# Raspberry Pi / PixHawk Drone Vision System

This document outlines the hardware and software required to build a vision system that extracts GPS coordinates from a Pixhawk, takes an image using a pi camera, processes the image for useful data, saves the image to a raspberry pi and transmits useful data as a string to an associated radio. It is assumed that the reader has access to a functional UAV controlled by a pixhawk with a GPS module. Familiarity with Python 2 and basic terminal commands would be useful.

**Hardware:**

* Raspberry Pi (model 3)
* Monitor, keyboard and mouse (any uni workstation)
* Pixhawk controlled drone
* Raspberry pi camera v2
* 2 x Sik radios (with USB cables)
* Pixhawk ‘telem’ cable, wires and soldering facilities

**Software:**

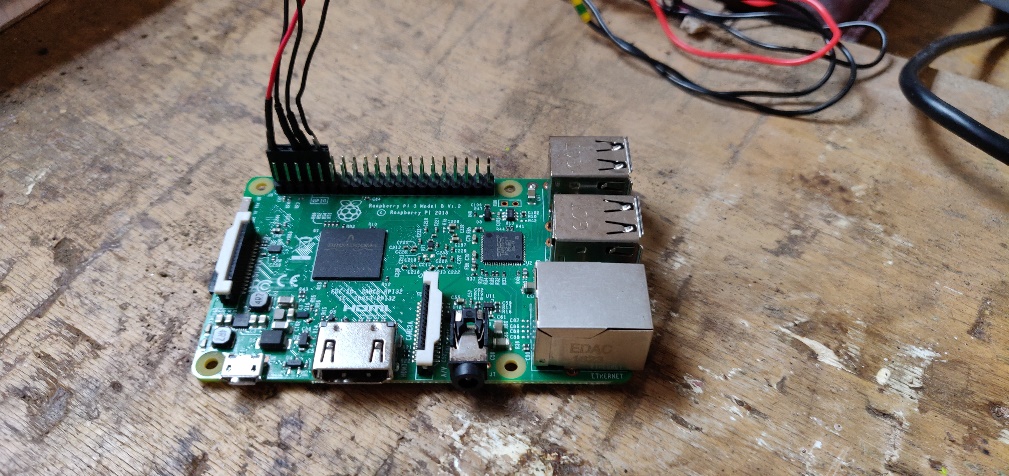
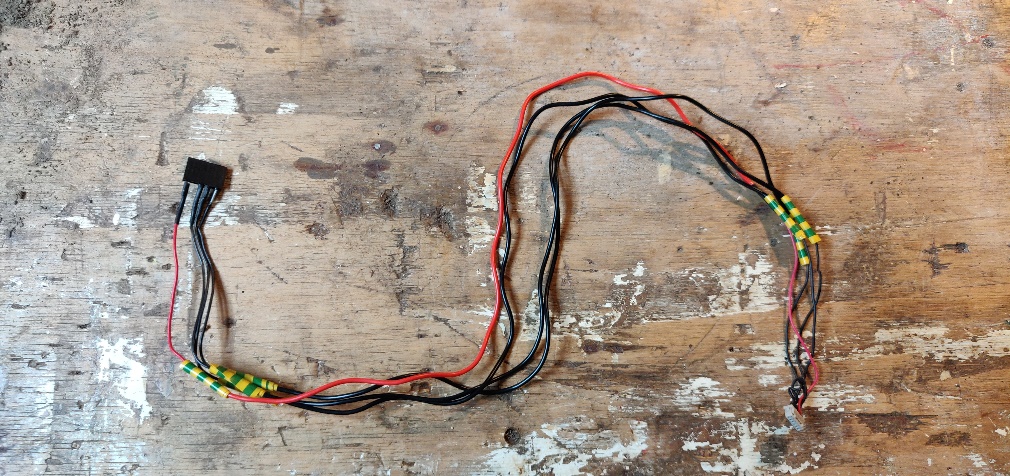
* Pi operating system - Raspbian (Jessie+)
* On the pi - Python 2+
* On your laptop – Python 2+

**Hardware Setup:**

*Connecting pi to pixhawk*

The pi communicates with the pixhawk controller using a UART serial connection (RX and TX lines). This requires a custom connector between the GPIO pins on the pi to the ‘telem 2’ on the pixhawk. The following link provides a breakdown of how to do this: <http://ardupilot.org/dev/docs/raspberry-pi-via-mavlink.html>.

