Building an Arduino Drone Attachment to Detect Western Corn Rootworm Beetles Using YOLOv5

Ayush Khot, Jeremiah Lin, Samuel Pasquesi and Pavan Pavithran[1]

Department of Physics PHYS371 & National Center for Supercomputing Applications (NCSA) & College of Agricultural, Consumer and Environmental Sciences
University of Illinois at Urbana-Champaign

E-mail: akhot2@illinois.edu, jl190@illinois.edu, smp6@illinois.edu, pavanp2@illinois.edu

Abstract: Western corn rootworms, also known as *Diabrotica virgifera virgifera* LeConte, are pests found in the Corn Belt, a midwest region of the United States. Rootworms can cause extreme damage to maize and soybeans, with an estimated annual economic loss between $1 and $2 billion. To combat this pest, researchers working with farmers have strategically placed sticky traps over their farmland and use pesticides once the number of western corn rootworm beetles is above a certain action threshold. Currently, they need to manually count the number of beetles on each trap. We sought to create a drone attachment that takes pictures of the traps and uses an object detection algorithm to count the number of beetles. We choose to use You Only Look Once version 5 (YOLOv5), an algorithm known for its high accuracy and speed, then trained it using a simulated dataset by cropping beetles and insects onto trap outlines. We then tested our various model iterations on the simulated dataset, images taken using an iPhone, and images taken using the Arducam on the drone attachment.

[1]Corresponding author.

Contents

1 Introduction

Western corn rootworms (WCR) are pests found in the Corn Belt, a midwest region of the United States. They can cause extreme damage to the yield, especially corn, usually from larval feeding on the roots [1–4]. There have been several methods to manage their population, such as rootworm insecticides and resistant crop strains [5, 6]. Since 2003, transgenic maize that produces insecticidal toxins from the bacterium *Bacillus thuringiensis* (Bt) have been used to manage their population and other important pests [6, 7]. However, six years after the release of Bt maize, the first cases of Bt resistance in the field were recorded [8]. Ever since, there have been many recorded instances of field-evolved resistance to all available Bt traits in western corn rootworms all throughout the US Corn Belt [6, 9–11].

To this day, western corn rootworms are one of the most dangerous pests of maize and soybeans in the US [12–14]. To combat these pests, some farmers work with researchers to assess the corn rootworm activity in their area using sticky traps, then determine if special treatment or crops is appropriate [15]. By strategically placing the sticky traps over the farmland, they can estimate the number of western corn rootworms per trap per day and use pesticides and resistant crop breeds if the number exceeds the action threshold, which then the estimated damage to the crops would affect the yield [16]. With this measurement, a farmer can effectively assess the damage and make purchases of various deters. The current implementation of this method is to have the researchers take the sticky traps back to their lab to count the number of WCR beetles [17].

Upon analyzing this problem, we aimed to create a device so the farmer and researchers could count the number of beetles per trap without walking to each trap. From this, we developed a drone-mountable contraption that can take photos and other readings for a machine-learning model to count the number of western corn rootworms per trap.

This solution was decided upon for multiple reasons. First and foremost, the use of drones in agriculture is already prevalent. Many comprehensive reviews of the uses of drones in agriculture and pest management are given in Ref. [18–20]. Current uses of drones for agricultural purposes include spraying needs, surveying, monitoring, and collecting samples. Many of these users already use the same key hardware (namely the camera), thus deploying this method would not be a large demand for farmers.

Another factor is the consistency of the environment. Crops such as corn and soybeans can grow quite tall, and approaching the traps by ground would prove difficult to reach, as the traps will be nestled deep into the field, and any attempt to access them by land would prove difficult. A drone, however, would not need to traverse by land, and thus have no such restriction.

We wanted to use machine learning because it was the best method for autonomous analysis while accounting for many different factors, such as location variation in beetle placements, trap placements and backdrops, and variance in lighting to name a few. Robust machine learning accounts for these factors simultaneously and does not rely on color filtering methods that need specific parameters to be accurate. Out of the many available deep learning structures, convolutional neural networks are one of the best architectures for computer vision since this project needs to identify the locations of the rootworms in order to count the number of rootworms on the yellow sticky trap [21, 22]. Object detection algorithms are usually divided into two categories: classification-based (two-stage) object detectors and regression-based (one-stage) object detectors [23]. One of the best one-stage object detection algorithms is You Only Look Once (YOLO), known for its high speed and robustness [24–26]. We aim to detect the number of western corn rootworm beetles by using the YOLOv5 architecture, a version of the YOLO algorithm.

Section 2 reviews the materials used to build the drone attachment, the design of the Printed Circuit Board (PCB) layout, and the workflow of taking pictures and sending them through the model. In section 3, we describe the given dataset and how we implemented preprocessing methods to create training data. Section 4 reviews the YOLOv5 model architecture. In Section 5 and 6, we analyze the results of applying the model through

simulated and real images. Lastly, in section 7, we discuss possible extensions to this project.

2 Design Approach and Methodology

2.1 Materials

This section contains the devices integrated into our circuit including our sensors, breakouts, and the power source. It does not include the materials used to create the enclosures nor the drone it is to be mounted upon.

1. **Arduino Mega2560 Board** [27]
   The Arduino Mega2560 is a microcontroller board with 16 analog inputs, 54 digital pins, 4 UART (hardware serial ports), 8Kb of SRAM, and a flash memory of 256 KB. We used the Arduino Mega to connect all the sensors and breakouts because it provided a sufficient number of inputs and outputs as well as functional processing capability.

2. **Adafruit BME 680** [28]
   The Adafruit BME680 has the capability of reading temperature (±1.0 degree Celsius accuracy), humidity (±3% accuracy), barometric (±1 hPa accuracy), and altitude (±1 meter accuracy). We used the BME 680 to analyze these data points and see correlations between the measurements and the concentrations of beetles per trap. It communicates with the microcontroller via an I2C connection.

3. **Arducam Mini Module Camera Shield with OV2640 2 Megapixels Lens** [29]
   The Arducam Mini OV2640 is a 2MP (megapixels) SPI camera with JPEG compression mode, single and multiple shoot mode, and a burst read operation. It supports both I2C and SPI interfaces but does not have an auto-focusing feature. We used the Arducam Mini for live photos of the traps to be processed by the cropping and machine learning programs. The 2 megapixel camera provided sufficient detail for our model to detect and pre-coded image filters.

4. **Adafruit MicroSD Card Breakout Board** [30]
   The Adafruit MicroSD Card Breakout Board is a MicroSD card reader supporting FAT and FAT32 SD cards. We used the breakout board to write our data to a microSD in the form of a CSV (comma-separated value) and save images captured by the Arducam. It communicates with the microcontroller via a SPI connection.

5. **5-Cell AA Battery Pack**
   The 5-Cell AA Battery Pack is a battery pack with a chassis mount and wire leads spanning 127mm. We used this battery pack to power the circuit without a computer present, allowing for autonomous functions.

6. **ANVISION 4mm x 10mm 5V Cooling Fan**
   The ANVISION Cooling fan is a compact 5V fan with airflow capable of 5.32 CFM

(cubic feet/minute) and a speed of 6000 RPM (revolutions/min). We used this fan to cool the Arducam, as it had a tendency to overheat.

7. **Adafruit Ultimate GPS Breakout** [31]
   The Adafruit Ultimate GPS Breakout is a module that communicates with 33 satellites on 99 channels to provide location updates. It has an accuracy of < 3 meters and an update rate of 1 to 10 Hz. We used the GPS breakout to provide a location to be written into the CSV and help validate which trap the camera has taken a picture of. The GPS location validation works in conjunction with the QR Code scanner. It communicates with the microcontroller via a UART connection.

8. **HC-SR04 Ultrasonic Distance Measuring Sensor Module** [32]
   The HC-SR04 Ultrasonic Sensor Module is a 5V sensor using 40kHz pulses to measure distance by analyzing the time it takes for the pulse signal to return. It detects ranges from 2 to 400 cm in distance, with a range from 10 to 250cm for the most accurate results. We used the ultrasonic module to gauge distance for our photos. As our camera did not have an auto-focusing feature, we had to take our photos from a specific range.

2.2 Enclosure Design

When creating the enclosures, we had to keep a few things in mind. First, the size and weight of the circuit and enclosure were important. These factors would determine how efficient the overall design would be, as they would be key to the drone's maneuverability with the enclosures. We also needed to make sure that the camera could be oriented in different ways on the drone so that it could take photos from different angles. Part of ensuring the camera could be oriented in different ways is also designing the enclosure so that moving the camera will not significantly change the weight distribution of the enclosure. A different weight distribution could impact the drone's flight, thus maintaining it with multiple camera positions was paramount.

Secondly, the enclosure had to be able to accommodate all the sensors and breakouts that were required. This includes ensuring the GPS was able to communicate with their satellites, making sure the SD breakout was accessible, and fitting the distance sensor in a manner to provide accurate readings for the camera.

