Pet Monitor and Mini Treat Dispenser (IoT Final Project)

Dixon Liang, dixonl2@illinois.edu

November 28, 2021

Abstract

This report is for the final project of the course Internet of Things (CS-437) at the University of Illinois Urbana-Champaign as part of the Masters in Computer Science program. I created a pet monitor using a Raspberry Pi; I used a Pi Camera and TensorFlow Lite to monitor the activities of my pet. Additionally, I added functionality of Twilio to send my SMS updates and a servo arm for a basic treat dispenser. A video of a summary of the project and brief demo can be found here: https://youtu.be/eyrnKKqTISk. The code for the project can be found here: https://github.com/dixonliang/petmonitor

1 Introduction

This report is broken down into six sections: Motivation, Technical Approach, Implementation Details, Results, and Things Learned/Further Improvements. The inspirations for this project came from the following: https://github.com/EdjeElectronics/TensorFlow-Object-Detection-on-the-Raspberry-Pi/blob/master/Pet_detector.py (Pet Detection) and https://www.explainingcomputers.com/pi_zero_projects_video.html (Treat Dispenser).

2 Motivation

The main motivation for this project was my pet dog, Leo. Because I am sometimes away from my home for extended periods of time during the day for work or other activities, I get curious sometimes what he is spending his time doing. I do own a video camera which can see him at times, but it only gives me a glimpse of the moment I decide to tune in. It would be nice to get a recap of what he has done or where he has spent his time during the day. I can imagine for many pet owners, they have the same type of curiosity. Although there are video camera monitors and remote treat dispensers out there in the mass market, they have the same problem that it requires actually watching the camera feed at all times or watch a recording which requires a lot of time. It's not practical for someone to just be watching their pet all day while away from the home. I wanted to create something that monitors Leo, but also is able to alert me when he does something as well as with a summary of his day. I figured that I can't be the only pet owner who wants the same for his or her pet.

In addition to the monitoring capabilities of the project, I wanted to add some interaction functionality. I decided to add a time-released treat dispenser for an addition function to the system. My intention was to make the project into both a hardware and software project.

As per the the introduction, I was able to find two starting points for the pet detection and dispenser aspects for my project. I based my project off these as a starting point and used similar ideas such as SMS based off time-tracking location monitoring and a servo-based dispenser.

3 Technical Approach

3.1 Hardware

The hardware used for the project were as followed: 1) Raspberry Pi 2) Pi Camera 3) PiCar 4-WD Hat 4) External Battery 5) Servo Motor. Figure 1 shows a view of the set up from above. It is important to note that these materials were used because they what I had available. If a direct connection between the servo motor and the Raspberry Pi could be established with another servo product, there would be no need for the 4-WD Hat or External Battery.

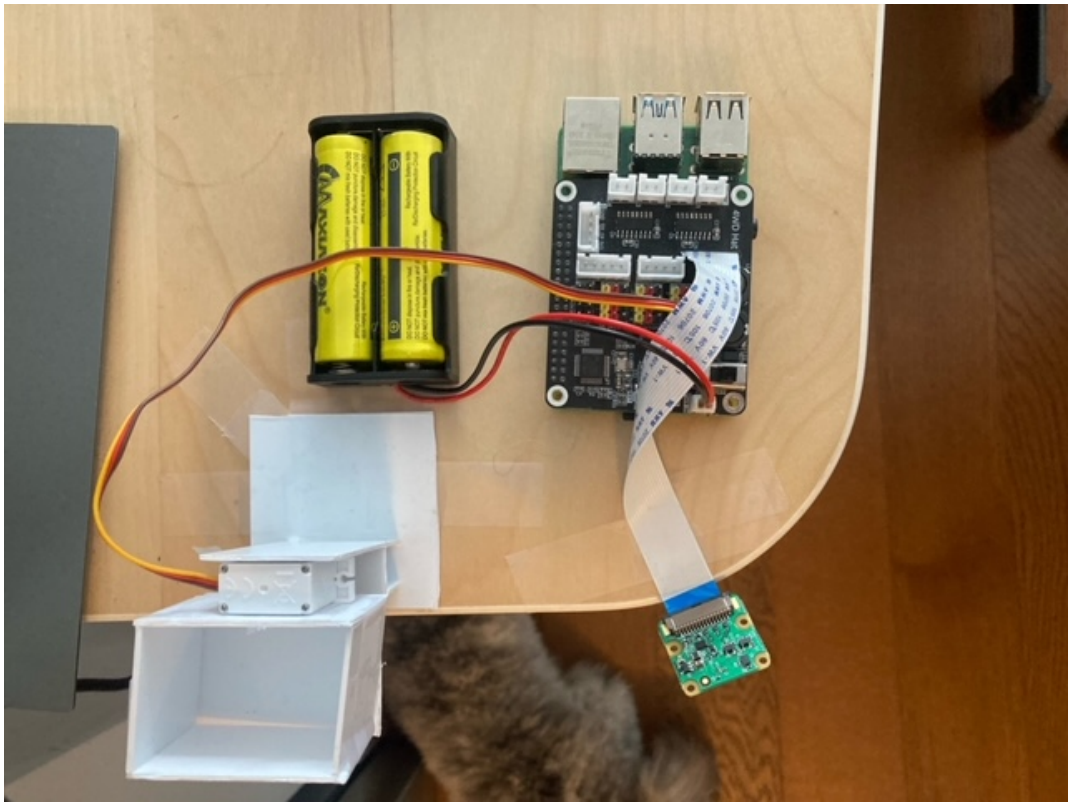

Figure 1: Layout of the project

Figure 2 shows the prototype dispenser that I created out of plasticard and liquid poly. This can also be replaced with other materials. The servo motor closes and releases the bottom latch which then releases the treats.

