CatDAR: A Rudimentary Automatic Litterbox System

Kristie Olds
College of Engineering: Computer Science
University of Idaho
Moscow, Idaho

***Abstract*- Using a combination of a Raspberry Pi, ESP32 and a Windows desktop with a graphics card, this project enables the user to keep track of litter box usage with multiple animals. This is possible with the use of a camera system used by an object detection algorithm to detect if an animal entered the room when passing a laser barrier.**

Keywords-Raspberry Pi, Adafruit Huzzah32 (ESP32), Windows, Machine Vision, Embedded System, Object Detection, Tensorflow, Keras, Teachable Machine, Python, C++, Sockets, Server/Client

I. INTRODUCTION

This project was an effort to combine embedded systems with machine vision. The idea was to be able to detect when an animal, a cat in this case, entered a room where their litterbox was located and add a tally to its "use". This is something that is very helpful to know when you and your roommate are busy college students and have 3 cats living with you; sometimes that's one of the last things you're thinking about. Systems similar to this are quite expensive, around $400 [1]; while they do add a feature of cleaning out the box for you, this alert system came in around $200 for all the components (before the chip shortage).

Fig. 1: Connection diagram

The system works by using socket connections between the 3 subsystems: (1) The Raspberry Pi, (2) an ESP32 and (3) a Windows desktop. The ESP32 is what manages the connections between the Pi and the Desktop and tells them when to start and stop processes. The Raspberry Pi has the task of starting and stopping a HiQT camera and making it accessible across the network. The Desktop then takes that camera stream and runs the object detection algorithm.

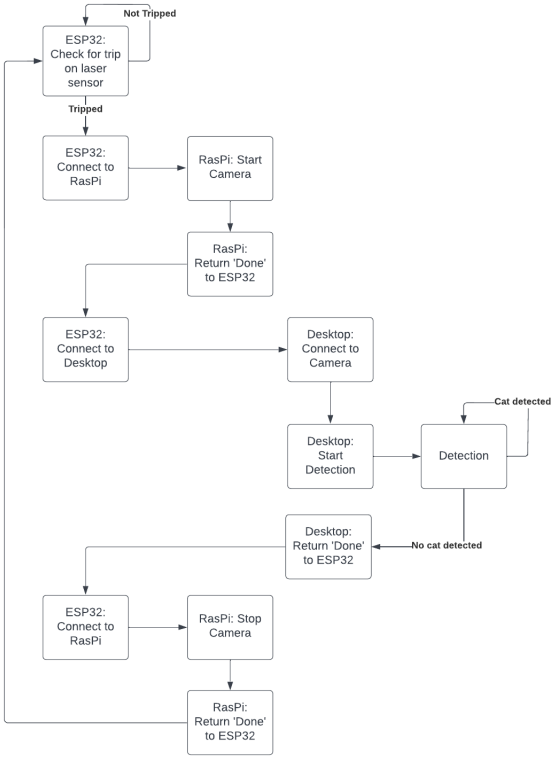

Fig. 2: Flow diagram of system

II. ESP32

The ESP32 is on an Adafruit Huzzah32; the board sports a Wi-Fi/Bluetooth connection, a bunch of GPIO pins and many other things that aren't super relevant here. This board was picked for its ability to

connect to a Wi-Fi connection, one of the main pillars of this project. On an attached breadboard, there are 2 status LEDs, a button and a laser sensor.