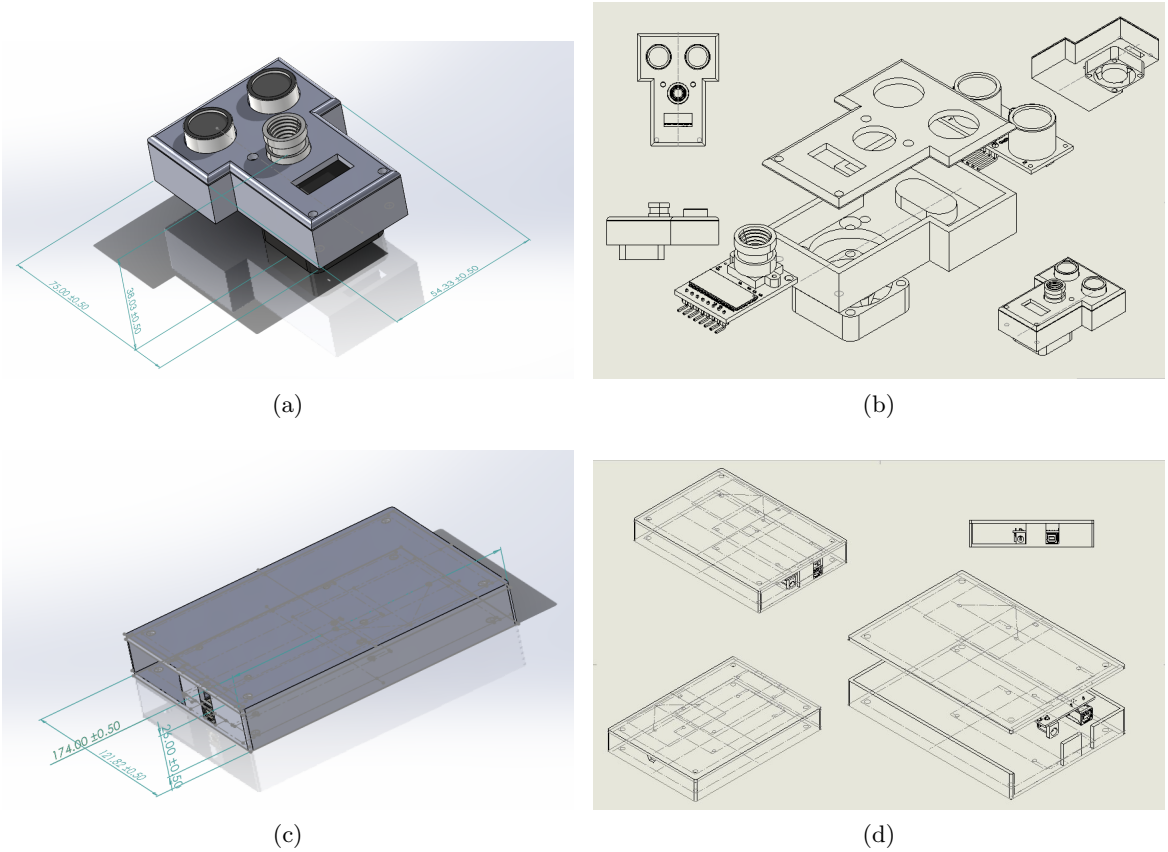
Finally, we needed to make sure that the design was easy to troubleshoot. Between the precise calibration required with the camera and ultrasonic sensor and the need for possible micro-adjustments to certain components, it was important that the enclosure could be adjusted without having to completely redesign it each time. This would save time and money.

To effectively accommodate these requirements, we decided to create two enclosures: one for the Arducam camera, ultrasonic sensor, and fan, and one for the PCB (with the Micro SD Breakout, BME Sensor, Arduino Mega, and Ultimate GPS mounted). This design choice preserves the weight distribution by allowing the camera orientation to independently change from the enclosure. Having separate enclosures also isolates the Arducam

and ultrasonic sensor from the rest of the circuit, which, in turn, also isolates their calibrations and allows for easier calibrations.

The design for the circuit housing is relatively simple. The enclosure is designed to fit as close as possible to the PCB while still allowing for micro-adjustments. This is only possible because the PCB (with the installed sensors) is the only component housed in this case. To mount the battery pack, we have added holes for both the screws and the wires to access the PCB inside. One side has a large hole to allow access to the SD card breakout and Arduino, while the opposite side has a small hole to allow wires to be fed into the PCB.

The camera enclosure is designed to have the ultrasonic sensor and camera lens as close as possible to allow the distance reading to accurately match the distance from the camera aperture to the target. Both a hole in the front of the enclosure and a fan on the back provide constant airflow to account for the camera's tendency to overheat during the prototyping phase. Figure 1 shows both the enclosures. More information on the enclosures and the PCB layout is available on our GitLab [33].



(a) (b)

(c) (d)

Figure 1. Images of the (a) camera enclosure (cm), (b) camera enclosure drawing, (c) PCB enclosure (cm), and (d) PCB enclosure drawing.

2.3 Photo Range Determination

Because the Arducam did not possess an auto-focusing ability, we designed the circuit to take a photo when the ultrasonic sensor measured a distance that fell within the pre-focused threshold. The camera enclosure was designed to have the aperture of the camera at the same length as the distance sensor. This helped the ultrasonic sensor measure the distance from the camera aperture to the target trap. However, the HC-SR04 Ultrasonic Module has a varying amount of error depending on the distance.

To find the best range to take our photos, we conducted tests comparing the actual distance to the distance measured by the ultrasonic module. The results are shown in Figure 2 with the negative Measured Percent Error denoting how the measured distance was less than the actual distance. Between 20 and 30 cm, the percent error measured was minimal and provided a large range (10 cm) of minimal fluctuations. Thus, we decided to code the circuit to take photos between a range of 20 cm and 30 cm.

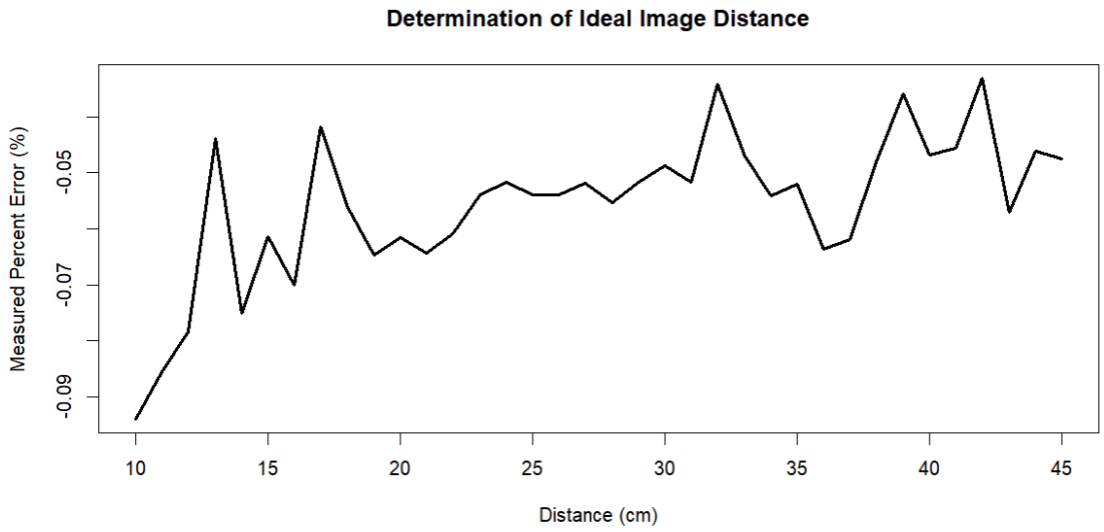


Figure 2. Graph comparing the HC-SR04 Ultrasonic Distance Measuring Sensor Module recorded distance and the actual distance

2.4 Arduino Code

The driver code for the Arduino Mega2560 is split into two main functions: the setup and the loop. The full driver code is available on our GitLab [33].

In the setup function, we initialized all the components which include the BME680 sensor, the SD card, the Arducam, and the ultimate GPS module. We also initialized the settings for the Arducam which include the format of the image captured, the resolution size, and the light mode. After the SD card is initialized, we created/opened a CSV file to write our sensor data to.

In the loop function, we used the ultrasonic sensor to detect the distance every 1 second. If the distance is in the desired range, between 20 cm and 30 cm, the data from

the Ultimate GPS module, the BME680, and the distance sensor is read and written to the CSV file. This distance was chosen due to the clarity and resolution of the image captured at that distance. A picture is then captured by the Arducam and saved to the SD card with the corresponding file name which was also written to the CSV file. The device continues to capture images as long as the distance is in the desired range and stops once it is out of it. Image capture and writing data take between 1 and 2 seconds. Our current workflow for object detection is shown in Figure 3.

