Nature Photography Robot

Authors: Justin Kiefel, Sidhant Motwani, Fernando Paulino
Affiliation: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—Nature photography is a mundane and time-intensive process. Photographers must wait for sparse animal appearances and spend even more time editing the photographs. Remote-controlled photography robots exist, but these systems still require constant human attention. Our solution to this problem is a robotic system capable of performing nature photography and photo editing. We developed a photography pipeline, where the robot searches for, tracks, and photographs animals then performs automatic editing.**

***Index Terms*—Computer vision, design, motion planning, object detection, photo enhancement, robot, search**

1 INTRODUCTION

Animal photographs are widely used across the internet and social media. From advertisements to raising awareness for conservation efforts, there is a clear demand for high-quality animal photos [1]. However, the process of acquiring these photos is far from easy. Animals are constantly moving and often avoid humans. The photographer may need to wait an extended period of time to have the opportunity to capture a photo. Afterward, more human effort is required to edit the photos. The cost of human time and labor in this process is undoubtedly high. As a solution, we propose a photography robot, which can find and photograph animals. This robot will be designed to sit stationary in nature. It will then rotate to capture images of animals in its environment and automatically edit the photos. While a photographer needs to be paid per hour or day, a robot only requires a one-time payment. Furthermore, a robot will not get distracted or tired while waiting. This difference will enable companies to search for photographs for longer hours and on more days. The reduced costs may also allow companies to purchase multiple systems and survey a larger area for the same cost. The idea of photography robots is not new. For example, a remote-control buggy has been used to take photos of dangerous animals from close up [2]. In our research, we found many examples of remote-controlled photography robots, but this approach does not solve the problem of high human labor costs. Furthermore, the autonomous photography systems we found were for very simple applications, like photographing a stationary object from a close distance [3]. Our system extends the idea of autonomous photography to solve a shortcoming of modern animal photography. Our goal for this project is to prove that this approach is feasible by producing a functioning robotic system.

2 USE-CASE REQUIREMENTS

To adequately emulate the capturing of animals in nature, the system must be able to detect animals up to 25 meters (the necessary distance to photograph birds in nearby trees) away with a recall rate of 75 percent. Recall is a more important metric than precision or accuracy, because animal appearances are sparse and removing irrelevant photos is a quick process. An autonomous system with 75 percent recall would photograph as many animals as a perfect human would in just 33 percent more time. While humans certainly do not have 100 percent recall, we decided that this is a reasonable trade off considering the reduced human effort. Animals are not stationary, so the detection must happen in a timely manner and the robot must follow animals after detection. We decided that the system must detect animal within 15 seconds. Photographing a running animal or flying bird is difficult even for many humans (especially when done without a professional grade camera), but the system should be able to follow and photograph a walking animal. A walking animal moves approximately 2m/s and could walk the entire search diameter in 25 seconds. Assuming animals will only walk through the center of our search radius is unreasonable, so we decided that 15 seconds is a more practical time window. Furthermore, the robot should be capable of following an animal moving at 2 m/s to continue taking pictures. Performing professional level photography will be difficult, as the cost of most DSLR cameras far exceeds our budget. However, the most common and accessible form of photography is phone photography, and we think this is an obtainable goal. Most modern phones have 8-12MP cameras and in-app editing software. In addition to the technical capabilities, human photographers have the ability to properly zoom and focus when capturing photos. Along with being shot with an 8MP sensor, the photo should be of a quality indistinguishable from a human shot and edited photograph. To quantitatively measure this, we will have human testers attempt to distinguish our photos from human captured photos. These testers should not do better than guessing (50 percent accuracy) with any statistical significance when labeling photos as robot or human pictures. By this metric, the system's camera should also be capable of at least 2x optical zoom. The system should also be able to pan at least camera 180° in both the x and y-directions to be able to track/detect all accessible animals in its vicinity

Table 2.

Table 1: Use-Case Requirements

System	Task	Requirement
Detection	Recall	75 Percent
Search	Search Time	15 Seconds
Tracking	Max Tracking Speed	2m/s
Photography	Picture Quality	8MP
System	Range of Motion	180 Degrees Pan and Tilt
Editing	Differentiable from Human Editing	50 Percent

3 ARCHITECTURE AND/OR PRINCIPLE OF OPERATION

Figure 1 depicts a software diagram for the system's principle of operation. The camera pans and scans for animals in its environment. In the case of multiple animals in frame, the system chooses the first detected animal. Once an animal is detected, the system's KLT begins to track and follow the target, assuring that it is appropriately centered and that the camera has zoomed in so that the target is occupying approximately 40 percent of the photo frame. Meanwhile, photographs are taken and then run through the system's photo editing algorithm to produce a more professional grade photo of the target animal. If the target exits the KLT frame of tracking and is lost, the KLT is halted and the camera begins to search for an animal target again.

Figure 2 contains a hardware diagram of the systems architecture. The camera is an Arducam 8MP IMX219 PTZ camera that is connected to a MIPI CSI-2 port. The camera is mounted on a PTZ stage consisting of 4 Servo motors (one motor for each of the pan, tilt, zoom, and focus functionalities of the cameras). The Nvidia Jetson Nano draws power from its USB-C port, the pan/tilt motors draw power from a 9V battery, and the zoom/focus motors draw power from the Jetson Nano. The servo motors are controlled by the SDA and SCL pins in the GPIO Relay of the Nano.