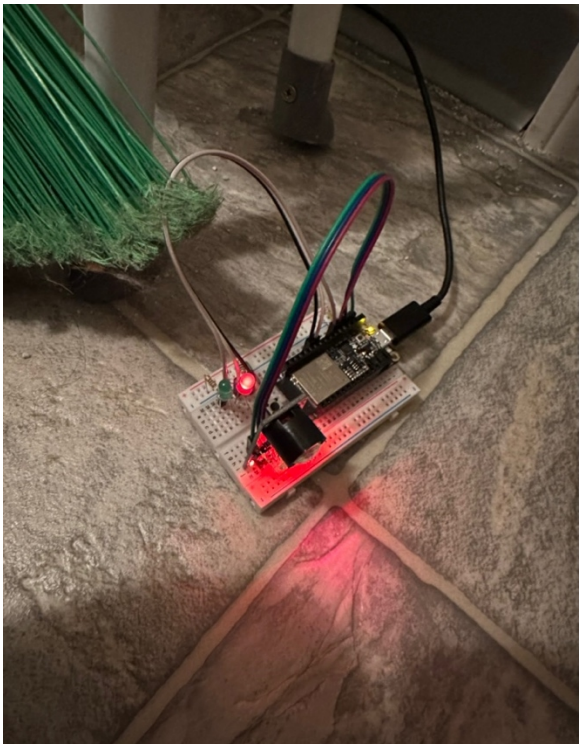

Fig. 3: Image of ESP32 set-up outside of box enclosure

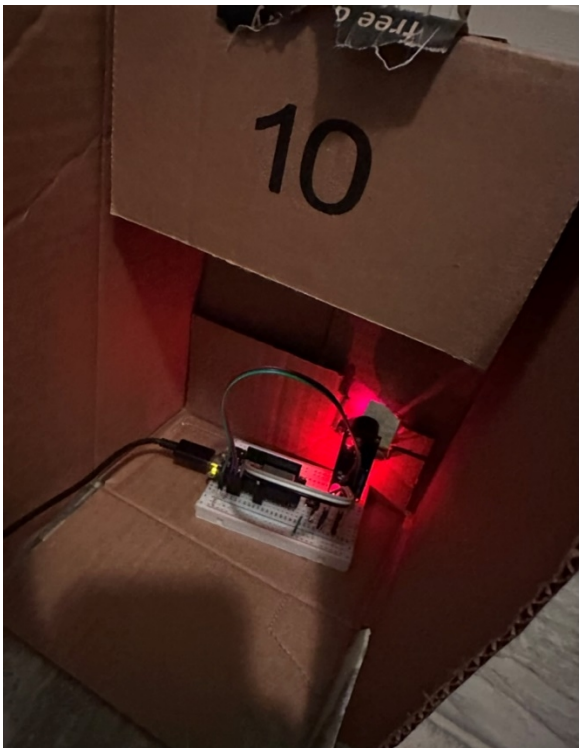

Fig. 4: ESP32 set-up in box (inside view)

Fig. 5: Outside box view

The 2 LEDs are to help with running this in a “headless” state, not attached to a device reading the serial output. The color-coded LEDs signify if laser sensor was tripped and to run, or is running, the detection algorithm (Red) or if its waiting to see a trip on the laser sensor (Green).

The button is for shutting down the software on the other 2 machines; this helps with port connections for the sockets; however, this does require reconnecting to these machines and restarting the software. Since software is always running on the ESP32 board, there is no “off” state; which means you need to get creative and make your own. This was done by making an infinite loop that flashes both LEDs to signify that it is in this loop state. The only way to get out of this state is to press the reset switch on the board manually; presumably after you have already restarted the software on the other machines, not doing so can cause some unexpected socket behavior.

The laser sensor is what starts the detection sequence, acting as a “tripwire”. This was done to save computational resources by allowing the detection algorithm to only run when the system knows there is most likely something in the room. This laser is set up to point straight across the doorway to a mirror that reflects it towards the ceiling. This was done because of a reflection of the laser beam back towards the sensor when something is in front of it; causing it to ‘trip’.

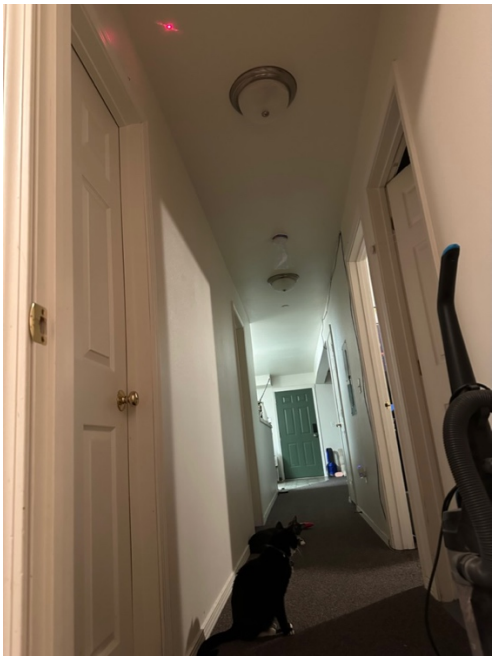

Fig. 6: Laser redirected to ceiling

When the software starts, it goes through a basic set up of connecting to the Wi-Fi network and setting up all the GPIO pins. The software sequence starts when the laser sensor is tripped by something walking in front of it, breaking the laser beam and causing a reflection to occur, pausing the main loop. It will then send a "command" via the socket to the Raspberry Pi to start up the camera; this can take a little bit for the Pi to get it set up, so there is a small delay right after. Once that's done, it then sends another "command" to the Desktop to start the detection software. This socket connection will wait until it gets a reply from the Desktop, this only happens once the detection is complete. When it does receive that message, it proceeds to send a "command" back to the Pi to shut off the camera and change the status LEDs, putting it back into the main loop.