3.2 Architecture

Figure 3 shows the circuit diagram for the hardware of the project. Besides the 4-WD Hat attachment, there were only three connections made with the Raspberry Pi: 1) Pi Camera 2) Servo Arm 3) Battery Pack.

3.3 Software

The basic flow of data from the software perspective is that frames are captured by the Pi Camera which are then fed to the Raspberry Pi. The Raspberry Pi then runs through the main file which then leads to the actionable items which are to send a SMS using Twilio or to move the servo arm which utilizes the dispenser. More detail about the software will be discussed in the Implementation Details.

4 Implementation Details

4.1 Packages

The code was written using Python. For the computer vision aspect of my project, I used TensorFlow Lite. Within this package, there were files that included the model, labels, and annotator. The model used for classification was the EfficientDet-Lite. The associated labels used were Coco (Common Objects in Context). In order to move the servo arm, I used the code from the Picar 4-WD package, specifically the servo code in order to move the arm. This is a portion of the code that could be replaced with another servo-only package. For the SMS portion of the project, I used Twilio.

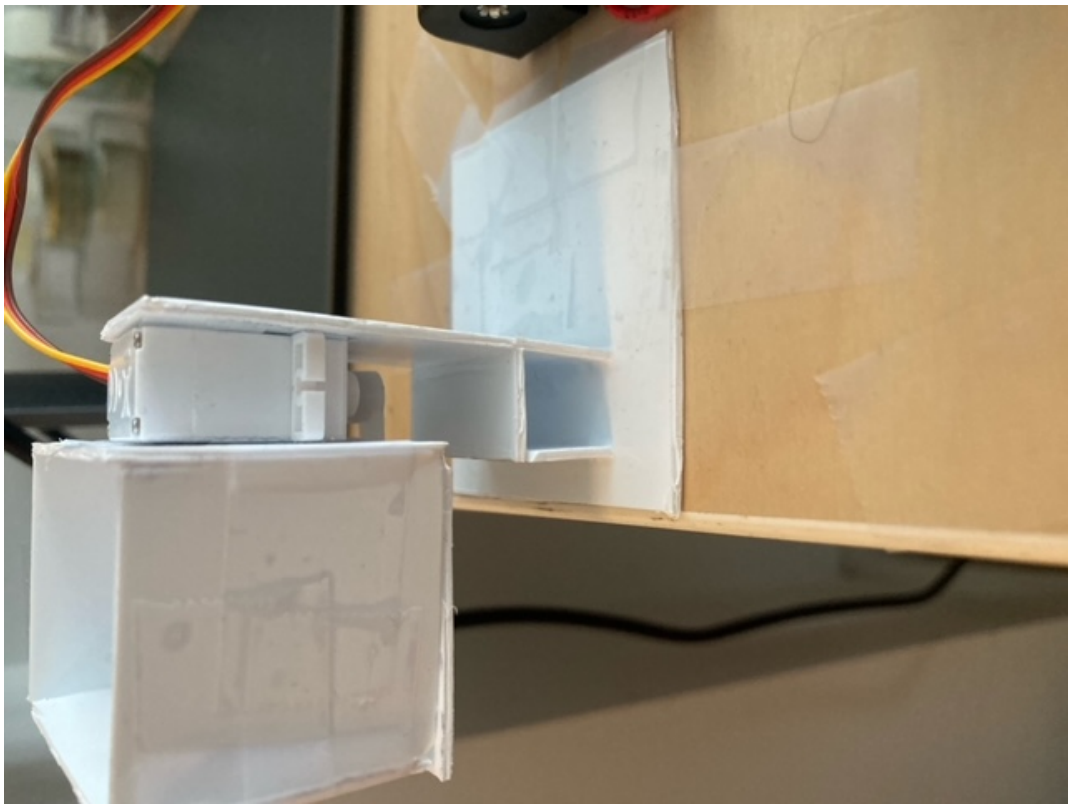

Figure 2: Close up of the Dispenser and Servo

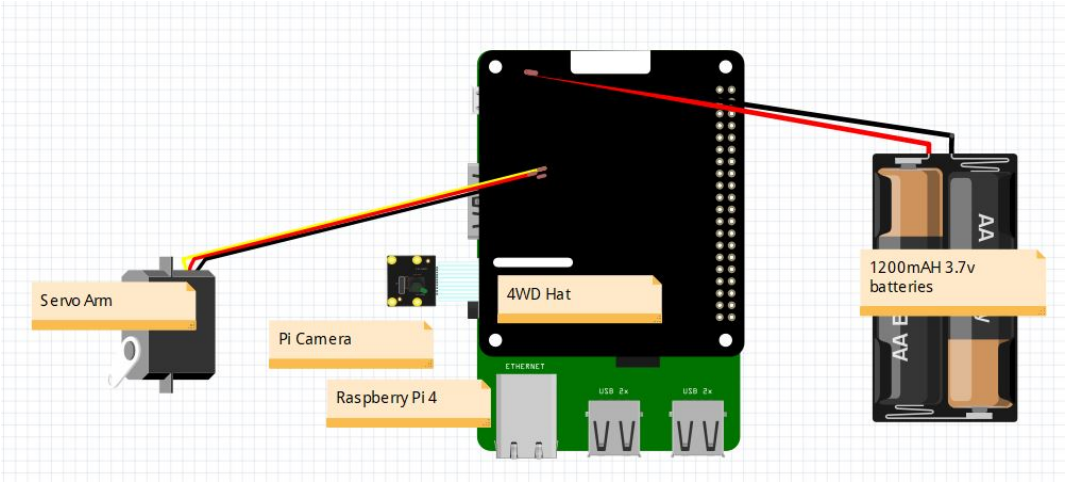

Figure 3: Fritzing Diagram of the Raspberry Pi Connections

4.2 Starter Code

As mentioned above, the main algorithm was built off using the baseline TensorFlow Lite model code which can be found here:https://github.com/tensorflow/examples/blob/master/lite/examples/object_detection/raspberry_pi/README.md.

From the starter code in object-detector.py, I used the functions: load-labels(), set-input-tensor(), get-output-tensor(), annotate-objects(), and detect-object(). load-labels() loads the Coco labels used for recognition by the model into the file. set-input-tensor() and get-output-tensor() are directly related to the model process. detect-objects() uses the tensors generated to process the images from the camera and classify. annotate-objects() provides a boundary based on the objects classified. After setting parameters, the pipeline for the image recognition and associated actions using these functions is done in main().

