Autonomous Drone (Raspberry Pi 4B and Pixhawk)

■ **D.Manoj**

Introduction

In this project, we explore the exciting world of building a autonomous drone using a Raspberry Pi and Pixhawk autopilot. This project involves a combination of hardware and software components to create a fully functional drone capable of recognizing faces during autonomous. Below, we outline the steps involved in setting up this project.

Prerequisites

Before getting started, you'll need to gather the following components:

A general Pixhawk Drone: Pixhawk is a popular open-source autopilot system used in many autonomous vehicle projects. It provides high-level control to your drone.

Raspberry Pi: The Raspberry Pi serves as the onboard computer for your drone. It handles tasks such as image processing and communication with the Pixhawk.

Servo Motor (Micro Servo 9G - SG90): This servo motor is commonly used for controlling the drone's payload release mechanism.

Raspberry Pi Camera: The camera is essential for capturing images and video that the Raspberry Pi can process. It plays a crucial role in face recognition.

Wifi Dongle (For Hotspot): A Wi-Fi dongle allows your Raspberry Pi to establish a wireless connection, which is useful for remote control and data transfer.

Dronekit Python: Dronekit is a Python library that simplifies communication with Pixhawk and allows you to write scripts for autonomous flight.

Mavlink: Mavlink is a lightweight communication protocol used for communicating with Pixhawk and other UAV systems.

Face recognition Algorithm for Raspberry Pi: This algorithm is used for recognizing faces in captured images, a valuable feature for a autonomous drone.

Software Setup

Raspberry Pi OS Installation

Obtain the **Raspios-Buster**. This is the software that uploads operating system for your Raspberry Pi.

Use an IP scanner, such as Advanced IP Scanner, to find the Raspberry Pi's IP address when connected to Wi-Fi. This address will be important for remote communication.

To ensure the Raspberry Pi is up to date, use the following commands:

sudo apt-get update

sudo apt-get upgrade

sudo reboot

Face Recognition Library Setup

Before installing the face_recognition library, you need to install dlib, a prerequisite. Run the following commands to install the required dependencies on your Raspberry Pi:

**sudo apt-get install -y --fix-missing **

**build-essential **

**cmake **

**gfortran **

**git **

**wget **

**curl **

**graphicsmagick **

**libgraphicsmagick1-dev **

**libatlas-base-dev **

**libavcodec-dev **

**libavformat-dev **

**libgtk2.0-dev **

**libjpeg-dev **

**liblapack-dev **

**libswscale-dev **

**pkg-config **

**python3-dev **

**python3-numpy **

**software-properties-common **

**zip **

&& sudo apt-get clean && sudo rm -rf /tmp/* /var/tmp/*

Clone the dlib repository and install it:

the repository for the "dlib C++ Library." Dlib is an open-source C++ library primarily used for machine learning, computer vision, and deep learning tasks. It provides a wide range of tools and utilities for various applications in these domains, making it a popular choice for researchers and developers.

git clone https://github.com/davisking/dlib.git

cd dlib

mkdir build; cd build; cmake ..; cmake --build .

cd ..

sudo python3 setup.py install

Finally, install the face_recognition library using pip:

pip3 install face_recognition

Raspberry Pi to Pixhawk Communication Setup

To enable communication between the Raspberry Pi and Pixhawk, you'll need to install several libraries and configure some settings.

Install necessary packages:

sudo pip install future

sudo apt-get install screen

sudo apt-get install python-wxgtk4.*

sudo apt-get install libxml2-dev

sudo apt-get install libxslt1-dev

pip install lxml

sudo pip3 install pyserial

sudo pip3 install dronekit

sudo pip3 install geopy

config-setup

Configure the Raspberry Pi by enabling the camera, VNC (Virtual Network Computing), serial port for TX (Transmit) and RX (Receive). Use the following command:

sudo raspi-config

Identify the port connection for communication between the Raspberry Pi and Pixhawk. This step helps you find the correct serial port:

ls /dev/ttyAMA0

Open the configuration file to enable UART (Universal Asynchronous Receiver-Transmitter) and disable Bluetooth. This step ensures the UART port is available for data transfer:

sudo nano /boot/config.txt

Add the following lines at the bottom of the file if not already present:

enable_uart=1

dtoverlay=disable-bt

Testing the Connection

To test the connection between the Raspberry Pi and Pixhawk, use the following command:

mavproxy.py --master=/dev/ttyAMA0

This command opens a connection to Pixhawk, allowing you to view telemetry data from the drone.

