



MREN318 Course project report

Automatic Pet Feeder

Queen's University, Kingston, Canada

Department of Electrical and Computer Engineering

Group Members:

 Name Student Number

Lucas Balog 20320245

Skyler Zhou 20321898

Samuel Sin 20324814

Overview:



Figure 1: Final prototype

Our pet feeder is an automated system that uses two sensors, an Arduino Uno Rev 4 Wifi, and a servo motor. It is primarily 3D printed, with a plastic container attached for food storage.

Sensors:

Two sensors were used in our final prototype: the camera and the ultrasonic distance sensor. However, many other sensors were considered and tested but were determined to be a poor fit for our design.

Camera:

Our camera system was designed to take a picture when a distance sensor senses an animal approaching. This image would then be sent to an AI algorithm to give an output to the Arduino. This output tells the system the breed of the animal, which in turn dispenses food based on the breed. Several issues were encountered when implementing the camera, including outdated libraries, saving images, and integration with the rest of the system.

The first issue occurred when initializing the camera. When setting up the camera, the Arducam Wiki Quick Start Guide [1] was used. Despite following the simple instructions exactly, none of the example code was able to compile. After some research, we realized that the Arduino Uno Rev 4 was not officially supported by the ArduCAM Mini. Luckily, user Keeeal on GitHub had updated the library to support Uno Rev 4 [2], and using this new library allowed the code to compile.

Next, we had to get the camera to automatically take pictures and store those images in an easily accessible place for integration. To achieve this, we tried using the ArduCAM_Mini_2MP_OV2640_functions example code [3] to activate when an infrared sensor

sends a signal. Each component worked separately however, the camera would only take pictures if the signal comes from the ArduCAM Host App [4]. When receiving the signal from the IR, the Host App prints the initialization and capture text, but no image appears. Due to this, we decided to bypass the Host App entirely and instead save the image to an SD card using the ArduCAM_Mini_2MP_Plus_Multi_Capture2SD example code [5].

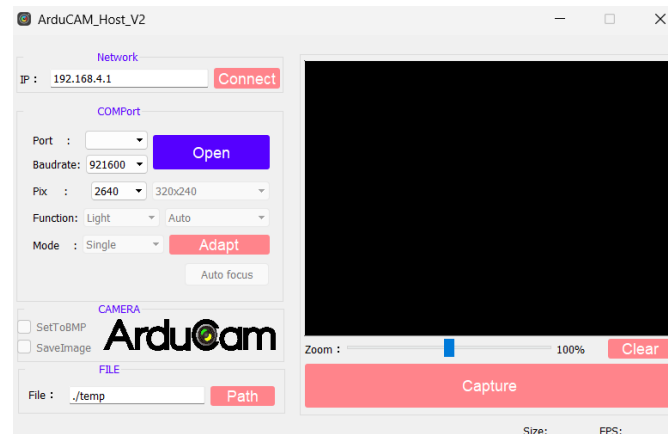


Figure 2: ArduCAM Host App V2

The camera system was difficult to integrate and had several issues. Most libraries are example code are written for outdated chips and codebases for Arduino, RaspberryPi, and Micropython Systems. Moreover, most of the SPI addresses were under documented and it was unclear what data was expected to be received. This created an inability to receive and properly process data from the camera in any custom system. Future iterations of the design should utilize a fully embedded camera system designed for and utilizing wireless communication protocol to share images.

Distance Sensors:

The feeder is designed to utilize a distance sensor to detect if an animal was within a certain range. To do this, the distance sensor would activate at the beginning of each cycle and return the distance of any object in front of it. We had initially decided to use the IR sensor as some animals can hear ultrasonic waves, except overtime the sensor's circuitry degraded. The sensor began outputting the same value regardless of the distance of the object in front of it and would only behave as expected if pressure was applied to the back of the device. It is suspected that this problem occurred due to cheap soldering. For this reason, the ultrasonic sensor was ultimately used.

Weight Sensors:

Our original design involved using sensors for portion control. Under the food bowl there would be a sensor reading the amount of food dispensed, stopping distribution once the desired weight was reached. However, we decided that none of the sensors provided were fit for this purpose.

The sensor we first chose was the SENS-5 load cell, as it has the most surface area and is the easiest to balance a food bowl on. When attempting to calibrate the sensor, we found that the readings were very inconsistent. Even when averaging multiple readings, there was a complete lack of pattern in the output. Furthermore, each time the Arduino was reset, there would be an even bigger difference in readings. The sensor would also drift after a minute of activation. With the completely inaccurate outputs, we decided to try one of the pressure sensors instead.