4 DESIGN REQUIREMENTS

In order to pass the use-case requirements outlined in Section 2, our proposed hardware and software layout must pass as a set of more specific design requirements. For example, to detect animals in a 25m radius within 15 seconds with 75 percent recall, **we must certainly have an animal detection algorithm with at least 75 percent recall.** The equation for recall is shown in Equation 1. Even with a perfect searching algorithm, reaching the desired recall level would otherwise be impossible. Furthermore, to enable a complete search of the 25m radius, **the cameras must be able to pan at least 180 degrees and tilt 90 degrees up and down.** The time constraint outlined by the detection requirement also outlines a joint requirement for the detection algorithm and the embedded computer. The more images that can be processed in 15 seconds, the more complete the system's search can be. As a result, **once the 75 percent recall level is reached for the detection algorithm, the speed of the hardware/detection algorithm pairing should be maximized.**

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{1}$$

In order to properly track and photograph animals moving at 2m/s, the system must pass another set of necessary benchmarks. At 2m/s, an animal 5m away could escape our camera's field of view from the center in under half a second. With this in mind it is essential our tracking algorithm can process multiple frames in this time to estimated and react to the animal's motion. We believe **at least 15 frames per second will be necessary for the tracking algorithm**, though testing will give a more accurate number. Our goal for photo quality is to have the robot's photos be indistinguishable from photos taken by a novice using a smart phone. In order to meet this goal, our photo editing algorithm must have the ability to make photo adjustments seen in smart phones. **As a result, we will require the implementation of the most commonly used algorithms: temperature, tint, exposure, contrast, vibrancy, saturation, and sharpness [5].** Additionally, the **camera will need to be 8MP** or above to meet the smartphone quality use-case requirement. It is possible to meet the 8MP requirement using a lower quality camera and a up sampling algorithm. However, completing this method effectively would be quite complex and require a large time commitment. As a result, we have decided to avoid this route and buy an adequate camera.

5 DESIGN TRADE STUDIES

Meeting the use-case and design requirements is a difficult task. There is no clear-cut way to accomplish our goals, and choices benefiting our progress towards one requirement may hinder our progress towards another. With this in mind, it is essential to evaluate tradeoffs between potential implementations for each of our sub systems. Where possible, we use pre-existing research for these evaluations, but some tradeoffs must be evaluated through our own testing.

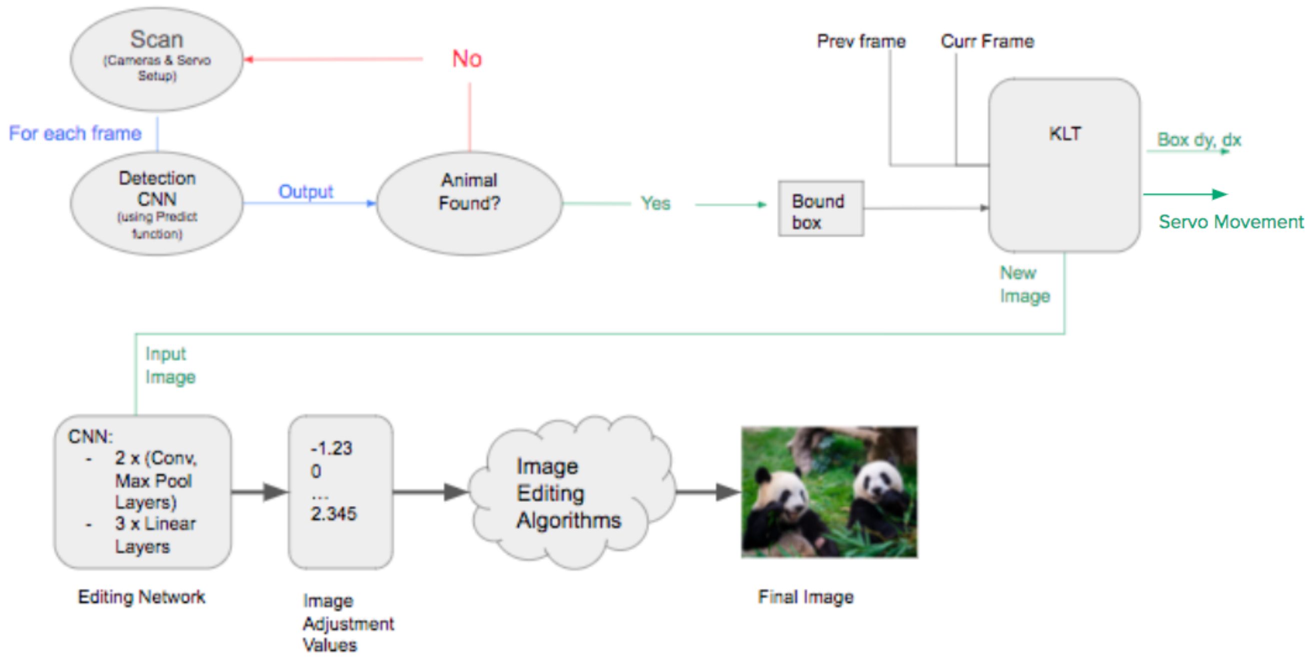


Figure 1: The software architecture of the Nature Photography Robot.