Running the Project

Connect the Raspberry Pi and Pixhawk with wires, following the specific wiring diagram for your setup.

Write a Python script for your autonomous drone. For example, you can use the provided **autonomous_drone_yt.py** script as a starting point. This script should handle tasks such as flight control, payload release, and face recognition.

Run the Python script with the following command, specifying the connection port to Pixhawk:

python3 autonomous_drone_yt.py --connect /dev/ttyAMA0

This command initiates the autonomous flight and autonomous process, including face recognition, payload release, and other flight control operations.

Code to check connection establishment

```
# Import DroneKit-Python
from dronekit import connect, VehicleMode


# Connect to the Vehicle.
print("Connecting to vehicle on: %s" % (connection_string,))
vehicle = connect(connection_string, wait_ready=True)


# Get some vehicle attributes (state)
print "Get some vehicle attribute values:"
print " GPS: %s" % vehicle.gps_0
print " Battery: %s" % vehicle.battery
print " Last Heartbeat: %s" % vehicle.last_heartbeat
print " Is Armable?: %s" % vehicle.is_armable
print " System status: %s" % vehicle.system_status.state
print " Mode: %s" % vehicle.mode.name    # settable


# Close vehicle object before exiting script
vehicle.close()
# Shut down simulator
```

```
sitl.stop()
print("Completed")
```

code for co-ordinate navigation

```
# -*- coding: utf-8 -*-

from __future__ import print_function
import time
from dronekit import connect, VehicleMode, LocationGlobalRelative

# Set up option parsing to get connection string
connection_string = '/dev/ttyACM0'
# Connect to the Vehicle
print('Connecting to vehicle on: %s' % connection_string)
vehicle = connect(connection_string, wait_ready=True)

def arm_and_takeoff(aTargetAltitude):
    print("Arming motors")
    # Copter should arm in GUIDED mode
    vehicle.mode = VehicleMode("GUIDED")
    vehicle.armed = True

    while not vehicle.armed:
        print(" Waiting for arming...")
        time.sleep(1)

    print("Taking off!")
    vehicle.simple_takeoff(aTargetAltitude)  # Take off to target altitude

    # Wait until the vehicle reaches a safe height before processing the goto
```

```
    #  (otherwise the command after Vehicle.simple_takeoff will execute
    #   immediately).
    while True:
        print(" Altitude: ", vehicle.location.global_relative_frame.alt)
        # Break and return from function just below target altitude.
        if vehicle.location.global_relative_frame.alt >= aTargetAltitude * 0.95:
            print("Reached target altitude")
            break
        time.sleep(1)

arm_and_takeoff(5)

print("Set default/target airspeed to 3")
vehicle.airspeed = 2

print("Going towards first point for 30 seconds ...")
point1 = LocationGlobalRelative(13.011654, 77.705102, 5)
vehicle.simple_goto(point1)

# sleep so we can see the change in map
time.sleep(10)

print("Returning to Launch")
vehicle.mode = VehicleMode("RTL")

# Close vehicle object before exiting script
print("Close vehicle object")
vehicle.mode = VehicleMode("LAND")
vehicle.close()
# Shut down simulator if it was started.
```

