LONDON'S GLOBAL UNIVERSITY

Tracking Wildlife Counts Using the Internet Of Things

Matthew Bell[1]
m.bell@cs.ucl.ac.uk

BSc Computer Science

Supervisor: Dr Kevin Bryson

Submission date: Day Month Year

[1]**Disclaimer:** This report is submitted as part requirement for the Computer Science BSc at UCL. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

Abstract

Conservation experts, park rangers, and biologists frequently aim to try to track the location and count of various species of animals. This is, more often than not, extremely time consuming, since researchers have to install camera traps with motion sensing shutters, and manually look back through images to identify and count the animals.

This project and report explores the possibility of using a low-power computer with sensors, connected to a web server over a wireless Internet connection (a paradigm frequently referred to as *The Internet of Things (IoT)*) to automate this task to save researchers hours of time when conducting studies using camera traps.

The project also explores various methods in which species of animals could be identified automatically, given various constraints of how the system can work, including the availability and speed of the network link.

To-Do (Delete Before Submission)

Contents

Chapter 1

Introduction and background

1.1 Motivation and need

Wildlife conservation experts constantly need to keep track of the location and movements of wildlife for a number of reasons, including monitoring species migration patterns and population counts [7]. Other options exist, like line transect surveys (counting animals or traces of animals, like tracks and droppings) or track surveys (physically visiting the area and counting animals); but in the comparison study by Silveira et al [11], they found that camera traps, despite their longer initial setup time and cost, “can be handled more easily and with relatively [lower] costs in a long term run”.

Based on this, it would be logical to assume that running costs and analysis time could be reduced further by automating the classification of photos taken by camera traps and sending the results back to a web server. This would enable research teams to store, access, analyse and visualise wildlife counts with ease, using an Application Programming Interface (API) provided by the web service.

The biggest advantage of automating the classification stage of camera trap studies is that it would save a lot of time after the main study has ended, and results can be analysed as soon as possible.

1.2 Requirements

A list of requirements for the solution were reasonably easy to devise. Most of the requirements are defined by the limits of the environments where this system may be deployed, such as woodlands, grasslands, and national parks.

Most of the locations where this solution could be deployed may have very limited cell network coverage, and definitely would not have WiFi connectivity available. Therefore, the system would have to use LoRaWAN, which has a theoretical range of up to twenty kilometres. However, the lower data rate of LoRaWAN means that photos captured by the camera traps would not be able to be sent back to a web server, so any kind of image processing and classification would have to be performed on-device.

Another limitation is the devices being used for the project. The main “base station” device is the Creator Ci40 developer board, designed to “allow developers to rapidly create connected products” [9]. This ability to rapidly prototype on the board, which has a -based processor and runs the OpenWRT Linux distribution. It also contains a WiFi radio, useful for communicating with the device and debugging code on it during development, and a 6LoWPAN radio, useful for communicating with nearby devices.

The sensor devices were also provided as part of the project. They consist of each sensor board integrated onto a *MikroElektronika* 6LoWPAN clicker board [3]. This board runs a Real-Time Operating System (RTOS), which allows the board to respond very quickly to changes in sensor input, as well as the ability to be battery powered and the inclusion of a 6LoWPAN radio, to communicate with the base station.

Another requirement arising from the intended deployment scenario is that the system needs to be able to run on battery power, or indeed a low-voltage power source, for a considerable amount of time. The intention of the project is to reduce long-term study costs, and a need to replace sensor batteries regularly would be failing this. Consequently, the system would have to rely on hardware interrupts as much as possible, so that the devices can remain in a “sleep mode” when they are not needed.

