Rapport labo 3

Auteurs : Cédric Rosat, Costantino Cecchet

Date : 30.04.2024

Transfert des informations de l'UART vers STM32

Valeur de la fréquence de la FC : `IAALAB03: CL FREQ = 50'000'000`. Cette dernière est envoyée depuis le GAP8 et récupérée par le STM32 avant d'être affichée dans la console du client Crazyflie.

Création d'une tâche WiFi

Voici un exemple d'affichage dans la console du client Crazyflie lorsqu'on se connecte au drone :

```
CPX: ESP32: I (7374) WIFI: station:60:3e:5f:35:aa:4e join, AID=1 <= Indique
qu'un client s'est simplement connecté au WiFi
CPX: ESP32: I (11624) WIFI: Connection accepted <= Le client a ouvert un
socket (p. ex avec le script connect.py)
CPX: ESP32: I (11634) WIFI: Client connected <= Message venant de l'ESP32
indiquant que la connection est établie
CPX: CPX connected <= La librairie CPX indique que la connection est
établie
CPX: GAP8: Wifi client connection status: 1 <= Notre print qui indique que
le client a ouvert un socket
CPX: GAP8: Client connected...CPX: GAP8:  starting capture.
CPX: GAP8: Stop image capture
CPX: GAP8: Send image of size: 200x200
CPX: ESP32: I (11934) WIFI: Client disconnected <= L'ESP32 indique que le
socket a été fermé par le client
CPX: CPX disconnected <= Même message de la part de la librairie CPX
CPX: ESP32: I (11934) WIFI: Waiting for connection <= L'ESP32 est prêt pour
ouvrir un nouveau socket
CPX: GAP8: Wifi client connection status: 0 <= Notre print indiquant que le
socket a été fermé
```

Les messages de l'ESP32 et de la librairie CPX sont gérés par la `xTask` `rx_wifi_task` qui créé un "listener" permettant d'accepter et de gérer des connections wifi. Quand on ouvre/ferme un socket, dans notre `rx_wifi_task` on reçoit un paquet `rxp` qu'on peut lire afin de définir si le client a ouvert/fermé le socket.

Lorsqu'un premier client se connecte au WiFi du drone, sa connection est acceptée et il peut ensuite ouvrir un socket. Si maintenant un second client tente de se connecter pendant que le premier est encore connecté, la connection ne peut se faire. Cependant, on ne voit pas dans notre console qu'une seconde connection a été refusée. Le client affiche simplement un message pour dire que le WiFi est inaccesible sans autres informations. Ce comportement est bizarre sachant que, selon le datasheet du module EPS32

NINA W102, ce dernier prend en charge le WiFi 802.11b/g/n qui devrait accepter jusqu'à 200 connections en simultané.

Transfert des informations du GAP8 vers le PC via WiFi

La taille maximale de packet a envoyer à la fois est de 50. Ceci est défini par la structure donnée ci-dessous:

```
typedef struct {
  WiFiCTRLType cmd;
  uint8_t data[50];
} __attribute__((packed)) WiFiCTRLPacket_t;
```

Notre protocole de communication est le même que l'exemple `wifi_image_streamer.c` se trouvant dans la machine virtuelle fournie, mais sans le footer.

Il sagit d'un header qu'il faut envoyer en premier pour indiquer la taille de l'image, le type (RAW ou JPEG) et une constante magique (souvent utilisée pour indiquer le début d'une en-tête). Ce header est important car l'image est ensuite envoyée par paquets de 50 octets et nous devons indiquer au destinataire le quantité de données qui sera envoyée.

```
typedef struct {
    uint8_t magic; // Identifiant pour indiquer le début de l'en-tête
    uint16_t width;
    uint16_t height;
    uint8_t depth; // Profondeur de l'image: nuances de gris (=1), RGB (=3)
et RGB-D (=4)
    uint8_t type; // Encodage: RAW (=0) ou JPEG (=1)
    uint32_t size;
} __attribute__((packed)) img_header_t;
```

Nous avons décidé de laisser tout le header afin de rendre le code évolutif. Cela permet d'envoyer aussi des images qui seraient prétraitées en JPEG, ou alors des images en couleur / couleur-profondeur. Pour notre cas, nous envoyons des images en nuances de gris et encodées en RAW.

