Edge and Cloud-based IoT for Bird Ecology

Jeffrey Stockman
School of Engineering and Technology
University of Washington - Tacoma
Redmond, WA, USA
stockmaj@uw.edu

Viktoriya Grishkina
School of Engineering and Technology
University of Washington – Tacoma
Tacoma, WA, USA
vias2010@gmail.com

***Abstract*— In this work, we demonstrate a practical application of IoT architecture, edge and cloud computing services, and data visualization and analysis to automate traditionally manual bird watching and bird population studies. We employ motion detection to capture high-resolution images that are compared to a machine learning algorithm and assist in species identification and counts.**

Keywords*— *Internet of Things [IoT], edge computing, fog computing, AWS S3, Azure Custom Vision, Node-Red, InfluxDB, Grafana, GrovePi, Raspberry Pi, smart bird feeder.

I. Introduction

Significant Ecological studies occur in bird tracking to identify migration patterns, environmental changes, and bird populations. Most of these studies are manual and are volunteer-based [1]. There are limited connected devices and associated services that compile this data. Most work in this area is limited to disconnected devices like game cameras where the information is stored locally and must be retrieved. Internet connectivity is limited, and these devices are not able to leverage fog or cloud-based technologies. Or, the work completed to study birds is highly manual, open to observer subjectivity, and must be transcribed to a database for analysis.

The motivation of the project is to leverage IoT connected devices and sensors, fog computing, and cloud services to automate the Ecological studies around overall bird population health, where migratory and environmental patterns can be tracked automatically and systematically, image can be stored and retained for viewing or to improve the vision learning model, and all of the data can be used for further analysis for scientists, ecologists, and ornithologists.

II. Related Work

Beyond the initial prototyping phase, we are cognizant of the need for scalability of this particular problem statement. One bird feeder will not satisfy the need to improve scientific data collection and analysis, but can make a large impact with thousands of devices deployed worldwide. Thus, we attempted to build our design on 1) readily-accessible hardware, 2) cloud services that are highly available and without scale or latency issues across multiple geographical regions, and 3) with software components that are able to be containerized. Future iterations of this prototype, as we will discuss in the later sections, should be able to be containerized to allow independent, structured development, have publisher/subscriber features for features like notifications across mobile or local networks , as well as unified logging of mass amounts of data points for researchers to leverage in their studies [2].

III. System Architecture and Design

The system is built on a Raspberry Pi Model 3B. This form factor is powerful enough to manage local and fog computing requirements, but lightweight enough to control costs and could be adapted to use solar power. The prototype also uses a Grove Pi + sensor board for ease-of-use, and allows seamless connection with the DHT11 Temperature / Humidity Gauge, LS06-S Grove Light Sensor, and Grove Ultrasonic Distance Sensor. A Raspberry Pi HQ Camera is connected directly to the Raspberry Pi board. The Raspberry Pi and light sensors are encased in a waterproof case, and the remaining sensors are protected against wind and water without impacting performance.

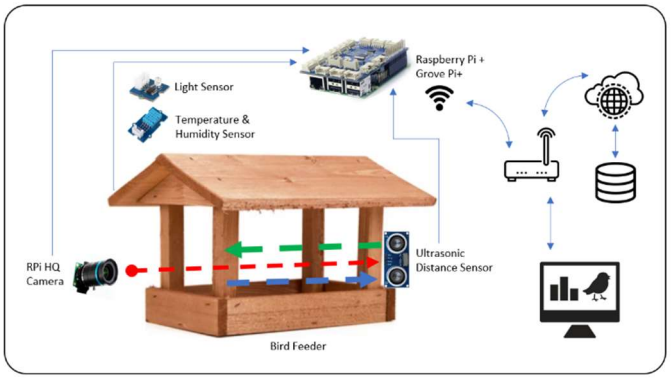

Figure 1. Bird Feeder schematic.

The Raspberry Pi OS employed is the standard Raspbian. Node Red drives the main functionality of the system via browser-based workflows, and uses JavaScript as the back-end programming language. Locally, InfluxDB is used as a time-based database to store sensor data and bird species & counts. Two cloud services are utilized in the system. Azure Custom Vision service is used as a bird identification model using an object detection machine learning algorithm. An AWS S3 bucket stores all images for dashboards and archives, and a public HTML / JavaScript website is also contained therein. Locally, Grafana Visualization is employed for dashboarding (image display, sensor information, and analytics). Also, a webcam is accessible via the local browser [2].

The system workflow is as follows. All sensor data (light, temperature, humidity) is captured and stored locally to InfluxDB once a minute. The ultrasonic ranger senses distance on the perch of the bird feeder every 3 seconds. If the distance decreases (a bird lands), an image is captured and sent to Azure

for identification. A birdPresent flag is used to ensure only one bird image is captured, and one count is incremented. If the probability of the image exceeds 50%, the image is submitted to the S3 bucket for archiving. Once the distance resets (the bird flies away), the process starts over. All data and images are then presented on the Grafana dashboard. Images can also be viewed from any browser – an HTML & JavaScript script is hosted within the S3 photo album [3].