1.3 Initial research and similar problems

Chapter 2

Design

2.1 System Architecture

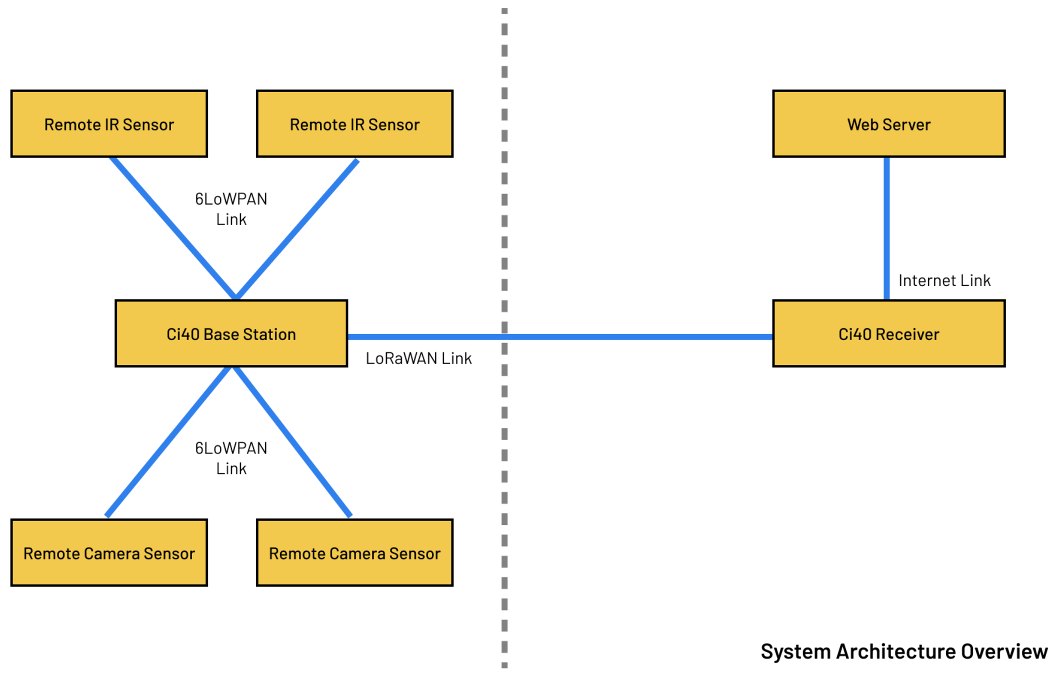

Figure 2.1: System architecture overview, showing how the high-level components of the system are connected.

Figure 2.1 shows the connections between the high-level components in the system, including the sensor deveices and the base stations, as well as the web server used to store and retrieve data.

2.1.1 Base Station

As previously mentioned in the requirements, the base station is a *Creator Ci40 IoT Hub* device [9]. It's a development board that is highly specialised for rapidly building Internet of Things applications, due to its abundance of sockets, pins and radios for I/O.

Hardware Design Features

The Ci40 board includes a 6LoWPAN radio, which is crucial in allow it to conduct two-way communication with nearby sensing devices at a reasonably high data rate. The main drawback is the range, which is typically a couple of tens of meters [2]. However, this is perfect for communicating with the nearby sensor devices, which won't be too far away from the base station in the first place, and there is also the potential to use a mesh-style network, where communication might happen via one or more intermediary nodes. However, this would require sensor devices to be constantly active, thus preventing them from entering a lower power state and resulting in a much higher power usage.

Software Design

The base station provides a couple of very important tasks. Firstly, it acts as the main router, or coordinator, for nearby motion detector and camera pairs. It receives a motion alert from the motion sensor and sends a command to the related camera to capture an image. This program would also need to keep a list of sensor pairs, to ensure each command gets sent to the correct sensor.

In addition to this, the base station runs a program to process incoming images from the camera sensors and calculates the count and species of any wildlife in the photo. Since this is an image recognition problem, it was decided that the best solution would be to use a convoluted neural network, trained on a dataset of similar images, to estimate to a reasonable accuracy the species of wildlife in the image. However, since the Ci40 board uses a 32-bit processor architecture, it makes running neural networks a little more difficult, since most frameworks will only support 64-bit processors, as the larger word length results in larger computations being made possible.

Finally, the base station needs to act as a LoRaWAN transmitter, to send the count data to an Internet-connected base station, which can then in turn upload the results to the web server. So, the base station is running three programs at the same time to deal with different tasks. All of the code that deals with the 6LoWPAN connection or LoRaWAN connection is written in either C or Python, and uses the *LetMeCreate* library provided by the board manufacturer to increase ease of development.

2.1.2 Remote Sensors

include graphic of camera pairs in trees etc?

The system also utilises a set of remote sensors that wirelessly connect with the base station using a 6LoWPAN connection. There's two kinds of sensors being used in the system, motion detector sensors and camera sensors. Both sensors use the same base board, a *MikroElektronika* 6LoWPAN Click Board [3], with either a camera or motion detector attachment.

Hardware Design

Include image of 6LoWPAN clicker

The *MikroElektronika* 6LoWPAN Click Board, as previously mentioned, runs a Real-Time Operating System (RTOS) called Contiki. According to the Contiki project website [1], it's ideal for low-power IoT projects since it supports the full IPv4 and IPv6 networking suites, as well a being able to run on "tiny systems, only having a few kilobytes of memory available". *Creator*, the manufacturers of the Ci40 base station, provide a toolchain for writing programs to the click board, as well as abstraction libraries for network and sensor interfaces [10].

Creator also provide a set of guides to help set up and write programs to the flash memory onboard the clicker device [8], which have proven invaluable. Code is written in a slightly modified version of the C programming language; the main difference being that all of the code runs in process threads that can be suspended, resumed and interrupted.

