i] <- paste(hortusSunData$date[i],
format(strptime(answer_json[["results"]][["sunset"]], "%I:%M:%S %p") + hours(2),
"%H:%M:%S"))
# }
# hortusSunData$sunrise <- as.POSIXct(hortusSunData$sunrise, tz = "UTC")
# hortusSunData$sunset <- as.POSIXct(hortusSunData$sunset, tz = "UTC")
#
# hortusSunData$sunrise_oneHour <- round_date(hortusSunData$sunrise, "1 hour")
# hortusSunData$sunset_oneHour <- round_date(hortusSunData$sunset, "1 hour")
#
#
# hortusSunData$cameraID <- as.integer(hortusSunData$cameraID)
#
# # Save hortusSunData in order to use it without having to access the api
# save(hortusSunData, file = "hortusSunData.Rda")

load("input/RDAFiles/hortusSunData.Rda")

nestedIDs <- hortusSunData %>%
  group_nest(cameraID)

hortusData <- left_join(x = hortusData, y = nestedIDs, by = c("cameraID")) %>%
  as_tibble() %>%
  mutate(dayOrNight = map2_int(dateTime, data, ~+any(.x %within%
interval(.y$sunrise_oneHour, .y$sunset_oneHour)))) %>%
  select(-data)


# Also add an extra column containing the full treatment
fullTreatment <- c()
for (i in c(1:nrow(hortusData))) {
  treatmentName <- c()
  if (!is.na(hortusData$screenColor[i])) {
    treatmentName <- paste0(treatmentName, paste0(hortusData$screenColor[i], "_"))
  }
  if (!is.na(hortusData$screenOrientation[i])) {
    treatmentName <- paste0(treatmentName, paste0(hortusData$screenOrientation[i],
"_"))
  }
  if (!is.na(hortusData$screenTexture[i])) {
    treatmentName <- paste0(treatmentName, paste0(hortusData$screenTexture[i]))
  }
  if (!is.na(hortusData$screenVersion[i])) {
    treatmentName <- paste0(treatmentName, paste0(hortusData$screenVersion[i]))
  }
  fullTreatment <- append(fullTreatment, treatmentName)
}

hortusData$fullTreatment <- fullTreatment
```

```
# My goal is to obtain the average number of individuals per hour. However, since
hortusData only contains entries of detected individuals, I still need to create a
dataset that contains the full hours during which nothing was caught. This is what
is done during the following steps.
# First, create a table that holds every single 1 hour mark well within the
experimental period (in this case, from April 24th until May 30th).
hortusNoData <- data.frame(cameraID = as.numeric(rep(cameraNumbers,
                                                     each =
length(seq(as.POSIXct("2023-04-24 00:00:00",

  format = "%Y-%m-%d %H:%M:%OS",

  tz = "UTC"),
                                                                  +
as.POSIXct("2023-05-30 23:59:00",

    format = "%Y-%m-%d %H:%M:%OS",

    tz = "UTC"),
                                                                  by = "1
hour")))),
                                dateTime = rep(seq(as.POSIXct("2023-04-24 00:00:00",
                                                              format = "%Y-%m-%d
%H:%M:%OS",
                                                              tz = "UTC"),
                                                   + as.POSIXct("2023-05-30
23:59:00",
                                                                format = "%Y-%m-%d
%H:%M:%OS",
                                                                tz = "UTC"),
                                                   by = "1 hour"), 6))

# Just like before, add the count of the day and hour mark to the table.
hortusNoData$dayCount <- as.integer(as.Date(hortusNoData$dateTime) - as.Date("2023-
04-24")) + 1
hortusNoData$hourCount <- as.integer(substr(format(hortusNoData$dateTime, "%Y-%m-%d
%H:%M:%OS"), 12, 13)) + 1

# Select and filter the 1 hour intervals on which the cameras were actually on and
treated
nestedIDs <- uptimeDates_Hortus %>%
  group_nest(cameraID)

hortusNoData <- left_join(x = hortusNoData, y = nestedIDs, by = "cameraID") %>%
  as_tibble() %>%
  mutate(cameraKeepUptime = map2_int(dateTime, data, ~+any(.x %within%
interval(.y$startDateTime_oneHour, .y$endDateTime_oneHour - 1)))) %>%
```

```
  select(-data)


nestedIDs <- treatmentDates_Hortus %>%
  group_nest(cameraID)

hortusNoData <- left_join(x = hortusNoData, y = nestedIDs, by = "cameraID") %>%
  as_tibble() %>%
  mutate(cameraKeepTreatment = map2_int(dateTime, data, ~+any(.x %within%
interval(.y$startDateTime_oneHour, .y$endDateTime_oneHour - 1)))) %>%
  select(-data)


hortusNoData <- subset(hortusNoData, cameraKeepUptime == 1 & cameraKeepTreatment ==
1)

nestedIDs <- hortusSunData %>%
  group_nest(cameraID)

hortusNoData <- left_join(x = hortusNoData, y = nestedIDs, by = c("cameraID")) %>%
  as_tibble() %>%
  mutate(dayOrNight = map2_int(dateTime, data, ~+any(.x %within%
interval(.y$sunrise_oneHour, .y$sunset_oneHour)))) %>%
  select(-data)

# Create a function that creates an overview of the hourly counts of a given
subset.
countHortus <- function() {
  # Create a new dataset from hortusNoData with the rows that are already
represented in the subset removed
  hortusNoData_alt <- anti_join(hortusNoData, hortusSubset, by = c("cameraID",
"dayCount", "hourCount"))
  # For the remaining data, add columns containing the treatment
  nestedIDs <- treatmentDates_Hortus %>%
    mutate(across(c("startDateTime", "endDateTime"), ymd_hms)) %>%
    group_nest(cameraID)

  hortusNoData_alt <- left_join(x = hortusNoData_alt, y = nestedIDs, by =
"cameraID") %>%
    as_tibble() %>%
    mutate(treatmentNumber = map2(dateTime, data, ~which(.x %within%
interval(.y$startDateTime, .y$endDateTime)))) %>%
    select(-data)

  hortusNoData_alt$treatmentNumber <- as.integer(hortusNoData_alt$treatmentNumber)

  hortusNoData_alt <- subset(hortusNoData_alt, !is.na(treatmentNumber))

  addition <- data.frame(matrix(nrow = nrow(hortusNoData_alt), ncol = 5))
```

```
  hortusNoData_alt <- cbind(hortusNoData_alt, addition)
  colnames(hortusNoData_alt)[(ncol(hortusNoData_alt) - 4):ncol(hortusNoData_alt)]
<- c("screenColor", "screenTexture", "screenOrientation", "screenVersion",
"fullTreatment")

  for (i in c(1:nrow(hortusNoData_alt))) {
    hortusNoData_alt[i, c((ncol(hortusNoData_alt) - 3):ncol(hortusNoData_alt) - 1)]
<- subset(treatmentDates_Hortus, cameraID == hortusNoData_alt[i,
]$cameraID)[hortusNoData_alt[i, ]$treatmentNumber, ][2:5]
  }

  # Also add an extra column containing the full treatment
  fullTreatment <- c()
  for (i in c(1:nrow(hortusNoData_alt))) {
    treatmentName <- c()
    if (!is.na(hortusNoData_alt$screenColor[i])) {
      treatmentName <- paste0(treatmentName,
paste0(hortusNoData_alt$screenColor[i], "_"))
    }
    if (!is.na(hortusNoData_alt$screenOrientation[i])) {
      treatmentName <- paste0(treatmentName,
paste0(hortusNoData_alt$screenOrientation[i], "_"))
    }
    if (!is.na(hortusNoData_alt$screenTexture[i])) {
      treatmentName <- paste0(treatmentName,
paste0(hortusNoData_alt$screenTexture[i]))
    }
    if (!is.na(hortusNoData_alt$screenVersion[i])) {
      treatmentName <- paste0(treatmentName,
paste0(hortusNoData_alt$screenVersion[i]))
    }
    fullTreatment <- append(fullTreatment, treatmentName)
  }

  hortusNoData_alt$fullTreatment <- fullTreatment

  # Select the relevant columns from hortusNoData_alt and after that add a column n
containing only zeroes, since all rows represent a count of zero individuals
  hortusNoData_alt <- hortusNoData_alt[, c("cameraID", "dayCount", "hourCount",
"screenColor", "screenTexture", "screenOrientation", "screenVersion",
"fullTreatment", "dayOrNight")]
  hortusNoData_alt <- cbind(hortusNoData_alt, n = 0)

  # Create another dataset with the counts of the selected subset aggregated by
hour
  hortusCounts <- hortusSubset %>%
    count(cameraID, dayCount, hourCount, screenColor, screenTexture,
screenOrientation, screenVersion, fullTreatment, dayOrNight)
```

```
  # Bind the hortusNoData_alt dataset to this dataset
  hortusCounts <- rbind(hortusCounts, hortusNoData_alt)

  # Assign the correct data types to the columns
  hortusCounts$n <- as.integer(hortusCounts$n)
  hortusCounts$dayOrNight <- as.factor(hortusCounts$dayOrNight)
  hortusCounts$fullTreatment <- factor(hortusCounts$fullTreatment, levels =
c("vertical_v2", "vertical_v1", "blue_vertical_rough", "blue_vertical_smooth",
"white_vertical_rough", "white_vertical_smooth", "yellow_vertical_rough",
"yellow_vertical_smooth", "yellow_horizontal_rough", "yellow_horizontal_smooth"))
  hortusCounts$cameraID <- as.factor(hortusCounts$cameraID)

  # Add a column containing the normalized day count
  hortusCounts$dayCount_normalized <- scale(hortusCounts$dayCount)

  hortusCounts
}


# # Create a function that makes a plot grid containing plotted mean counts divided
across several variables. This section is commented to save time since I do not
require these plot grids for my final product.
# plotsHortus <- function(counts, plotText){
#   ###############
#   hortusCounts_newScreens <- subset(counts, !is.na(screenColor) &
screenOrientation == "vertical")
#
#   hortusCounts_newScreens_meanSD <- hortusCounts_newScreens %>%
#     group_by(screenColor, screenTexture, dayOrNight) %>%
#     summarise_at(vars(n), list(mean = mean, sd = sd)) %>%
#     as.data.frame()
#
#   plot1 <- ggplot(hortusCounts_newScreens_meanSD, aes(x = screenColor, y = mean))
+
#     geom_bar(stat = "identity") +
#     geom_errorbar(aes(ymin = mean - sd, ymax = mean + sd), width = 0.2) +
#     facet_grid(cols = vars(screenTexture, dayOrNight), labeller =
labeller(dayOrNight = c("0" = "Night",
#
        "1" = "Day")))
#   ###############
#
#   ###############
#   hortusCounts_orientation <- subset(counts, screenColor == "yellow")
#
#   hortusCounts_orientation_meanSD <- hortusCounts_orientation %>%
#     group_by(screenOrientation, screenTexture, dayOrNight) %>%
#     summarise_at(vars(n), list(mean = mean, sd = sd)) %>%
#     as.data.frame()
```