4.3 Parameters

Before going through the main algorithm, there are several parameters that must be set. With regards to the camera and model, the three parameters are the camera resolution, frame rate, and threshold for classification. For my purposes, I set the resolution to 500x500. The frame rate is 2 frames per second. Threshold for classification was set at 0.5. It's important to note that if there are any changes to these parameters, other parts of the code will need to be adjusted. For example, the locations on the frame of certain areas and the time counter which uses frames per second to keep track will also need to be changed.

The rest of the parameters are related to either the locations (bed and water bowl) relative to the frame and keeping track of various elapsed times from sleeping time to day time. For testing purposes, the total time of the day is set as 30 seconds; an 8-hour day would be set at 28,800 seconds. Below is a total list of these parameters.

```
## coordinates relative to frame for locations for Leo including his water bowl and bed
### bed coordinates ###
bed_xmin = 0.15
bed_xmax = 0.65
bed_ymin = -0.10
bed_ymax = 0.60
#######################
#### water bowl ####
water_xmin = -0.10
water_xmax = 0.85
water_ymin = 0.45
water_ymax = 0.95
#######################
###### parameters ################################################
day_time = 30 # set how many seconds are to elapse before sending a summary
total_time_elapsed = 0 # how long the program has been running
time_dog = 0 # time Leo on screen
bed_time = 0 # time spent in bed
water_time = 0 # time to measure if drinking water
water_count = 0 # times Leo drank water
total_sleep_time = 0 # total time Leo has slept
currently_sleeping = False # check if Leo is sleeping
done_sleeping = True # check if Leo was sleeping, default state is that he is not sleeping
summary_sent = False # check if summary sent to make sure not spammed
treats = False # check if treats have been released
```

4.4 main() (Algorithm)

The code runs through a loop based on the two frames per second. The rest of the algorithm runs within a for loop of a continuous capture on the Pi Camera. Each image from the frame is then fed into the detect-objects() function which then outputs an array of results which contains the coordinates, class, confidence score, and count