When choosing between the two pressure sensors, we decided that the FSR would be better than the Flexiforce sensor. This is because we used the FSR in lab, and the recommended circuit for the Flexiforce had a capacitor [6], which we did not have. We were concerned that without the capacitor, we would not be able to predict the output behaviour of the sensor, and therefore would not be able to calibrate it appropriately. The FSR was set up as instructed in our lab, and we found that it would consistently output 441 when there was no load. Using this as our offset, we then recorded the output values with weights on the sensor. As seen in Table 1, the readings became inconsistent once weight was applied. Additionally, the sensor experienced very bad hysteresis with the outputs differing greatly when adding weights compared to subtracting. Lastly, the sensing area is also very small which forces the weights to be exactly centered for accurate readings. This would be impossible to maintain if an animal starts pushing the food around while eating. Due to the overall unreliability of the sensors, we decided to implement portion control without the use of sensors.

Table 1: FSR Readings

Weight	Test 1	Test 2	Test 3
100 g	252	296	297
150 g	326	351	359
200 g	358	391	390
250 g	400	418	402

Actuators:

In our original design, the stepper motor was going to be used for its increased torque and unidirectional rotation system. This was changed for two reasons: mounting and driver issues. Since the motor shaft was to be mounted to smooth 3D printed parts, there was not enough friction to hold it in place. As for the second issue, the stepper driver constantly overloaded the Arduino's current protector causing the system to shutdown. In an attempt to fix this problem, we integrated an external circuit that should have resolved things. It instead heated the motor and shield to unsafe temperatures. As the stepper motor proved both unsafe and unusable, it was abandoned.

On the other hand, the servo motor allowed for better power efficiency as the coils were not made to maintain position like the stepper motor. The design was changed to rotate back and forth between 0 and 180 degrees, dispensing food at each maximum. However, due to this change, cereal would occasionally get stuck in the corners of the pushing mechanism as it lacks torque. We

decided that this was okay as kibble is smaller and easier to crush than stale Froot Loops, and the servo therefore had enough torque.

Software:

We created two main pieces of software for this pet feeder: an app to control the feeder, and an AI algorithm for pet identification.

App:

For app control and notifications, the system uses the Arduino IOT Cloud platform. This is an MQTT server hosted by Arduino that includes standardized variables and objects including a scheduling object. This scheduling object allows easy setup and reoccurrence setting by a user that only sends a message to the Arduino when the time is met. This reduces time processing computation on the Arduino itself. The server can track time for multiple devices. The system also includes triggers to send notifications on variable changes. This was used to alert users when the pet was fed. The Arduino app also showed the status of the system in terms of if a feed time is met and waiting for a pet, is a pet is detected, and setting feed portions.

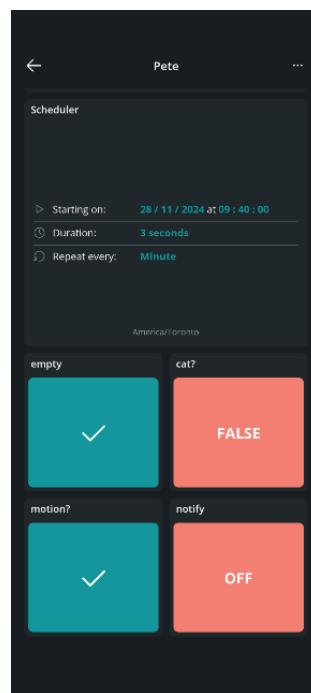


Figure 3: App Homepage

Pet identification:

The pet feeder uses a trained classification model to identify pets by their breeds. This identification feature would allow the pet feeder to feed multiple pets and control portions for different pets. The classification is done by computer vision. The trained model would take an input image and then output the breed with the highest probability that the image belongs to. The Oxford-

IIIT Pet Dataset was used for training and evaluation, which consists of 37 pet breeds with approximately 200 images per breed [7]. ResNet152 was chosen for this project due to its depth and use of residual connections, which allow it to effectively learn complex features. Its availability in PyTorch with pretrained weights on ImageNet enabled efficient transfer learning for classifying pet breeds.

The dataset had 37 pet breeds with approximately 200 images per breed a total of around 700 images (Figure 4). 20 images of each breed were split from the dataset into the validation set and the rest into the training set. Data augmentation was used on the training set to create more data for improving model's performance. The data augmentation code used in this project was adapted from the PyTorch official tutorials [8]. The normalization values used ([0.485, 0.456, 0.406] for the mean and [0.229, 0.224, 0.225] for the standard deviation) are derived from the ImageNet dataset [9]. This is commonly used in image classification tasks to standardize the input data for models pretrained on ImageNet.