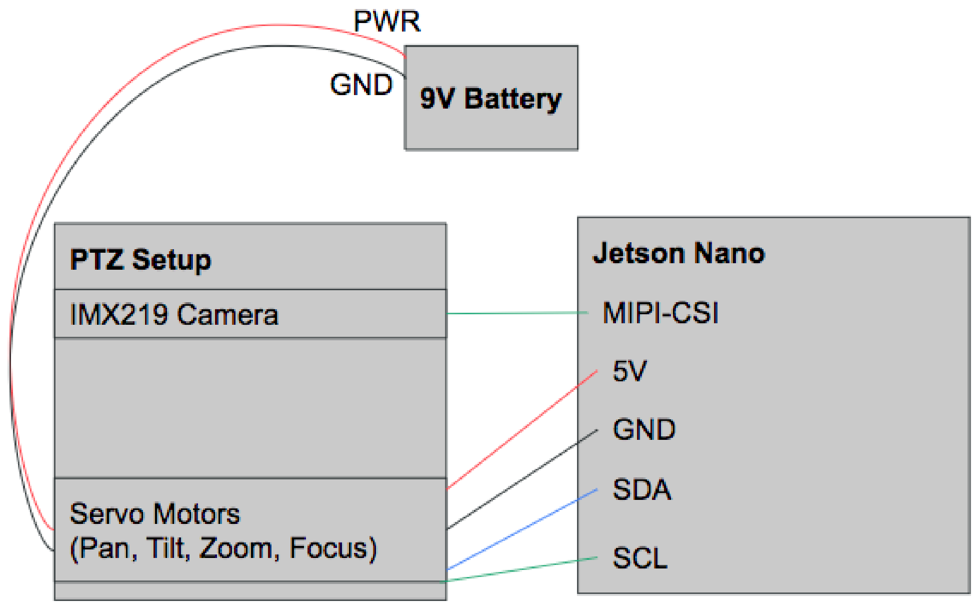


Figure 2: The hardware architecture of the Nature Photography Robot.

5.1 Hardware

Our design requirements require the animal detection algorithm to perform with 75 percent recall and the maximum possible speed. When considering embedded systems, the two choices available in our class inventory were the Raspberry Pi's and NVIDIA Jetson Nano. We found that no popular CNN backbones ran faster than 3FPS on a Raspberry Pi, so we decided to use a Jetson for our project [6].

When buying cameras, we sought to find the most affordable and easy to use hardware that met our design requirements. In order to meet our range requirement of 25m, we preferred a camera with adjustable zoom. This feature allows our robot to search for far distance animals without compromising the detection of close up animals. We also wanted a camera with native Jetson Nano compatibility to reduce setup time. There are a very limited number of cameras that meet these two requirements. We chose the cheapest camera in this group, the Arducam IMX219 PTZ Camera. This camera met our 8MP design requirement as well, which was ideal. Uctronics made an affordable PTZ base for this camera that met our mobility requirements, so we purchased this as well.

Our original plan was to use two cameras for the robot. The idea was that one camera could zoom in to take a close up photograph. Meanwhile the second camera could track the animal while at a farther zoom. This approach would enable closer photos to be taken of the animals. However, we ran into difficulties while implementing the two camera robot. First, our model of the Jetson Nano only has one MIPI-CSI port. We purchased a multi-camera adapter to avoid this problem, but only one camera stream can use the MIPI-CSI port at a time. This limitation prevents two MIPI-CSI cameras from simultaneously streaming, soiling our idea of a real time two camera tracker. Another problem with a two camera setup is the necessity of aligning the cameras. Logistically, the cameras must be vertically stacked. This means there are two dimensions that the cameras must align over (tilt and zoom) [6]. This procedure is not trivial and also adds computation to the time sensitive tracking process. While the perfect outcome is a closer photo, the more realistic outcome is frequent alignment errors. We could have used a USB camera to achieve multiple streams, but we decided the benefits of a two camera implementation were not worth the time and resources.

5.2 Detection Algorithm

Most modern papers on CNN's do not present recall as a metric, but their accuracies on challenging datasets far exceeds 75 percent [5]. Furthermore, our algorithm only needs to detect one class, animal. In contrast, research papers use datasets with many classes that are closely related, so our network should outperform the results presented in papers. It appears that most state-of-the-art approaches for object detection will perform well enough for our recall requirement, so our focus in selecting an algorithm is instead on speed. After researching detection algorithms, we decided to use yoloV5. This network uses a single stage compared to multi stage object detectors. Tests have shown that yoloV5 is faster than other popular object detectors using EfficientNet and ResNet backgrounds [7]. In addition, we found yolov5 to be one of the most well documented and accessible state of the art models [8].

Several design trade offs in the search and detection pipeline could be fully explored through research alone. For example, decisions like number of training epochs require validation testing. Also, the confidence threshold for animal detection should be determined through testing, because it is unknown how the model will generalize to the real world.

5.3 Tracking Algorithm

Initially, a Numpy based Bakers-Matthews tracker was considered for the base tracker of the system. Upon further analysis, it was decided that a simpler and faster algorithm would be more adequate for this use case. Many animals are capable of moving at considerably high speeds, and so a Lucas-Kanade tracker was chosen to optimize the speed of the frame-to-frame analysis.

An OpenCV based tracker [9] was chosen over the original Numpy based tracker made from scratch primarily for the support already provided by the OpenCV library. Real-time rendering of each video frame was bottle necked by the Matplotlib library's imshow function, which would require a wait period of 0.01 seconds after displaying each frame (before displaying the next one). Moreover, OpenCV's library allowed for easy implementation of interactivity between the tester and window in which a frame/video is diplayed, allowing for easier and more adjustable testing via a click event led bounding box.

The OpenCV based KLT was also capable of fine tuning via its parameters (window size and max pyramid level). Section 7 includes a test that shows the capabilities of the KLT at three different window sizes and three different max pyramid levels for animal pictures at three different distances from the camera. Smaller window sizes with higher pyramid levels were more appropriate for animals that were placed farther from the camera, while animals that were closer were easier to track using a medium/large window size and low pyramid level.