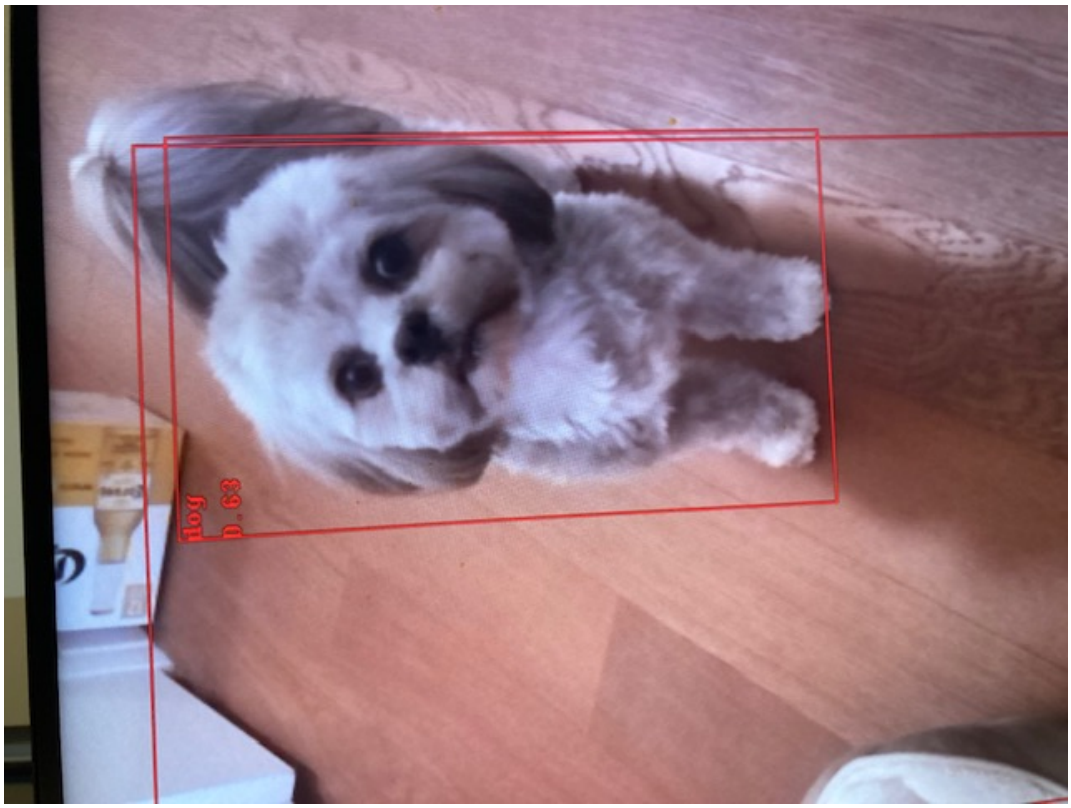

Figure 4: Dog Classification

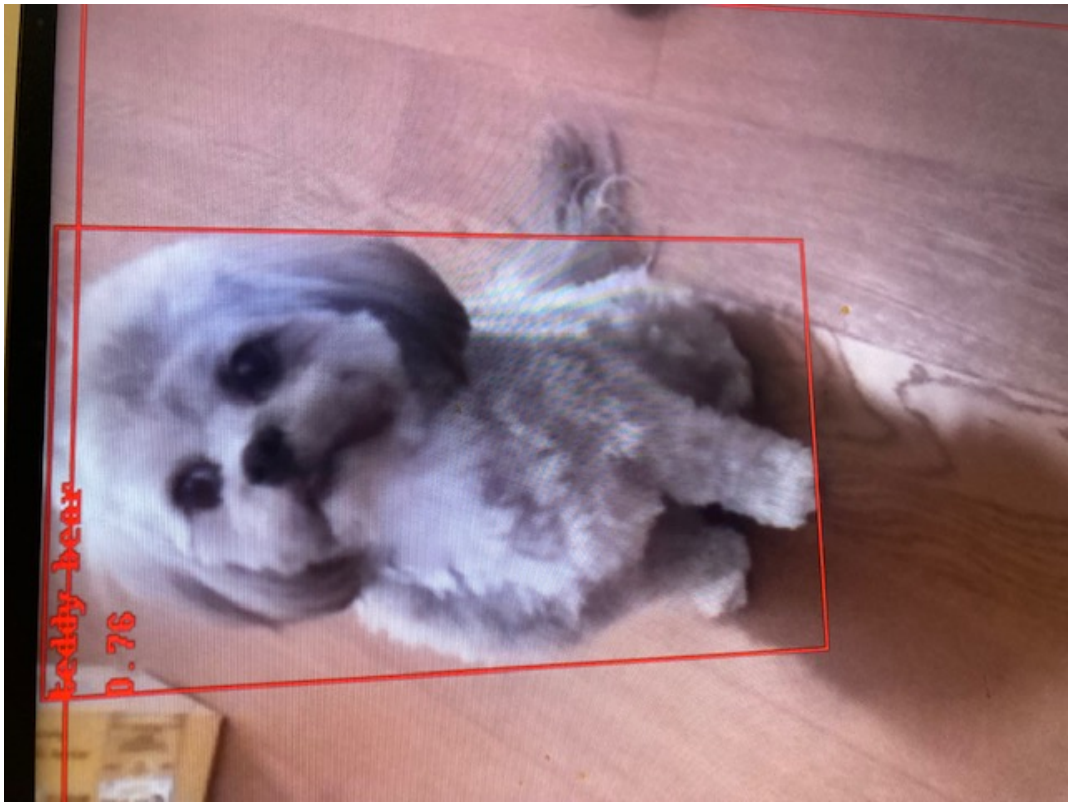

Figure 5: Teddy Bear Classification

of an object that is detected. From this output, we then check if what is detected is either 1) dog 2) cat 3) teddy bear. From experimentation, these are the three categories that my dog was categorized as. Figures 4 and 5 show these classifications from the camera.

