Low-Res Images from a "LoRa" Distance

Robert W Senser, PhD

March 24, 2022

Abstract

LoRa can be used to transmit information over relatively long distances, but at a low bandwidth. We used this capability to transmit reduced images, using a Raspberry Pi, Camera and LoRa hardware. We have found this approach to be effective for certain use cases.

1 Introduction

Our goal was to render low-resolution images, taken by a lower quality Raspberry Pi camera at a significant distance away. The communication from the Raspberry Pi to the rendering PC was handled via LoRa (Long Range Wide Area).

A sample use of this might be to determine if a vehicle was parked in a declared location at a remote location. Here we assumed that WiFi and cellular service were not available.

2 Overall Architecture

Figure 1 shows our hardware architecture. The wavy line from the LoRa Shield to the LoRa Board denotes the LoRa connection. The transmission components included a Raspberry Pi Zero, a common Raspberry Pi Camera and an Adafruit LoRa "Bonnet". The receiver components included an Adafruit LoRa board and a microprocessor. The microprocessor connected to the PC via USB.

3 Example Use Cases

3.1 Is a Person Sitting in the Chair?

Figure 2 shows three small images. These are monochrome, icon-sized images and they represent three different states of a person sitting in a chair, moving away and then returning. Note the lack of the "dark area" in the center of the middle image – the person was not present.

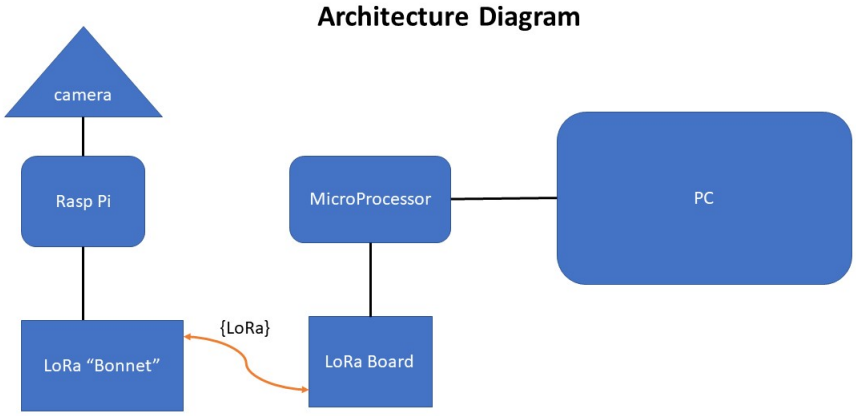

Figure 1: Hardware Architecture

(a) In Chair (b) Away from Chair (c) Again In Chair

Figure 2: Time Series: Sit, Move Away then Sit

3.2 Pilot's View While Changing Headings

Figure 3 also shows three images in a similar format to Figure 2. In this set, the images represent the simulated view from the cockpit of an airplane turning left, flying level and then turning right. Red lines were manually inserted to show the border between the instrument panel and then windshield. The area above the red line was the sky, the area below was the cockpit display, blacked out.

4 LoRa with Raspberry Pi Camera

The camera took a snapshot, with **raspistill**, at regular intervals. The resulting image was processed with **cv2** and **convert** to generate a 258 byte monochrome image. This image was transmitted via LoRa. The processed images could optionally be logged in the **archive** subdirectory.

(a) Left Turn (b) Level (c) Right Turn

Figure 3: Time Series: Left, Level then Right

5 LoRa Receiver using Serial Interface

The Adafruit Feather M0 or Adafruit Pro Trinket converted the LoRa transciever input into a serial stream that was send to the PC via a USB/serial connection. Both of these executed an Arduino shield that managed the LoRa device and passed the raw stream of bits received to the PC.

6 PC-based Serial Receiver and Image Viewer

The serial receiver module (**serialV6.py**) received the serial data and converted it to the needed format. Error handling was also performed and images could be logged.