Python Code for drone delivery

```
from dronekit import connect, VehicleMode, LocationGlobal,
LocationGlobalRelative
from pymavlink import mavutil
import time
import math
import socket
import argparse
import geopy.distance

import face_recognition
import picamera
import numpy as np



#connect to drone
def connectMyCopter():
  parser =  argparse.ArgumentParser(description='commands')
  parser.add_argument('--connect')
  args = parser.parse_args()

  connection_string = args.connect
  baud_rate = 57600
  print("\nConnecting to vehicle on: %s" % connection_string)
  vehicle = connect(connection_string,baud=baud_rate,wait_ready=True)
  return  vehicle


#arm and takeoff to meteres
def arm_and_takeoff(aTargetAltitude):
    """
    Arms vehicle and fly to aTargetAltitude.
    """

    print("Basic pre-arm checks")
    # Don't let the user try to arm until autopilot is ready
    while not vehicle.is_armable:
        print(" Waiting for vehicle to initialise...")
        time.sleep(1)


    print("Arming motors")
    # Copter should arm in GUIDED mode
    vehicle.mode = VehicleMode("GUIDED")
    vehicle.armed = True

    while not vehicle.armed:
        print(" Waiting for arming...")
        time.sleep(1)
```

```
    print("Taking off!")
    vehicle.simple_takeoff(aTargetAltitude) # Take off to target
altitude

    # Wait until the vehicle reaches a safe height before processing
the goto (otherwise the command
    #  after Vehicle.simple_takeoff will execute immediately).
    while True:
        print(" Altitude: ",
vehicle.location.global_relative_frame.alt)
        if
vehicle.location.global_relative_frame.alt>=aTargetAltitude*0.95:
#Trigger just below target alt.
            print("Reached target altitude")
            break
        time.sleep(1)


def drop_parcel():
    msg = vehicle.message_factory.command_long_encode(
    0, 0,    # target_system, target_component
    mavutil.mavlink.MAV_CMD_DO_SET_SERVO, #command
    0, #confirmation
    9,    # servo number
    1000,          # servo position between 1000 and 2000
    0, 0, 0, 0, 0)    # param 3 ~ 7 not used
    print("dropping parcel...")
    # send command to vehicle
    vehicle.send_mavlink(msg)
    print("parcel dropped...")



def get_dstance(cord1, cord2):
    #return distance n meter
    return (geopy.distance.geodesic(cord1, cord2).km)*1000



# Get a reference to the Raspberry Pi camera.
# If this fails, make sure you have a camera connected to the RPi and
that you
# enabled your camera in raspi-config and rebooted first.
camera = picamera.PiCamera()
camera.resolution = (640, 480)
output = np.empty((480, 640, 3), dtype=np.uint8)
# Load a sample picture and learn how to recognize it.
print("Loading known face image(s)")
rahul_image = face_recognition.load_image_file("faces/manoj11.jpeg")
rahul_face_encoding = face_recognition.face_encodings(manoj_image)[0]

# Initialize some variables
face_locations = []
face_encodings = []


#connect to drone
```

```
vehicle = connectMyCopter()



def goto_location(to_lat, to_long):

    print(" Global Location (relative altitude): %s" %
vehicle.location.global_relative_frame)
    curr_lat = vehicle.location.global_relative_frame.lat
    curr_lon = vehicle.location.global_relative_frame.lon
    curr_alt = vehicle.location.global_relative_frame.alt

    # set to locaton (lat, lon, alt)
    to_lat = to_lat
    to_lon = to_long
    to_alt = curr_alt

    to_pont = LocationGlobalRelative(to_lat,to_lon,to_alt)
    vehicle.simple_goto(to_pont, groundspeed=1)

    to_cord = (to_lat, to_lon)
    while True:
        curr_lat = vehicle.location.global_relative_frame.lat
        curr_lon = vehicle.location.global_relative_frame.lon
        curr_cord = (curr_lat, curr_lon)
        print("curr location: {}".format(curr_cord))
        distance = get_dstance(curr_cord, to_cord)
        print("distance ramaining {}".format(distance))
        if distance <= 2:
            print("Reached within 2 meters of target location...")
            break
        time.sleep(1)


def identify_person():
    found = False
    while True:
        print("\nCapturing image.")
        # Grab a single frame of video from the RPi camera as a numpy
array
        camera.capture(output, format="rgb")

        # Find all the faces and face encodings in the current frame of
video
        face_locations = face_recognition.face_locations(output)
        print("Found {} faces in image.".format(len(face_locations)))
        face_encodings = face_recognition.face_encodings(output,
face_locations)

        # Loop over each face found in the frame to see if it's someone
we know.
        for face_encoding in face_encodings:
            # See if the face is a match for the known face(s)
            match =
face_recognition.compare_faces([manoj_face_encoding], face_encoding)
            name = "<Unknown Person>"

            if match[0]:
```

```
                name = "manoj"
                found = True
                print("Found {} ".format(name))


            print("I see someone named {}!".format(name))

        if found:
            #found customer
            break



def my_mission():
    arm_and_takeoff(3)
    goto_location(25.806476,86.778428)
    identify_person()
    drop_parcel()
    time.sleep(2)
    print("Returning to Launch")
    vehicle.mode = VehicleMode("RTL")



# invoke the mission
my_mission()
```

Conclusion

Building a autonomous drone using a Raspberry Pi and Pixhawk is an exciting project that combines hardware and software to create a fully functional system. By following the steps outlined in this guide, you can create your own autonomous drone capable of recognizing faces during autonomous tasks. This project opens up a world of possibilities for applications in autonomous autonomous and beyond.

This article provides a comprehensive guide to setting up a Raspberry Pi and Pixhawk-based autonomous drone project, from software installation to testing the connection and running the project. With the right components and software, you can embark on your own journey of building intelligent and autonomous drones.