5.4 Editing Algorithm

Photo editing is a far less studied problem than CNN's for object detection. Most methods existing use pixel-by-pixel manipulations using GAN's or other Deep Neural Networks [10]. These methods may work well, but they do not align with our goals. Our use case is to emulate human photography on a phone, which involves applying a certain number of algorithms like sharpening. Using a pixel-by-pixel approach can distort an image making the original impossible to reconstruct. It also provides no explainability or modifiability to the result. Automatic editing algorithms like we are looking for exist in photoshop and on iOS but are not opensource. With this in mind we will need to experiment with our own methods.

We tested two ideas to solve this problem. The first method follows heuristic editing rules. This approach is inspired by histogram equalization (?), which normalizes the cumulative density functions of image values to avoid sharp peaks and improve contrast. Our idea is to construct a set of rules to transform an image and improve its appearance. We are concerned that this approach is too 'one size fits all', and it will be difficult to create rules for problems like blur. There are also documented difficulties getting histogram equalization to work for color images (?). The second idea is to train a CNN to take an input image and returns values to apply image editing algorithms. The motivation is to create a more adaptable method for image editing.

6 SYSTEM IMPLEMENTATION

6.1 Setup

In our purchases, we tried to minimize the setup work. We still needed to use several libraries to receive data and control our camera and servos. For our camera, we used GStreamer in addition to OpenCV to read frames [11]. We also used the smbus to access the GPIO pins on the Jetson Nano [12]. The SDA and SCL ports were used to communicate with the motors, and this library enabled this.

6.2 Search and Detection

```
while animal not found:
        if x = max_x or x = min_x:
                step y
                change x direction
        if y = max_y or y = min_y:
                step zoom
                change y direction
        if zoom = max_zoom
                set zoom to original
        step x
        get frame
        run detection model
```

Figure 3: Pseudocode for the search algorithm

Our search algorithm is shown in Figure 3. The method uses a constant scan of the search space. We assume that animals are equally likely in all locations, so a complete scan is the most efficient approach. We did have to find the ideal step size for the camera. We tested and chose the values that minimized overlap between consecutive frames. This maximizes the speed of the complete search.

For our detection model we used yoloV5, a deep convectional neural net. We chose this specific model because of its speed and convenience. Yolo networks divide the image into multiple sections and uses a single forward pass. This approach is considerably quicker than other recurrent detection networks. The v5 implementation applies a new technique, cross stage partial networks, to further speed up the detection [13]. For our final model we decided on a .45 confidence threshold for predictions and 20 training epochs. These values were decided by validation testing.

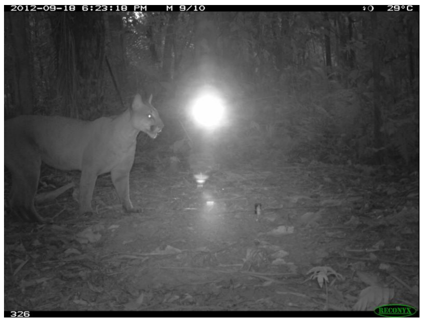


Figure 4: A picture from the WCS Camera Trap dataset. Notice the non-ideal lighting and quality. These obstructions are important for training a robust model.

We used the WCS Camera Trap dataset to train our detection model. This dataset was collected from cameras pointed at feeding sources all around the world. As a result there are 675 species and 375,000 images [14]. The large amount of data and variety of animals made this dataset perfect for our application. Many of these images are also lower quality and contain obstructed animals, adding to the potential for generalization. We ultimately used 20,000 of these images due to storage and time limitations.

6.3 Tracking Algorithm

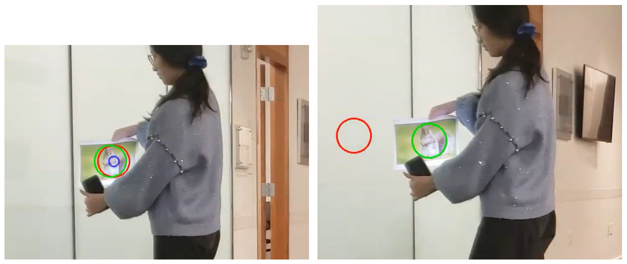


Figure 5: KLT test with the red circle indicating the initial position of the target (Tamal the squirrel), and the green circle indicating the tracker's prediction of the target's new location.

Using the OpenCV library, each frame of the video source of interest is read, with each pair of frames acting as inputs to the OpenCV KLT, along with the x and y coordinates of the target (animal) in the old frame in this pair. The KLT then returns the location of the target in the current/new frame. The KLT works with a window size of 30 pixels x 30 pixels, and a max pyramid level of 2. These parameter values are sufficient in tracking farther animals (20 meters from the camera) that appear to be smaller and closer animals (5 meters from the camera) that seem bigger and easier to follow.

Both sets of points are indicated by a circle as shown in Figure 5. The difference between these two point coordinates is then translated into motor steps for the servo motors that pan and tilt the camera.

6.4 Photo Editing

6.4.1 Heuristic Editing Approach

Our attempt of producing a heuristic editing algorithm began with running histogram equalization on our image. Histogram equalization augments the pixel intensity values such that the CDF of their distribution is linear [15]. The problem with this approach is that it only works for individual image channels. To solve this problem we tried applying the algorithm to the Red,Green,and Blue channels separately. This approach created large color distortions. As a second attempt, we converted the image to HSV space and performing the operation on the value channel. We also searched to find a heuristic approach to apply sharpening/blurring, but our research and attempts were unsuccessful.

