Drona Aviation: The Pluto Drone Swarm Challenge

Task 1 Report

Primary Team ID: 17
Secondary Team ID: 34

Contents

Working Principles

Sockets

- We connected to Pluto first by connecting our device to the drone's hotspot, then we used TCP sockets to connect to the drone using IPV4 scheme and `socket` library in Python to do the same
- The server is hosted at `192.168.4.1:23`
- We sent the MSP Packet data encoded as bytes to Pluto using sockets, the details about the packet is explained below

MSP Packets

- We used MSP Packet scheme to communicate with Pluto, ie send commands, request data and calibrate the drone
- The data of the packet was first encoded into bytes and then sent as a byte stream (array of bytes) to the drone
- The basic structure of the packet has been explained below

Header	Direction	Message Length	Type of Payload	Message Data	Checksum
2 Bytes	1 Byte	1 Byte	1 Byte	N Bytes	1 Byte
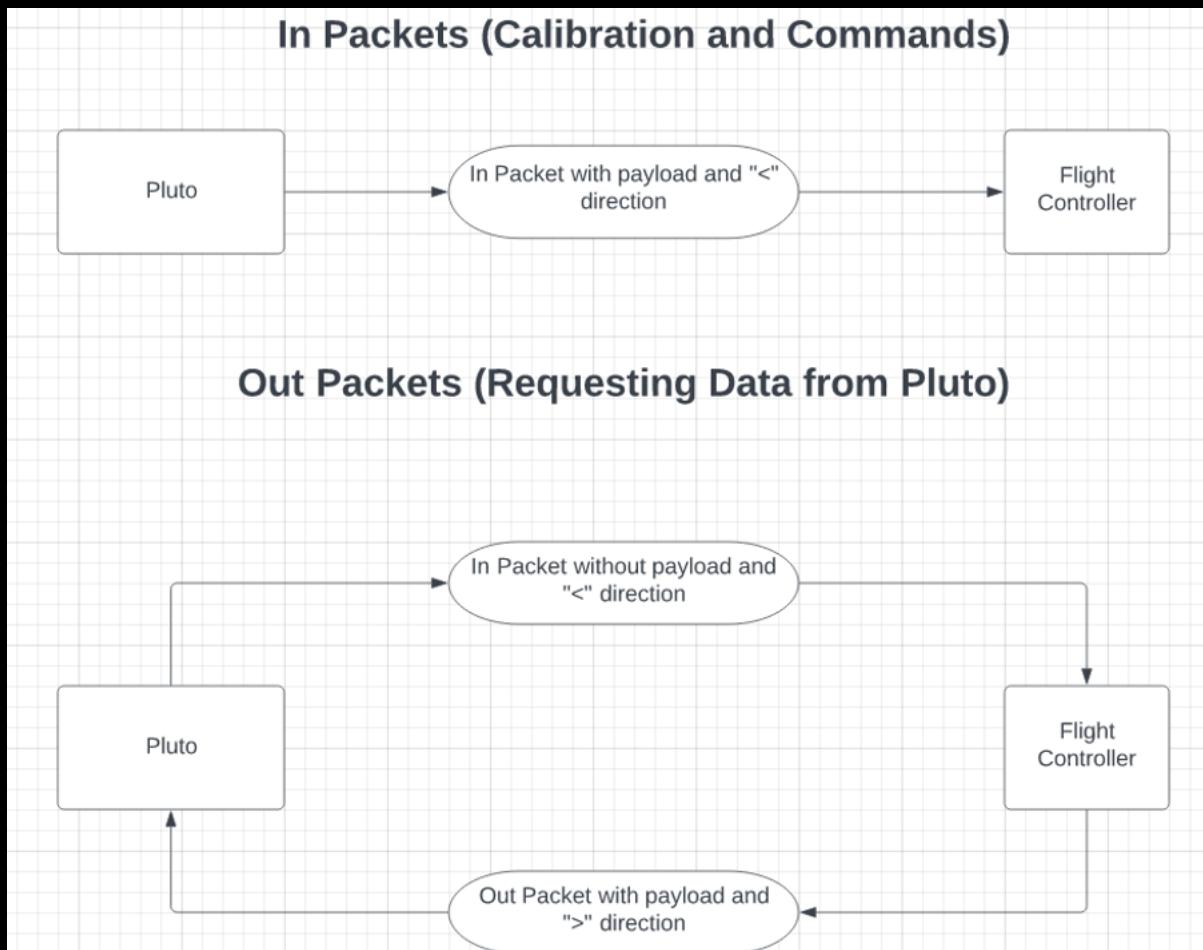
*Structure of the Packet

Type of Byte	ASCII	Hexadecimal
Header	$M	0x24 0x4d
Direction	'<' or '>'	0x3c (to the drone) or 0x3e (from the drone)
Message Length		0x00 - 0xff
Type of Payload		0x01 - 0xff
Payload		Message Body encoded into N bytes
Checksum		XOR of Bytes of "Msg length", "Command" and all bytes of "Payload"

*Details of the Packet

MSP Packets (Contd.)

- As defined in the MSP Packet scheme, every packet has a different packet code, along with it's direction, packet size and payload format specified
- The parameters of the payload are encoded according to byte sizes specified by it's data types

*Flow of MSP Packets with Pluto

NOTE: The drone requires packets continuously to function properly, as soon as the packets stop coming to the drone, it stops taking instructions and lands, rendering the further delayed commands useless and ineffective. To avoid this, run the script provided in our tutorial, which will continuously send packets to the drone so it does not run out of commands

For proper functioning of the drone do refer to our tutorial very thoroughly

link to the tutorial

Our Approach

Code

- We have used Classes in Python to make modules for different functionalities
- The two of tbe most important class which will be used for connecting to the drone and controlling the drone are `PlutoConnection` and `PlutoControl`

PlutoConnection

- This module is inherited from the socket class in Python
- Hence it inherits the socket module's connect function which we utilise to connect to our Pluto Drone's server at `192.168.4.1:23`, while being connected to the drone's network
- Hence, the object of this class becomes a connection object to the drone which will be used to send the messages to the drone in the form of bytes as already discussed above

PlutoControl

- This module will be used to control the drone, it has certain functions defined, like `take_off`, `land`, `pitch_forward`, `pitch_backward`, `roll_right`, `roll_left`, `yaw_right` and `yaw_left`
- This modules takes in the `PlutoConnection` object as a parameter to communicate with the drone
- This module utilises two other modules namely `SetRawRC` and `SetCommand`.
  - `SetRawRC`: This module sets the RC Params **(Roll, Pitch, Yaw and Throttle)** to the drone and sends it via the connection object defined above
  - `SetCommand`: This module set's the type of operation we need the drone to perform, Take Off and Land being the common examples
- Internally, as we initialise the object for the `PlutoControl` class, within it, two objects one of `SetRawRC` module and one for `SetCommand` module are initialised
- The `take_off()` function first arms the drone using `SetRawRC` object and then sets the command type to **TAKE_OFF** using the `SetCommand` object
- Similarly the `land()` function sets the command type to **LAND**
- The other commands for **Roll**, **Pitch** and **Yaw** simply utilise the `SetRawRC` object to control the drone using RC Params.

Flowchart

- The following flowchart may help you to understand the basic flows to operate Pluto