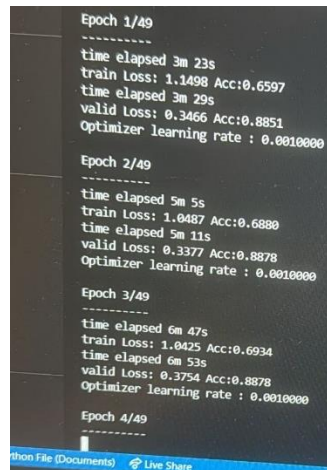
Breed	Count	Breed	Count
American Bulldog	200	Abyssinian	198
American Pit Bull Terrier	200	Bengal	200
Basset Hound	200	Birman	200
Beagle	200	Bombay	200
Boxer	199	British Shorthair	184
Chihuahua	200	Egyptian Mau	200
English Cocker Spaniel	196	Main Coon	190
English Setter	200	Persian	200
German Shorthaired	200	Ragdoll	200
Great Pyrenees	200	Russian Blue	200
Havanese	200	Siamese	199
Japanese Chin	200	Sphynx	200
Keeshond	199	Total	2371
Leonberger	200	2.Cat Breeds	
Miniature Pinscher	200		
Newfoundland	196	Family	Count
Pomeranian	200	Cat	2371
Pug	200	Dog	4978
Saint Bernard	200	Total	7349
Samoyed	200	3.Total Pets	
Scottish Terrier	199		
Shiba Inu	200		
Staffordshire Bull Terrier	189		
Wheaten Terrier	200		
Yorkshire Terrier	200		
Total	4978		

Figure 4: Dataset Statistics

The pre-trained model used is ResNet 152. ResNet152 is a deep convolutional neural network designed to handle very deep architectures using residual connections, which help prevent problems like vanishing gradients during training [10]. With 152 layers, it is highly effective for image classification tasks, especially when using pretrained weights from the ImageNet dataset [10]. These features make it a popular choice for transfer learning.

The model was trained using the Adam optimizer with a learning rate schedule. The learning rate scheduler used a step size of 5 epochs and a decay factor of 0.1. The loss function used was Negative Log-Likelihood Loss. The model was trained for 50 epochs with a batch size of 8. The number of epochs and batch size could be increased for better performance. But this might lead to longer training time and increase the burden on equipment (CPU/GPU). The device used for training

is NVIDIA GeForce RTX 3070 Laptop GPU. The training code used in this project was adapted from the PyTorch official tutorials for transfer learning and fine-tuning [8]. After each epoch the accuracy and time would be displayed shown in **Error! Reference source not found..** The total time used was approximately 160 minutes and the maximum accuracy was around 0.91 at the 47th epoch. Therefore, the lowest 2 output layers would be modified with the results of the 47th epoch and saved as a new trained model.



```

Epoch 1/49
-----
time elapsed 3m 23s
train loss: 1.1498 Acc:0.6597
time elapsed 3m 29s
valid loss: 0.3466 Acc:0.8851
Optimizer learning rate : 0.0010000

Epoch 2/49
-----
time elapsed 5m 5s
train loss: 1.0487 Acc:0.6880
time elapsed 5m 11s
valid loss: 0.3377 Acc:0.8878
Optimizer learning rate : 0.0010000

Epoch 3/49
-----
time elapsed 6m 47s
train loss: 1.0425 Acc:0.6934
time elapsed 6m 53s
valid loss: 0.3754 Acc:0.8878
Optimizer learning rate : 0.0010000

Epoch 4/49
-----

```

Figure 5: Output Display of Training Model

The trained model successfully predicts the breed of a pet from an input image, outputting an index corresponding to a specific breed. A .json file (Figure 6) was created to map the index to the breed's name for easier interpretation. However, the model has certain limitations. It cannot identify when an image contains no pets; instead, it outputs the breed with the highest predicted probability. Similarly, the model struggles with images containing multiple pets, as it is unable to differentiate between them. Furthermore, the model cannot recognize identical breeds of cats or dogs as separate entities within the same image.

```

1  {"1":"american bulldog","12":"american pit bull terrier","23":"basset hound","32":"beagle", "33":"boxer",
2  "34":"chihuahua","35":"english cocker spaniel","36":"english setter","37":"german shorthaired",
3  "2":"great pyrenees","3":"havanese","4":"japanese chin","5":"keeshond","6":"leonberger",
4  "7":"miniature pinscher","8":"newfoundland","9":"pomeranian","10":"pug","11":"saint bernard",
5  "13":"samoyed","14":"scottish terrier","15":"shina inu","16":"staffordshire bull terrier","17":"wheaten terrier",
6  "18":"yorkshire terrier","19":"abyssinian","20":"bengal","21":"birman","22":"bombay","24":"british shorthair",
7  "25":"egyptian mau","26":"main coon","27":"persian","28":"ragdoll","29":"russian blue","30":"siamese","31":"sphynx"}

```

Figure 6: json file used to map the index to the breed's name

Objectives:

Our final prototype meets all core objectives, as well as five desired objectives, with the potential for a sixth.