```
#
#   plot2 <- ggplot(hortusCounts_orientation_meanSD, aes(x = screenOrientation, y =
mean)) +
#     geom_bar(stat = "identity") +
#     geom_errorbar(aes(ymin = mean - sd, ymax = mean + sd), width = 0.2) +
#     facet_grid(cols = vars(screenTexture, dayOrNight), labeller =
labeller(dayOrNight = c("0" = "Night",
#
        "1" = "Day")))
#   ###############
#
#   ###############
#   hortusCounts_oldScreens <- subset(counts, !is.na(screenVersion))
#
#   hortusCounts_oldScreens_meanSD <- hortusCounts_oldScreens %>%
#     group_by(screenVersion, dayOrNight) %>%
#     summarise_at(vars(n), list(mean = mean, sd = sd)) %>%
#     as.data.frame()
#
#   plot3 <- ggplot(hortusCounts_oldScreens_meanSD, aes(x = screenVersion, y =
mean)) +
#     geom_bar(stat = "identity") +
#     geom_errorbar(aes(ymin = mean - sd, ymax = mean + sd), width = 0.2) +
#     facet_grid(cols = vars(dayOrNight), labeller = labeller(dayOrNight = c("0" =
"Night",
#                                                                             "1" =
"Day")))
#   ###############
#
#   plot_grid(ggdraw() +
#               draw_label(
#                 plotText,
#                 fontface = 'bold',
#                 x = 0,
#                 hjust = 0
#               ) +
#               theme(
#                 plot.margin = margin(0, 0, 0, 7)
#               ), plot1, plot2, plot3)
# }


# Create subsets of interest. The code generating the plots is commented to save
time since I do not require them for my final product.
hortusSubset <- subset(hortusData, class == "Insecta" & object_order_probability >=
0.5)
hortusCounts_insects <- countHortus()

# hortusPlots_insects <- plotsHortus(hortusCounts_insects, "All insects")
```

```r
hortusSubset <- subset(hortusData, order == "Lepidoptera" &
object_order_probability >= 0.5)
hortusCounts_lepidoptera <- countHortus()

# hortusPlots_lepidoptera <- plotsHortus(hortusCounts_lepidoptera, "Order:
Lepidoptera")


hortusSubset <- subset(hortusData, order == "Hymenoptera" &
object_order_probability >= 0.5)
hortusCounts_hymenoptera <- countHortus()

# hortusPlots_hymenoptera <- plotsHortus(hortusCounts_hymenoptera, "Order:
Hymenoptera")


hortusSubset <- subset(hortusData, order == "Diptera" & object_order_probability >=
0.5)
hortusCounts_diptera <- countHortus()

# hortusPlots_diptera <- plotsHortus(hortusCounts_diptera, "Order: Diptera")



# Create distribution plots for taxa of interest to use for determining the model's
distribution family
distributionPlot_insects <- ggplot(hortusCounts_insects, aes(x = n)) +
  geom_bar()

distributionPlot_lepidoptera <- ggplot(hortusCounts_lepidoptera, aes(x = n)) +
  geom_bar()

distributionPlot_hymenoptera <- ggplot(hortusCounts_hymenoptera, aes(x = n)) +
  geom_bar()

distributionPlot_diptera <- ggplot(hortusCounts_diptera, aes(x = n)) +
  geom_bar()


# Create a vector to hold the significance indicators for the comparisons
significanceValues <- function(calculatedModelValues) {
  significance <- c()
  for (i in c(1:nrow(calculatedModelValues))) {
    if (calculatedModelValues$p[i] <= 0.001) {
      significance <- c(significance, "***")
    } else if (calculatedModelValues$p[i] <= 0.01) {
      significance <- c(significance, "**")
```

```
    } else if (calculatedModelValues$p[i] <= 0.05) {
      significance <- c(significance, "*")
    } else if (calculatedModelValues$p[i] <= 0.1) {
      significance <- c(significance, "°")
    } else {
      significance <- c(significance, "")
    }
  }
  significance
}



# I will also include visual indicators (pictures) for the plot's x-axis tick
labels.
# Get a vector with the treatments.
treatments <- levels(hortusCounts_insects$fullTreatment)
# Write HTML code for additional text underneath the visual indicators.
treatmentsLabels <- c("v2<br>vertical", "v1<br>vertical",
"Blue<br>rough<br>vertical", "Blue<br>smooth<br>vertical",
"White<br>rough<br>vertical", "White<br>smooth<br>vertical",
"Yellow<br>rough<br>vertical", "Yellow<br>smooth<br>vertical",
"Yellow<br>rough<br>hori-<br>zontal", "Yellow<br>smooth<br>hori-<br>zontal")

# Generate the tick label HTML code using pre-made pictures of the treatments (the
pictures are taken from the HTML table schedule)
labels <- c()
for (i in c(1:length(treatments))) {
  labels <- c(labels, paste0("<img src='input/figures/", treatments[i], ".png", "'
width='40' /><br>", treatmentsLabels[i]))
}


# Finally, create the plot using ggplot
createCountPlot <- function(calculatedModelValues, calculatedSignificanceValues,
plotTitle) {
  tempPlot <- ggplot(calculatedModelValues, aes(x = fullTreatment, y = response,
color = dayOrNight)) +
    guides(color = guide_legend(title = "Day or night")) +
    scale_color_manual(labels = c("Night", "Day"), values = c("darkblue",
"darkorange")) +
    geom_point(size = 2, position = position_dodge(0.5)) +
    scale_y_log10() +
    geom_vline(xintercept = 1.5, linetype = "dashed", alpha = 0.5) +
    geom_errorbar(aes(ymin = standardErrorLower, ymax = standardErrorUpper), width
= 0.1, position = position_dodge(0.5)) +
    labs(y = "Counts per hour") +
    scale_x_discrete(name = "Treatment",
                     labels = labels) +
```

```
    ggtitle(plotTitle) +
    theme_bw() +
    theme(axis.text.x = element_markdown(color = "black", size = 11),
          axis.text.y = element_text(size = 11),
          axis.title = element_text(size = 14),
          legend.text = element_text(size = 11),
          legend.title = element_text(size = 14),
          plot.title = element_text(hjust = 0.5)) +
    geom_text(aes(label = calculatedSignificanceValues, group = dayOrNight),
              position = position_dodge(width = 1.7), size = 6,
              show.legend = FALSE)+
    geom_text(aes(label = round(response, 1)),
              position = position_dodge(width = 1.7),
              vjust = 1,
              show.legend = FALSE)
  tempPlot
}



#### General data figures ####
# Figures to show the total individual counts for both classes and orders
classCounts_hortus <- subset(hortusData, object_order_probability >= 0.5) %>%
mutate_at(c("class", "dayOrNight"), as.factor) %>% count(class, dayOrNight, .drop =
FALSE)
classCounts_hortus$label_y <- NA
counter <- 0
for (i in c(1:(nrow(classCounts_hortus)/2))) {
  counter <- counter + 2
  classCounts_hortus$label_y[(counter - 1): counter] <-
sum(classCounts_hortus$n[(counter - 1): counter])
}

hortusClassCounts_plot <- ggplot() +
  geom_col(data = classCounts_hortus, aes(x = class, y = n, fill = dayOrNight)) +
  guides(fill = guide_legend(title = "Day or night")) +
  scale_fill_manual(labels = c("Night", "Day"), values = c("darkblue",
"darkorange")) +
  ylim(0, 600) +
  geom_text(data = classCounts_hortus, aes(x = class, y = label_y, color =
dayOrNight, label = n), vjust = rep(c(-2.5, -0.75),
length(classCounts_hortus$n)/2)) +
  scale_color_manual(labels = c("Night", "Day"), values = c("darkblue",
"darkorange"), guide = "none") +
  labs(y = "Total count", x = "Class") +
  ggtitle("Hortus Botanicus") +
  theme_bw() +
  theme(axis.text = element_text(size = 11),
        axis.title = element_text(size = 14),
```

```
        legend.text = element_text(size = 11),
        legend.title = element_text(size = 14),
        plot.title = element_text(hjust = 0.5))

hortusClassCounts_plot

ggsave("output/figures/hortusClassCounts_plot.png", width = 500, height = 270,
units = "px", scale = 4.7)



orderCounts_hortus <- subset(hortusData, class == "Insecta" &
object_order_probability >= 0.5) %>% mutate_at(c("order", "dayOrNight"), as.factor)
%>% count(order, dayOrNight, .drop = FALSE)
orderCounts_hortus$label_y <- NA
counter <- 0
for (i in c(1:(nrow(orderCounts_hortus)/2))) {
  counter <- counter + 2
  orderCounts_hortus$label_y[(counter - 1): counter] <-
sum(orderCounts_hortus$n[(counter - 1): counter])
}

hortusOrderCounts_plot <- ggplot() +
  geom_col(data = orderCounts_hortus, aes(x = order, y = n, fill = dayOrNight)) +
  guides(fill = guide_legend(title = "Day or night")) +
  scale_fill_manual(labels = c("Night", "Day"), values = c("darkblue",
"darkorange")) +
  ylim(0, 200) +
  geom_text(data = orderCounts_hortus, aes(x = order, y = label_y, color =
dayOrNight, label = n), vjust = rep(c(-2.5, -0.75),
length(orderCounts_hortus$n)/2)) +
  scale_color_manual(labels = c("Night", "Day"), values = c("darkblue",
"darkorange"), guide = "none") +
  labs(y = "Total count", x = "Order") +
  ggtitle("Hortus Botanicus") +
  theme_bw() +
  theme(axis.text = element_text(size = 11),
        axis.title = element_text(size = 14),
        axis.text.x = element_text(angle = 45, hjust = 1),
        legend.text = element_text(size = 11),
        legend.title = element_text(size = 14),
        plot.title = element_text(hjust = 0.5))

hortusOrderCounts_plot

ggsave("output/figures/hortusOrderCounts_plot.png", width = 500, height = 270,
units = "px", scale = 4.7)
```

```r
# A figure to show the number of active hours of the cameras
hourCounts_hortus <- hortusCounts_insects %>% count(cameraID, dayOrNight, .drop =
FALSE)
hourCounts_hortus$label_y <- NA
counter <- 0
for (i in c(1:(nrow(hourCounts_hortus)/2))) {
  counter <- counter + 2
  hourCounts_hortus$label_y[(counter - 1): counter] <-
sum(hourCounts_hortus$n[(counter - 1): counter])
}

hortusHourCounts_plot <- ggplot() +
  geom_col(data = hourCounts_hortus, aes(x = cameraID, y = n, fill = dayOrNight)) +
  guides(fill = guide_legend(title = "Day or night")) +
  scale_fill_manual(labels = c("Night", "Day"), values = c("darkblue",
"darkorange")) +
  ylim(0, 800) +
  geom_text(data = hourCounts_hortus, aes(x = cameraID, y = label_y, color =
dayOrNight, label = n), vjust = rep(c(-2.5, -0.75), length(hourCounts_hortus$n)/2))
+
  scale_color_manual(labels = c("Night", "Day"), values = c("darkblue",
"darkorange"), guide = "none") +
  labs(y = "Hour count", x = "Camera ID") +
  ggtitle("Hortus Botanicus") +
  theme_bw() +
  theme(axis.text = element_text(size = 11),
        axis.title = element_text(size = 14),
        legend.text = element_text(size = 11),
        legend.title = element_text(size = 14),
        plot.title = element_text(hjust = 0.5))

hortusHourCounts_plot

ggsave("output/figures/hortusHourCounts_plot.png", width = 500, height = 270, units
= "px", scale = 4.7)