Although more complicated models could likely distinguish between the three, this "hack" fulfills the requirements for this project. Once it is determined that one of these three categories is detected, the code goes through a few conditionals to decide what to do next. While this is happening, time counters are being incremented. Time elapsed adds on regardless of what is detected, and time in frame increments when the pet is detected on the frame. The two conditions that are run based on the location parameters are if my dog was drinking water or sleeping. Below is a code snippet of the sleeping portion of the code.

```
        if (obj['bounding_box'][0] > bed_xmin and obj['bounding_box'][1] > bed_ymin and
obj['bounding_box'][2] < bed_xmax and obj['bounding_box'][3] < bed_ymax): # check if Leo
        sleeping in bed
            bed_time = bed_time + 0.5
            total_sleep_time = total_sleep_time + 0.5
        else:
            if ((done_sleeping == False) and bed_time > 0):
             done_sleeping = True # set that he is done sleeping
             currently_sleeping = False # set back to default that he is not sleeping
             bed_time = 0 # clear if Leo not in bed
             message = client.messages.create( # send message to tell me that Leo is
             done sleeping
               body = "Leo is done sleeping!",
               from_=twilio_number,
               to=my_number
                )
            else:
             bed_time = 0 # clear if Leo not in bed
```

The main logic for this is that if my dog was detected within the boundaries of the parameters earlier, then a time counter starts for the respective activity. The first check is if Leo is drinking water; if he is detected in the boundaries, then the water counter starts going. In this case, I have used the scenario of 1.5 seconds (3 frames) in that area to count that he is drinking water. Once the counter hits 1.5 seconds, the code then sends a message that Leo has drank water. If he is not detected in the water drinking area, then the water counter resets and then goes to the next check. Figure 6 shows the location of the water bowl on screen, which is roughly the top right corner.

As per the code above, the next check is if Leo is sleeping. The logic is similar to the water drinking although to make sure Leo is in fact sleeping, the counter takes longer before the message is sent. A message is also sent when Leo is done sleeping. There is a Boolean setting that is also set. Figure 7 shows the location of sleeping on screen, which is roughly the middle left side.

The last two aspects of the code are the servo based treat-dispenser which is also based on a time counter relative the length of the time of day. Currently, it is set to release at the half way point of the day. After that portion of the code, the summary of the day message is sent once a time elapsed counter passes the length of the time of the day. Figure 8 shows the servo released and Figure 9 shows an example text.

5 Results

5.1 SMS Notifications

The notification system using Twilio which depended on the computer vision classification model worked exactly as planned. Once the counters reached the designated times, the messages were sent to my phone seamlessly. I received messages as planned for the water counter, begin sleeping, end sleeping, and summary of the day. The frame rate of 2 per seconds made counting the elapsed time for each setting work very well. Figure 10 shows the summary of the day text.