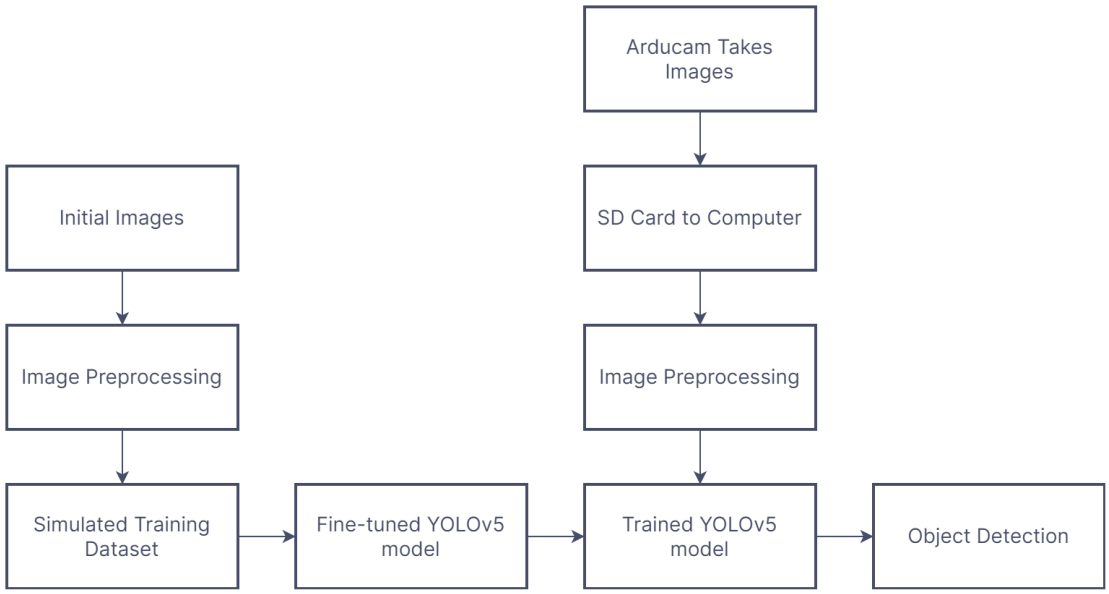


Figure 3. Workflow of our current beetle detection setup

3 Dataset

From now, we will refer to machine learning terminology, accessible in the Glossary B. We were provided with an initial dataset of 262 images of traps which were manually taken in various fields throughout Illinois [34]. Out of the given images, 30 of them had traps with at least one western corn rootworm beetle while the rest had none. Of the traps that had beetles, 41 beetles were counted in total. Most of the traps in the images also had dirt, sand, and other such undesirable contributions that can perturbate the image analysis.

In addition, some of the traps in the images were positioned at an angle in the field, while some were positioned perpendicular to the ground. This change in orientation can affect the number of insects or beetles caught in the trap by a significant amount. Figure 4 shows examples of the images given in the dataset.

(a) (b)

Figure 4. Images of the uncropped traps with (a) no beetles and sparse amounts dirt and (b) two beetles and numerous other insects.

3.1 Image Preprocessing

Images taken by the drone have the potential to be highly variable in perspective, position, and background. Processing images to reduce this variation is important for achieving high model accuracy.

We use cropping to eliminate the background and shift the perspective of the image to make it appear that the image was taken from directly above the trap, rather than at an angle. This allows us to take an image containing a trap, its mounting pole, and a plant-filled background, and reduce it to a rectangular image containing only the trap.

The OpenCV Python library provides a host of functions useful for image processing [35]. First, we seek to separate the white mounting pole from the trap and have it blend into the background. To do this, we set a threshold of "white-ish" values which correspond to the color of the pole. We then select all colors in this range and change them to a green color, blending the pole into the background and separating it from the page. In our testing, we found that this step reduced the tendency for the pole to be detected as part of the trap. Next, we convert the image to grayscale and use Otsu's Method [36] to separate the image pixels into two classes, foreground and background. Otsu's Method isn't perfect, resulting in jagged edges and small holes in the interior of the trap. Applying morphological transformations, we are able to clean up these artifacts. First, we perform

morphological closing to close the holes in the interior. Next, a morphological opening cleans up the jagged edges and reduces noise. Now that the trap is cleanly separated from everything else, we use OpenCV to identify the largest shape in the image. As long as the trap takes up the majority of the image, we are left with an outline around the trap. Since the images are often taken from an angle, this outline is a quadrilateral shape. Using OpenCV, we are able to perspective shift the image such that the trap's quadrilateral outline becomes rectangular. After cropping, we are left with a rectangular image containing only the trap. More information on the functions used to crop these traps is included on our GitLab [33].

(a) (b) (c) (d)

Figure 5. Stages of the image preprocessing: (a) original image of trap, (b) image after Otsu's method and morphology are applied, (c) bounding quadrilateral around trap, and (d) the final cropped and perspective-shifted image.

3.2 Simulated Dataset

Unfortunately, the images that we were given did not contain enough beetles to accurately train the object detection algorithm. Instead, we simulated a new dataset using our old images. As backgrounds, we used several images of clean traps with no beetles. From the given image dataset, we cropped out multiple beetles, as shown in Figure 6. We also cropped out other insects and dirt from the traps to ensure variability in the dataset and to better replicate real world conditions. Next, we randomly chose a background, a random number of beetles to paste, and a random number of insects to paste. Finally, we pasted randomly rotated cropped images of beetles, insects, and dirt onto the background to create a simulated training and testing dataset. We record the location, height, and width of the pasted beetles to store as labels. Using this method, we were able to greatly increase the size of our dataset. Furthermore, we iteratively improved our model by incrementally adding more complexity to the images with many types of beetles, dirt, and non-beetle bugs. Figure 7 shows examples of our simulated dataset.

Figure 6. Image of a cropped western corn rootworm beetle

(a) (b) (c)

Figure 7. Example images (a), (b), and (c) of the simulated dataset.

4 YOLOv5 Architecture

The YOLOv5 algorithm [37], a version of the YOLO algorithm was employed in this study as an accurate and high-speed object detection model. This computationally inexpensive algorithm was initially chosen to mount onto an Arduino Nano. However, we couldn't use the Arducam with the Arduino Nano, so we reverted to the Arduino Mega. This algorithm is able to detect objects quickly in real-time as people working on fields do not typically have access to high-performing computers. YOLOv5 was developed using PyTorch [38], an open-source framework based on Python that simplifies machine learning. This one-stage object detection algorithm consists of three main components: the backbone, the neck, and the head.

The backbone is based on the convolutional neural network Darknet53. The authors applied the Cross Stage Partial network strategy (CSPNet) [39] to reduce the number of floating-point operations per second (FLOPS) and the size of the model. This strategy resolves vanishing and exploding gradients due to the depth of the model and by truncating the gradient. This all leads to increasing the inference speed and accuracy of this

algorithm [40]. The main purpose of the backbone is to serve as a feature extractor by using convolutions and pooling to create different-size feature maps from the input.

The neck, also known as the path aggregation network (PANet), is used for feature fusion. In YOLOv5, the authors applied Spatial Pyramid Pooling (SPP) [41] and BottleNeckCSP [39] to increase the speed and segregate the most relevant features of the input, respectively. The neck takes in all the extracted features from the backbone to save and sends feature maps of three different sizes from deep layers to the head.

The head is responsible for object detection, consisting of three 1x1 convolutional layers[1] that predict the location of the bounding boxes (xmin, ymin, xmax, ymax), the confidence, and the object classes.

Figure 8 depicts the architecture of YOLOv5.

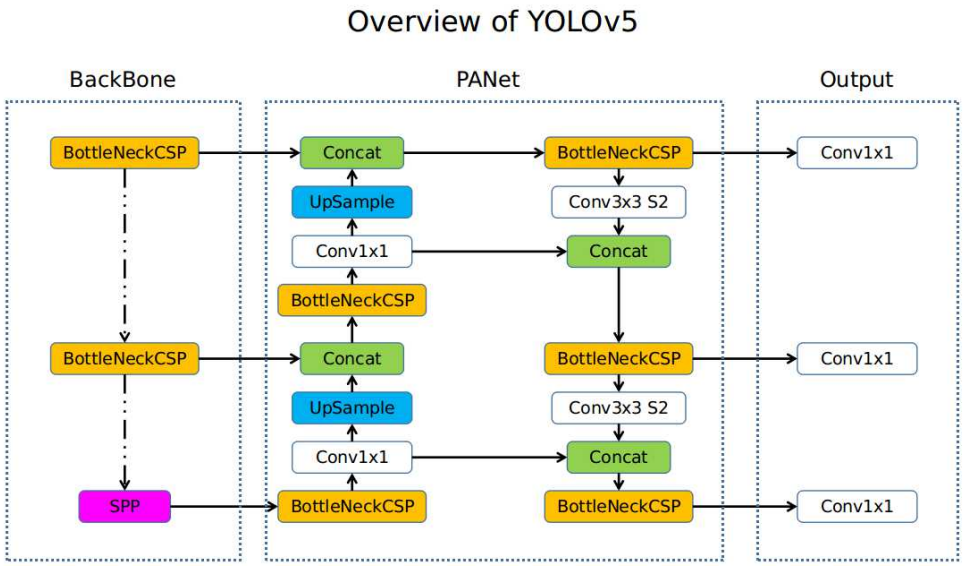


Figure 8. YOLOv5 architecture [37]

The loss function of YOLOv5 is the sum of the loss functions for the bounding box, classification loss, and confidence loss. The YOLOv5 loss function is defined as

$$L = \lambda_{bx} l_{bx} + \lambda_s l_s + \lambda_c l_c \tag{4.1}$$

where L is the overall loss function, l_{bx} is the regression loss function for the bounding box, l_s is the loss function for the classification, and l_c is the loss function for the confidence. $\lambda_b x$, λ_s, and λ_c are the coefficients for the loss functions l_{bx}, l_{bx}, l_s, l_c, respectively. l_{bx} is calculated using Complete Intersection over Union Loss (CIoU) while l_s and l_c are both calculated using Binary Cross Entropy with Logits Loss [42, 43].