#### Insects model ####

# Creating a model for insects:
# # After some trial and error, I created the following usable model. It does not
contain an interaction effect, as I was unable to create a usable model that
contained an interaction.
# M_insects_hortus <- glmmTMB(n ~ dayCount_normalized + dayOrNight + fullTreatment
+ (1 | cameraID), data = hortusCounts_insects, ziformula = ~ dayCount_normalized +
dayOrNight + (1 | cameraID), family = nbinom2, control = glmmTMBControl(optimizer =
optim, optArgs = list(method = "BFGS")))
#
```

```r
# # Dropping dayCount_normalized
# drop1(M_insects_hortus)
M_insects_hortus2 <- glmmTMB(n ~ dayOrNight + fullTreatment + (1 | cameraID), data
= hortusCounts_insects, ziformula = ~ dayOrNight + (1 | cameraID), family =
nbinom2, control = glmmTMBControl(optimizer = optim, optArgs = list(method =
"BFGS")))

testDispersion(M_insects_hortus2, alternative = "greater")
# Dispersion non-significant (p = 0.94)
plot(simulateResiduals(M_insects_hortus2))
# KS test non-significant (p = 0.08)
testOutliers(M_insects_hortus2, type = "bootstrap")
# Outlier test non-significant (p = 0.44 (May differ upon repeat due to the nature
of a bootstrapped test)

r2_zeroinflated(M_insects_hortus2)
# R2 = 0.357, R2 adj. = 0.354

# Create a dataframe for the predictions of the model
modelValues_insects_hortus <- as.data.frame(emmeans(M_insects_hortus2,
c("dayOrNight", "fullTreatment"), type = "response"))

# Reverse the order of day and night (1 and 0) in the dataframe so that the
createCountPlot function can handle the input
modelValues_insects_hortus <- modelValues_insects_hortus[
  with(modelValues_insects_hortus, order(fullTreatment, -xtfrm(dayOrNight))),
]

# Place the corresponding p values (p-values of v2 during day and night, and other
treatments compared to  v2 during either day and night) in a vector
pVector <- as.data.frame(coef(summary(M_insects_hortus2))$cond[, -3])[c(2, 1), 3]
pVector <- append(pVector, as.data.frame(pairs(emmeans(M_insects_hortus2,
c("dayOrNight", "fullTreatment"), type = "response"), adjust = "none"))[c(21, 2,
23, 4, 25, 6, 27, 8, 29, 10, 31, 12, 33, 14, 35, 16, 37, 18), 7])
modelValues_insects_hortus$p <- pVector

# Calculate the upper and lower limits of the standard errors which will be used
for the error bars in the plots
modelValues_insects_hortus$standardErrorUpper <-
modelValues_insects_hortus$response + modelValues_insects_hortus$SE
modelValues_insects_hortus$standardErrorLower <-
modelValues_insects_hortus$response - modelValues_insects_hortus$SE

# Change the lower range of the standard error of certain treatments to 0, since a
log-transformed axis is not able to visualize negative values
modelValues_insects_hortus$standardErrorLower[c(5:7)] <- rep(0, 3)
```

```r
# Use the significanceValues function to create the right significance indicators
(***, **, *, °) for in the plot
significance_insects_hortus <- significanceValues(modelValues_insects_hortus)


# Manually create the plot for this model, since there are some visual issues when
using the function
hortusCounts_insects_plot <- ggplot(modelValues_insects_hortus, aes(x =
fullTreatment, y = response, color = dayOrNight)) +
  guides(color = guide_legend(title = "Day or night")) +
  scale_color_manual(labels = c("Night", "Day"), values = c("darkblue",
"darkorange")) +
  geom_point(size = 2, position = position_dodge(0.5)) +
  scale_y_log10() +
  geom_vline(xintercept = 1.5, linetype = "dashed", alpha = 0.5) +
  geom_errorbar(aes(ymin = standardErrorLower, ymax = standardErrorUpper), width =
0.1, position = position_dodge(0.5)) +
  labs(y = "Counts per hour") +
  scale_x_discrete(name = "Treatment",
                   labels = labels) +
  ggtitle("Hortus Botanicus - Insects") +
  theme_bw() +
  theme(axis.text.x = element_markdown(color = "black", size = 11),
        axis.text.y = element_text(size = 11),
        axis.title = element_text(size = 14),
        legend.text = element_text(size = 11),
        legend.title = element_text(size = 14),
        plot.title = element_text(hjust = 0.5)) +
  geom_text(aes(label = significance_insects_hortus, group = dayOrNight),
            position = position_dodge(width = 1.7), size = 6,
            vjust = c(rep(0.5, 16), 1, 0.5, 0.5, -0.52),
            show.legend = FALSE) +
  geom_text(aes(label = round(response, 1)),
            position = position_dodge(width = 1.75),
            vjust = c(rep(1, 16), 1.75, 1, 1, -0.65),
            show.legend = FALSE)

hortusCounts_insects_plot


# Save the plot
ggsave("output/figures/hortusCounts_insects_plot.png", width = 520, height = 370,
units = "px", scale = 4.7)


#### Diptera model ####

# Creating a model for diptera:
```

```
M_diptera_hortus <- glmmTMB(n ~ dayCount_normalized + dayOrNight + fullTreatment +
(1 | cameraID), data = hortusCounts_diptera, ziformula = ~ dayCount_normalized +
dayOrNight + (1 | cameraID), family = nbinom2, control = glmmTMBControl(optimizer =
optim, optArgs = list(method = "BFGS")))

# drop1(M_diptera_hortus)
# # Model is already optimized



testDispersion(M_diptera_hortus, alternative = "greater")
# Dispersion non-significant (p = 0.48)
plot(simulateResiduals(M_diptera_hortus))
# KS test non-significant (p = 0.414)
testOutliers(M_diptera_hortus, type = "bootstrap")
# Outlier test non-significant (p = 0.44 (May differ upon repeat due to the nature
of a bootstrapped test)

r2_zeroinflated(M_diptera_hortus)
# R2 = 0.172, R2 adj. = 0.168

modelValues_diptera_hortus <- as.data.frame(emmeans(M_diptera_hortus,
c("dayOrNight", "fullTreatment"), type = "response"))

modelValues_diptera_hortus <- modelValues_diptera_hortus[
  with(modelValues_diptera_hortus, order(fullTreatment, -xtfrm(dayOrNight))),
]

pVector <- as.data.frame(coef(summary(M_diptera_hortus))$cond[, -3])[c(2, 1), 3]
pVector <- append(pVector, as.data.frame(pairs(emmeans(M_diptera_hortus,
c("dayOrNight", "fullTreatment"), type = "response"), adjust = "none"))[c(21, 2,
23, 4, 25, 6, 27, 8, 29, 10, 31, 12, 33, 14, 35, 16, 37, 18), 7])
modelValues_diptera_hortus$p <- pVector

modelValues_diptera_hortus$standardErrorUpper <-
modelValues_diptera_hortus$response + modelValues_diptera_hortus$SE
modelValues_diptera_hortus$standardErrorLower <-
modelValues_diptera_hortus$response - modelValues_diptera_hortus$SE

# Change the lower range of the standard error of all treatments to 0, since a log-
transformed axis is not able to visualize negative values
modelValues_diptera_hortus$standardErrorLower <- 0


significance_diptera_hortus <- significanceValues(modelValues_diptera_hortus)

hortusCounts_diptera_plot <- createCountPlot(modelValues_diptera_hortus,
significance_diptera_hortus, "Hortus Botanicus - Diptera")
```

```
hortusCounts_diptera_plot


# Save the plot
ggsave("output/figures/hortusCounts_diptera_plot.png", width = 520, height = 370,
units = "px", scale = 4.7)


#### Hymenoptera model ####

# Creating a model for hymenoptera:
# M_hymenoptera_hortus <- glmmTMB(n ~ dayCount_normalized + dayOrNight + (1 |
cameraID), data = hortusCounts_hymenoptera, ziformula = ~ dayCount_normalized +
dayOrNight + (1 | cameraID), family = nbinom2, control = glmmTMBControl(optimizer =
optim, optArgs = list(method = "BFGS")))
#
# drop1(M_hymenoptera_hortus)
M_hymenoptera_hortus2 <- glmmTMB(n ~ dayCount_normalized + (1 | cameraID), data =
hortusCounts_hymenoptera, ziformula = ~., family = nbinom2, control =
glmmTMBControl(optimizer = optim, optArgs = list(method = "BFGS")))
# Not able to create a sensible model that includes treatments due to lack of data


testDispersion(M_hymenoptera_hortus2, alternative = "greater")
# Dispersion non-significant (p = 0.524)
plot(simulateResiduals(M_hymenoptera_hortus2))
# KS test non-significant (p = 0.101)
testOutliers(M_hymenoptera_hortus2, type = "bootstrap")
# Outlier test non-significant (p = 1) (May differ upon repeat due to the nature of
a bootstrapped test)

r2_zeroinflated(M_hymenoptera_hortus2)
# R2 = 0.014, R2 adj. = 0.013


#### Lepidoptera model ####
# Creating a model for lepidoptera:
M_lepidoptera_hortus <- glmmTMB(n ~ dayCount_normalized + dayOrNight + (1 |
cameraID), data = hortusCounts_lepidoptera, ziformula = ~., family = nbinom2,
control = glmmTMBControl(optimizer = optim, optArgs = list(method = "BFGS")))

# drop1(M_lepidoptera_hortus)
# # Gives errors, therefore keeping this model


testDispersion(M_lepidoptera_hortus, alternative = "greater")
# Dispersion non-significant (p = 0.192)
plot(simulateResiduals(M_lepidoptera_hortus))
# KS test non-significant (p = 0.325)
```

```
testOutliers(M_lepidoptera_hortus, type = "bootstrap")
# Outlier test non-significant (p = 0.36) (May differ upon repeat due to the nature
of a bootstrapped test)

r2_zeroinflated(M_lepidoptera_hortus)
# R2 = 0.000, R2 adj. = -0.001
```

Script 4: Statistics Ooijpolder model

```
library(plyr)
library(tidyverse)
library(lubridate)
library(httr)
library(jsonlite)
library(cowplot)
library(glmmTMB)
library(DHARMa)
library(ggtext)
library(ggh4x)
library(performance)
library(emmeans)

