Project Progress

**Jargon**

**PIR:** [Passive InfraRed sensor](https://en.wikipedia.org/wiki/Passive_infrared_sensor). Commonly used by game cameras to detect motion.

**Raspberry Pi:** A very small Linux computer. Comes in several different models, [more info here](https://www.raspberrypi.org/help/).

**Pi:** Short for Raspberry Pi.

**Sleepy Pi 2:** A low power control board for the Raspberry Pi. [More here](https://spellfoundry.com/product/sleepy-pi-2/)

**Sleepy:** Short for Sleepy Pi 2

**Camera Trap:** A device used to take pictures of animals without human intervention. [More info here](https://en.wikipedia.org/wiki/Camera_trap).

**Trap:** Short for camera trap.

**Game Camera:** Used interchangeably with camera trap, for the purposes of this document.

**MCU:** [MicroController Unit](https://en.wikipedia.org/wiki/Microcontroller).

**Development Board:** A circuit board used to interact with an onboard microcontroller. [More here.](https://en.wikipedia.org/wiki/Microprocessor_development_board)

**SBC:** [Single board computer](https://en.wikipedia.org/wiki/Single-board_computer), such as the Raspberry Pi.

**Drone:** An autonomous/semiautonomous aircraft designed to fly without a human pilot onboard.

**Flight Controller:** The ‘brains’ of an aerial drone. [More info here](https://droneswithcamera.info/quadcopter-flight-controller/).

**Pixhawk:** A popular flight controller for autonomous drones. Several different models, the one considered in this document is the [mRo Pixhawk (or Pixhawk 1)](http://ardupilot.org/copter/docs/common-pixhawk-overview.html).

**Background**

Camera traps are tools widely used by researchers to study free roaming wildlife. A typical camera trap consists of a motion activated game camera, targeted at a piece of bait or other animal attraction (game trail, marking tree, etc.). Existing game cameras suffer from several limitations such as manual data recovery and limited customizability. The cameras typically store image and/or video data on a SD card that must be retrieved by hand. This becomes time consuming when a system of camera traps is deployed across wide areas, especially those with rough terrain. Arduous trips to the traps can be wasted if they have not collected any useful data since the last access. Conversely, collected data could be days or even weeks old which limits research tactics.

**Project Description**

The goal of this project is to solve the limitations of camera traps described above by retrieving data from the traps remotely, using an aerial drone. Instead of retrieving data by hand, it will be uploaded from the camera trap to a drone over a wireless connection. The drone will be able to fly over difficult terrain that would impede travel by foot or wheel. Importantly, the drone can operate autonomously to reduce the time burden on researchers and allow them to check traps more often. No existing game cameras are capable of interfacing with such a drone, so a custom camera system must be created as well.

**Camera Trap**

A basic camera tap was designed and built during the summer. It is currently capable of entering a low power state, booting up upon motion activation, and taking time-lapse photos with timestamps. As with most commercial game cameras, it uses a PIR module to detect motion.

The greatest challenge when creating the camera trap is minimizing power consumption. While using a single board computer such as the Raspberry Pi makes programming and interfacing easy, they draw too much current (even when “powered off”) to operate on a reasonably sized battery. Basic microcontrollers, on the other hand, offer exceptional low power performance but are more difficult to interface with cameras and large amounts of data (on the scale of GB).

These challenges can be overcome by incorporating both a low power MCU and a SBC in a single camera trap. In this design, the computer is responsible for taking photos, managing data, and uploading data to the drone. The MCU is used to control power to the main computer, turning it on when an animal or the drone is detected and then completely disconnecting power once it has finished its work.

While there are a ton of SBCs out there, the Raspberry Pi family of devices was chosen due to their community support and easy camera integration. Pi’s were one of the first cheap SBCs to hit the market and, as such, have a huge userbase that can be drawn upon for help. Several Pi models are available, with varying features and clock speeds. The Raspberry Pi Zero W was chosen from the lineup because it has lower size, power consumption, and cost than the A/B models while still having wireless connectivity.

Raspberry Pi’s have a dedicated onboard camera connector, designed to be used with the [Pi camera module (v2)](https://www.raspberrypi.org/products/camera-module-v2/). As you would expect, this makes adding camera functionality to a Pi extremely easy. Pictures can be taken directly from the command line using the “raspicam” program. Python and C libraries are also available for creating custom programs. Further information on setting up the Pi Zero W and the camera module can be found in Appendix B.

Turning now to power control, the first MCU investigated was the MSP430. This choice was made purely out of convenience, as a [MSP-EXP430FR5994 development board](http://www.ti.com/tool/MSP-EXP430FR5994) is used by ECE 306 (intro to embedded systems). The code written for the course could be used as a software base for the project, aiding development. The MSP430 also has all the basic MCU features (low power modes, [hardware interrupts](https://en.wikipedia.org/wiki/Interrupt), etc.) that are needed to minimize power consumption.