YOLOv5 has several versions, including YOLOv5n, YOLOv5s, YOLOv5m, YOLOv5l, and YOLOv5x. These models are ordered from the smallest size to the largest size. We used YOLOv5m for most of our process because, in our trial attempts, it offered exceptional

[1]Convolutional layers are the building block of a convolutional neural network, described in Glossary B.

accuracy with a smaller size. To decrease the amount of time training the model, we used pre-trained model weights for transfer learning. Pre-trained model weights are the weights gained from training the same model on a different dataset. In this case, it was trained on the COCO dataset [44], a comprehensive dataset widely used in object detection to evaluate model performance. We chose to use the default hyperparameters but with a batch size of 16 and an image size of 1280x1280 pixels for our initial models. We used 1000 images for the training dataset and 250 for the test dataset, using an 80-20 train-test split. We had only one class: beetles. This would ensure that the model would distinguish WCR beetles from other insects. To address data scarcity, YOLOv5 uses data augmentation techniques during training, such as scaling and rotations, to allow for better model performance. This augmented dataset is shown in Figure 9. We trained our initial models for 50 epochs, and we trained three final models, YOLOv5s, YOLOv5m, and YOLOv5x, for 100 epochs to ensure loss minimization. We also trained another YOLOv5x model with an image size of 640x640 pixels. We tested the algorithm using the simulated dataset, images from our initial dataset, and images taken from the Arducam. We then recorded the information to evaluate the model's performance and robustness.

5 Results

5.1 Evaluation Metrics

In order to evaluate the performance of the trained object detection models, we used three different metrics: precision, recall, and mean average precision (mAP). These values can vary when the confidence threshold is changed, so it is possible to use them as metrics to characterize the model performance.

Precision measures the percentage of correct detections out of total ones, and can be calculated using the following equation

$$P = \frac{TP}{TP + FP} \tag{5.1}$$

where P is the precision, TP is the number of true positives, and FP is the number of false positives.

Recall measures the percentage of objects that are detected correctly. This can be calculated using the following equation

$$R = \frac{TP}{TP + FN} \tag{5.2}$$

where R is the recall and FN is the number of false negatives.

Average Precision (AP) is the area under the precision-recall curve, and mAP is the average of the AP over all classes. In this case, the mAP and AP are equal to each other since we only have one class. mAP_0.5 is the mAP where we consider an object detected if it has an Intersection over Union (IoU) greater than or equal to 0.5. mAP_0.5:0.95 is the average of mAP over different IoU thresholds from 0.5 to 0.95. All of these metrics are necessary to evaluate the robustness and performance of the object detection algorithm.

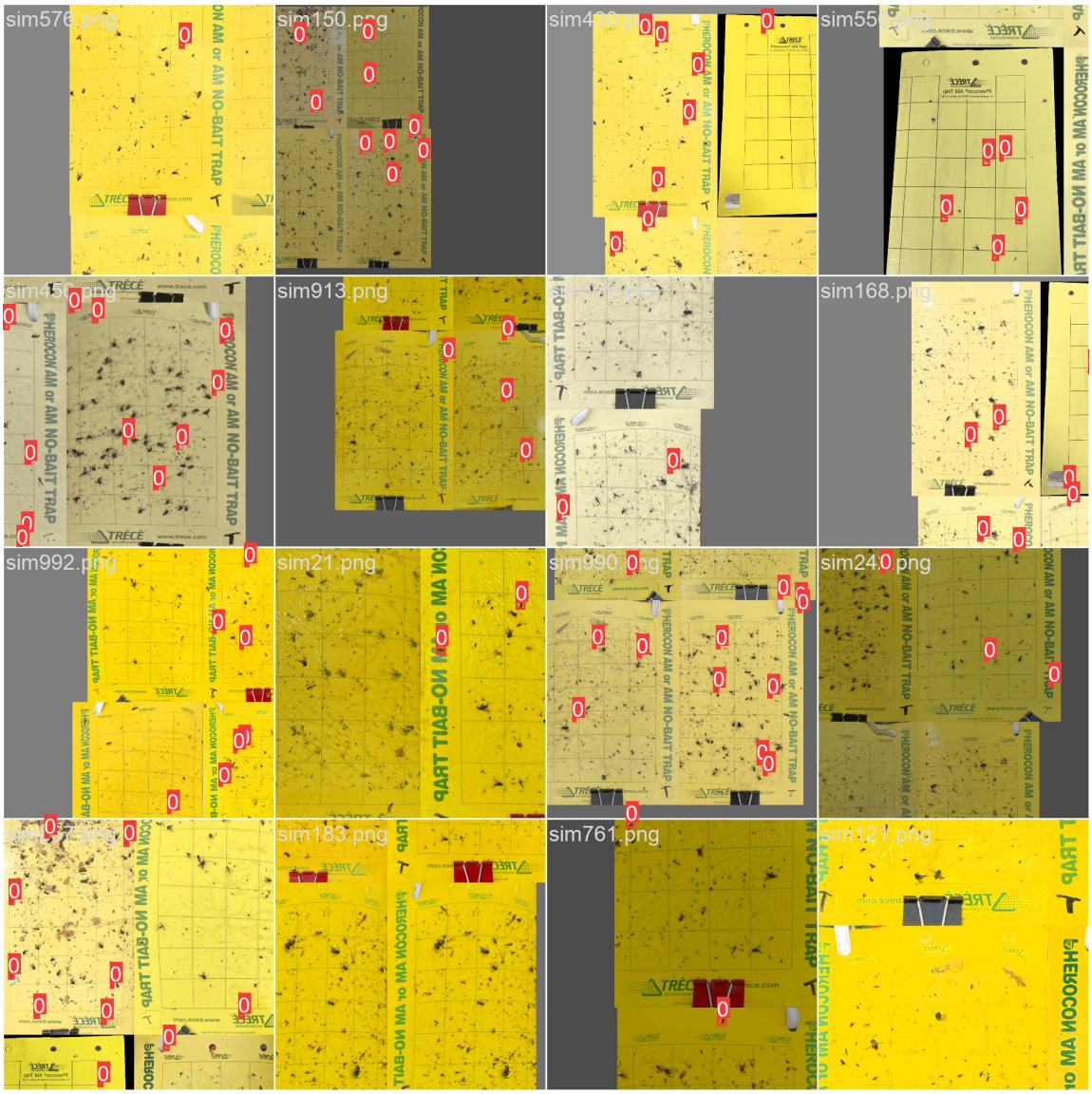


Figure 9. Examples of images in the augmented training dataset. The original image and 3 random images are appended to each other, and YOLOv5 performs online imagespace and colorspace augmentations to ensure variability in this dataset.

5.2 Simulated Dataset Evaluation

Using the metrics listed previously, the model was evaluated on the simulated testing dataset from the randomized simulated images. The precision, recall, and mAP for all the models are shown in Table 1. We also plotted the evaluation metrics across 100 epochs for the final YOLOv5x model with an image size of 1280x1280 because that model had the highest mAP_0.5:0.95, meaning that, on the simulated dataset, it was the most accurate in identifying the beetles with the correct bounding boxes. This is expected because YOLOv5x is the largest model as it has the highest number of parameters (Params) in Table 1. Figure

10 shows the graphs of these metrics as training progresses for this model.

Architecture	Description	Params (M)	Precision	Recall	mAP_0.5	mAP_0.5:0.95
YOLOv5m	6 beetles	21.2	0.998	0.998	0.995	0.935
	6 beetles, 7 insects	21.2	0.996	0.999	0.999	0.934
	6 beetles, 7 insects, 3 bkgs	21.2	1	0.998	0.995	0.947
	20 beetles, 48 insects, 21 bkgs (Final)	21.2	0.999	0.999	0.995	0.932
YOLOv5s	20 beetles, 48 insects, 21 bkgs (Final)	7.2	0.997	0.998	0.995	0.886
YOLOv5x	20 beetles, 48 insects, 21 bkgs (Final)	86.7	1	1	0.995	0.968
	20 beetles, 48 insects, 21 bkgs (Final), 640 img	86.7	0.996	0.999	0.995	0.871

Table 1. Performance of YOLOv5 models for different simulated datasets. Params refers to the number of parameters of the model in millions (M).