##### Ooijpolder #####

# Start by defining the camera numbers of interest
cameraNumbers <- c("222", "223", "224", "225", "227", "228")

# The following code individually loads each camera's dataset, removes all rows
from before the starting date of the experiment, assigns the corresponding camera
number (cameraID) to each row, and finally combines all datasets into one
ooijpolderData table
ooijpolderData <- as.data.frame(c())
for (i in cameraNumbers) {
  currentDataset <- paste0("DIOPSIS", i, "_data")
  assign(currentDataset, read.csv(paste0("input/rawData/RUN-Ooijpolder-2022-2023-
20230727T105952Z-001/RUN-Ooijpolder-2022-2023/Camera ", i, "_2023-", i,
"_data_dedup.csv")))
  assign(currentDataset, subset(eval(parse(text = currentDataset)), uid >
20230531000000))
  assign(currentDataset, cbind(eval(parse(text = currentDataset)), "cameraID" = i))
  ooijpolderData <- rbind(ooijpolderData, eval(parse(text = currentDataset)))
}


# Import a file containing the treatment intervals of each camera
treatmentDates_Ooijpolder <-
read.csv2("input/rawData/treatmentDates_Ooijpolder.csv")

# The following steps will assign the correct treatment to each individual row in
the ooijpolderData dataset
# First, create a table that has all treatments with their time intervals nested by
cameraID
nestedIDs <- treatmentDates_Ooijpolder %>%
  mutate(across(c("startDateTime", "endDateTime"), ymd_hms)) %>%
  group_nest(cameraID)
# Change the class of the cameraIDs to integer so that it is compatible in the
following steps
```

```
ooijpolderData$cameraID <- as.integer(ooijpolderData$cameraID)

# Change uid to a usable date format in a new dateTime column
ooijpolderData <- ooijpolderData %>%
  mutate(dateTime = uid) %>%
  mutate(across("dateTime", ymd_hms))

# Add a new column to ooijpolderData that contains each row's treatment, indicated
by the treatment's row number in the nestedIDs table
ooijpolderData <- left_join(x = ooijpolderData, y = nestedIDs, by = "cameraID") %>%
  as_tibble() %>%
  mutate(treatmentNumber = map2(dateTime, data, ~which(.x %within%
interval(.y$startDateTime, .y$endDateTime)))) %>%
  select(-data)
# Change the class of the treatment number to integer make it compatible in the
next steps
ooijpolderData$treatmentNumber <- as.integer(ooijpolderData$treatmentNumber)


# Remove the rows containing NAs in the treatmentNumber column from ooijpolderData,
since this indicates that there was no treatment or no treatment of interest during
those time intervals
ooijpolderData <- subset(ooijpolderData, !is.na(treatmentNumber))

# Add four new columns to ooijpolderData to hold the actual treatments of each row
addition <- data.frame(matrix(nrow = nrow(ooijpolderData), ncol = 4))
ooijpolderData <- cbind(ooijpolderData, addition)
colnames(ooijpolderData)[(ncol(ooijpolderData) - 3):ncol(ooijpolderData)] <-
c("screenColor", "screenTexture", "screenOrientation", "screenVersion")

# Add the treatments to these new columns. The treatment is picked from
treatmentDates_Ooijpolder
for (i in c(1:nrow(ooijpolderData))) {
  ooijpolderData[i, c((ncol(ooijpolderData) - 3):ncol(ooijpolderData))] <-
subset(treatmentDates_Ooijpolder, cameraID == ooijpolderData[i,
]$cameraID)[ooijpolderData[i, ]$treatmentNumber, ][2:5]
}



# Import a file containing the uptime intervals of each camera
uptimeDates_Ooijpolder <- read.csv2("input/rawData/uptimeDates_Ooijpolder.csv")

# The following steps will remove any rows that contain data outside of the uptime
intervals of the cameras. This is done in order to remove any data gathered from
photos that were taken during repairs/troubleshooting of the camera.
# First, create a table that has all uptime intervals nested by cameraID
nestedIDs <- uptimeDates_Ooijpolder %>%
  mutate(across(c("startDateTime", "endDateTime"), ymd_hms)) %>%
```

```
  group_nest(cameraID)

# Add a new column to ooijpolderData that contains the status of the camera (1 =
up, 0 = down) for each row
ooijpolderData <- left_join(x = ooijpolderData, y = nestedIDs, by = "cameraID") %>%
  as_tibble() %>%
  mutate(cameraStatus = map2_int(dateTime, data, ~+any(.x %within%
interval(.y$startDateTime, .y$endDateTime)))) %>%
  select(-data)

# Select only those rows containing data when the cameras were up
ooijpolderData <- subset(ooijpolderData, cameraStatus == 1)




# To implement date in the linear model, we will use several proxies for time
# First, create a column containing the day-number of the experiment. The count
starts with 31-05-2023 as day 1.
ooijpolderData$dayCount <- as.integer(as.Date(ooijpolderData$dateTime) -
as.Date("2023-05-31")) + 1
# Then, create a column that contains the hour-number of that day, with hour 1
starting at 00:00 of each day.
ooijpolderData$hourCount <- as.integer(substr(format(ooijpolderData$dateTime, "%Y-
%m-%d %H:%M:%OS"), 12, 13)) + 1


# In the uptimeDates_Ooijpolder dataset, add new columns that contain new dates
rounded up and down to the nearest hour. The following steps will use this data to
remove any data that is not part of a 1 hour time interval during which the camera
was online.
uptimeDates_Ooijpolder$startDateTime_oneHour <-
ceiling_date(ymd_hms(uptimeDates_Ooijpolder$startDateTime), "1 hour")
uptimeDates_Ooijpolder$endDateTime_oneHour <-
floor_date(ymd_hms(uptimeDates_Ooijpolder$endDateTime), "1 hour")

# Update nestedIDs to include the newly created columns
nestedIDs <- uptimeDates_Ooijpolder %>%
  group_nest(cameraID)

# For each row, add an indicator whether the row should be kept or not (0 =
discard, 1 = keep) based on whether the row is part of a complete 1 hour interval.
ooijpolderData <- left_join(x = ooijpolderData, y = nestedIDs, by = "cameraID") %>%
  as_tibble() %>%
  mutate(cameraKeepUptime = map2_int(dateTime, data, ~+any(.x %within%
interval(.y$startDateTime_oneHour, .y$endDateTime_oneHour)))) %>%
  select(-data)
```

```
# Repeat the process of rounding up and down to the nearest full hour for the
treatment dates.
treatmentDates_Ooijpolder$startDateTime_oneHour <-
ceiling_date(ymd_hms(treatmentDates_Ooijpolder$startDateTime), "1 hour")
treatmentDates_Ooijpolder$endDateTime_oneHour <-
floor_date(ymd_hms(treatmentDates_Ooijpolder$endDateTime), "1 hour")

nestedIDs <- treatmentDates_Ooijpolder %>%
  group_nest(cameraID)

ooijpolderData <- left_join(x = ooijpolderData, y = nestedIDs, by = "cameraID") %>%
  as_tibble() %>%
  mutate(cameraKeepTreatment = map2_int(dateTime, data, ~+any(.x %within%
interval(.y$startDateTime_oneHour, .y$endDateTime_oneHour)))) %>%
  select(-data)

# Remove the the data points that are not part of a complete 1 hour cycle
ooijpolderData <- subset(ooijpolderData, cameraKeepUptime == 1 &
cameraKeepTreatment == 1)

# The following code is commented because it uses an API and will therefore not
function without an internet connection or when the API is down. An Rda file of
taxaTable_ooijpolder created on 18-09-2023 will be loaded instead. It is also
possible to un-comment this code in order to create the taxaTable_ooijpolder
dataframe yourself.

# # I will now proceed to list the correct names of the taxa levels of each
individual. The original dataset contains an object_order column that contains only
the lowest taxon level name (e.g. Chironomidae), which is not very useful for data
analysis.
# # First, create a list that contains all the different taxon names in the
object_order column.
# object_orderFixed <- levels(as.factor(ooijpolderData$object_order))
# # Change the taxa names that deviate from the standard format
# object_orderFixed <- sub("Auchenorrhyncha indet.", "Auchenorrhyncha",
object_orderFixed)
#
# # Create a table for holding all the correct taxa levels of each taxon name
# taxa <- c("kingdom", "phylum", "class", "order", "family", "genus", "species")
# taxaTable_ooijpolder <- data.frame(matrix(NA, nrow = length(object_orderFixed),
ncol = length(taxa)))
# colnames(taxaTable_ooijpolder) <- taxa
#
# # Fill taxaTable_ooijpolder with the correct taxa levels/names of each taxon name
given in object_order. This is done using the GBIF (Global Biodiversity Information
Facility) API.
# for (i in object_orderFixed) {
#   answer <-  GET("https://api.gbif.org/v1/species",
```

```
#                   query = list(name = i))
#   answer_json <- fromJSON(rawToChar(answer$content))
#   tempTaxa <- c()
#   for (j in taxa) {
#     if (is.null(answer_json[["results"]][[j]][1])) {
#       tempTaxa <- append(tempTaxa, NA)
#     } else {
#       tempTaxa <- append(tempTaxa, answer_json[["results"]][[j]][1])
#     }
#   }
#   taxaTable_ooijpolder[match(i, object_orderFixed), ] <- tempTaxa
# }
#
# taxaTable_ooijpolder$object_order <-
levels(as.factor(ooijpolderData$object_order))
#
# # Save taxaTable_ooijpolder in order to use it without having to access the api
# save(taxaTable_ooijpolder, file = "taxaTable_ooijpolder.Rda")

load("input/RDAFiles/taxaTable_ooijpolder.Rda")

# Add the columns from taxaTable_ooijpolder to ooijpolderData
ooijpolderData <- ooijpolderData %>%
  left_join(taxaTable_ooijpolder, by = "object_order")


# The following code is commented because it uses an API and will therefore not
function without an internet connection or when the API is down. An Rda file of
ooijpolderSunData created on 18-09-2023 will be loaded instead. It is also possible
to un-comment this code in order to create the ooijpolderSunData dataframe
yourself.
# # Add a variable that indicates whether the individual was observed during the
day (0) or the night (1), based on sunrise and sunset
# ooijpolderLat <- c(51.8553, 51.8347, 51.8444, 51.8519, 51.8402, 51.8474)
# ooijpolderLon <- c(5.926, 5.9943, 5.9423, 5.9339, 5.959, 5.9618)
#
# ooijpolderSunData <- data.frame(cameraID = rep(cameraNumbers, each =
length(levels(as.factor(format(ooijpolderData$dateTime, "%Y-%m-%d"))))),
#                                 date =
rep(levels(as.factor(format(ooijpolderData$dateTime, "%Y-%m-%d"))),
length(cameraNumbers)),
#                                 lat = rep(ooijpolderLat, each =
length(levels(as.factor(format(ooijpolderData$dateTime, "%Y-%m-%d"))))),
#                                 lon = rep(ooijpolderLon, each =
length(levels(as.factor(format(ooijpolderData$dateTime, "%Y-%m-%d"))))),
#                                 sunrise = NA,
#                                 sunset = NA)
#
#
```