6.4.2 Image Processing Library

In order to make a functioning photo editing algorithm, our first step was implementing the necessary image editing algorithms. After researching photo editing, we decided on a set of seven algorithms (exposure, contrast, temperature, tint, vibrance, saturation, and sharpening)

Exposure: Exposure in photography refers to the amount of light allowed in the camera. We emulated this using an exponentially scaled multiplier shown in Equation 2 [16]. The input, val, can fall in the range $[-2, 2]$, which was determined experimentally.

$$out = \text{clip}(in * 2^{val}, 0, 255) \tag{2}$$

Contrast: Contrast in photography refers difference between light and dark pixels. We used an implementation by DRKStudios for this algorithm, which is shown in equation 3 [17]. For this algorithm val can be in the range $[-255, 255]$

$$out = \text{clip}(\frac{259(val + 255)}{255(259 - val}(in - 128) + 128, 0, 255) \tag{3}$$

Temperature and Tint: We implemented temperature and tint as pixel wise adjustment to the images colors in the range $[-255, 255]$. Temperature is an adjustment on a blue/yellow scale, and tint is an adjustment on a purple/-green scale [4].

Saturation: Saturation is also a pixel wise adjustment to the images colors in the range $[-255, 255]$. However this augmentation adds and subtracts from all color channels [4].

Blur and Sharpen: We implemented blur and sharpening as positive and negative adjustments using a single algorithm. For blurring we applied a gaussian filter to the image, we adjusted the blur by setting the gaussian covariance to the input value[18]. For sharpening we convolved the image with an odd sized kernel that has -1 values in the middle column and row. The center value is set to keep the kernel's magnitude to 1, to not change the energy of the image. An example of a 3x3 kernel is shown in equation 4. We originally set the kernel size equal to the input value, but this approach only allows for values that are odd integers. To resolve this problem, we calculate the image using the next largest and smallest valid kernels. Afterwards we use a weighted combination of these images as the output.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{4}$$

Vibrance: The idea behind the vibrance algorithm is to increase the color of dull shades while maintaining the color of bright shades. This approach prevents the image from becoming washed out like with the saturation algorithm. To accomplish this effect we use a shifted and stretched tanh function shown in equation 5. To apply this algorithm, the image is converted into hue, saturation, and value space. The algorithm is only applied to the saturation channel. The range for this function is $[0, inf)$

$$out = \text{clip}(255 * \frac{\tanh(\frac{val*in}{255})}{\tanh(1)} + .5, 0, 255) \tag{5}$$

It was necessary to create inverses for our image processing algorithms to create data for our editing network. This way we can take an edited image, apply an inverse transformation, and know the necessary editing amounts to restore the image. For most algorithms the inverse was mathematically calculable. However, blur and sharpening do not have a mathematical inverse. These algorithms are roughly opposite of each other, so we attempted to roughly align these algorithms. Our approach involved applying blur and then sharpening for multiple values. We found the pairs with the minimum MSE with the original image and estimated the relationship between the two algorithms. This relationship was roughly linear, and shown in figure 5. It is important to note that the inverse functions we created are not perfect. Values are rounded to integers and clipped between 0 to 255. As a result, large applications of any algorithm cannot be well restored.

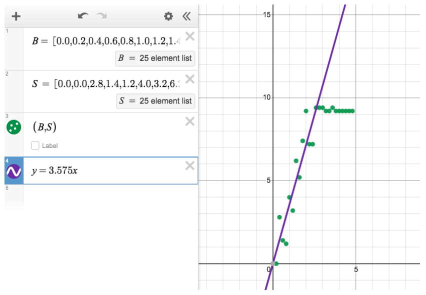


Figure 6: The relationship observed between our blur and sharpening algorithms

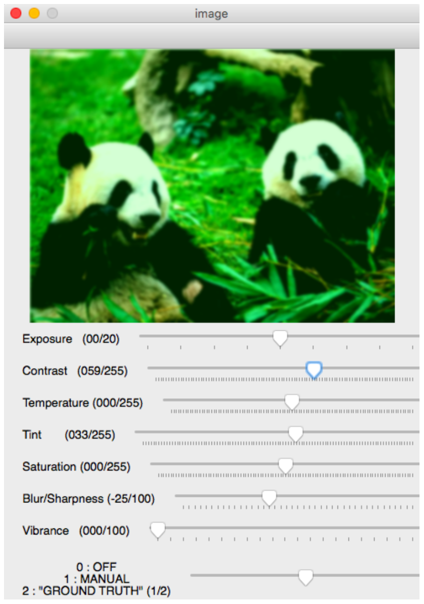
6.4.3 Neural Editing Approach

For our editing network we used the Kaggle Animal Classification dataset [19], because most of these images were well edited pictures of animals. We randomly augmented the images with our inverse functions and stored the restoration values as the output. We applied two random augmentations of each image, leading to a custom editing dataset with about 10,000 images/editing value pairs. For our network we used a convolution neural net with the following structure

- 2d Convolution layer with 6 output channels and a 5x5 kernel, followed by ReLu activation and max pooling
- 2d Convolution layer with 16 output channels and a 5x5 kernel, followed by ReLu activation and max pooling
- Linear layer with output size 120 and ReLu activation.
- Linear layer with output size 84 and ReLu activation.
- Linear layer with output size 7.

We used ADAM and MSE loss for the optimization.