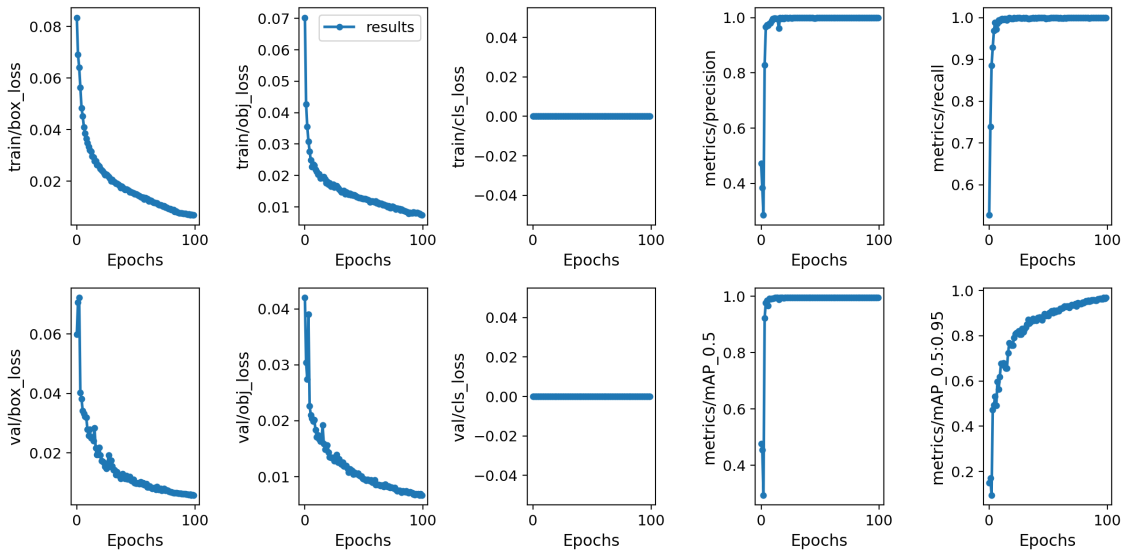


Figure 10. YOLOv5x (Image Size 1280x1280) Training Summary. The train/cls_loss and val/cls_loss is expected to be 0 since there is only one class, so we wouldn't expect to predict the class incorrectly.

We can see that the loss is minimizing over the training process, so the model is improving. All of the precision, recall, and mAP_0.5 values for all of the models are near 1, so these metrics are not a good measurement for the performance of the model. Instead, it is better to look at mAP_0.5:0.95. Using this evaluation metric and only looking at the final dataset, the YOLOv5s model performed much worse than YOLOv5m and both YOLOv5x models. YOLOv5s has a mAP_0.5:0.95 of 0.811 while YOLOv5m has a mAP_0.5:0.95 of 0.886 and YOLOv5x with an image size of 1280x1280 has a mAP_0.5:0.95 of 1. Therefore, going forward, it is more effective to use YOLOv5x of both image sizes to test the robustness of this object detection algorithm on the simulated dataset, real images taken from the initial dataset, and images taken from the Arducam. We can also compare how the different image sizes affect object detection.

On the simulated dataset, both YOLOv5x models performed extremely well, accurately identifying all beetles and rarely having any instances of misclassification. Figures A1 and A2 show some of the predictions on the simulated dataset on both YOLOv5x models. Each prediction is shown along with the confidence level. We have plotted a histogram of the confidence scores in the simulated test dataset for both YOLOv5x models in Figure 11. The average confidence of the 1280x1280 model is 0.938 while the average confidence of the 640x640 model is 0.934. The YOLOv5x model with an image size of 1280x1280 has slightly higher confidence predictions than the model with an image size of 640x640, possibly attributed to the larger image size. This would then allow the model to use more data to predict the beetle's locations. Since both models have extremely high precision and recall on the simulated dataset, it is better to compare the models on non-simulated data taken from a high-quality camera and the Arducam.

(a) (b)

Figure 11. Using the simulated test dataset, histogram of the confidence scores using the YOLOv5x model with an image size of (a) 640x640 and (b) 1280x1280.

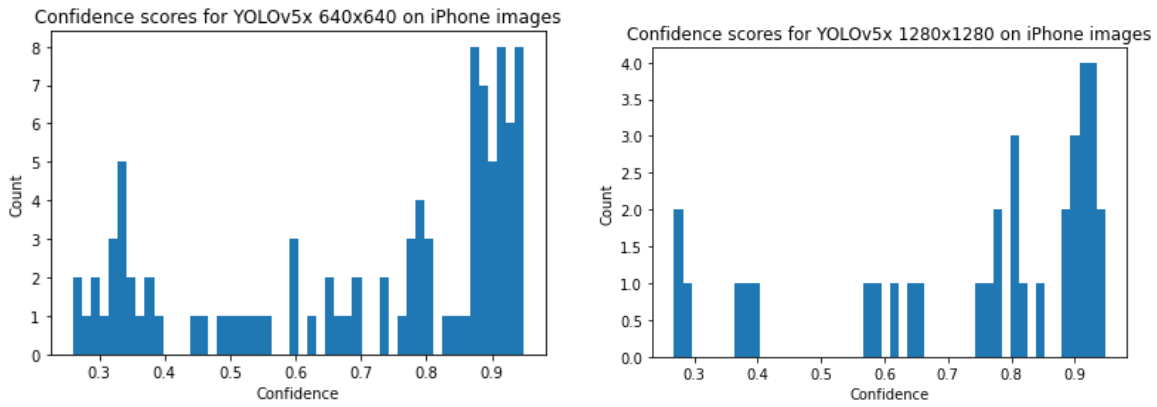
5.3 Real World Performance Evaluation (Iphone and Arducam)

The results for the precision and recall on both datasets are shown in Table 2. On the high-quality camera, an iPhone, there were a few instances of bad cropping due to the lighting and the horizon in the picture, so some of the images were incorrectly cropped. We did not use these in our final results nor conclusions. Also, we were not given specific information about the version of the iPhone camera. As shown in Figure A3, both models were able to identify the beetles that were used in the simulated dataset with relative ease, but there were a lot of false positives from many similar-sized insects. This is further shown as there are 40 predictions for the 1280x1280 model and 94 predictions for the 640x640 model. The histogram in Figure 12 depicts the much more confident YOLOv5x 1280x1280 model. The average confidence on iPhone images is 0.707 for the 640x640 model and 0.740 for the 1280x1280 model. Figure A4 shows an example of these varying confidences. Let P be the number of predicted beetles and let E be the number of effective beetles. To see how the model predicts for different traps, we plotted 2D histograms of $P - E$ vs E and

P/E vs E. We could not use any of the traps with $E = 0$ for the latter plot. Figures 13 shows these histograms. The 640x640 model has much more guesses than the 1280x1280 model since the 640x640 model has some values of $P - E = 11$. These large number of predictions come from insects that look like beetles, causing the smaller model to give a wide range of guesses. The 1280x1280 model has much fewer predictions, and the model is too strict since $P - E$ is often negative. This shows that it's not predicting as much as it should. The P/E vs E further confirms that the 640x640 model performs worse since P/E is often 2 or 3 on the 640x640 model and P/E is usually 1 on the 1280x1280 model. This depicts that the 1280x1280 model is usually predicting the correct number of beetles for each trap. The model with an image size of 1280x1280 pixels usually had less number of and more confident predictions while the model with an image size of 640x640 pixels was more numerous and labeled beetles based on their size rather than their patterns. As we decrease the image size, the model has a high recall but a low precision as it tries to guess many similar-sized insects as the beetles. The higher image size model also makes predictions based on the yellow color, as shown in Figure A5. Table 2 depicts that the higher image size model has a higher precision, showing it is able to distinguish what is a WCR beetle. Overall, on iPhone pictures, the model with a higher image size performs better because it has a much higher precision with the same recall.

Model Name	Precision (iPhone)	Recall (iPhone)	Precision (Arducam)	Recall (Arducam)
YOLOv5x (1280x1280)	0.675	0.675	1.0	0.025
YOLOv5x (640x640)	0.287	0.651	0.5	0.625

Table 2. Precision and Recall of YOLOv5x models on both the iPhone and Arducam images.



(a) (b)

Figure 12. Using the iPhone images, histogram of the confidence scores using the YOLOv5x model with an image size of (a) 640x640 and (b) 1280x1280.

We have also taken pictures on the Arducam using western corn rootworm beetles preserved in 70% alcohol and the same yellow sticky traps seen in the previous images. We tested in a variety of conditions with the lighting, orientation, and amount of foliage on the trap. Using these, the YOLOv5x model with an image size of 640x640 pixels performed