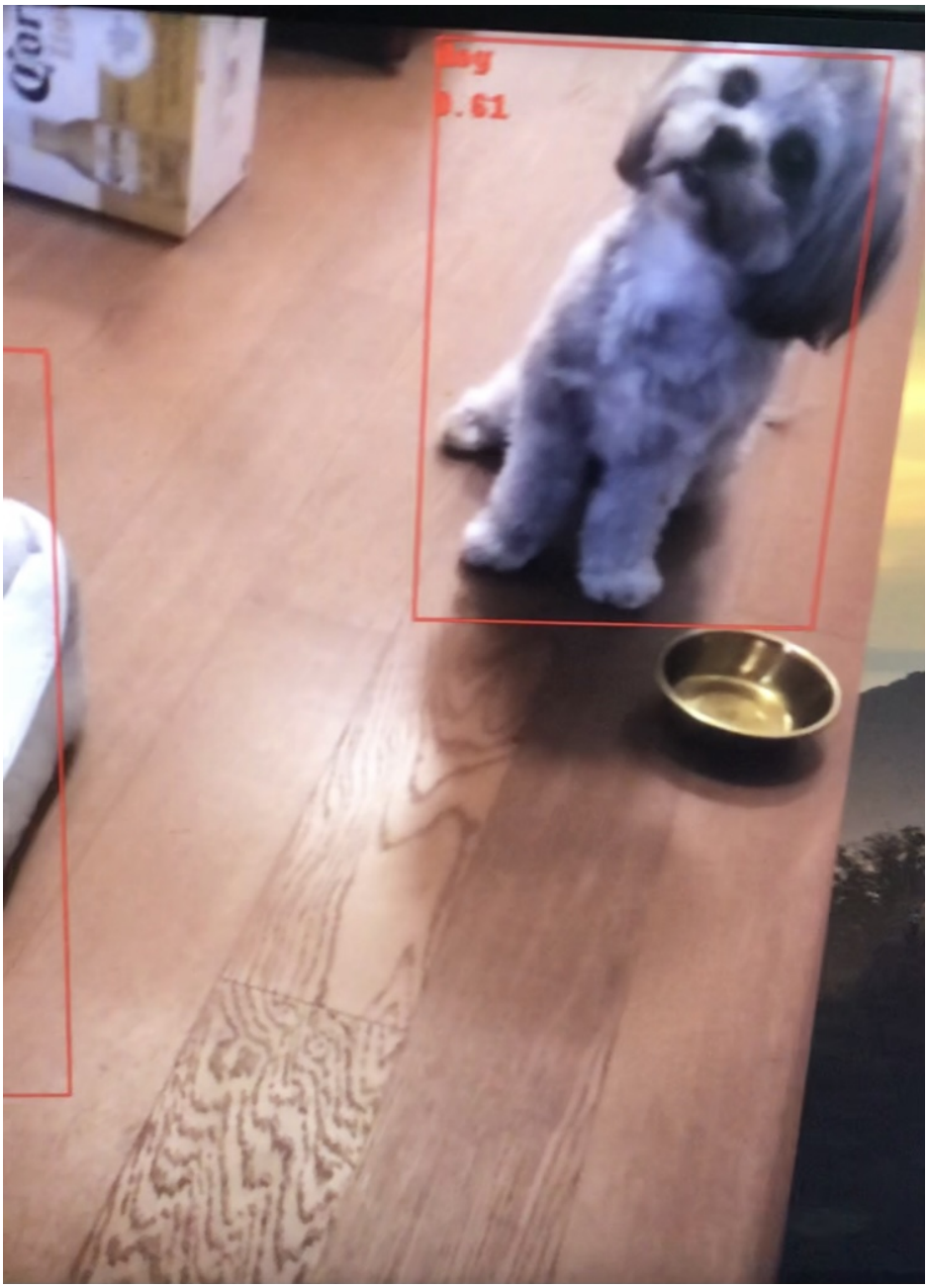

Figure 6: Water Bowl Area

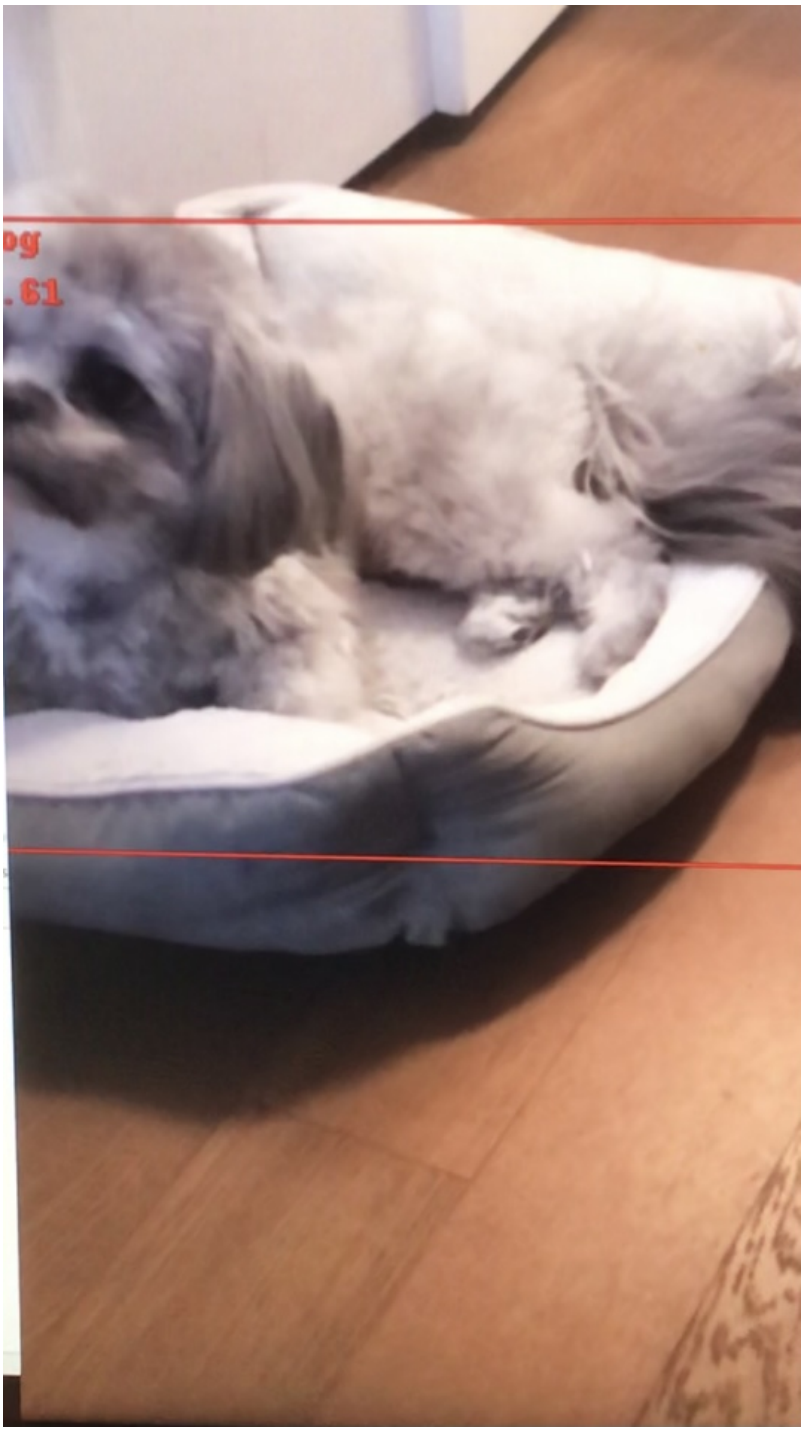

Figure 7: Sleeping Area

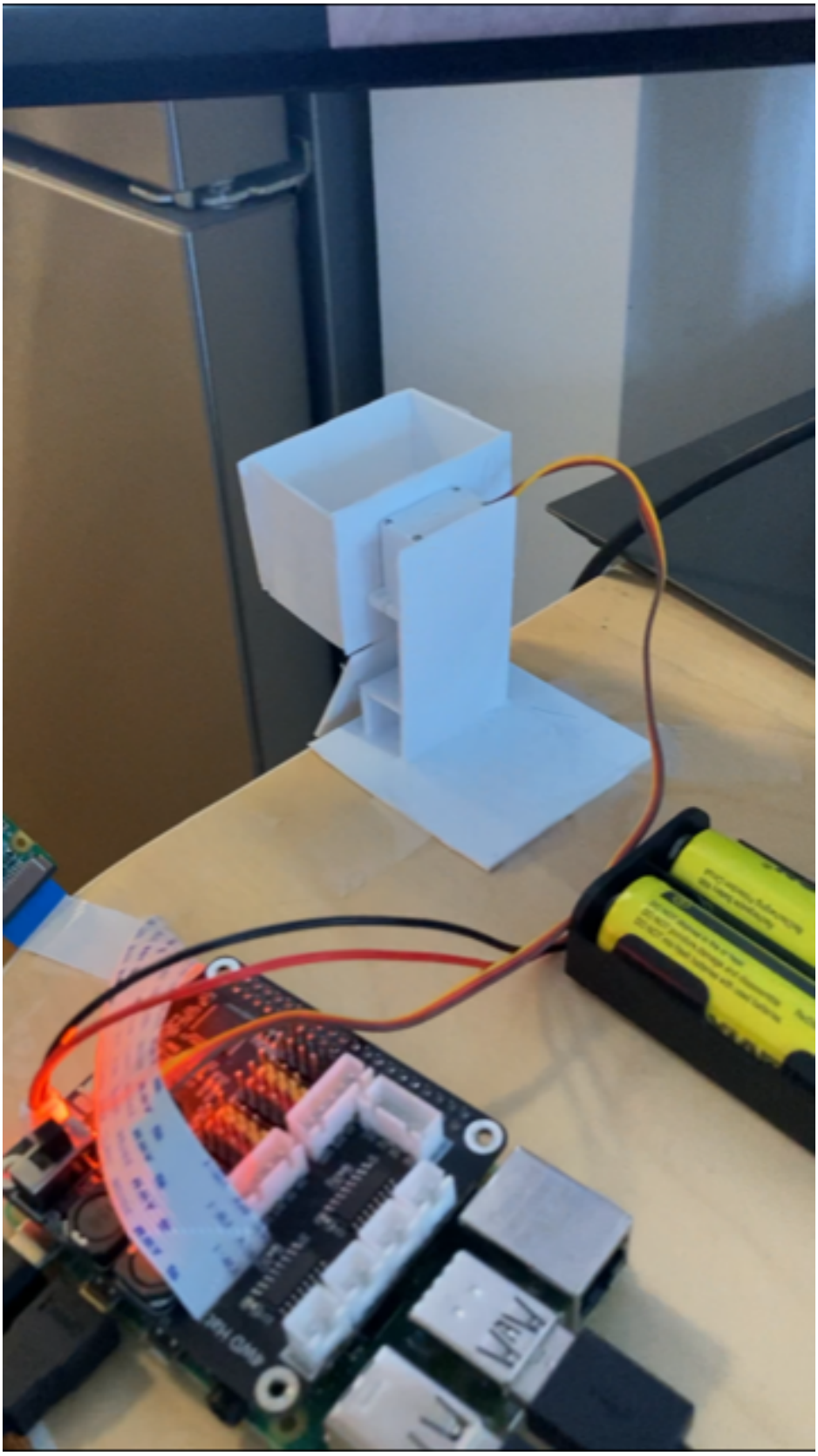

Figure 8: Servo Released

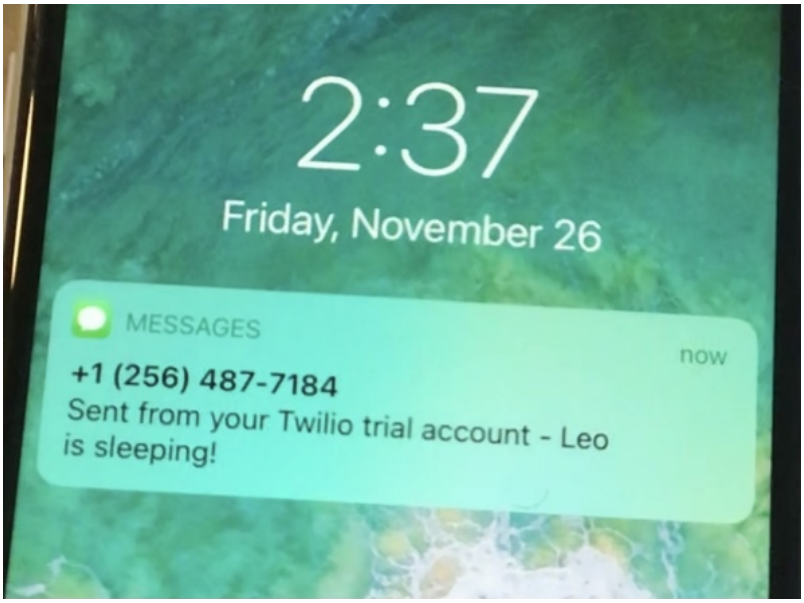

Figure 9: Sleeping Text

5.2 Servo Treat Dispenser

The servo based treat dispenser worked very well. The servo arm closed and released the latch of the dispenser exactly as planned. As with the SMS notifications, the time counter for the servo worked exactly as expected too.

5.3 Difficulties