III. RASPBERRY PI

The Raspberry Pi 3 Model B+ that was used is a compact computer using an ARM processor. You could run the Desktop's sequence on this, but this wasn't done due to software issues with Tensorflow Lite and lack of support for Tensorflow on ARM; this is because regular Tensorflow doesn't work on ARM processors natively and instead uses Tensorflow Lite for embedded systems like the Raspberry Pi. It also would have required a lot of adjustments to the software to change everything from regular Tensorflow to its Lite counterpart, which time did not allow for.

The camera connected to the Pi is a Raspberry Pi High Quality Camera with an AruCam 6mm 3MP Wide Angle Lens. When looking for a camera and lens, it needed to be compact but not for sacrificing quality; this one was probably a bit overdoing it, but having never purchased/used cameras like this, better over than under.

Fig. 7: Raspberry Pi set up above cabinet

When the software starts, it goes through and sets up the socket and attaches it to a port, it then waits and listens to the port for a "command" from the ESP32. The software sequence starts when it receives a "command" from the ESP32, it will then parse the message to figure out what it needs to do:

- If the "command" is 'Stop', it will break out of the sequence, close the socket and exit the program.
- If the "command" is 'Start', the sequence will continue and fork into 2 processes, the child process will run an execl function and run the camera routine. The parent process will continue the loop back and wait for the 'Off' "command".
- If the "command" is 'Off, the sequence will continue and runs a system call to kill the child process, turning off the camera.

A. *Camera Process*

The camera is able to be accessed by anything on the network, either by its IP:8000/index.html in a browser, or by its IP:8000/stream.mjpg; which is how the detection algorithm can access the camera. This script creates a webpage and updates it with the feed from the camera, which is helpful for troubleshooting. This was made with help over at *Random Nerd Tutorials* [2].

Fig. 8: Camera view from browser

```
pi@raspberrypi:~ $ ./ESPsock
Buffer: Start.
Forking
Parent continuing
child made...
Buffer: Off.
Buffer: Start.
Forking
Parent continuing
child made...
192.168.254.20 - - [16/Nov/2022 14:01:50] "GET /index.html HTTP/1.1" 200 -
192.168.254.20 - - [16/Nov/2022 14:01:50] "GET /stream.mjpg HTTP/1.1" 200 -
Buffer: Off.
```

Fig. 9: Testing of starting and stopping camera process

IV. DESKTOP

The desktop that was used runs Windows OS and has an Intel i9-9900K and a NIVIDA GeForce 2060 graphics card for creating the first object model and running the detection algorithm. The desktop was used because it has the horsepower to run these computationally intensive algorithms in a manageable time frame. While things like the Raspberry Pi could so some of the things the Desktop did, it would do so at a slower pace and would require more time dedicated to get software to work with each other.

Every other 'main' program was written in C++, while all the subroutines that were called off that were in Python. This complicates things for when you need to run a Python script from within a C++ program. To get that to work you need to turn your Python code into an executable with Pyinstaller [3] so that you can run it in an execl call. However, Windows OS is built where things like fork don't work like they do in Unix systems to create parent/child processes. To work around this, socket creation and running the detection all happens in one 'main' Python program.

When the software starts, it goes through, like on the Raspberry Pi, and sets up the socket and attaches it to a port. It will then wait and listen to the port for a "command" from the ESP32. After it receives the "command" it will parse the message:

- If the command is 'Stop', it will break out of the sequence, close the socket and exit the program.
- If the command was 'Start', it will continue the sequence and run the detection algorithm. This algorithm will run until the probability that a cat is in the frame is 0; or in the case of testing, the 'esc' key is hit. Afterwards, it will increment a counter and send a return message to the ESP saying that it was completed or had an error. Then it will loop back to the beginning and wait for a new "command".

A. *First Model*

The first model was one that was made using ImageAI functions. This was a YOLO model that gave a bunch of compatibility issues, which may have been why the model didn't work so well. To get it to work, edits to the yolo.py file needed to be made; this was due to a function, to_float(), no longer existing with the version of Tensorflow that was being used and instead had to be manually cast.

```
count       = tf.reduce_sum(object_mask)
count_noobj = tf.reduce_sum(1 - object_mask)
#detect_mask = tf.to_float((pred_box_conf*object_mask) >= 0.5)
detect_mask = tf.cast(((pred_box_conf*object_mask) >= 0.5),tf.float32)
#class_mask  = tf.expand_dims(tf.to_float(tf.equal(tf.argmax(pred_box_class, -1), t
class_mask  = tf.expand_dims(tf.cast((tf.equal(tf.argmax(pred_box_class, -1), true_
#recall50    = tf.reduce_sum(tf.to_float(iou_scores >= 0.5 ) * detect_mask  * class
recall50    = tf.reduce_sum(tf.cast((iou_scores >= 0.5 ),tf.float32) * detect_mask
#recall75    = tf.reduce_sum(tf.to_float(iou_scores >= 0.75) * detect_mask  * class
recall75    = tf.reduce_sum(tf.cast((iou_scores >= 0.75),tf.float32) * detect_mask
```