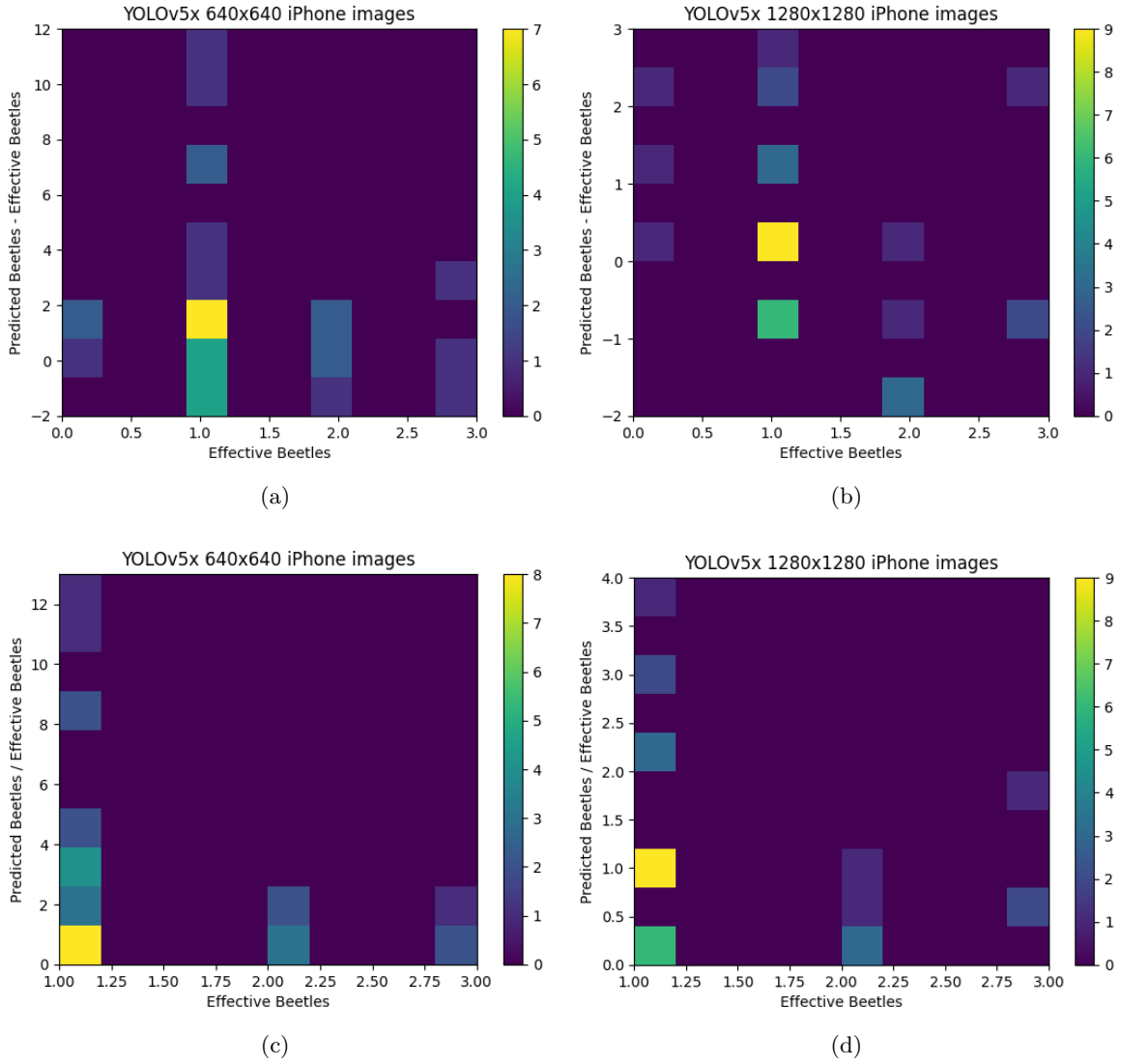


Figure 13. Using the iPhone images, 2D histograms of $P - E$ vs E using the YOLOv5x model with an image size of (a) 640x640 and (b) 1280x1280. Using the iPhone images, 2D histograms of P/E vs E using the YOLOv5x model with an image size of (c) 640x640 and (d) 1280x1280.

much better than the model with the higher image size (1280x1280 pixels). The results for precision and recall on the Arducam images are shown in Table 2. Out of the 8 images that were cropped properly with 50 total beetles, the model with the 1280x1280 pixel size only made 1 prediction, shown in Figure A6. Figures A6 and A7 show that the model with the lower image size had a higher recall and higher precision, getting some of the clearly identifiable beetles. It is able to identify most of the beetles if there are not any other insects or plants in the image. Figure A8 shows that the model's precision is increased in the sun, most likely due to the simulated dataset being constructed from images in the sun. Therefore, as a result of the training methods in different image sizes, both YOLOv5x models are better in different scenarios.

When comparing the iPhone and Arducam performances, a possible explanation for the result can be found in our simulated dataset. To create the images, we cropped the beetles and took traps from the iPhone images. This had the unintended effect of having the iPhone dataset (while not completely identical) quite similar to the training datasets. Regarding respective resolutions (or comparing 640x640 iPhone to 640x640 Arducam and 1280x1280 iPhone to 1280x1280 Arducam), a possible explanation for the iPhone dataset having a higher recall than the Arducam dataset would be the similarity between the iPhone dataset and the training data, making the model better at detecting potential beetles. It would also explain how the 1280x1280 model precision on the Arducam dataset was higher than on the iPhone images. With the Arducam dataset having less of a semblance to the training dataset, beetles would have to more closely resemble a beetle to be identified as a potential from the model. On the other hand, the 640x640 model had a lower precision on the Arducam dataset, most likely due to the smaller resolution inhibiting the model to predict insects that are of a similar size to the beetles.

6 Conclusion

Our circuit allows for the user to take and process images from the Arducam with a model that (in the end) performs flawlessly in our simulated environment.

However, our current setup is not accurate enough for field applications quite yet. The 640x640 images had a precision of 0.5 with a recall of 0.625. The 1280x1280 pixel images had a precision of 1 with a recall of 0.025. The difference in precision and recall for the two models most likely was because of the difference in image detail. The 1280x1280 images are stricter with beetle detection as more detail is required to identify an object as a beetle. This increases the precision, as the few objects that are detected as potential beetles have already passed a strenuous check. The 640x640 images are less strict with their detection of potential beetles because of the lack of detail, and this decreases the overall precision.

This in turn means incorporation of the Arducam and the hardware yields a maximum precision of 0.5 with a maximum recall of 0.625. With industry standards generally lying around 70% as a good accuracy, it's clear that our model requires further development in the form of increasingly varied training sets and data that better emulates the hardware conditions.

7 Possible Extensions

Due to time restrictions we faced because of the nature of the project being intended for a semester long course, there were a few additions to the project that we could not achieve. One of these additions is the integration of our device with the drones that are currently being used in fields. Once this is achieved, we would be able to test our system in an accurate environment.

We also were unable to fully implement a QR code system for better geolocalization. On each trap would be a QR code corresponding to the location of that trap in the field. Alongside GPS coordinates from the Adafruit Breakout, we could precisely tag each trap

image with the location it was taken. This information would be vital in understanding if beetles are more or less prevalent in specific areas. It could even be used to detect beetle movement patterns over time. Utilizing a heat map, this data could be comprehensively relayed to a farmer, aiding in more precise pesticide application. For QR detection, we used a pretrained YOLOv7 model. We opted for a pretrained model because QR codes are fairly standardized and training would have taken a substantial amount of time. Although the model worked well with iPhone-quality images of QR codes, in our testing, it did not perform well on Arducam-quality images. Furthermore, our dataset of trap images did not have QR codes in them, so we were unable to test the pretrained model in any real-world scenarios specific to our use-case.

One major issue we had faced was the limitations of the ArduCAM OV2640. While being able to capture high resolution images, the camera module has restrictions such as manual focus and slow auto exposure locking. In the future, we would like to attempt to use the camera built into the drone to capture images rather than have an external camera. On the high-quality camera, there were a few instances of bad cropping due to the lighting and the horizon in the picture, so we would like to make a more thorough cropping software that can be perform highly even with bad lighting and other quadrilateral-like shapes in the picture.

Incorporating wireless data streaming to make the process more efficient would be beneficial and more streamlined for the user. At the moment, physical transfer of the micro SD card from the device to a computer is required.

Other possible future applications of this project include a way to temporarily change the trap visibility based on drone proximity. One of the easiest ways to change the trap visibility would be extending the trap upward over the crop for photos from which a drone would be able to access.

We would like to increase the size of our simulated dataset by including cropped insects and backgrounds from our initial images. We also want to take more images at various farms to introduce variability in the lighting, background, and insects. This would ensure the model is more robust in the future.

Implementing the newest version of the YOLO algorithm, YOLOv8, and possibly other object detection algorithms could increase the precision and recall of the model. We would then compare the performance and speed of these algorithms to YOLOv5. We would then test the preprocessing and inference speed of this algorithm across the different algorithms to understand if it is possible for real-time beetle identification.