6.4.4 Editing GUI



Exposure (00/20)
Contrast (059/255)
Temperature (000/255)
Tint (033/255)
Saturation (000/255)
Blur/Sharpness (-25/100)
Vibrance (000/100)
0 : OFF
1 : MANUAL
2 : "GROUND TRUTH" (1/2)

Figure 7: The editing GUI

In testing, we saw limited effectiveness for both the heuristic and neural editing methods. With this in mind, we still wanted to permit users to edit their photographs. We used the OpenCV sliders functionality to create a GUI to edit their photographs. This GUI is shown in Figure 6.

7 TEST & VALIDATION

7.1 Results for Detection Recall

Our detection requirement was 75 percent recall. In training, our results far exceeded our requirement. We saw a recall of 92.8 percent on our validation data. These results are shown in figure 7.

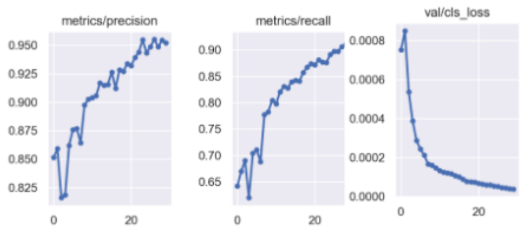


Figure 8: The yoloV5 training results

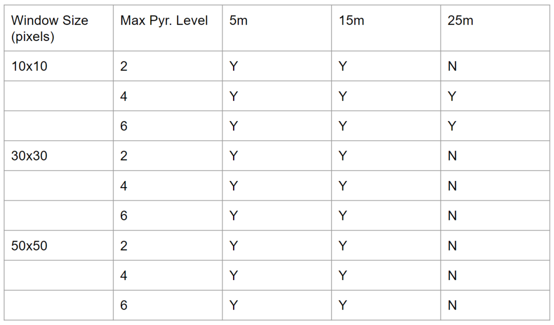
When we tested our detection algorithm with our camera in the real world, we saw a considerable drop in recall. To perform these tests, we placed an image of an animal in front of the camera at various distances. We saw a recall of 82 percent under 15 meters,and a recall of 54 percent between 15 and 25 meters. We fell below our goal in this farther range, but we believe a large part of this failure our low camera quality. Additionally, we saw a larger number of human/non-animal false defections. We think this was due to the low number of human and empty frames in our data.

7.2 Results for Search Time

To test our search time, we ran our search algorithm for a complete pan/tilt cycle. In our first run through we found that the process was very slow. Every run of the detection model took 4.6 seconds. We were using CPU to run the model, but we were able to speed this up by switching to CUDA on the Jetson Nano. With CUDA the entire cycle only took approximately 21 seconds, which is slightly above our goal of 15 seconds. With more powerful computational resources, we believe that the 15 seconds goal is obtainable.

7.3 Results for Tracking Capability

To test the KLT's capabilities, a human was given a printed image of an animal and asked to stand 5 meters, 15 meters, then 25 meters away from the camera. Once the wait period passed, the tester walked or slowly jogged from side to side while holding the picture. The KLT would then be tested with the specific parameters listed in Figure 9. Three different square window sizes were tested along with 3 differnt numbers of KLT pyramid levels. Each pyramid level represents a sub-sampled version of the frame's image, with each proceeding level containing half the number of pixels as the previous level. The higher the level, the more sampled the image. Under mildly lit - good lighting conditions, the tracker worked adequately for animals 5m and 15m away. Once the animals reached a distance of 25m from the camera, the tracker had some difficulties maintaining the bounding box on target, and would need a smaller initial window size and more pyramid levels for accuracy. A successfully tracked animal remained within the bounding box for at least 3 seconds.



Window Size (pixels)	Max Pyr. Level	5m	15m	25m
10x10	2	Y	Y	N
	4	Y	Y	Y
	6	Y	Y	Y
30x30	2	Y	Y	N
	4	Y	Y	N
	6	Y	Y	N
50x50	2	Y	Y	N
	4	Y	Y	N
	6	Y	Y	N

Figure 9: Results for KLT test on animals at different distances. Successfully tracked targets indicated by "Y", unsuccessfully tracked targets indicated by "N".

7.4 Results for Photo Editing

To test the photo editing algorithms, we chose 150 images not used in training and divided them into 3 sets of 50. One set for no editing, one for heuristic editing, and one for neural editing. Each image is augmented, edited, and shown to a tester with the original image. The tester has to guess what the original image is. An ideal image editing algorithm will have the testers guess correct 50 percent of the time.

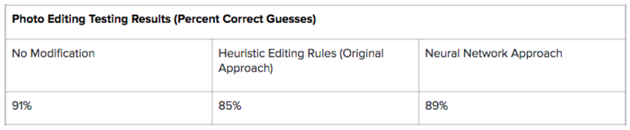


Photo Editing Testing Results (Percent Correct Guesses)		
No Modification	Heuristic Editing Rules (Original Approach)	Neural Network Approach
91%	85%	89%

Figure 10: Photo editing testing results

Our results are shown in figure 9 . Originally, we thought that the testing results for the CNN were higher, but this was a coding bug accidentally showing the ground truth edit. This problem has been corrected from the design report. Overall, neither of our editing approaches performed well, especially considering our goal was 50 percent tester accuracy. We believe the heuristic approach was too simple and "one size fits all" for the many possible augmentations. For the CNN we suspect two issues. First, we believe that our inverse algorithms were not effective because information is lost by clipping the original augmentation. This creates low quality data. Second, we believe our architecture was inappropriate for the task. Overall, this problem is largely unstudied except for major corporations like apple and adobe, and this was an ambitious task to attempt.

8 PROJECT MANAGEMENT

8.1 Schedule

We planned to divide the Setup, Detection, Tracking and Editing sections of our system. Our plan includes finishing the physical setup and include time for the detection, tracking and editing phases as well as preliminary testing right after the physical setup. This is followed by the final integration and testing of the system as highlighted in Figure 5 below.