Communication

The remote sensors communicate with the base station using JSON-encoded strings sent via TCP over 6LoWPAN. A set of commands are defined and recognised by the sensors and the base station server alike, to allow messages to be sent that are both brief and human-readable, which is invaluable when debugging. The messages take a form similar to this:

```
{ "device_id": 1, "command": "heartbeat" }
```

A unique `device_id` is provided by every sensor to identify itself to the base station server, and on its first connection it will also provide a `pair_id` which tells the server which unique camera/motion detector pair the device belongs to. Each of these are provided to the sensor program at compile time, using environment variables.

Add list of commands?

Motion Detector Sensor

include image of motion click

The motion detector sensor uses the 6LoWPAN Click board described above, with a *MikroElektronika* Motion Click device [6] integrated onto the board using the "mikroBUS" port. According to the product page [6], it has a range of up to four metres which is probably not sufficient for real world usage. However, for prototyping purposes, it's perfect because of the ease of integration, thanks to the aforementioned *LetMeCreate* library [10].

The code runs in a continuous loop, that yields the main thread until it is resumed by an event interrupt. This event could be one of:

- a timer expiring,
- a TCP event (received packets, lost connection, et cetera),
- a motion detection event received from the motion sensor.

any other events?

For the prototype version of this sensor, the code sends a "heartbeat" command to the base station every twenty seconds, for debugging purposes. But in a production version, the processor would only be be interrupted by TCP events or by motion detection events, as this would result in fewer interrupts over time and thus reduce the power usage of the device. The development version of the program also makes use of a debug server running on the base station. The clicker board does not always have a serial output available, so printing to console (i.e., using the `printf()` function) does not work. Therefore, Contiki includes a `PRINTF` macro that sends the string to a server using 6LoWPAN, if available.

Camera Sensor

include image of camera click

The camera sensor comprises of the same 6LoWPAN Clicker board, but instead of a motion sensor, there is a *MikroElektronika* Camera Click board [4] installed onto the mikroBUS port. The board contains a digital camera sensor which, according to the specification page, has a maximum resolution of 640 by 480 pixels. It also contains an extra microcontroller, which "outputs the camera image to the target board microcontroller through the mikroBUS SPI interface". Essentially, it appears to transform the raw data stream from the camera into a data stream that can be sent using Serial Peripheral Interface bus (SPI) to the 'target' board (in this case, the 6LoWPAN clicker). However, a lot of the board's inner workings—save for the board schematics, available on the product page—is largely closed-source.

The code examples provided by *MikroElektronika* [5] helped to provide a little bit of insight into how the camera can be operated. For example, it provides a list of opcodes that the camera click accepts, such as requesting an image, or getting/setting a register on the camera sensor itself.

The main objective of the camera sensor is to respond to a message sent from the base station (over 6LoWPAN) commanding it to take a photo and send the photo back to the base station over the same connection.

2.1.3 Web Server

The web server serves the purpose of storing incoming species counts and types from any number of base stations, as well as keeping track of the locations of the base stations and sensors. Since the sensors and devices don't possess geolocation capabilities, this would be something that a research team using this solution would have to manually input.

As well as providing a way of uploading and storing this information, the web server would also have to provide methods of retrieving the data, as well as displaying the data. To this end, an API is the best solution for the problem.

build the damn API, then talk about it more

Chapter 3

Implementation and Testing

Chapter 4

Conclusion

Appendix A

Glossaries

Glossary

Symbols

6LoWPAN Short-range wireless data transmission standard. Short for "IPv6 over LOw Power Wireless Personal Area Networks"; alternative to protocols like Bluetooth and Zigbee. 3, 5–7

C

Contiki A Real-Time Operating System (RTOS) designed specifically for the Internet of Things. Contains a full network stack and can run on a minimal system, with lower power consumption. 6

L

LoRaWAN Wireless data transmission standard designed for long range communication at low power, at the cost of a lower data transmission rate. 2, 5

M

mikroBUS Bus standard for integrating IoT sensors onto development boards. Contains pins for SPI, analog data transmission, power, and an interrupt pin. 6, 7

MIPS Multiprocessor without Interlocked Pipeline Stages, a type of processor architecture. 3

O

opcode Short for operation code, a command or instruction that may be part of a device's instruction list. 7

Acronyms

Appendix B

Bibliography

[1] Contiki. *Contiki: The Open Source Operating System for the Internet of Things.* URL: http://www.contiki-os.org (visited on 04/13/2018).