Acknowledgements

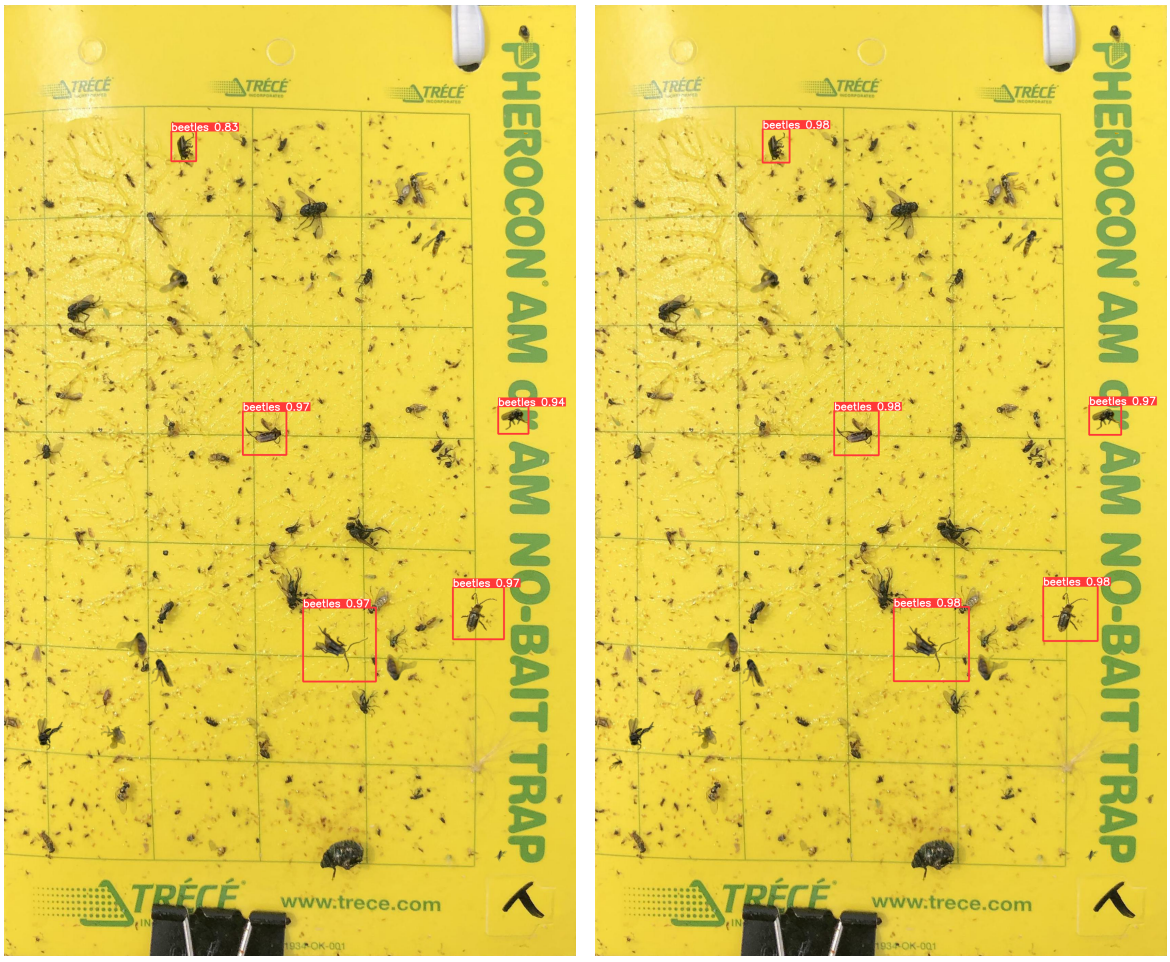
We would like to thank Dr. Riccardo Longo and the Center for Artificial Intelligence Innovation at the NCSA for support through our affiliation. We would like to thank Dr. Joseph L. Spencer and Sagnika Das at the College of Agricultural, Consumer and Environmental Sciences for their help in providing images and information about western corn rootworm beetles. We would like to thank Jenny Campbell, Matthew Hoppesch, and Kristopher Young for useful discussions, comments, and suggestions. This research is

part of the Delta research computing project, which is supported by the National Science Foundation (award OCI 2005572), and the State of Illinois. Delta is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications. This work utilizes resources supported by the National Science Foundation's Major Research Instrumentation program, grant 1725729, as well as the University of Illinois at Urbana-Champaign.

A YOLOv5 images

(a) (b)

Figure A1. Using the simulated dataset, predictions of beetles using the YOLOv5x model with an image size of (a) 640x640 and (b) 1280x1280.



(a) (b)

Figure A2. Using the simulated dataset, predictions of beetles using the YOLOv5x model with an image size of (a) 640x640 and (b) 1280x1280.

(a) (b)

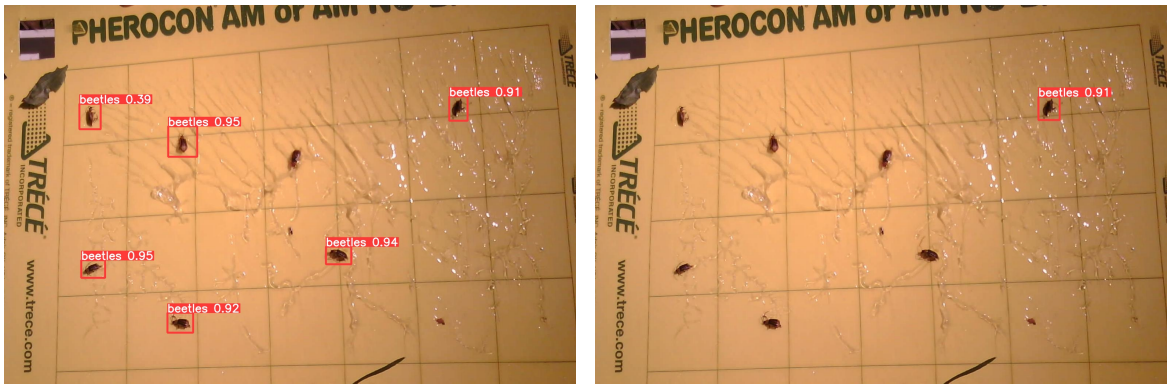
Figure A3. Using the high-definition dataset, predictions of beetles using the YOLOv5x model with an image size of (a) 640x640 and (b) 1280x1280.

(a) (b)

Figure A4. Using the high-definition dataset, predictions of beetles using the YOLOv5x model with an image size of (a) 640x640 and (b) 1280x1280.

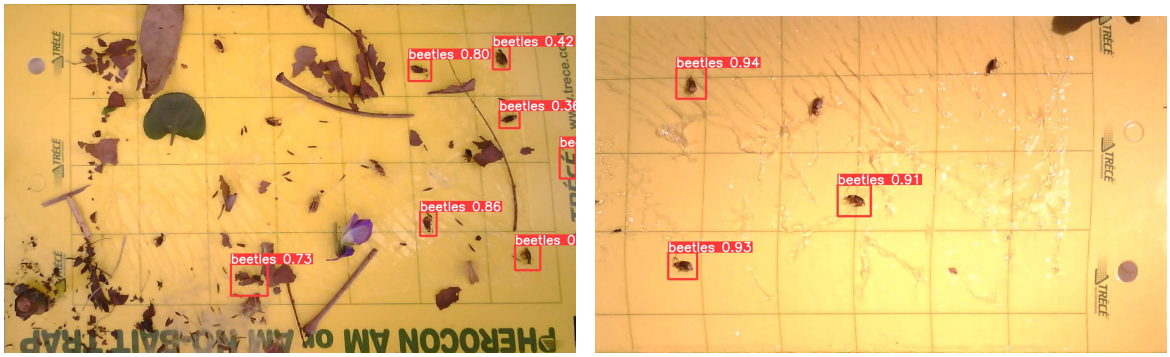
(a) (b)

Figure A5. Using the high-definition dataset, predictions of beetles using the YOLOv5x model with an image size of (a) 640x640 and (b) 1280x1280.



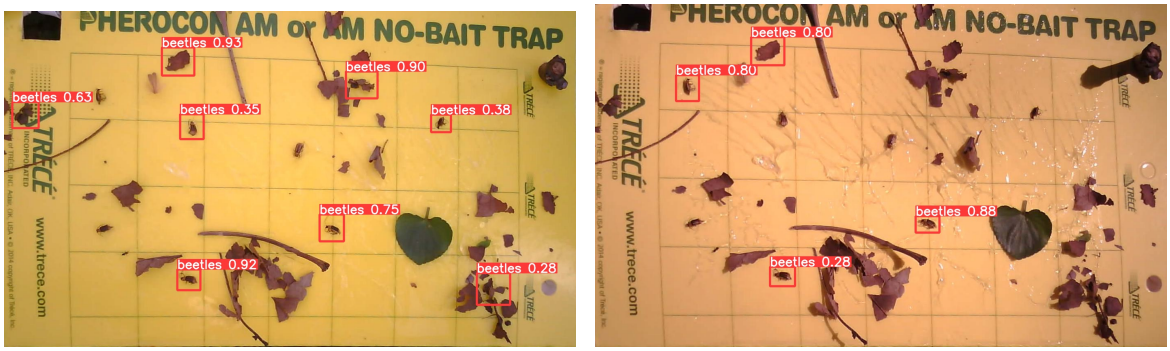
(a) (b)

Figure A6. Using images taken by the Arducam, predictions of beetles using the YOLOv5x model with an image size of (a) 640x640 and (b) 1280x1280. Both of these images have 7 beetles.



(a) (b)

Figure A7. Using images taken by the Arducam, these are predictions of beetles using the YOLOv5x model with an image size of 640x640. They each have (a) 10 beetles and (b) 5 beetles.



(a) (b)

Figure A8. Using images taken by the Arducam, these are predictions of beetles using the YOLOv5x model with an image size of 640x640. They both have 6 beetles each. They are each taken in the (a) shade and (b) sun.

B Glossary

Batch Size is the number of images processed before the model parameters are updated. 12

Binary Cross Entropy with Logits Loss is a loss function used to compare the predicted probabilities to the actual output, which should be either 0 or 1 [45]. 11

Complete Intersection over Union Loss is a loss function that uses the overlap area and central point between the predicted and true bounding box as well as the aspect ratio of both boxes to compute the loss [46]. 11

Convolutional Neural Network is a deep learning network architecture particularly useful for patterns and image recognition of objects and classes. A filter of a certain size, (1x1 for example) is passed over an image and outputs a matrix of a different size. 10

Data Augmentation is a way to alter training sets to increase variety in what models see and learn from. Some examples include cropping, changing colors, and shifting perspectives [47]. 12

Epochs are the total number of iterations of all training data in one cycle for training. 12

Intersection over Union is a metric used to compare the accuracy of two bounding boxes. The ratio is the area of overlap over the area of union. 12

Loss Function is a function evaluating how well an algorithm models a dataset. We seek to minimize the loss function to make the model more accurate. 11

Otsu's Method is a image processing method of separating an image into a foreground and background based on pixel grayscale intensity values. 8

Testing Dataset is the part of a dataset used to evaluate the final model performance on a the dataset. 9

Training Dataset is the part of a dataset to train a model for prediction and/or classification. 9

Transfer Learning is a method of using the weights from a model trained on another task as the starting weights for another model. 12

References

[1] J. Moeser and B.E. Hibbard, A synopsis of the nutritional ecology of larvae and adults of diabrotica virgifera virgifera (leconte) in the new and old world - nouvelle cuisine for the invasive maize pest diabrotica virgifera virgifera in europe?, Western corn rootworm: ecology and management (2004) 41–65.