Fig. 10: Casting values to floats in yolo.py

The training was on 500+ images of the 3 cats in various angles and groupings along with their annotations made using Label Studio. Another note to make, the original images were 3000x4000 and didn't get resized when training the model, so it took 2 days

to complete. After testing some detection with the model, it was better to find an alternative.

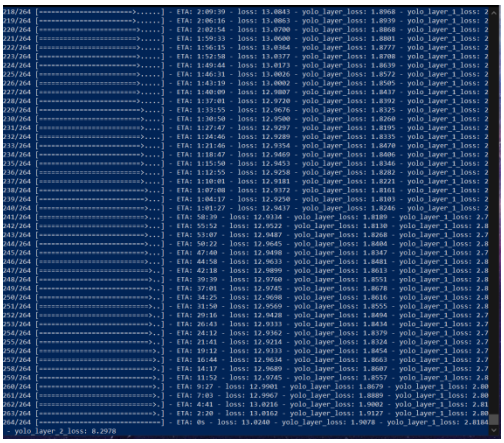

Fig. 11: Part of 1 epoch in the training of first model

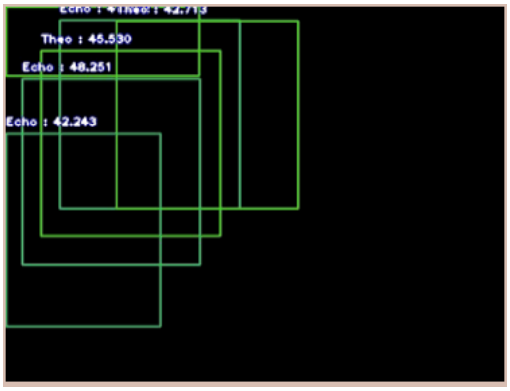

Fig. 12: Testing of first model on a black image

B. Teachable Machine

Since the first model wasn't something that was useable Teachable Machine [4] was able to build a better model instead. The process was similar to ImageAI's; feed it some images, give it a moment and then you have a model you can test.

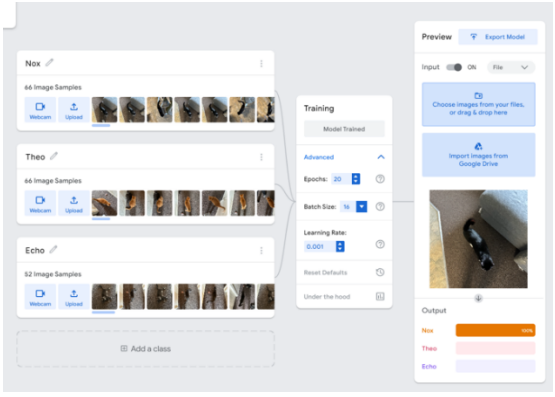

Fig. 13: Training a model on Teachable Machine

Here you can see the training history for the new model:

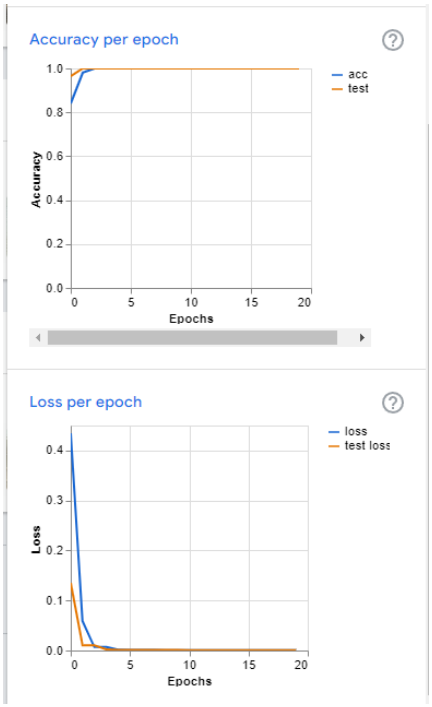

Fig. 14: Training data from Teachable Machine

V. FUTURE WORK

This project is far from being at the limit of what it can do. Some things that can be worked on to make it better or add features:

- Get detection working on the Raspberry Pi so the only devices in the system are the ESP32 and the Pi.
- Tracking the cat through the room to make sure it's using the litter box and not just wandering into the room.
- Using a different "tripwire" solution. The cats are very interested in the laser on the ceiling and chirp at it from time to time.
- Make a PCB for the ESP32 and solder components to it instead of using a breadboard.
- 3D printing a housing for the ESP32. Amazon box is fine, but some cats like to eat it.
- Battery for ESP, so it doesn't need to be connected to a wall outlet, more compact.
- Ability to send SMS to us to tell us when we should clean the boxes.

