KB4YG IoT Hardware Documentation

4/20/2022

[Summary](#_5ejxssnksdcd) 3

[**Software**](#_tjomnl1iluv) **3**

[Networking](#_rtx0j1b44dh4) 3

[Connect to AWS backend](#_m9rx50hryy5t) 3

[Get Weather Data](#_er70wph8e3v4) 3

[Camera](#_l0ni3gdljgr8) 4

## Summary

This document serves as the complete guide for the hardware and software for the KB4YG capstone project. Each section will contain detailed information and troubleshooting steps for any of the potential issues.

# Software

## Networking

Each hardware node connects to Sixfabs backend through cellular networks. The cell service is paid for through Sixfabs dashboard. Each hardware node is under the same account ([teamKB4YG@gmail.com](mailto:teamKBYG@gmail.com)). Here you will be able to see all of the nodes in the system and remote terminal into each one.

### Remote terminal into pi

1. Download OpenVPN connect, connect to local VPN network
2. In terminal, command ssh pi@fittongreen.lan
3. Use login pi with password raspberry
4. Should now be connected to the raspberry pi and is able to run functionalities

### Connecting to the Pi through Sixfab

1. Login at <https://connect.sixfab.com/#/login>
2. Once logged in click devices on the left hand side under CORE
3. Click on the device you want to connect to
4. Click on remote terminal on the right hand side
5. Once the terminal has loaded (it may take up to a minute depending on the connection signal) you will be logged in as the user sixfab@fittongreen
6. This is the incorrect user, so enter the following command into the terminal to switch to the pi user:
7. $ sudo su - pi
8. This will automatically authenticate and switch to the correct user.
9. Here you can now access the code and other services on the Pi

## Code Structure Breakdown ([Github](https://github.com/KB4YG/iot))

### Backup

Contains a full Raspberry Pi OS backup including all of the software and correct parameters set in the event of a SD card getting corrupted or lost. Follow the directions in the readme to properly restore a SD card.

### SRC

This is the file that contains most of the code needed to run the operations of the system.

### SRC/Demo

This file contains some test images and code for testing the system from network communication to running images through the object detection algorithm.

### SRC/Hardware

This directory contains the code used to interface with the different pieces of hardware including the camera, and the power management hat.

### SRC/Services

This directory contains the code used to communicate over the network to OpenWeatherApi, and send packets to the backend for storage.

### SRC/Setup

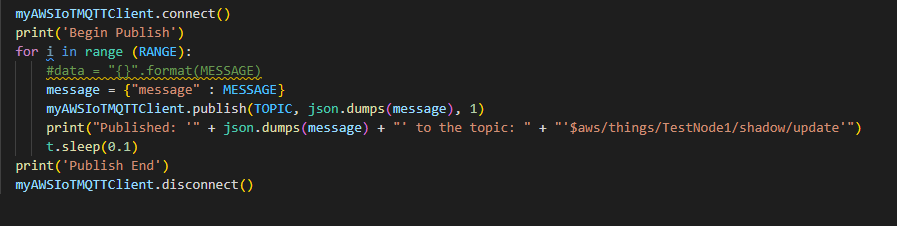
This file contains the code for setting up a brand new Raspberry Pi OS to automatically replicate all the code necessary for connecting and operating the system.

## Connect to AWS backend

All of the data is sent via MQTT messaging service to AWS. You will need the certificates from AWS in order to send data properly. Check this [link](https://docs.aws.amazon.com/iot/latest/developerguide/mqtt.html) to view the AWS documentation on how to properly set up an endpoint for the hardware to connect to. The code is located [here](https://github.com/KB4YG/iot/blob/main/src/services/api.py).

With MQTT there is no return signal saying that AWS handled the request, like a standard HTTP request would. So, it is helpful to be logged into the AWS backend to make sure that the requests are being processed correctly by the lambda functions.