```
# for (i in c(1:nrow(ooijpolderSunData))) {
#   answer <- GET("https://api.sunrise-sunset.org/json",
#                 query = list(lat = ooijpolderSunData$lat[i],
#                              lng = ooijpolderSunData$lon[i],
#                              date = ooijpolderSunData$date[i]))
#   answer_json <- fromJSON(rawToChar(answer$content))
#   ooijpolderSunData$sunrise[i] <- paste(ooijpolderSunData$date[i],
format(strptime(answer_json[["results"]][["sunrise"]], "%I:%M:%S %p") + hours(2),
"%H:%M:%S"))
#   ooijpolderSunData$sunset[i] <- paste(ooijpolderSunData$date[i],
format(strptime(answer_json[["results"]][["sunset"]], "%I:%M:%S %p") + hours(2),
"%H:%M:%S"))
# }
# ooijpolderSunData$sunrise <- as.POSIXct(ooijpolderSunData$sunrise, tz = "UTC")
# ooijpolderSunData$sunset <- as.POSIXct(ooijpolderSunData$sunset, tz = "UTC")
#
# ooijpolderSunData$sunrise_oneHour <- round_date(ooijpolderSunData$sunrise, "1
hour")
# ooijpolderSunData$sunset_oneHour <- round_date(ooijpolderSunData$sunset, "1
hour")
#
# ooijpolderSunData$cameraID <- as.integer(ooijpolderSunData$cameraID)
#
# # Save ooijpolderSunData in order to use it without having to access the api
# save(ooijpolderSunData, file = "ooijpolderSunData.Rda")

load("input/RDAFiles/ooijpolderSunData.Rda")

nestedIDs <- ooijpolderSunData %>%
  group_nest(cameraID)

ooijpolderData <- left_join(x = ooijpolderData, y = nestedIDs, by = c("cameraID"))
%>%
  as_tibble() %>%
  mutate(dayOrNight = map2_int(dateTime, data, ~+any(.x %within%
interval(.y$sunrise_oneHour, .y$sunset_oneHour)))) %>%
  select(-data)

# Also add an extra column containing the full treatment
fullTreatment <- c()
for (i in c(1:nrow(ooijpolderData))) {
  treatmentName <- c()
  if (!is.na(ooijpolderData$screenColor[i])) {
    treatmentName <- paste0(treatmentName, paste0(ooijpolderData$screenColor[i],
"_"))
  }
  if (!is.na(ooijpolderData$screenOrientation[i])) {
```

```
    treatmentName <- paste0(treatmentName,
paste0(ooijpolderData$screenOrientation[i], "_"))
  }
  if (!is.na(ooijpolderData$screenTexture[i])) {
    treatmentName <- paste0(treatmentName, paste0(ooijpolderData$screenTexture[i]))
  }
  if (!is.na(ooijpolderData$screenVersion[i])) {
    treatmentName <- paste0(treatmentName, paste0(ooijpolderData$screenVersion[i]))
  }
  fullTreatment <- append(fullTreatment, treatmentName)
}

ooijpolderData$fullTreatment <- fullTreatment



# My goal is to obtain the average number of individuals per hour However, since
ooijpolderData only contains entries of detected individuals, I still need to
create a dataset that contains the hours during which nothing was caught. This is
what is done during the following steps.
# First, create a table that holds every single 1 hour mark well within the
experimental period (in this case, from May 31st until July 5th).
ooijpolderNoData <- data.frame(cameraID = as.numeric(rep(cameraNumbers,
                                          each = length(seq(as.POSIXct("2023-05-31
00:00:00",
                                                                                for
mat = "%Y-%m-%d %H:%M:%OS",
                                                                                    tz =
"UTC"),
                                                                            +
as.POSIXct("2023-07-05 23:59:00",
                                                                                f
ormat = "%Y-%m-%d %H:%M:%OS",
                                                                                t
z = "UTC"),
                                                                            by = "1
hour")))),
                            dateTime = rep(seq(as.POSIXct("2023-05-31 00:00:00",
                                                                    format = "%Y-%m-%d
%H:%M:%OS",
                                                                tz = "UTC"),
                                                            + as.POSIXct("2023-07-05
23:59:00",
                                                                    format = "%Y-%m-
%d %H:%M:%OS",
                                                                    tz = "UTC"),
                                                            by = "1 hour"), 6))

# Just like before, add the count of the day and hour to the table.
```

```
ooijpolderNoData$dayCount <- as.integer(as.Date(ooijpolderNoData$dateTime) -
as.Date("2023-05-31")) + 1
ooijpolderNoData$hourCount <- as.integer(substr(format(ooijpolderNoData$dateTime,
"%Y-%m-%d %H:%M:%OS"), 12, 13)) + 1

# Select and filter the 1 hour intervals on which the cameras were actually on and
treated
nestedIDs <- uptimeDates_Ooijpolder %>%
  group_nest(cameraID)

ooijpolderNoData <- left_join(x = ooijpolderNoData, y = nestedIDs, by = "cameraID")
%>%
  as_tibble() %>%
  mutate(cameraKeepUptime = map2_int(dateTime, data, ~+any(.x %within%
interval(.y$startDateTime_oneHour, .y$endDateTime_oneHour - 1)))) %>%
  select(-data)


nestedIDs <- treatmentDates_Ooijpolder %>%
  group_nest(cameraID)

ooijpolderNoData <- left_join(x = ooijpolderNoData, y = nestedIDs, by = "cameraID")
%>%
  as_tibble() %>%
  mutate(cameraKeepTreatment = map2_int(dateTime, data, ~+any(.x %within%
interval(.y$startDateTime_oneHour, .y$endDateTime_oneHour - 1)))) %>%
  select(-data)


ooijpolderNoData <- subset(ooijpolderNoData, cameraKeepUptime == 1 &
cameraKeepTreatment == 1)

nestedIDs <- ooijpolderSunData %>%
  group_nest(cameraID)

ooijpolderNoData <- left_join(x = ooijpolderNoData, y = nestedIDs, by =
c("cameraID")) %>%
  as_tibble() %>%
  mutate(dayOrNight = map2_int(dateTime, data, ~+any(.x %within%
interval(.y$sunrise_oneHour, .y$sunset_oneHour)))) %>%
  select(-data)

# Create a function that creates an overview of the hourly counts of a given
subset.
countOoijpolder <- function() {
  # Create a new dataset from ooijpolderNoData with the rows that are already
represented in the subset removed
  ooijpolderNoData_alt <- anti_join(ooijpolderNoData, ooijpolderSubset, by =
c("cameraID", "dayCount", "hourCount"))
```

```
  # For the remaining data, add columns containing the treatment
  nestedIDs <- treatmentDates_Ooijpolder %>%
    mutate(across(c("startDateTime", "endDateTime"), ymd_hms)) %>%
    group_nest(cameraID)

  ooijpolderNoData_alt <- left_join(x = ooijpolderNoData_alt, y = nestedIDs, by =
"cameraID") %>%
    as_tibble() %>%
    mutate(treatmentNumber = map2(dateTime, data, ~which(.x %within%
interval(.y$startDateTime, .y$endDateTime)))) %>%
    select(-data)

  ooijpolderNoData_alt$treatmentNumber <-
as.integer(ooijpolderNoData_alt$treatmentNumber)

  ooijpolderNoData_alt <- subset(ooijpolderNoData_alt, !is.na(treatmentNumber))

  addition <- data.frame(matrix(nrow = nrow(ooijpolderNoData_alt), ncol = 5))
  ooijpolderNoData_alt <- cbind(ooijpolderNoData_alt, addition)
  colnames(ooijpolderNoData_alt)[(ncol(ooijpolderNoData_alt) -
4):ncol(ooijpolderNoData_alt)] <- c("screenColor", "screenTexture",
"screenOrientation", "screenVersion", "fullTreatment")

  for (i in c(1:nrow(ooijpolderNoData_alt))) {
    ooijpolderNoData_alt[i, c((ncol(ooijpolderNoData_alt) -
3):ncol(ooijpolderNoData_alt) - 1)] <- subset(treatmentDates_Ooijpolder, cameraID
== ooijpolderNoData_alt[i, ]$cameraID)[ooijpolderNoData_alt[i, ]$treatmentNumber,
][2:5]
  }

  # Also add an extra column containing the full treatment
  fullTreatment <- c()
  for (i in c(1:nrow(ooijpolderNoData_alt))) {
    treatmentName <- c()
    if (!is.na(ooijpolderNoData_alt$screenColor[i])) {
      treatmentName <- paste0(treatmentName,
paste0(ooijpolderNoData_alt$screenColor[i], "_"))
    }
    if (!is.na(ooijpolderNoData_alt$screenOrientation[i])) {
      treatmentName <- paste0(treatmentName,
paste0(ooijpolderNoData_alt$screenOrientation[i], "_"))
    }
    if (!is.na(ooijpolderNoData_alt$screenTexture[i])) {
      treatmentName <- paste0(treatmentName,
paste0(ooijpolderNoData_alt$screenTexture[i]))
    }
    if (!is.na(ooijpolderNoData_alt$screenVersion[i])) {
```

```
      treatmentName <- paste0(treatmentName,
paste0(ooijpolderNoData_alt$screenVersion[i]))
    }
    fullTreatment <- append(fullTreatment, treatmentName)
  }

  ooijpolderNoData_alt$fullTreatment <- fullTreatment

  # Select the relevant columns from ooijpolderNoData_alt and after that add a
column n containing only zeroes, since all rows represent a count of zero
individuals
  ooijpolderNoData_alt <- ooijpolderNoData_alt[, c("cameraID", "dayCount",
"hourCount", "screenColor", "screenTexture", "screenOrientation", "screenVersion",
"fullTreatment", "dayOrNight")]
  ooijpolderNoData_alt <- cbind(ooijpolderNoData_alt, n = 0)

  # Create another dataset with the counts of the selected subset aggregated by
hour
  ooijpolderCounts <- ooijpolderSubset %>%
    count(cameraID, dayCount, hourCount, screenColor, screenTexture,
screenOrientation, screenVersion, fullTreatment, dayOrNight)
  # Bind the ooijpolderNoData_alt dataset to this dataset
  ooijpolderCounts <- rbind(ooijpolderCounts, ooijpolderNoData_alt)

  # Assign the correct data types to the columns
  ooijpolderCounts$n <- as.integer(ooijpolderCounts$n)
  ooijpolderCounts$dayOrNight <- as.factor(ooijpolderCounts$dayOrNight)
  ooijpolderCounts$fullTreatment <- factor(ooijpolderCounts$fullTreatment, levels =
c("vertical_v2", "vertical_v1", "blue_vertical_rough", "blue_vertical_smooth",
"white_vertical_rough", "white_vertical_smooth", "yellow_vertical_rough",
"yellow_vertical_smooth", "yellow_horizontal_rough", "yellow_horizontal_smooth"))
  ooijpolderCounts$cameraID <- as.factor(ooijpolderCounts$cameraID)