An [Arduino](https://www.arduino.cc/en/Guide/Introduction) board could be used instead, offering easier programming in exchange for performance. The Arduino ‘language’ (essentially C with added libraries) used to program the devices abstracts the complexities of MCUs making for simple and easy programming. Anyone without prior MCU programming experience should defer to using an Arduino over a bare-bones MCU.

The downside to using a general purpose board is that it requires additional hardware to function as a power controller. A buck converter is needed to reduce battery voltage (upwards of 6.7V) down to a level useable by the MCU and SBC (generally 5V). A normal pin on the MCU would not be able to power the Pi directly, so additional power circuitry would also be needed.

The solution to these (and more) problems is to use an application specific piece of hardware, such as the Sleepy Pi 2. The board’s features are explained at length [on its webpage](https://spellfoundry.com/product/sleepy-pi-2/), but the main takeaway is that it includes all of the hardware and software needed to control power to the Pi. It can be programmed using the Arduino IDE and comes with a variety of [software examples](https://github.com/SpellFoundry/SleepyPi2/tree/master/examples). Documentation, FAQ’s, and other useful info can be found on Spell Foundry’s website. Appendix C holds additional tips and tricks to augment (but not replace) official documentation.

The Sleepy can handle an input voltage between 5.5-30V, allowing for a wide range of battery options. This is a plus, as battery size and cost constraints will vary depending on the camera trap user’s needs. Lead acid car batteries are a good choice if size is not an issue, as they are robust and hold considerable charge. LiPo batteries offer much higher energy density (and thus reduced size) but require greater care and expense.

While solar panels may seem like another great option for powering the trap, their drawbacks render them unusable for most applications. They are difficult to camouflage and would draw attention to the device, attracting unwanted attention from wildlife. The animals may avoid the area, bias their behavior, or even attack and destroy the camera trap.

**Aerial Drone**

While the drone didn’t get off the ground during the summer, research was done to select appropriate hardware and develop a software plan. The basic design is based on drones created for the Namibia WAO study abroad program.

Any drone type and size could theoretically be used, so long as it can carry hardware to interact with the camera trap. The two types considered for this project were fixed wing or multi-rotor drones. While a fixed wing aircraft would provide better theoretical performance, creating a quadcopter design was found to be much more feasible.

Other than the ability to interact with the camera trap, the most important features of the drone are flight time and range; fixed wing drones are far superior to multirotors in both categories. Unfortunately, the fixed wings that make flight more efficient also make autonomous operation more difficult. Landing/takeoff requires a runway and is very difficult to achieve without human intervention. Fixed wing craft are also unable to hover directly over the camera trap, meaning that the trap must transmit data very quickly or have enough range to reach the drone while it circles above.

A multirotor design avoids the above issues, at the cost of flight range. It can takeoff/land from almost any flat surface and is able to hover over the camera trap during communication and data transfer. A quadcopter design was chosen due to its lower cost and complexity when compared with hexa- or octocopter drones. Additional information about quadcopters [can be found online](https://oscarliang.com/quadcopter-hardware-overview/).

When creating the drone, the most important piece of hardware to select was the flight controller and possible companion computer. The original plan was to use a [Beaglbone Blue](https://beagleboard.org/blue), which is a single board computer similar to the better known Beaglebone Black. This decision arose from a talk Jason Kridner, the co-founder of Beagleboard.org, gave at NC State. [The presentation was about creating a drone using the Beaglebone Blue, which had recently received official ArduPilot support](https://beagleboard.org/p/jkridner/beaglebone-blue-220mm-quadcopter-9801d8). Unlike other dedicated flight controllers, the BBBlue is a complete Linux computer and it seemed to be a great option for achieving both flight control and camera trap interaction in a single device.

Upon further research and development, utilizing the BBBlue was found more difficult to implement than expected. Because it is new on the scene, there is little support for using it as a flight controller and some features appear to still be in development. Implementing camera trap functionality on top of a potentially unstable base was going to take greater knowledge and development time than was feasible at the time, as a single part of the larger project.

A more viable alternative is to use a dedicated flight controller in tandem with a companion computer. In this setup, the “companion computer” is a piece of hardware that can interact with the ground camera without the overhead of performing flight control. A popular setup is to use a Pixhawk flight controller and a Raspberry Pi. There is documentation online about how to setup both of the devices, namely on [ArduPilot’s offical website](http://ardupilot.org/dev/docs/raspberry-pi-via-mavlink.html). The two devices function independently and communicate using the MAVLink protocol.