The biggest difficulty was from the location capturing. Because the location capturing depends on the relative location relative to the frame, there was some difficulties in managing the coordinates especially if the camera moved slightly. This required some adjusts of the coordinates of the locations from time to time or physically adjusting the camera if the locations were not registering as intended. Some possible improvements to this is discussed in the next section.

6 Things Learned and Further Improvements

There are few additions in functionality that could be made to the project. Something that could be added is a live video stream from mobile. This would likely require some type of application development. On a similar note, some type of live interaction with the servo motor could also be implemented using an application; instead of the servo being moved by a counter, it would be moved by some type of interaction in the application.

As per the difficulties noted, some improvements could be made to the camera. The camera needs to be fixed in a location where it cannot move at all. With a better permanent set up, the adjustments needed to be periodically made in case of movements would be lessened.

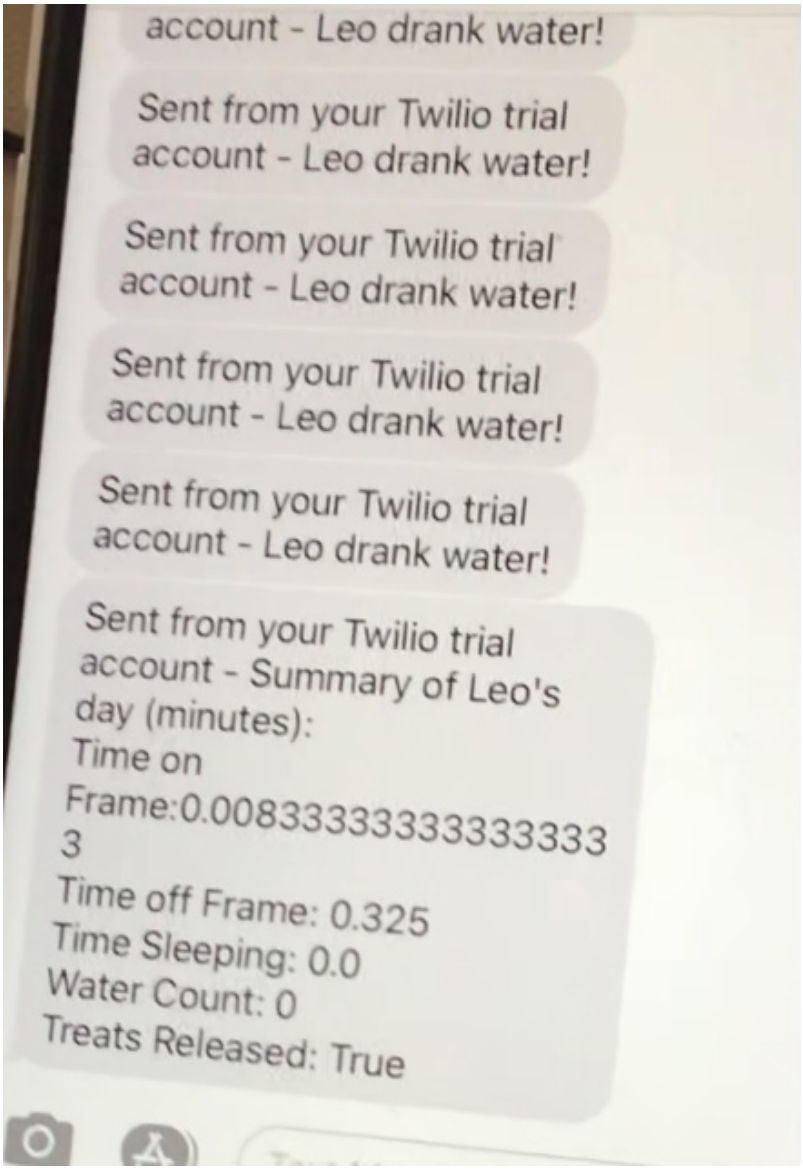

Figure 10: Summary Text