  # Add a column containing the normalized day count
  ooijpolderCounts$dayCount_normalized <- scale(ooijpolderCounts$dayCount)

  ooijpolderCounts
}


# # Create a function that makes a plot grid containing plotted mean counts divided
across several variables. This section is commented to save time since I do not
require these plot grids for my final product.
# plotsOoijpolder <- function(counts, plotText){
#   ###############
#   # Create a subset of only the vertical screens
#   ooijpolderCounts_newScreens <- subset(counts, !is.na(screenColor) &
screenOrientation == "vertical")
#
```

```
#   # Calculate the means and standard deviations of the counts of grouped screen
color, screen texture, and day/night variables, and place these in a new dataset
#   ooijpolderCounts_newScreens_meanSD <- ooijpolderCounts_newScreens %>%
#     group_by(screenColor, screenTexture, dayOrNight) %>%
#     summarise_at(vars(n), list(mean = mean, sd = sd)) %>%
#     as.data.frame()
#
#   # Create a plot to visualize these calculated means
#   plot1 <- ggplot(ooijpolderCounts_newScreens_meanSD, aes(x = screenColor, y =
mean)) +
#     geom_bar(stat = "identity") +
#     geom_errorbar(aes(ymin = mean - sd, ymax = mean + sd), width = 0.2) +
#     facet_grid(cols = vars(screenTexture, dayOrNight), labeller =
labeller(dayOrNight = c("0" = "Night",
#
        "1" = "Day")))
#   ###############
#
#   # Repeat these steps with modifications to the selected data, but with the same
general code
#   ###############
#   ooijpolderCounts_orientation <- subset(counts, screenColor == "yellow")
#
#   ooijpolderCounts_orientation_meanSD <- ooijpolderCounts_orientation %>%
#     group_by(screenOrientation, screenTexture, dayOrNight) %>%
#     summarise_at(vars(n), list(mean = mean, sd = sd)) %>%
#     as.data.frame()
#
#   plot2 <- ggplot(ooijpolderCounts_orientation_meanSD, aes(x = screenOrientation,
y = mean)) +
#     geom_bar(stat = "identity") +
#     geom_errorbar(aes(ymin = mean - sd, ymax = mean + sd), width = 0.2) +
#     facet_grid(cols = vars(screenTexture, dayOrNight), labeller =
labeller(dayOrNight = c("0" = "Night",
#
        "1" = "Day")))
#   ###############
#
#   ###############
#   ooijpolderCounts_oldScreens <- subset(counts, !is.na(screenVersion))
#
#   ooijpolderCounts_oldScreens_meanSD <- ooijpolderCounts_oldScreens %>%
#     group_by(screenVersion, dayOrNight) %>%
#     summarise_at(vars(n), list(mean = mean, sd = sd)) %>%
#     as.data.frame()
#
#   plot3 <- ggplot(ooijpolderCounts_oldScreens_meanSD, aes(x = screenVersion, y =
mean)) +
```

```
#     geom_bar(stat = "identity") +
#     geom_errorbar(aes(ymin = mean - sd, ymax = mean + sd), width = 0.2) +
#     facet_grid(cols = vars(dayOrNight), labeller = labeller(dayOrNight = c("0" =
"Night",
#                                                                              "1" =
"Day")))
#   ###############
#   # Combine the three plots into one grid
#   plot_grid(ggdraw() +
#               draw_label(
#                 plotText,
#                 fontface = 'bold',
#                 x = 0,
#                 hjust = 0
#               ) +
#               theme(
#                 plot.margin = margin(0, 0, 0, 7)
#               ), plot1, plot2, plot3)
# }


# Create subsets of interest. The code generating the plots is commented to save
time since I do not require them for my final product.
ooijpolderSubset <- subset(ooijpolderData, class == "Insecta" &
object_order_probability >= 0.5)
ooijpolderCounts_insects <- countOoijpolder()

# ooijpolderPlots_insects <- plotsOoijpolder(ooijpolderCounts_insects, "All
insects")


ooijpolderSubset <- subset(ooijpolderData, order == "Lepidoptera" &
object_order_probability >= 0.5)
ooijpolderCounts_lepidoptera <- countOoijpolder()

# ooijpolderPlots_lepidoptera <- plotsOoijpolder(ooijpolderCounts_lepidoptera,
"Order: Lepidoptera")


ooijpolderSubset <- subset(ooijpolderData, order == "Hymenoptera" &
object_order_probability >= 0.5)
ooijpolderCounts_hymenoptera <- countOoijpolder()

# ooijpolderPlots_hymenoptera <- plotsOoijpolder(ooijpolderCounts_hymenoptera,
"Order: Hymenoptera")


ooijpolderSubset <- subset(ooijpolderData, order == "Diptera" &
object_order_probability >= 0.5)
```

```
ooijpolderCounts_diptera <- countOoijpolder()

# ooijpolderPlots_diptera <- plotsOoijpolder(ooijpolderCounts_diptera, "Order:
Diptera")



# Create distribution plots for taxa of interest to use for determining the model's
distribution family
distributionPlot_insects <- ggplot(ooijpolderCounts_insects, aes(x = n)) +
  geom_bar()

distributionPlot_lepidoptera <- ggplot(ooijpolderCounts_lepidoptera, aes(x = n)) +
  geom_bar()

distributionPlot_hymenoptera <- ggplot(ooijpolderCounts_hymenoptera, aes(x = n)) +
  geom_bar()

distributionPlot_diptera <- ggplot(ooijpolderCounts_diptera, aes(x = n)) +
  geom_bar()



# Create a function to create a vector to hold the significance indicators for the
comparisons
significanceValues <- function(calculatedModelValues) {
  significance <- c()
  for (i in c(1:nrow(calculatedModelValues))) {
    if (calculatedModelValues$p[i] <= 0.001) {
      significance <- c(significance, "***")
    } else if (calculatedModelValues$p[i] <= 0.01) {
      significance <- c(significance, "**")
    } else if (calculatedModelValues$p[i] <= 0.05) {
      significance <- c(significance, "*")
    } else if (calculatedModelValues$p[i] <= 0.1) {
      significance <- c(significance, "°")
    } else {
      significance <- c(significance, "")
    }
  }
  significance
}



# I will also include visual indicators (pictures) for the plot's x-axis tick
labels.
# Get a vector with the treatments.
treatments <- levels(ooijpolderCounts_insects$fullTreatment)
```

```
# Write HTML code for additional text underneath the visual indicators.
treatmentsLabels <- c("v2<br>vertical", "v1<br>vertical",
"Blue<br>rough<br>vertical", "Blue<br>smooth<br>vertical",
"White<br>rough<br>vertical", "White<br>smooth<br>vertical",
"Yellow<br>rough<br>vertical", "Yellow<br>smooth<br>vertical",
"Yellow<br>rough<br>hori-<br>zontal", "Yellow<br>smooth<br>hori-<br>zontal")

# Generate the tick label HTML code using pre-made pictures of the treatments (the
pictures are taken from the HTML table schedule)
labels <- c()
for (i in c(1:length(treatments))) {
  labels <- c(labels, paste0("<img src='input/figures/", treatments[i], ".png", "'
width='40' /><br>", treatmentsLabels[i]))
}


# Finally, create the plot using ggplot
createCountPlot <- function(calculatedModelValues, calculatedSignificanceValues,
plotTitle) {
  tempPlot <- ggplot(calculatedModelValues, aes(x = fullTreatment, y = response,
color = dayOrNight)) +
    guides(color = guide_legend(title = "Day or night")) +
    scale_color_manual(labels = c("Night", "Day"), values = c("darkblue",
"darkorange")) +
    geom_point(size = 2, position = position_dodge(0.5)) +
    scale_y_log10() +
    geom_vline(xintercept = 1.5, linetype = "dashed", alpha = 0.5) +
    geom_errorbar(aes(ymin = standardErrorLower, ymax = standardErrorUpper), width
= 0.1, position = position_dodge(0.5)) +
    labs(y = "Counts per hour") +
    scale_x_discrete(name = "Treatment",
                     labels = labels) +
    ggtitle(plotTitle) +
    theme_bw() +
    theme(axis.text.x = element_markdown(color = "black", size = 11),
          axis.text.y = element_text(size = 11),
          axis.title = element_text(size = 14),
          legend.text = element_text(size = 11),
          legend.title = element_text(size = 14),
          plot.title = element_text(hjust = 0.5)) +
    geom_text(aes(label = calculatedSignificanceValues, group = dayOrNight),
              position = position_dodge(width = 1.7), size = 6,
              show.legend = FALSE) +
    geom_text(aes(label = round(response, 1)),
              position = position_dodge(width = 1.7),
              vjust = 1,
              show.legend = FALSE)
  tempPlot
}
```

```
#### General data figures ####
# Figures to show the total individual counts for both classes and orders
classCounts_ooijpolder <- subset(ooijpolderData, object_order_probability >= 0.5)
%>% mutate_at(c("class", "dayOrNight"), as.factor) %>% count(class, dayOrNight,
.drop = FALSE)
classCounts_ooijpolder$label_y <- NA
counter <- 0
for (i in c(1:(nrow(classCounts_ooijpolder)/2))) {
  counter <- counter + 2
  classCounts_ooijpolder$label_y[(counter - 1): counter] <-
sum(classCounts_ooijpolder$n[(counter - 1): counter])
}

ooijpolderClassCounts_plot <- ggplot() +
  geom_col(data = classCounts_ooijpolder, aes(x = class, y = n, fill = dayOrNight))
+
  guides(fill = guide_legend(title = "Day or night")) +
  scale_fill_manual(labels = c("Night", "Day"), values = c("darkblue",
"darkorange")) +
  ylim(0, 60000) +
  geom_text(data = classCounts_ooijpolder, aes(x = class, y = label_y, color =
dayOrNight, label = n), vjust = rep(c(-2.5, -0.75),
length(classCounts_ooijpolder$n)/2)) +
  scale_color_manual(labels = c("Night", "Day"), values = c("darkblue",
"darkorange"), guide = "none") +
  labs(y = "Total count", x = "Class") +
  ggtitle("Ooijpolder") +
  theme_bw() +
  theme(axis.text = element_text(size = 11),
        axis.title = element_text(size = 14),
        legend.text = element_text(size = 11),
        legend.title = element_text(size = 14),
        plot.title = element_text(hjust = 0.5))

ooijpolderClassCounts_plot

ggsave("output/figures/ooijpolderClassCounts_plot.png", width = 500, height = 270,
units = "px", scale = 4.7)



orderCounts_ooijpolder <- subset(ooijpolderData, class == "Insecta" &
object_order_probability >= 0.5) %>% mutate_at(c("order", "dayOrNight"), as.factor)
%>% count(order, dayOrNight, .drop = FALSE)
orderCounts_ooijpolder$label_y <- NA
counter <- 0
for (i in c(1:(nrow(orderCounts_ooijpolder)/2))) {
```