The **Viewer7.py** module formated the images for display in two windows. Figure 8 shows an example screen display of our viewer. The image shown on the left was the most currently received image and the image on the right could be scrolled through the previously received images.

7 Next Steps

7.1 Major Enhancements

1. Allow run-time update of image transmission settings, including duration between images and image resolution.
2. Allow for re-transmission of interesting images at a different resolution.
3. Use Deep Learning to optionally augment the displayed images.

7.2 Incremental Enhancements

1. Improve LoRa Receiver design to be less complex, using minimal Arduino C code.

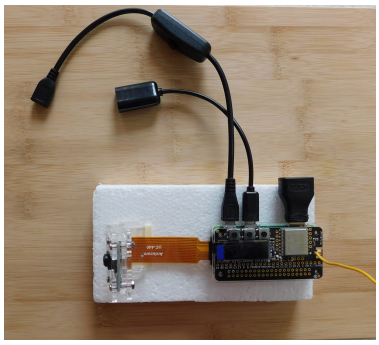

Figure 4: Camera & Rasp. Pi Bonnet Top View

Figure 5: Camera, Rasp. Pi & Rasp. Pi Bonnet Side View

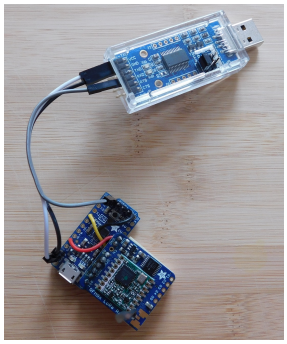

Figure 6: Pro Trinket-based Receiver

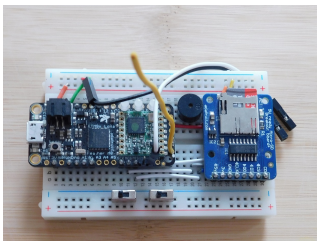

Figure 7: Feather M0-based Receiver

2. Use a smaller and lighter transmission design, such as an ESP32-CAM with an attached LoRa board.

3. Combine PC python software into one module, with additional correction.

7.3 List of Hardware:

This section denotes the hardware used in the first version. We considered this as a proof-of-concept version.

7.3.1 Raspberry Pi Zero with attached Camera

The actual Raspberry Pi used was a "Zero W". The added WiFi support made testing easier. This was programmed with the **transmitterV1.py** Python code

Figure 8: View of Received Images

and the **clear** and **run** scripts.

7.3.2 AdAfruit Raspberry Pi LoRa Bonnet

This component was Adafruit part number 4074.

7.3.3 Adafruit Feather M0 with LoRa

This component was Adafruit part number 3178.
Component was programmed with the **Tracker_9X_RX5_M0.ino** software.

7.3.4 Adafruit Pro Trinket V5

This component was Adafruit part number 2000.
Component was programmed with the **Tracker_9X_Raw.ino** software.

7.3.5 Adafruit LoRa Board for 915Mhz

This component was Adafruit part number 3072.

7.3.6 Note

Either a Feather with LoRa or Pro Trinket with a LoRa board was used, not both.

7.4 List of Arduino software:

- **Tracker_9X_Raw.ino** (for Adafruit Pro Trinket)
- **Tracker_9X_RX5_M0.ino** (for Adafruit Feather M0)

7.5 Raspbian Scripts and Python

- **clear** script code

```
rm archive/*.bin
rm archive/*.jpg
echo "cleared"
```

- **run** script code

```
python3 transmitterV1.py -t 12 -n
```

- **transmitterV1.py** Python code
  See GIT.

7.6 PC Scripts and Python

- **clear.bat** "batch" file contents

```
del archive\f1*.bin
echo "** cleared **"
pause
```

- **serialInV6.py** Python code
  See GIT.
- **Viewer7.py** Python code
  See GIT.

7.7 Scripts and code available in GIT

- **transmitterV1.py**
- **serialV6.py**
- **Viewer7.py**

LoRaCam public GIT repository URL:

`https://github.com/rwsenser/LoRaCam`