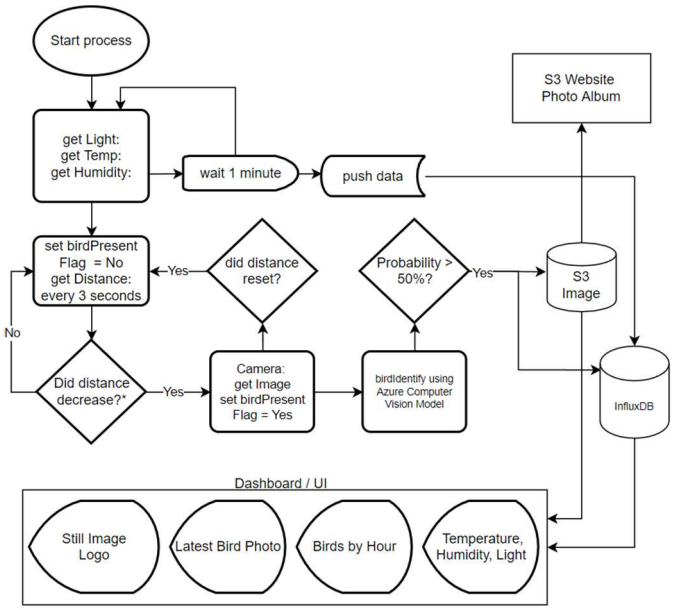

Figure 2. Process flowchart.

The Azure Custom Vision model was initially trained using over 150 photos (courtesy Google images) across 7 native western Washington state birds. Every image taken by the system is sent to Azure, where verification, re-tagging, and/or model retraining can occur.

IV. Discussion

Initial system prototyping was conducted in a controlled office environment, and all sensors and systems were stable. As the prototype was deployed outdoors, several issues appeared rather quickly. Sensor data often fluctuated without cause, connectivity of sensor data to the Grafana dashboard was sporadic, and the overall functionality that controlled the image capture & identification process and Custom Vision model needed improvements. It was noted than when temperatures dropped below 10°C and/or higher than 75% humidity, connectivity issues and data loss occurred.

The Azure Custom Vision model provided a 75.0% precision score upon initial training. As bird images were captured and predictions validated and/or tags updated, the second iterations of the model increased to 92.4%. This algorithm is quite accurate, and can see this model becoming increasingly proficient in species identification with successive retraining with situational images. This is particularly impressive because most traditional bird images are from the side, but in this practical example we capture all sides of the birds – front, back, side, bending over, flying away, etc. This model is accurate in detecting the species from all angles.

The counting process is not as accurate as it could be, which is a core requirement for this data to be effective in bird population studies. The limits of the system prevent count increments if 1) two or more birds are at the feeder at the same time, and 2) if the bird is not identified with at least a 50% probability score. As the custom vision model improves, we anticipate point 2 to be addressed. In two images, the vision model did detect two birds, but the limits of the JSON response from Azure prevented the count increment from returning an accurate count.

V. Challenges & Future Work

There were a number of limitations of our initial prototype. As indicated, the Ultrasonic Distance Sensor provided sporadic measurements, we believe during high humidity levels (evenings, during rain). This caused us to modify our design, and could potentially prevent the presence capture of birds if they were far enough away from the sensor (greater than 12 cm). This could lead to errors in overall bird counts.

The construct of the Ultrasonic Distance Sensor combined with the HQ Camera prevented dual workstreams to occur at the same time, notably both a live webcam and image capture). In the future, we would look to redevelop this technology first with a live webcam, and secondly with the ability to capture images and / or submit data to Azure or perhaps a local ML algorithm for object detection. Equally, it is of note that we are using both the free tiers for both Azure and AWS, and during the initial prototyping over the course of a week, we received throughput warnings and cost warnings. To properly scale this type of ecosystem, we would need to optimize for the overall costs in using these services.

The Grafana dashboard proved to be limited in our ability to analyze the data in meaningful ways. We attempted to improve the Grafana dashboard with non-numerical data and analysis capabilities. However, we learned there is a limitation when using InfluxDB exclusively as the backend database. In the future, we would employ a more standard SQL database that can also capture non-numerical data, such as bird species strings, which would give us a greater ability to develop better analysis charts and graphs.

Significant development and improvements can be made in the deployment of a scalable network of smart bird feeders to assist in data collection across thousands of devices. Containerization could assist with the deployment of software updates over-the-air and by regional requirements and / or development requirements. In this prototype, mobile apps were not considered for development, but could be used to improve the user experience, share data, analyze information remotely, and provide a more enjoyable experience.

VI. Conclusion

This bird watching prototype provided significant improvements over traditional bird-watching. The automated capture of important environmental and ornithological data presents an opportunity to collect, analyze and learn even more about changing ecological factors that impacts bird migrations,

bird populations, and feeding habits. Several improvements would allow this device to drastically improve the collection of bird-related data to better understand their environmental cues, migratory patterns, and habits.

REFERENCES

[1] Audubon Society. Backyeard Bird Count. https://www.birdcount.org/

[2] DevOps for the Urban IoT. Urb-IoT'16, May 24-25, 2016, Tokyo, Japan

[3] RPi-Cam-Web-Interface. https://elinux.org/RPi-Cam-Web-Interface

[4] Tutorial. Hosting a static website using Amazon S3. https://docs.aws.amazon.com/AmazonS3/latest/userguide/WebsiteHosting.html