```
  counter <- counter + 2
  orderCounts_ooijpolder$label_y[(counter - 1): counter] <-
sum(orderCounts_ooijpolder$n[(counter - 1): counter])
}

ooijpoldersOrderCounts_plot <- ggplot() +
  geom_col(data = orderCounts_ooijpolder, aes(x = order, y = n, fill = dayOrNight))
+
  guides(fill = guide_legend(title = "Day or night")) +
  scale_fill_manual(labels = c("Night", "Day"), values = c("darkblue",
"darkorange")) +
  ylim(0, 32000) +
  geom_text(data = orderCounts_ooijpolder, aes(x = order, y = label_y, color =
dayOrNight, label = n), vjust = rep(c(-2.5, -0.75),
length(orderCounts_ooijpolder$n)/2)) +
  scale_color_manual(labels = c("Night", "Day"), values = c("darkblue",
"darkorange"), guide = "none") +
  labs(y = "Total count", x = "Order") +
  ggtitle("Ooijpolder") +
  theme_bw() +
  theme(axis.text = element_text(size = 11),
        axis.title = element_text(size = 14),
        axis.text.x = element_text(angle = 45, hjust = 1),
        legend.text = element_text(size = 11),
        legend.title = element_text(size = 14),
        plot.title = element_text(hjust = 0.5))

ooijpoldersOrderCounts_plot

ggsave("output/figures/ooijpoldersOrderCounts_plot.png", width = 500, height = 270,
units = "px", scale = 4.7)


# A figure to show the number of active hours of the cameras
hourCounts_ooijpolder <- ooijpolderCounts_insects %>% count(cameraID, dayOrNight,
.drop = FALSE)
hourCounts_ooijpolder$label_y <- NA
counter <- 0
for (i in c(1:(nrow(hourCounts_ooijpolder)/2))) {
  counter <- counter + 2
  hourCounts_ooijpolder$label_y[(counter - 1): counter] <-
sum(hourCounts_ooijpolder$n[(counter - 1): counter])
}

ooijpolderHourCounts_plot <- ggplot() +
  geom_col(data = hourCounts_ooijpolder, aes(x = cameraID, y = n, fill =
dayOrNight)) +
  guides(fill = guide_legend(title = "Day or night")) +
```

```
  scale_fill_manual(labels = c("Night", "Day"), values = c("darkblue",
"darkorange")) +
  ylim(0, 800) +
  geom_text(data = hourCounts_ooijpolder, aes(x = cameraID, y = label_y, color =
dayOrNight, label = n), vjust = rep(c(-2.5, -0.75),
length(hourCounts_ooijpolder$n)/2)) +
  scale_color_manual(labels = c("Night", "Day"), values = c("darkblue",
"darkorange"), guide = "none") +
  labs(y = "Hour count", x = "Camera ID") +
  ggtitle("Ooijpolder") +
  theme_bw() +
  theme(axis.text = element_text(size = 11),
        axis.title = element_text(size = 14),
        legend.text = element_text(size = 11),
        legend.title = element_text(size = 14),
        plot.title = element_text(hjust = 0.5))

ooijpolderHourCounts_plot

ggsave("output/figures/ooijpolderHourCounts_plot.png", width = 500, height = 270,
units = "px", scale = 4.7)


#### Insects model ####

# Creating a model for insects:
M_insects_ooijpolder <- glmmTMB(n ~ dayCount_normalized + dayOrNight*fullTreatment
+ (1 + dayCount_normalized | cameraID), data = ooijpolderCounts_insects, ziformula
= ~ dayCount_normalized + (1 + dayCount_normalized | cameraID), family = nbinom2)

# drop1(M_insects_ooijpolder)
# # Removing dayCount_normalized from the fixed effects decreases AIC by 2,
however, this is a very small amount, and doing this results in a convergence
problem:
# M_insects_ooijpolder_1 <- glmmTMB(n ~ + dayOrNight*fullTreatment + (1 +
dayCount_normalized | cameraID), data = ooijpolderCounts_insects, ziformula = ~
dayCount_normalized + (1 + dayCount_normalized | cameraID), family = nbinom2)
# # Therefore, I will keep it in the model.

# Perform some model validation tests as per recommendation by the DHARMa
documentation (https://cran.r-project.org/web/packages/DHARMa/DHARMa.pdf)
testDispersion(M_insects_ooijpolder, alternative = "greater")
# Dispersion non-significant (p = 1)
plot(simulateResiduals(M_insects_ooijpolder))
# KS test non-significant (p = 0.244)
testOutliers(M_insects_ooijpolder, type = "bootstrap")
# Outlier test non-significant (p = 0.16) (May differ upon repeat due to the nature
of a bootstrapped test)
```

```
r2_zeroinflated(M_insects_ooijpolder)
# R2 = 0.998, R2 adj. = 0.998

# Create a dataframe for the predictions of the model
modelValues_insects_ooijpolder <- as.data.frame(emmeans(M_insects_ooijpolder,
c("dayOrNight", "fullTreatment"), type = "response"))

# Reverse the order of day and night (1 and 0) in the dataframe so that the
createCountPlot function can handle the input
modelValues_insects_ooijpolder <- modelValues_insects_ooijpolder[
  with(modelValues_insects_ooijpolder, order(fullTreatment, -xtfrm(dayOrNight))),
]

# Place the corresponding p values (p-values of v2 during day and night, and other
treatments compared to  v2 during either day and night) in a vector
pVector <- as.data.frame(coef(summary(M_insects_ooijpolder))$cond[-2, -3])[c(2, 1),
3]
pVector <- append(pVector, as.data.frame(pairs(emmeans(M_insects_ooijpolder,
c("dayOrNight", "fullTreatment"), type = "response"), adjust = "none"))[c(21, 2,
23, 4, 25, 6, 27, 8, 29, 10, 31, 12, 33, 14, 35, 16, 37, 18), 7])
modelValues_insects_ooijpolder$p <- pVector

# Calculate the upper and lower limits of the standard errors which will be used
for the error bars in the plots
modelValues_insects_ooijpolder$standardErrorUpper <-
modelValues_insects_ooijpolder$response + modelValues_insects_ooijpolder$SE
modelValues_insects_ooijpolder$standardErrorLower <-
modelValues_insects_ooijpolder$response - modelValues_insects_ooijpolder$SE

# Use the significanceValues function to create the right significance indicators
(***, **, *, °) for in the plot
significance_insects_ooijpolder <-
significanceValues(modelValues_insects_ooijpolder)

# Use the createCountPlot function to create the plot
ooijpolderCounts_insects_plot <- createCountPlot(modelValues_insects_ooijpolder,
significance_insects_ooijpolder, "Ooijpolder - Insects")

ooijpolderCounts_insects_plot

# Save the plot
ggsave("output/figures/ooijpolderCounts_insects_plot.png", width = 520, height =
370, units = "px", scale = 4.7)



#### Diptera model ####

# Creating a model for diptera:
```

```
# M_diptera_ooijpolder <- glmmTMB(n ~ dayCount_normalized +
dayOrNight*fullTreatment + (1 + dayCount_normalized | cameraID), data =
ooijpolderCounts_diptera, ziformula = ~ dayCount_normalized + (1 +
dayCount_normalized | cameraID), family = nbinom2)
# # Does not create a usable model
# Use optim instead:
M_diptera_ooijpolder_optim <- glmmTMB(n ~ dayCount_normalized +
dayOrNight*fullTreatment + (1 + dayCount_normalized | cameraID), data =
ooijpolderCounts_diptera, ziformula = ~ dayCount_normalized + dayOrNight + (1 +
dayCount_normalized | cameraID), family = nbinom2, control =
glmmTMBControl(optimizer = optim, optArgs = list(method = "BFGS")))


# drop1(M_diptera_ooijpolder_optim)
# # The current model is already optimal

# Perform some model validation tests as per recommendation by the DHARMa
documentation (https://cran.r-project.org/web/packages/DHARMa/DHARMa.pdf)
testDispersion(M_diptera_ooijpolder_optim, alternative = "greater")
# Dispersion non-significant (p = 0.788)
plot(simulateResiduals(M_diptera_ooijpolder_optim))
# KS test significant (p = 0.024)
testOutliers(M_diptera_ooijpolder_optim, type = "bootstrap")
# Outlier test non-significant (p = 0.5) (May differ upon repeat due to the nature
of a bootstrapped test)

r2_zeroinflated(M_diptera_ooijpolder_optim)
# R2 = 0.994, R2 adj. = 0.994


modelValues_diptera_ooijpolder <- as.data.frame(emmeans(M_diptera_ooijpolder_optim,
c("dayOrNight", "fullTreatment"), type = "response"))

modelValues_diptera_ooijpolder <- modelValues_diptera_ooijpolder[
  with(modelValues_diptera_ooijpolder, order(fullTreatment, -xtfrm(dayOrNight))),
]

pVector <- as.data.frame(coef(summary(M_diptera_ooijpolder_optim))$cond[-2, -
3])[c(2, 1), 3]
pVector <- append(pVector, as.data.frame(pairs(emmeans(M_diptera_ooijpolder_optim,
c("dayOrNight", "fullTreatment"), type = "response"), adjust = "none"))[c(21, 2,
23, 4, 25, 6, 27, 8, 29, 10, 31, 12, 33, 14, 35, 16, 37, 18), 7])
modelValues_diptera_ooijpolder$p <- pVector

modelValues_diptera_ooijpolder$standardErrorUpper <-
modelValues_diptera_ooijpolder$response + modelValues_diptera_ooijpolder$SE
modelValues_diptera_ooijpolder$standardErrorLower <-
modelValues_diptera_ooijpolder$response - modelValues_diptera_ooijpolder$SE
```

```
significance_diptera_ooijpolder <-
significanceValues(modelValues_diptera_ooijpolder)

ooijpolderCounts_diptera_plot <- createCountPlot(modelValues_diptera_ooijpolder,
significance_diptera_ooijpolder, "Ooijpolder - Diptera")

ooijpolderCounts_diptera_plot


# Save the plot
ggsave("output/figures/ooijpolderCounts_diptera_plot.png", width = 520, height =
370, units = "px", scale = 4.7)




#### Hymenoptera model ####

# Creating a model for hymenoptera:
# M_hymenoptera_ooijpolder <- glmmTMB(n ~ dayCount_normalized +
dayOrNight*fullTreatment + (1 + dayCount_normalized | cameraID), data =
ooijpolderCounts_hymenoptera, ziformula = ~ dayCount_normalized + (1 +
dayCount_normalized | cameraID), family = nbinom2)
# # Does not create a usable model
# Use optim instead:
M_hymenoptera_ooijpolder_optim <- glmmTMB(n ~ dayOrNight*fullTreatment + (1 |
cameraID), data = ooijpolderCounts_hymenoptera, ziformula = ~ dayOrNight + (1 |
cameraID), family = nbinom2, control = glmmTMBControl(optimizer = optim, optArgs =
list(method = "BFGS")))