[1] "Boqii automatic cat litter box," Amazon Marketplace. [Online]. Available: https://www.amazon.com/boqii-Automatic-Ultra-Large-Multi-cat-Recognition/dp/B0B6Z37J4J. [Accessed: 28-Nov-2022].

[2] A. Lelieveld and S. Santos, "Video streaming raspberry pi camera," Random Nerd Tutorials, 01-Nov-2019. [Online]. Available: https://randomnerdtutorials.com/video-streaming-with-raspberry-pi-camera/. [Accessed: 21-Sep-2022].

[3] F. Andrade, "How to easily convert a python script to an executable file (.exe)," Towards Data Science, 12-Oct-2021. [Online]. Available: https://towardsdatascience.com/how-to-easily-convert-a-python-script-to-an-executable-file-exe-4966e253c7e9. [Accessed: 16-Nov-2022].

[4] "Teachable Machine," Google. [Online]. Available: https://teachablemachine.withgoogle.com/. [Accessed: 20-Sep-2022].

[5] "Laser Sensor," Waveshare. [Online]. Available: https://www.waveshare.com/laser-sensor.htm. [Accessed: 16-Sep-2022].

[6] John HarrisJohn Harris 34133 silver badges66 bronze badges, Venkatesh WadawadagiVenkatesh Wadawadagi 2, Gustavo GeoDronesGustavo GeoDrones 14133 silver badges55 bronze badges, Dinusha ThilakarathnaDinusha Thilakarathna 41266 silver badges1313 bronze badges, GarryGarry 32255 silver badges1717 bronze badges, Soma BanerjeeSoma Banerjee 4111 bronze badge, QasQas 32244 silver badges1111 bronze badges, Rami AlloushRami Alloush 1, and elSergioelSergio 7144 bronze badges, "Access IP camera in Python opencv," Stack Overflow, 25-May-2018. [Online]. Available: https://stackoverflow.com/questions/49978705/access-ip-camera-in-python-opencv. [Accessed: 19-Sep-2022].

[7] dgreathouse, "Send SMS Messages from Raspberry Pi," Raspberry Pi Forums - Index page, 12-Feb-2014. [Online]. Available: https://forums.raspberrypi.com/viewtopic.php?t=69286. [Accessed: 18-Oct-2022].

[8] A. Rosebrock, "OpenCV people counter," PyImageSearch, 24-Jul-2022. [Online]. Available: https://pyimagesearch.com/2018/08/13/opencv-people-counter/. [Accessed: 20-Oct-2022].

[9] "Installing cuDNN," NVIDIA Documentation Center, 27-Nov-2015. [Online]. Available: https://docs.nvidia.com/deeplearning/cudnn/install-guide/index.html#install-windows.

[10] M. Olafenwa and J. Olafenwa, "Official English documentation for imageai!," Official English Documentation for ImageAI! - ImageAI 2.1.6 documentation. [Online]. Available: https://imageai.readthedocs.io/en/latest/.

[11] M. Olafenwa and J. Olafenwa, "Custom training: Prediction," Custom Training: Prediction - ImageAI 2.1.6 documentation. [Online]. Available: https://imageai.readthedocs.io/en/latest/custom/index.html.

[12] "How to start a executable within a C++ program and get its process ID (in linux)?," Stack Overflow, 18-Aug-2014. [Online]. Available: https://stackoverflow.com/questions/13557853/how-to-start-a-executable-within-a-c-program-and-get-its-process-id-in-linux. [Accessed: 20-Oct-2022].

[13] S. Bansal, "System() function in C/C++," Tutorials Point, 20-Jan-2020. [Online]. Available: https://www.tutorialspoint.com/system-function-in-c-cplusplus#:~:text=The%20system()%20function%20is,included%20to%20call%20this%20function.

[14] N. Jennings, "Socket programming in python (guide)," Real Python, 21-Feb-2022. [Online]. Available: https://realpython.com/python-sockets/#echo-server. [Accessed: 20-Nov-2022].