Core Objectives:

Portion control and feeding routines are managed through the app. The user can select a start date and time as well as repeating intervals ranging from one minute to one year. The number of rotations per distribution can also be selected, dispensing approximately $\frac{1}{2}$ a cup of kibble per rotation. This is currently listed at duration in the app from prototyping purposes.

Safety is ensured by minimizing the number of moving parts and blocking access to the center of the device with the pusher. Electronics and wires are kept in a separate box to prevent the animals from being able to chew on them. Production designs should incorporate internal wiring paths to ensure no external wires are available.

Secure food storage was achieved with a plastic food storage container on top with a screw on lid. As we did not have access to injection mold manufacturing, a candy package was used as a prototype. We tested and could not remove or open the container without unscrewing the top or cutting the tape and glue used to mount the container.

Affordability was achieved by reducing our filament usage and choosing the cheapest components. We utilized the servo motor and ultrasonic sensors as they were the lower cost components. We also put as much processing on server-side computing as possible to allow the lowest cost microcontrollers to be use in a production run. We estimate a standard ESP32 typically ranging from \$4-\$12 is the only processing device needed on a production run.


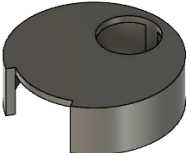
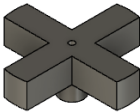

	Food Funnel
	Top cover to keep pet safe
	Pusher that blocks access to internals when in off state
	Base with slide to guide food falling out to bowl

Figure 7: Pet Feeder CAD Design

Desired Objectives:

As the app is the primary way to adjust the feeder, its simple user-friendly interface allows the user to make changes without being physically next to the feeder. Moreover, it is set up to send notifications when feeding variable has been changed, resulting in easy remote monitoring. Animal detection and feeding schedules can also be seen and adjusted in the app.

Using the AI model has allowed for pet identification. This feature allows for different animals to get different amounts of food without concern of the feeder dispensing food for the wrong animal. Currently, the system has only been tested for two pets, but has the capacity to support more, each with its own schedule and portion size.

Energy efficiency is achieved by having the most costly processing done in cloud (AI and scheduling). The servo motor is also more energy efficient than stepper for its reduced current required to move and maintain position.

Since app communicates through WiFi, the pet feeder holds the potential to integrate with Google Assistant. Furthermore, motion and AI sensors can be connected to the Google Home app.

References

- [1] "Arducam," [Online]. Available: <https://docs.arducam.com/Arduino-SPI-camera/Legacy-SPI-camera/Software/Quick-Start-Guide/>. [Accessed 17 October 2024].
- [2] Keeeal, "GitHub," 28 September 2024. [Online]. Available: <https://github.com/ArduCAM/Arduino/pull/595/files>. [Accessed 19 October 2024].
- [3] ad2d819, "GitHub," 26 July 2019. [Online]. Available: https://github.com/ArduCAM/Arduino/blob/master/ArduCAM/examples/mini/ArduCAM_Mini_2MP_OV2640_functions/ArduCAM_Mini_2MP_OV2640_functions.ino. [Accessed 19 October 2024].
- [4] f7e209f, "GitHub," 13 August 2018. [Online]. Available: https://github.com/ArduCAM/Arduino/tree/master/ArduCAM/examples/host_app. [Accessed 19 October 2024].
- [5] c6fcbde, "GitHub," 7 August 2018. [Online]. Available: https://github.com/ArduCAM/Arduino/tree/master/ArduCAM/examples/mini/ArduCAM_Mini_2MP_Plus_Multi_Capture2SD. [Accessed 27 October 2024].
- [6] Tekscan Inc., "Flexiforce Sensors User Manual," 30 May 2019. [Online]. Available: <https://www.tekscan.com/sites/default/files/FlexiForce%20Sensors%20RevL.pdf>. [Accessed 17 November 2024].
- [7] O. M. a. V. A. a. Z. A. a. J. C. V. Parkhi, "The Oxford-IIIT Pet Dataset," in *Computer Vision and Pattern Recognition*, Jun. 2012.
- [8] S. Chilamkurthy, "Transfer Learning Tutorial," Linux Foundation umbrella, [Online]. Available: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html#transfer-learning-for-computer-vision-tutorial. [Accessed 10 2024].
- [9] W. D. R. S. L. L. K. L. a. F. L. J. Deng, "ImageNet: A large-scale hierarchical image database," in *Comput. Vis. Pattern Recognit*, 2009.
- [10] X. Z. S. R. a. J. S. K. He, "Deep Residual Learning for Image Recognition," in *Comput. Vis. Pattern Recognit*, 2016.