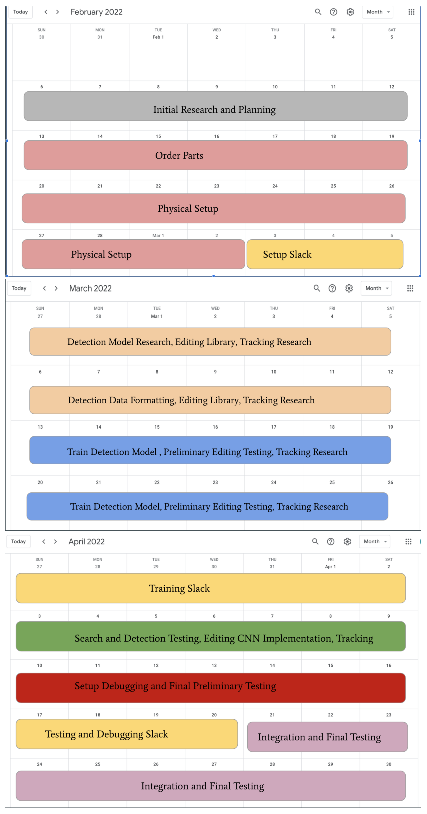


Figure 11: Gantt Chart of the Schedule

Compared to our initial plans, ordering parts and creating the physical setup took longer than anticipated. Due to problems faced during data formatting and tracking, we were forced to push back future tasks and had to dedicate less time integration and final testing.

8.2 Team Member Responsibilities

We divided the responsibilities among our team by different sections of the project as outlined in the schedule. To be specific:

- Justin took charge of Image editing and put together the physical setup of the robot and the interactive editing User Interface.
- Justin played a major role in integration of the project along with Sid.
- Sid was the Search and Detection lead for the project and was responsible for the electrical setup of the system.
- Sid was in charge of the Testing of the integrated system along with Justin.
- Fernando was in charge of the Tracking part of the project and software setup.

8.3 Bill of Materials and Budget

The Bill of Materials mentions a few parts that we did not use in our system. Our initial plan highlighted using a two camera system which called for the use of an adapter. Due to problems faced while integrating the system, we were forced to pivot our approach to use one camera. As a result, the final system featured the use of 1 Arducam 8MP PTZ Camera and 1 Digital Servo Kit as well as Jetson NANO, for computation, and a battery pack which was used as a power source (These have not been included in the Bill of Materials).

The final Bill of Materials needed for the project which highlight the use of the budgets, can be found in Table 2.

8.4 Risk Management

The critical risk factors we identified for the project is the ability to detect animals in the defined time frame and accurately. If we are unable to detect the animal while it is in the related environment, we will not be able to capture a photograph of the animal since to the system, it does not exist. Therefore, detecting the animal with a bounding box around it is a pivotal aspect of the project. We used a pre-trained CNN model which already has established validity to prevent this from being a factor in our project. In case we faced possible failure with this due to an inaccurate model, we planned to train with a larger data set and for more epochs.

Another major risk we identified over the course of the project was issues created by version and system incompatibility. To be specific, running a variety of algorithms on the system meant importing a large number of libraries. This caused issues with conflicting versions and dependencies that required large amounts of time for debugging. The major fallback for this was trying to identify these bottlenecks beforehand and attempt to allocate time for debugging that may be required during integration phases of the project.

In the context of personnel and schedule, the risks faced over the semester were due to problems faced in different speeds of development in the different phases of the project. The initial editing library was ready and tested before the tracking and detection models could be tested. This pushed back integration of that aspect of the system and caused slight wastage in time. To deal with this, we made changes to the schedule and began other work simultaneously to get ahead in other areas of the project. Once the search and detection model was ready, we integrated that with the entire system and resumed testing. However we were forced to pivot our approach and use case since the tracking algorithm was not ready and could not be included in the system.

9 ETHICAL ISSUES

A great deal of the ethical issues pertaining to the Photorobo are centered on the user of the system and

Table 2: Bill of materials

Description	Manufacturer	Seller	Quantity	Cost @	Total
8MP Pan Tilt Zoom Camera	Arducam	UCTronics	2	$94.99	$189.98
2 DoF Pan Tilt Digital Servo Kit	Arducam	UCTronics	2	$89.99	$179.98
Multi Camera Adapter module V2.2	Arducam	UCTronics	1	$49.99	$49.99
					$ 460.00

their intentions in detecting and tracking various species of wildlife. In the wrong hands, the system may be used to facilitate the poaching of animals, particularly those that are endangered or of high value in poachers' markets. The system would have to be administered by a trustworthy organization that would be conscious of its customers and perhaps capable of running background checks or surveys of the like to gauge their clients' intentions/trustworthiness.

Of course this raises the possibility of selling the robot in second-hand markets, meaning it would inevitably fall into the wrong hands. For the sake of protecting endangered/frequently targeted animals, the system's detector should be trained to avoid detecting these animals, with administrative access to the detector's trained CNN. This would prevent any malicious users from replacing an ethically trained CNN with one more suitable for finding animals that are illegally hunted. The CNN may also be replaced by another that was trained in detecting more niche or unrelated targets that are more deserving of privacy (i.e. specific humans, humans in general, passenger vehicles, etc). This administrative access would most likely be granted to wildlife preservation organizations.