**Trap/Drone Communication**

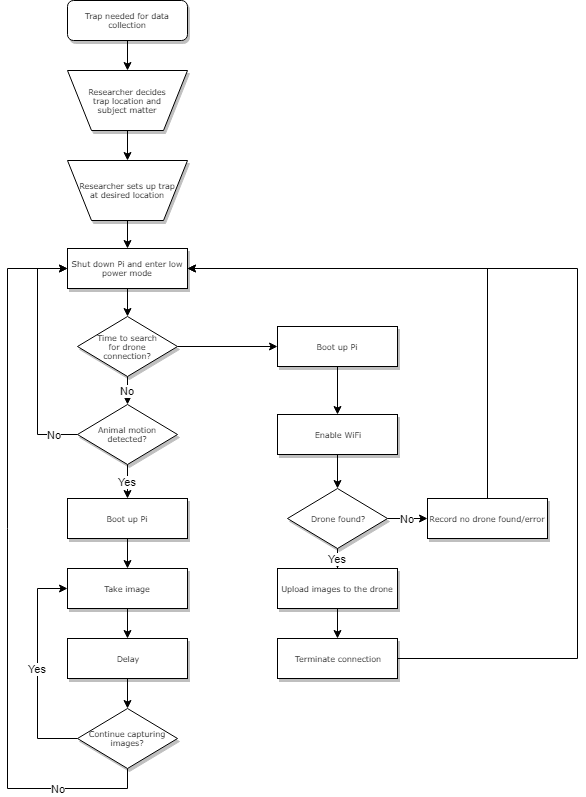
Little progress was made on setting up the companion computer or connecting it to the camera trap. The drone’s most challenging tasks involve connecting to the camera trap, so this should be a point of focus going forward. One method is to set up the drone’s Pi as a WiFi access point that the camera trap can search for and connect to. There are several tutorials online on how to set up a Raspberry Pi as an AP, [such as this official guide](https://www.raspberrypi.org/documentation/configuration/wireless/access-point.md).

While the easiest solution is to use the WiFi connection for both detecting the drone/camera trap and transmitting data, searching for a connection will consume considerable power. The Pi must be powered on, with WiFi enabled, every time it wants to look for the drone. An alternate method would be to use a separate wireless module to communicate directly with the Sleepy, even while the Pi is off.

There are countless hardware options to fill this need. Wireless modules utilizing WiFi, Bluetooth Low Energy (BLE), or the Zigbee protocol are readily available. Examples include ESP8266 modules such as [this](https://www.sparkfun.com/products/13678) or [this](https://learn.adafruit.com/adafruit-huzzah-esp8266-breakout/using-arduino-ide), BLE modules such as [these](https://www.adafruit.com/categories/255), and XBee modules found [here](https://www.sparkfun.com/pages/xbee_guide). These modules offer a wide range of capabilities but will add to the system’s complexity and minimum power consumption. If they do not offer substantial power reduction over using the Pi alone, they are probably not worth the effort (likely overkill).

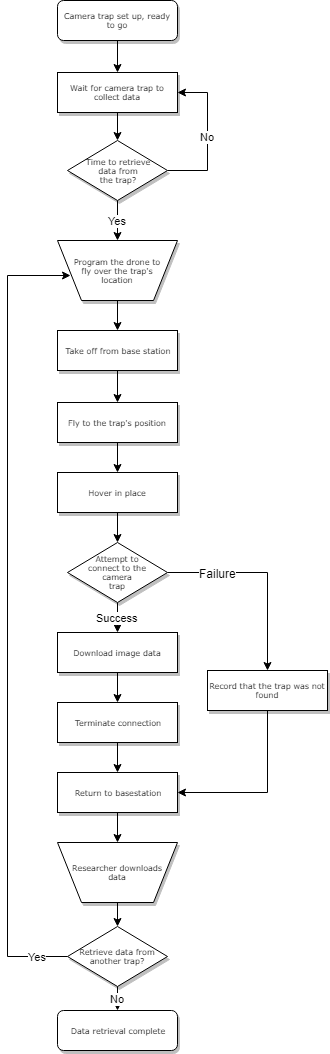
The lowest power option is to use a basic RF switch (basically a garage door opener) to ping the camera trap when the drone draws near. Modules [such as these](https://www.adafruit.com/product/1096) are cheap and drop dead simple to operate. The only drawback is that the camera trap would be unable to transmit meaningful data (status, battery life, etc.) back to the drone without booting up the Pi. This, however, is a small price to pay for a potentially huge reduction in power consumption.

**Appendix A: Operational Flowcharts**



Camera Trap Operational Flowchart

The diagram above shows one possible method for operating the camera trap. It assumes that a separate wireless module (RF, Zigbee, etc.) is not being used and the Pi must be booted to search for the aerial drone. It should also be noted that, in this setup, the trap does not exchange status information with the drone. The trap will operate until its battery dies, then shut off without warning. Both issues can (and ideally should) be eliminated with design improvements.



Drone Operational Flowchart

The diagram above shows one possible method for operating the aerial drone. It is designed to interact with the camera trap flowchart on the preceding page. This method does not collect data from multiple traps in a single flight, which is a desirable feature.

**Appendix B: Pi Zero W & Camera**

* First time setup
* Useful accessories
* Connecting to the Pi Zero
* Example systemd timelapse.service

**Appendix C: Sleepy Pi 2**

* Important features
* 3.3V I/O
* First time setup
  + Modifying init script
* Programming
* Headers
* Battery management
* Input voltage