[2] David Culler and Samita Chakrabarti. "6LoWPAN: Incorporating IEEE 802.15. 4 into the IP architecture". In: *IPSO Alliance, White paper* (2009).

[3] MikroElektronika d.o.o. *6LoWPAN clicker - a compact development board | MikroElektronika.* URL: https://www.mikroe.com/clicker-6lowpan (visited on 04/12/2018).

[4] MikroElektronika d.o.o. *Camera click — board with OV7670-VL2A CMOS image sensor.* URL: https://www.mikroe.com/camera-click (visited on 04/15/2018).

[5] MikroElektronika d.o.o. *LibStock - Camera Click - Example.* URL: https://libstock.mikroe.com/projects/view/1263/camera-click-example (visited on 04/15/2018).

[6] MikroElektronika d.o.o. *Motion click - pir500b motion detector sensitive only to live bodies.* URL: https://www.mikroe.com/motion-click (visited on 04/13/2018).

[7] K Ullas Karanth. "Estimating tiger Panthera tigris populations from camera-trap data using capture—recapture models". In: *Biological conservation* 71.3 (1995), pp. 333–338.

[8] Imagination Technologies Limited. URL: https://docs.creatordev.io/clicker/guides/quick-start-guide/ (visited on 04/13/2018).

[9] Imagination Technologies Limited. *Creator Ci40 IoT Hub - Build your Internet of Things Product.* URL: https://creatordev.io/ci40-iot-hub.html (visited on 04/12/2018).

[10] Imagination Technologies Limited. *CreatorDev/LetMeCreate.* URL: https://github.com/CreatorDev/LetMeCreate (visited on 04/13/2018).

[11] Leandro Silveira, Anah TA Jacomo, and Jose Alexandre F Diniz-Filho. "Camera trap, line transect census and track surveys: a comparative evaluation". In: *Biological Conservation* 114.3 (2003), pp. 351–355.

Appendix C

Code Listing

C.1 detect/detect.c

```c
#include "contiki.h"
#include "contiki-lib.h"
#include "contiki-net.h"

#include <sys/clock.h>
#include "letmecreate/core/network.h"
#include "letmecreate/core/common.h"
#include "letmecreate/click/motion.h"
//#include "letmecreate/core/debug.h"
#include <sys/etimer.h>
#include <leds.h>

#include <string.h>
#include <stdio.h>
#include <stdbool.h>

#define UDP_CONNECTION_ADDR "fe80:0:0:0:19:f5ff:fe89:1af0"
#define SERVER_PORT 9876
#define CLIENT_PORT 3001
#define BUFFER_SIZE 128
#define PROC_INTERVAL 20 * CLOCK_SECOND

#ifndef DEVICE_ID
#define DEVICE_ID -1
#endif

#ifndef PAIR_ID
#define PAIR_ID -1
```

```
#endif

static bool motion_detected;

static void motion_callback(uint8_t event)
{
  motion_detected = true;
}

PROCESS(detect_main, "Main process for detector");
AUTOSTART_PROCESSES(&detect_main);
PROCESS_THREAD(detect_main, ev, data)
{
  PROCESS_BEGIN();
  {
    static struct etimer et;
    static struct uip_conn *connection;
    static char buf[BUFFER_SIZE];
    static int res = 0;

    etimer_set(&et, CLOCK_SECOND * 2);
    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));

    leds_off(LED1);
    leds_on(LED2);

    connection = tcp_new_connection(SERVER_PORT, UDP_CONNECTION_ADDR);
    PROCESS_WAIT_TCP_CONNECTED();
    leds_on(LED1);
    leds_off(LED2);
    motion_detected = false;

    motion_click_enable(MIKROBUS_1);
    motion_click_attach_callback(MIKROBUS_1, motion_callback);

    etimer_set(&et, PROC_INTERVAL);
    while (true) {
      res = 0;
      PROCESS_YIELD();
      if (motion_detected) {
        sprintf(buf, "{\"device_id\":%d,\"command\":\"motion\"}", DEVICE_ID);
        tcp_packet_send(connection, buf, strlen(buf));
        PROCESS_WAIT_TCP_SENT();
        motion_detected = false;
        continue;
      }

      if (etimer_expired(&et)) {
        leds_off(LED1);
        leds_off(LED2);
        sprintf(buf, "{\"device_id\":%d,\"command\":\"heartbeat\"}", DEVICE_ID);
        res = tcp_packet_send(connection, buf, strlen(buf));
        PROCESS_WAIT_TCP_SENT();
        if (res == -1) {
          leds_on(LED2);
        }
        leds_on(LED1);
        etimer_restart(&et);
      }
    }

    motion_click_disable(MIKROBUS_1);
  }
  PROCESS_END();
}
```