Proper code for connecting to the backend:



Message Body MUST be in this exact format or else the system will silently fail:

"message": {

"state": {

"desired": {

"welcome": "aws-iot"

},

"reported": {

"welcome": "aws-iot",

"LocationId": "e3d3d8d8-e41c-4c00-a7f1-70da0c87cfdc",

"OpenGeneral": 10,

"OpenHandicap": 10,

"UsedGeneral": 2,

"UsedHandicap": 2,

"Temp": 42,

"Confidence": 80,

"ParkingLotId": "e3d3d8d8-e41c-4c00-a7f1-70da0c87cfdc",

"TotalHandicap": 0

}

}

}

Each numerical value in the message must be an integer. Other fields may be a string. Parking lot id is a value set in the backend for determining which parking lot the information needs to be inserted for. This Parking lot ID is the ID for Fitton Green parking lot 1. Other parking lot IDs can be generated as needed for additional locations. Location ID also must match Parking Lot ID. OpenGeneral represents a numerical value of the estimated number of spots open. OpenHandicap represents the number of Handicap parking spots open. UsedGeneral represents the number of spots recognized to be full by the system. Temp represents the current temperature at the parking lots location. Confidence is the value associated with how accurate the system thinks the number of spots open is.

## Get Weather Data

The temperature can make a large difference in the ability to store and generate power for the system. Temperatures below freezing will prevent the battery from being charged. Days with lots of cloud cover will prevent the solar panel(s) from generating an adequate amount of solar power. To help facilitate this we built a program for fetching the current weather data. If the conditions are unfavorable, the code will return a number specifying the current conditions. The codes are specified below:

* 0: All clear; Conditions are optimal for the system
* 1: Temperature is above freezing, but there is rain and or cloud cover
* 2: Temperature is below freezing, but the weather is favorable
* 3: Temperature is below freezing, and the weather is unfavorable

In addition to the number, the code returns an object detailing some extra details about the weather including the current temperature and the sunrise.

## Camera

[PiCamera Documentation](https://picamera.readthedocs.io/en/release-1.13/api_camera.html)

[LibCamera Documentation](https://libcamera.org/docs.html)

[Camera Code](https://github.com/KB4YG/iot/blob/main/src/hardware/camera.py)

The camera is a Raspberry Pi V2 camera that is supported with the PiCamera Python library. Custom parameters can be set for the camera settings for different times in the day such as night mode. Our code has two modes, one for day and one for night. The night mode supports longer exposure times and a different ISO to allow for noise free images. Day mode has a shorter shutter speed. Using both modes will enable clear images for the machine learning algorithm to run image classification for marking the vehicles. Refer to the PiCamera documentation for more details on setting custom parameters.

# Hardware

NOTICE: When connecting power to the system with the Power Hat installed, the power for the system must be plugged into the Power Hat NOT the Raspberry Pi itself.

Refer to our Steps to Build Hardware Documentation for detailed instructions on how to assemble parts.

## Raspberry Pi 4

We are using a Raspberry Pi 4 (2 gb) as the microprocessor to host our functionalities. All of the major components are managed using the Pi, and we are using Python to integrate them together. The Pi handles the networking, and executing of the code.

## Sixfab Cellular 4G-LTE Hat

The cellular hat connects the Pi to the cellular networks in the area. Additionally, Sixfab has a dashboard where we can remote terminal into each Pi and test code. The payments for the service are handled through the Sixfab dashboard. There are a few options for data plans, such as a low speed, low cost mode, depending on the service quality in the area. Data rates are inexpensive, and we estimate between 10 and $15 per month per node. Sixfab also supports as many nodes as needed per account, so future expansion is available.

## Voltaic V75 Battery

This battery can be directly connected to solar panels for ease of charging. Adding multiple solar panels in a series is required to add more than one panel. Our calculations estimate based on a number of factors that the battery can support the system between 14-25 days alone with no solar input.

## Raspberry Pi NoIR V2 Camera

This camera is the V2 raspberry pi camera and supports the PiCamera Python library. The V2 camera supports a number of features. Refer to the PiCamera [documentation](https://picamera.readthedocs.io/en/release-1.13/) for more info.

## Sixfab Outdoor Waterproof Case

This case acts as a protector for our hardware stack. It is transparent which allows us to see inside, as well as protects from water and moisture. The mounts on the sides enable us to attach it to a pole for a better vantage point in the future.

## Sixfab Power Management Hat

This addition to the Raspberry Pi is what regulates the battery and allows us to turn the device on and off. Because we have the voltaic battery and the solar panels, this device manages the integration of both.

## 

## 

## 