# drop1(M_hymenoptera_ooijpolder_optim)
# # The current model is already optimal

testDispersion(M_hymenoptera_ooijpolder_optim, alternative = "greater")
# Dispersion non-significant (p = 0.128)
plot(simulateResiduals(M_hymenoptera_ooijpolder_optim))
# KS test non-significant (p = 0.331)
testOutliers(M_hymenoptera_ooijpolder_optim, type = "bootstrap")
# Outlier test non-significant (p = 0.96) (May differ upon repeat due to the nature
of a bootstrapped test)

r2_zeroinflated(M_hymenoptera_ooijpolder_optim)
# R2 = 0.855, R2 adj. = 0.854

modelValues_hymenoptera_ooijpolder <-
as.data.frame(emmeans(M_hymenoptera_ooijpolder_optim, c("dayOrNight",
"fullTreatment"), type = "response"))

modelValues_hymenoptera_ooijpolder <- modelValues_hymenoptera_ooijpolder[
```

```
  with(modelValues_hymenoptera_ooijpolder, order(fullTreatment, -
xtfrm(dayOrNight))),
]

pVector <- as.data.frame(coef(summary(M_hymenoptera_ooijpolder_optim))$cond[, -
3])[c(2, 1), 3]
pVector <- append(pVector,
as.data.frame(pairs(emmeans(M_hymenoptera_ooijpolder_optim, c("dayOrNight",
"fullTreatment"), type = "response"), adjust = "none"))[c(21, 2, 23, 4, 25, 6, 27,
8, 29, 10, 31, 12, 33, 14, 35, 16, 37, 18), 7])
modelValues_hymenoptera_ooijpolder$p <- pVector

modelValues_hymenoptera_ooijpolder$standardErrorUpper <-
modelValues_hymenoptera_ooijpolder$response + modelValues_hymenoptera_ooijpolder$SE
modelValues_hymenoptera_ooijpolder$standardErrorLower <-
modelValues_hymenoptera_ooijpolder$response - modelValues_hymenoptera_ooijpolder$SE


# Remove the infinitely/extremely small values and standard errors of the blue
screens during the day
modelValues_hymenoptera_ooijpolder <- modelValues_hymenoptera_ooijpolder[-c(5, 7),
]
significance_hymenoptera_ooijpolder <-
significanceValues(modelValues_hymenoptera_ooijpolder)

# Manually create the plot for this model, since there are some visual issues when
using the function
ooijpolderCounts_hymenoptera_plot <- ggplot(modelValues_hymenoptera_ooijpolder,
aes(x = fullTreatment, y = response, color = dayOrNight)) +
    guides(color = guide_legend(title = "Day or night")) +
    scale_color_manual(labels = c("Night", "Day"), values = c("darkblue",
"darkorange")) +
    geom_point(size = 2, position = position_dodge(0.5)) +
    scale_y_log10() +
    geom_vline(xintercept = 1.5, linetype = "dashed", alpha = 0.5) +
    geom_errorbar(aes(ymin = standardErrorLower, ymax = standardErrorUpper), width
= 0.1, position = position_dodge(0.5)) +
    labs(y = "Counts per hour") +
    scale_x_discrete(name = "Treatment",
                     labels = labels) +
    ggtitle("Ooijpolder - Hymenoptera") +
    theme_bw() +
    theme(axis.text.x = element_markdown(color = "black", size = 11),
          axis.text.y = element_text(size = 11),
          axis.title = element_text(size = 14),
          legend.text = element_text(size = 11),
          legend.title = element_text(size = 14),
          plot.title = element_text(hjust = 0.5)) +
    geom_text(aes(label = significance_hymenoptera_ooijpolder, group = dayOrNight),
```

```
              position = position_dodge(width = 1.7), size = 6,
              vjust = c(rep(0.5, 4), 3.58, 3.58, rep(0.5, 12)),
              show.legend = FALSE) +
    geom_text(aes(label = round(response, 1)),
              position = position_dodge(width = 1.7),
              vjust = c(rep(1, 4), 6, 6, rep(1, 12)),
              show.legend = FALSE)


ooijpolderCounts_hymenoptera_plot


# Save the plot
ggsave("output/figures/ooijpolderCounts_hymenoptera_plot.png", width = 520, height
= 370, units = "px", scale = 4.7)




##### Lepidoptera model #####

# Creating a model for lepidoptera:
# M_lepidoptera_ooijpolder <- glmmTMB(n ~ dayCount_normalized +
dayOrNight*fullTreatment + (1 + dayCount_normalized | cameraID), data =
ooijpolderCounts_lepidoptera, ziformula = ~ dayCount_normalized + (1 +
dayCount_normalized | cameraID), family = nbinom2)
# # Does not create a usable model
# Use optim instead:
M_lepidoptera_ooijpolder_optim <- glmmTMB(n ~ dayOrNight*fullTreatment + (1 +
dayCount_normalized | cameraID), data = ooijpolderCounts_lepidoptera, ziformula = ~
dayOrNight + (1 + dayCount_normalized | cameraID), family = nbinom2, control =
glmmTMBControl(optimizer = optim, optArgs = list(method = "BFGS")))

# drop1(M_lepidoptera_ooijpolder_optim)
# # The current model is already optimal

testDispersion(M_lepidoptera_ooijpolder_optim, alternative = "greater")
# Dispersion non-significant (p = 0.424)
plot(simulateResiduals(M_lepidoptera_ooijpolder_optim))
# KS test significant (p = 0)
testOutliers(M_lepidoptera_ooijpolder_optim, type = "bootstrap")
# Outlier test non-significant (p = 0.64) (May differ upon repeat due to the nature
of a bootstrapped test)

r2_zeroinflated(M_lepidoptera_ooijpolder_optim)
# R2 = 0.994, R2 adj. = 0.994
```

```
modelValues_lepidoptera_ooijpolder <-
as.data.frame(emmeans(M_lepidoptera_ooijpolder_optim, c("dayOrNight",
"fullTreatment"), type = "response"))

modelValues_lepidoptera_ooijpolder <- modelValues_lepidoptera_ooijpolder[
  with(modelValues_lepidoptera_ooijpolder, order(fullTreatment, -
xtfrm(dayOrNight))),
]

pVector <- as.data.frame(coef(summary(M_lepidoptera_ooijpolder_optim))$cond[, -
3])[c(2, 1), 3]
pVector <- append(pVector,
as.data.frame(pairs(emmeans(M_lepidoptera_ooijpolder_optim, c("dayOrNight",
"fullTreatment"), type = "response"), adjust = "none"))[c(21, 2, 23, 4, 25, 6, 27,
8, 29, 10, 31, 12, 33, 14, 35, 16, 37, 18), 7])
modelValues_lepidoptera_ooijpolder$p <- pVector

modelValues_lepidoptera_ooijpolder$standardErrorUpper <-
modelValues_lepidoptera_ooijpolder$response + modelValues_lepidoptera_ooijpolder$SE
modelValues_lepidoptera_ooijpolder$standardErrorLower <-
modelValues_lepidoptera_ooijpolder$response - modelValues_lepidoptera_ooijpolder$SE

significance_lepidoptera_ooijpolder <-
significanceValues(modelValues_lepidoptera_ooijpolder)

ooijpolderCounts_lepidoptera_plot <-
createCountPlot(modelValues_lepidoptera_ooijpolder,
significance_lepidoptera_ooijpolder, "Ooijpolder - Lepidoptera")

ooijpolderCounts_lepidoptera_plot

# Save the plot
ggsave("output/figures/ooijpolderCounts_lepidoptera_plot.png", width = 520, height
= 370, units = "px", scale = 4.7)



# Actual counts vs expected counts
# Set a seed for replication purposes
set.seed(20230905)
# Randomly select 40 insect counts of each color: 20 sampled nighttime and 20 from
daytime
screenSamples <- as.data.frame(c())
for (i in c("yellow", "white", "blue")){
  screenSamples <- rbind(screenSamples, sample_n(subset(ooijpolderCounts_insects,
screenColor == i & dayOrNight == 1), 20))
  screenSamples <- rbind(screenSamples, sample_n(subset(ooijpolderCounts_insects,
screenColor == i & dayOrNight == 0), 20))
}
```

```r
screenSamples <- screenSamples[,-ncol(screenSamples)]
write_excel_csv2(screenSamples, "output/modifiedData/screenSamples.csv")

# Load the dataset with the manual counts added
screenSamples_filled <- read.csv2("input/rawData/screenSamples_filled.csv")
screenSamples_filled$dayOrNight <- as.factor(screenSamples_filled$dayOrNight)

# Create dataframes with manually set x and y-axis limits
df_scales_y <- data.frame(
  Panel = c(1:6),
  ymin = rep(0, 6),
  ymax = c(25, 40, 50, 5, 10, 5),
  n = c(5, 5, 5, 2, 3, 2)
)
df_scales_y <- split(df_scales_y, df_scales_y$Panel)

scales_y <- lapply(df_scales_y, function(x) {
  scale_y_continuous(limits = c(x$ymin, x$ymax), n.breaks = x$n)
})

df_scales_x <- data.frame(
  Panel = c(1:6),
  xmin = rep(0, 6),
  xmax = c(80, 80, 130, 2, 2, 15),
  n = c(5, 5, 5, 2, 2, 3)
)
df_scales_x <- split(df_scales_x, df_scales_x$Panel)

scales_x <- lapply(df_scales_x, function(x) {
  scale_x_continuous(limits = c(x$ymin, x$ymax), n.breaks = x$n)
})

# Add labels to relevant columns (to be used as the facet labels)
screenSamples_filled$screenColor <- factor(screenSamples_filled$screenColor, labels
= c("Blue", "White", "Yellow"))
screenSamples_filled$dayOrNight <- factor(screenSamples_filled$dayOrNight, labels =
c("Night", "Day"))


# Create the plot
screenSamples_filled_plot <- ggplot(screenSamples_filled, aes(x = countedN, y = n,
color = dayOrNight)) +
  geom_point() +
  guides(color = guide_legend(title = "Day or night")) +
  scale_color_manual(labels = c("Night", "Day"), values = c("darkblue",
"darkorange")) +
  scale_fill_manual(labels = c("Night", "Day"), values = c("darkblue",
"darkorange")) +
```

```
  facet_wrap(facets = vars(dayOrNight, screenColor), scales = "free", labeller =
label_parsed) +
  facetted_pos_scales(y = scales_y, x = scales_x) +
  geom_abline() +
  geom_smooth(method = lm, aes(fill = dayOrNight), alpha = 0.2) +
  guides(fill = "none") +
  labs(y = "Counts by the algorithm", x = "Manual counts") +
  theme_bw() +
  theme(axis.text = element_text(size = 11),
        axis.title = element_text(size = 14),
        legend.text = element_text(size = 11),
        legend.title = element_text(size = 14),
        strip.text = element_text(size = 11, margin = margin(t = 0, r = 0, b = 1, l
= 0, unit = "pt")))

screenSamples_filled_plot


# Save the plot
ggsave("output/figures/screenSamples_filled_plot.png", width = 520, height = 350,
units = "px", scale = 4.7)
```