[2] Z. Dun, P.D. Mitchell and M. Agosti, Estimating diabrotica virgifera virgifera damage functions with field trial data: Applying an unbalanced nested error component model, Journal of Applied Entomology **134** (2009) 409–419.

[3] N.A. Tinsley, R.E. Estes and M.E. Gray, Validation of a nested error component model to estimate damage caused by corn rootworm larvae, Journal of Applied Entomology **137** (2012) 161–169.

[4] L.J. Meinke, T.W. Sappington, D.W. Onstad, T. Guillemaud, N.J. Miller, J. Komáromi et al., Western corn rootworm (diabrotica virgifera virgiferaleconte) population dynamics, Agricultural and Forest Entomology **11** (2009) 29–46.

[5] C.R. Taylor, The economics of control of northern and western corn rootworms in illinois, Illinois Agricultural Economics **15** (1975) 11.

[6] A. Gassmann, Resistance to bt maize by western corn rootworm: Effects of pest biology, the pest–crop interaction and the agricultural landscape on resistance, Insects **12** (2021) 136.

[7] B.D. Siegfried and R.L. Hellmich, Understanding successful resistance management, GM Crops amp; Food **3** (2012) 184–193.

[8] A.J. Gassmann, Field-evolved resistance to bt maize by western corn rootworm, 2016 International Congress of Entomology (2016) .

[9] C.R. St. Clair, Abundance of western corn rootworm, injury to corn, and bt resistance in local landscapes, 2016 International Congress of Entomology (2016) .

[10] J.D. Reinders, E.E. Reinders, E.A. Robinson, B.W. French and L.J. Meinke, Evidence of western corn rootworm (diabrotica virgifera virgifera leconte) field-evolved resistance to cry3bb1 + cry34/ 35ab1 maize in nebraska, Pest Management Science **78** (2021) 1356–1366.

[11] E.M. Cullen, M.E. Gray, A.J. Gassmann and B.E. Hibbard, Resistance to bt corn by western corn rootworm (coleoptera: Chrysomelidae) in the u.s. corn belt, Journal of Integrated Pest Management **4** (2013) 1–6.

[12] M.E. Gray, T.W. Sappington, N.J. Miller, J. Moeser and M.O. Bohn, Adaptation and invasiveness of western corn rootworm: Intensifying research on a worsening pest, Annual Review of Entomology **54** (2009) 303 [https://doi.org/10.1146/annurev.ento.54.110807.090434].

[13] K. Estes, Increased insect densities reflected in annual corn and soybean survey, farmdoc (2017) .

[14] S. Wechsler and D. Smith, Has resistance taken root in u.s. corn fields? demand for insect control, American Journal of Agricultural Economics **100** (2018) 1136–1150.

[15] T.P. Kuhar and R.R. Youngman, Olson Yellow Sticky Trap: Decision-Making Tool for Sampling Western Corn Rootworm (Coleoptera: Chrysome Journal of Economic Entomology **91** (1998) 957 [https://academic.oup.com/jee/article-pdf/91/4/957/19240664/jee91-0957.pdf].

[16] N. Seiter, Western corn rootworm: Adult sampling and economic thresholds, farmdoc (2018) .

[17] T. Kuhar and R. YOUNGMAN, Sex ratio and sexual dhnorphism in western corn rootworm (coleoptera: Chrysomelidae) adults on yellow sticky traps in corn, Environmental Entomology **24** (1995) 1408.

[18] A. Rejeb, A. Abdollahi, K. Rejeb and H. Treiblmaier, Drones in agriculture: A review and bibliometric analysis, Computers and Electronics in Agriculture **198** (2022) 107017.

[19] A. Hafeez, M.A. Husain, S. Singh, A. Chauhan, M.T. Khan, N. Kumar et al., Implementation of drone technology for farm monitoring amp; pesticide spraying: A review, Information Processing in Agriculture (2022) .

[20] K.S. Subramanian, S. Pazhanivelan, G. Srinivasan, R. Santhi and N. Sathiah, Drones in insect pest management, Frontiers in Agronomy **3** (2021) .

[21] I. Ahmad, Y. Yang, Y. Yue, C. Ye, M. Hassan, X. Cheng et al., Deep learning based detector yolov5 for identifying insect pests, Applied Sciences **12** (2022) 10167.

[22] N.T. Nam and P.D. Hung, Pest detection on traps using deep convolutional neural networks, in Proceedings of the 1st International Conference on Control and Computer Vision, ICCCV '18, (New York, NY, USA), p. 33–38, Association for Computing Machinery, 2018, DOI.

[23] E. Onler, Real time pest detection using yolov5, .

[24] M. Sukkar, D. Kumar and J. Sindha, Real-time pedestrians detection by yolov5, in 2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT), pp. 01–06, 2021, DOI.

[25] N. Al-Qubaydhi, A. Alenezi, T. Alanazi, A. Senyor, N. Alanezi, B. Alotaibi et al., Detection of unauthorized unmanned aerial vehicles using yolov5 and transfer learning, Electronics **11** (2022) 2669.

[26] B. Aydin and S. Singha, Drone detection using yolov5, Eng **4** (2023) 416–433.

[27] T.A. Team, "Mega 2560 rev3: Arduino documentation."

[28] A. Industries, "Adafruit bme680 - temperature, humidity, pressure and gas sensor."

[29] Arducam mini 2mp plus - ov2640 spi camera module for arduino uno mega2560 board amp; raspberry pi pico, Feb, 2023.

[30] A. Industries, "Microsd card breakout board+."

[31] A. Industries, "Adafruit ultimate gps breakout - 66 channel w/10 hz updates."

[32] A. Industries, "Hc-sr04 ultrasonic sonar distance sensor + 2 x 10k resistors."

[33] A. Khot, J. Lin, S. Pasquesi and P. Pavithran, Akhot2 / group-01-phys371-sp2023 · gitlab, 2023.

[34] W.B. Detection, "Wcr beetle traps dataset." https://universe.roboflow.com/wcr-beetle-detection/wcr-beetle-traps, may, 2023.

[35] G. Bradski, The OpenCV Library, Dr. Dobb's Journal of Software Tools (2000) .

[36] N. Otsu, A threshold selection method from gray-level histograms, IEEE Transactions on Systems, Man, and Cybernetics **9** (1979) 62.

[37] G. Jocher, A. Chaurasia, A. Stoken, J. Borovec, NanoCode012, Y. Kwon et al.,

ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation, Nov., 2022. 10.5281/zenodo.7347926.

[38] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan et al., Pytorch: An imperative style, high-performance deep learning library, in Advances in Neural Information Processing Systems 32, pp. 8024–8035, Curran Associates, Inc. (2019), http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[39] C.-Y. Wang, H.-Y.M. Liao, I.-H. Yeh, Y.-H. Wu, P.-Y. Chen and J.-W. Hsieh, Cspnet: A new backbone that can enhance learning capability of cnn, 2019.

[40] R. Xu, H. Lin, K. Lu, L. Cao and Y. Liu, A forest fire detection system based on ensemble learning, Forests **12** (2021) .

[41] K. He, X. Zhang, S. Ren and J. Sun, Spatial pyramid pooling in deep convolutional networks for visual recognition, in Computer Vision – ECCV 2014, pp. 346–361, Springer International Publishing (2014), DOI.

[42] Q. Xu, Z. Zhu, H. Ge, Z. Zhang and X. Zang, Effective face detector based on yolov5 and superresolution reconstruction, Computational and Mathematical Methods in Medicine **2021** (2021) 7748350.

[43] N. Al-Qubaydhi, A. Alenezi, T. Alanazi, A. Senyor, N. Alanezi, B. Alotaibi et al., Unauthorized unmanned aerial vehicle detection using yolov5 and transfer learning, .

[44] T. Lin, M. Maire, S.J. Belongie, L.D. Bourdev, R.B. Girshick, J. Hays et al., Microsoft COCO: common objects in context, CoRR **abs/1405.0312** (2014) [1405.0312].

[45] Z. Zhang and M.R. Sabuncu, Generalized cross entropy loss for training deep neural networks with noisy labels, CoRR **abs/1805.07836** (2018) [1805.07836].

[46] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye and D. Ren, Distance-iou loss: Faster and better learning for bounding box regression, CoRR **abs/1911.08287** (2019) [1911.08287].

[47] C. Shorten and T.M. Khoshgoftaar, A survey on image data augmentation for deep learning, Journal of Big Data **6** (2019) 60.