10 RELATED WORK

After looking at the directory of past Carnegie Mellon ECE Capstone projects, we found related projects using Computer Vision solutions. Namely:

- Smart Wardrobe
- CV Studio
- Tartans Gambit

Over the course of the project we also noticed similarities with other current Capstone projects like:

- Project Projective
- Where the Milk?

During research we used a variety of online resources. Among these we also identified a few that had similar purposes and solutions implemented as our system.

11 SUMMARY

The system was not fully integrated, however each subsystem works well individually. With more time, all subsystems would work in tandem on the Jetson Nano. The camera's movement was rather jittery and abrupt, contributing to the blurry nature of many of the pictures taken by the Arducam. Perhaps a smaller motor step and smoother environment-scan can be implemented to fix this issue.

The KLT is capable of tracking animals moving at 2 m/s and those moving at faster speeds. With parameters tuned according to the target animal's distance from the camera, the tracker can follow targets up to 25 meters away, successfully meeting the design requirements that were initially set. Due to time constraints caused by difficulties met during the configuration of the Jetson Nano, this system was not integrated into the final product.

The photo editing subsystem has a wide range of tools used by photographers to enhance their shots, with each tool having a dedicated spectrum of intensity that would allow for fine-grained editing adequate for a wide variety of environments (lighting conditions, color temperatures, etc.). However, the photo editing is only manual, and would require a substantial amount of user input. With a CNN trained to apply various amounts of each editing tool to the initial images, the system would satisfy the automated editing design requirement.

Under strong lighting conditions, the system was able to consistently detect surrounding animals. Rooms under warm and/or slightly dimmed lighting negatively impacted the system's ability to detect the printed photos of animals. Besides a longer training duration with more epochs, the detector's subsystem would benefit from a training set that involved more empty images and images of humans to avoid false detections, which were not uncommon.

Many of the limits to the system are rooted in the hardware chosen for this project. The Arducam, while simple to program in conjunction with a Jetson computer, is not as capable as more specialized, professional grade cameras with stronger sensor resolutions, image stabilization functionality, wider variety of focal length options, etc. Indeed, a more advanced camera system could be used to significantly enhance the quality of the photos and remove some of the burden placed on the photo editing subsystem.

With more time, a suitable casing/ chassis would also be included, making the system insusceptible to water damage, heat damage, or even shock damage. All of these factors can easily be encountered in the wild.

The Jetson Nano came with a long list of difficulties in setting up a proper coding environment for this project. A considerable amount of systems knowledge would have been helpful in managing the dependencies needed to complete the project. These difficulties contributed to a long integration period, which was to be expected. Taking the time to set up an appropriate coding environment is key.

Glossary of Acronyms

Include an alphabetized list of acronyms if you have lots of these included in your document. Otherwise define the acronyms inline.

- DoF - Degrees of Freedom
- CNN - Convolutional Neural Network
- KLT - Kanade-Lucas Tracker

References

[1] Heggie, J. Take a photo save a species. https://www.nationalgeographic.com/photography/article/paid-content-take-a-photo-save-a-species-the-power-of-wildlife-photography

[2] Gallucci, M. These robots are transforming how we see wildlife. https://mashable.com/article/robots-wildlife-photography

[3] Dempsey, J. Photographers, are robots coming for your jobs. https://digital-photography-school.com/photographers-are-robots-coming-for-your-jobs

[4] REI. Photo Editing Basics. https://www.rei.com/learn/expert-advice/photo-editing-basics.html

[5] NVIDIA. Jetson Benchmarks. https://developer.nvidia.com/embedded/jetson-benchmarks

[6] OpenCV. Multi-Camera Calibration. https://docs.opencv.org/4.x/d2/d1c/tutorial$_{m}ulti_{c}amera_{m}ain.html$

[7] Dwivedi, P. YOLOv5 compared to Faster RCNN. https://towardsdatascience.com/yolov5-compared-to-faster-rcnn-who-wins-a771cd6c9fb4

[8] Nelson, J. YOLOv5 is here. https://blog.roboflow.com/yolov5-is-here/

[9] OpenCV. Optical Flow. https://docs.opencv.org/3.4/d4/dee/tutorial$_{o}ptical_{f}low.html$

[10] Meisner, B. What is the future of artificial ai in photo editing. https://www.forbes.com/sites/forbesbusinesscouncil/2022/01/20/what-is-the-future-of-artificial-intelligence-in-photo-editing/?sh=ef80e5c547c0

[11] Gstreamer. https://gstreamer.freedesktop.org/

[12] Shawn. How to use raspberry pi gpis pins. https://www.seeedstudio.com/blog/2020/02/19/how-to-use-raspberry-pi-gpio-pins-python-tutorial/

[13] Wang, C. Et al. CSPNet. https://arxiv.org/pdf/1911.11929.pc

[14] LILA. WCS Camera Traps. https://lila.science/datasets/wcsca

[15] Bhattacharyya, S. Histogram Equalization. https://towardsdatascience.com/histogram-equalization-a-simple-way-to-improve-the-contrast-of-your-image-bcd66596d815

[16] https://stackoverflow.com/questions/12166117/what-is-the-math-behind-exposure-adjustment-on-photoshop

[17] DFStudios. Contrast Adjustment. https://www.dfstudios.co.u programming-algorithms/image-processing-algorithms-part-5-contrast-adjustment/

[18] Edeza, T. Image Processing with Python. https://towardsdatascience.com/image-processing-with-python-blurring-and-sharpening-for-beginners-3bcebec0583a

[19] Banerjee, S. Animal Image Dataset. https://www.kaggle.com/datasets/iamsouravbanerjee/animal-image-dataset-90-different-animals?resource=download