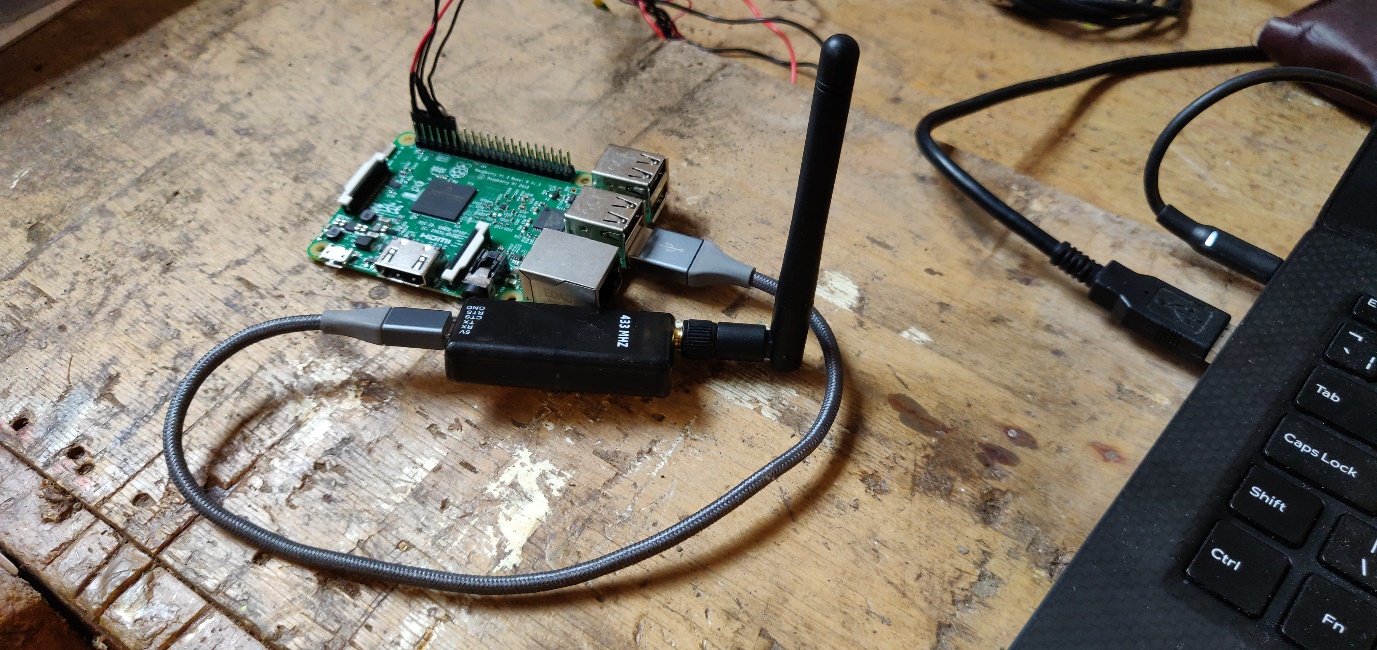
If you are using an off the shelf telem cable, you will have two redundant wires that can be cut short. The pi can be powered entirely from the 5V cable of this connector **so long as a monitor is not attached**.



*Connecting camera to pi*

The pi camera plugs into the pi using the ‘camera’ connector next to the hdmi port. Make sure to press the lever down fully or the camera will not be detected by the pi. Some initial setup may be required before using the camera: <https://thepihut.com/blogs/raspberry-pi-tutorials/16021420-how-to-install-use-the-raspberry-pi-camera>.

*Connecting SiK radios*

The SiK radios connect directly to a USB port on the pi. You should use the **same USB port** every time you attach the radio to the pi. This saves you having to change name of the port in the main script. The other radio should be attached in the same manner to a laptop.

This unit is enough to do everything we need. Note that for attachment to the drone, the pi needs a waterproof housing and the camera requires an unimpeded downward view.

**Code:**

*On the Pi*

You will need to have the following libraries installed and available to python; openCV (python-opencv), serial (pyserial), time, picamera and pymavlink. In the main script these must be imported;

import cv2

import serial

from pymavlink import mavutil

from time import sleep

from picamera import PiCamera

Images taken throughout the flight will need to be saved in a base directory;

BASE\_DIR = "/home/pi/Documents/"

Initialize the camera. A delay of two seconds is added to give the camera time to adjust to the light levels;

camera = PiCamera()

sleep(2)

A serial connection to the Sik radio must be established. The first argument is the USB port you have used to attach the radio which may be different on other models of pi. You can try listing the usb ports from the command line using ‘lsusb’ with the radio connected then disconnected. The port that changes is the radio.

port = serial.Serial("/dev/ttyUSB0", baudrate=57600, timeout=10)

Establish serial communication to the pixhawk using mavlink. port.write is how data is transmitted across the radio connection – note that strings must be given as byte arrays (put a ‘b’ before the string). wait\_heartbeat checks whether communication has been established. If unsuccessful, the code will not progress beyond this point;

master = mavutil.mavlink\_connection(

'/dev/serial0',

baud=57600)

port.write(b"Connecting to pixhawk...\n")

master.wait\_heartbeat()

port.write(b"Success!\n")

The actual data acquisition and transmission runs in an unending while loop;

while True:

try:

Retrieve the GPS cords from the pixhawk and format them correctly. **This data will only be streamed to the pi if the pixhawk has been told to do so in mission planner. This is done by adjusting the SR2 settings in the parameter list. GPS data was streamed at 10 Hz in this case**;

message = master.recv\_match(type="GLOBAL\_POSITION\_INT", blocking=True)

LAT\_STR = str(message.lat\*(10\*\*-7))

LON\_STR = str(message.lon\*(10\*\*-7))

ALT\_STR = str(message.alt\*(10\*\*-5))

Now an image can be captured. The image is saved to the base directory as a jpeg with its name being the latitude and longitude to ensure that each image is a unique location;

IMG\_NAME = LAT\_STR + "\_" + LON\_STR + ".jpg"

IMAGE = BASE\_DIR+IMG\_NAME

camera.capture(IMAGE)

The image is processed to identify the presence of a rectangular target. It is first converted to grayscale then a threshold is applied to identify bright regions. The contours of the image are identified. Any contours with four edges and an internal area within a given range are the target. This part of the code could do with expansion to include character recognition;

img = cv2.imread(IMAGE)

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

graycopy = gray.copy()

ret, thresh1 = cv2.threshold(gray,210,255,cv2.THRESH\_BINARY)

contours, heirarchy= cv2.findContours(thresh1,cv2.RETR\_TREE,

cv2.CHAIN\_APPROX\_SIMPLE)

CONT\_CORNERS\_TMP = False

for cnt in contours:

approx=cv2.approxPolyDP(cnt, 0.02\*cv2.arcLength(cnt, True), True)

if len(approx)==4 and 7000.0>cv2.contourArea(approx)>500:

cv2.drawContours(graycopy, [approx], 0, (0,255,0), 0)

CONT\_CORNERS\_TMP = True

If a target is identified, find the centre point and return its pixel coordinates;

M=cv2.moments(approx)

cX=int(M["m10"]/M["m00"])

cY=int(M["m01"]/M["m00"])

If an image shows a target, transmit the relevant data via radio;

if CONT\_CORNERS\_TMP:

TF = "True"

lat = "Lat: " + LAT\_STR + " - "

port.write(lat.encode())

lon = "Lon: " + LON\_STR + " - "

port.write(lon.encode())

alt = "Alt: " + ALT\_STR + " - "

port.write(alt.encode())

tf = "Signal: " + TF + " - "

port.write(tf.encode())

x = "xOffset: "+str(cX)+" - "

port.write(x.encode())

y = "yOffset: "+str(cY)+"\n"

port.write(y.encode())

If no target is identified, just transmit the current GPS cords;

else:

TF = "False"

lat = "Lat: " + LAT\_STR + " - "

port.write(lat.encode())

lon = "Lon: " + LON\_STR + " - "

port.write(lon.encode())

alt = "Alt: " + ALT\_STR + " - "

port.write(alt.encode())

tf = "Signal: " + TF + " - "

port.write(tf.encode())

x = "xOffset: "+"NA"+" - "

port.write(x.encode())

y = "yOffset: "+"NA"+"\n"

port.write(y.encode())

The following code cleans up any errors if they occur and relays them across the radio connection;

except Exception as e:

error = str(e)

port.write(error.encode())

# end sequence

port.close()

*On the laptop*

The following script runs continuously on a laptop, retrieving data transmitted across the radio link. Again, you will need to find the USB port to which your radio is attached;

import serial

ser = serial.Serial(port="COM14", baudrate=57600, timeout = 5)

while True:

try:

inp = ser.readline()

print(inp)

except:

ser.close()

ser.close()

**Improvements:**

*Exposure time –* Blur in the images could be reduced by decreasing the exposure time. This is done in the python script by increasing the frame rate.

*Improved geolocation* – Roll and pitch of the aircraft mean that the image captured is never exactly downward. The GPS cords at the centre of the image will be different from those recorded. To resolve this, the attitude data can be streamed from the Pixhawk in the same way as the GPS coordinates. This could then be used to calculate a more precise position. Alternatively, a gimble could be used to ensure the image is always vertically below the drone.

*Increasing field of view* – A significant limitation of the current system is the field of view. A fish eye lens could be applied to the camera and the image processing could be adjusted to account for skewed shapes. Increasing the altitude would provide some improvement but this is dependent on the drone. Alternatively, the pi camera could be easily replaced by a camera with a wider field of view.

*Character Recognition* – Various approaches can be taken to perform character recognition, however neural networks such as tesseract are likely to be too slow when run on the pi. They are also wholly unnecessary in this case. An alternative method is to identify the contours of the detected letter then measure features such as area and the number of edges. To do this, it is necessary to extract the innermost contour of the target (the letter). Code for this can be requested from the author (Oliver Stocks) but is not included here. Note that the target square is of a consistent size and shape so could potentially be used as a reference feature for comparison with the letter.