Fonctionnnement du script de récupération des images

Pour récupérer les images, nous utilisons un script python qui, une fois la machine hôte connectée sur le wifi du drone, ouvre un socket et attend de recevoir des données. Une fois les données reçues, le script affiche un flux d'image.

Il faut flash le GAP8 avec la commande suivante:

```
make build image flash
```

Ceci fait, le drone doit être redémarré afin d'exécuter le nouveau code et d'envoyer les images.

Pour lancer le script python, il faut lancer la commande suivante:

```
python3 image_retriever.py
```

Dans un premier temps, le header est récupéré :

```
packetInfoRaw = rx_bytes(4)
[length, routing, function] = struct.unpack('<HBB', packetInfoRaw)

imgHeader = rx_bytes(length - 2)
[magic, width, height, depth, format, size] = struct.unpack('<BHHBBI',
imgHeader)
```

Puis, si le magic paquet est correcte, l'image sera récupérée dans un tableau ;

```
imgStream = bytearray()
while len(imgStream) < size:
  # Récupération de l'image
```

Pour finir, le buffer est lu et convertit en image :

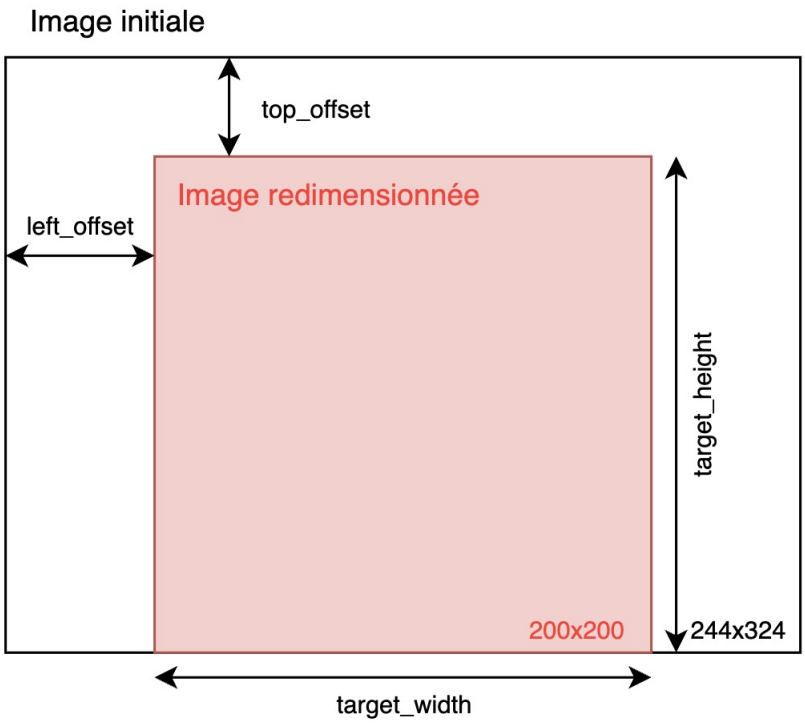
```
bayer_img = np.frombuffer(imgStream, dtype=np.uint8)
bayer_img.shape = (200, 200)
cv2.imshow('Raw', bayer_img)
```

Traitement des images

La taille de l'image de base est de 244x324. Cette dernière est trop grande pour nos manipulations, nous allons donc devoir la redimensionner. Cette partie est directement faite sur le drone et permet de réduire la quantité de données qui doit être transférée vers le client. Voici le code qui est utilisé pour cela :

```
for (int i = 0; i < CROPPED_HEIGHT; i++) {
    for (int j = 0; j < CROPPED_WIDTH; j++) {
        cropped_image[i * CROPPED_WIDTH + j] = imgBuff[(i + ORIGINAL_HEIGHT
- CROPPED_HEIGHT) *
                                                       ORIGINAL_WIDTH + j +
(ORIGINAL_WIDTH -
                                                       CROPPED_WIDTH)/2];
    }
}
```

Explication de l'algorithme



Dans la figure ci-dessus, on peut voir l'image originale et l'image redimensionnée. Nous avons décidé de centrer cette dernière sur son axe horizontal mais de la placer au plus bas sur son axe vertical. Cela nous permet de garder les informations visuelles qui se situes au plus proche du drone (voir image dans la partie exemple).

Quant au fonctionnement de l'algorithme (voir code plus haut), ce dernier utilise 2 boucles imbriquées qui vont parcourir les pixels de l'image redimensionnés (image rouge). La première permet de parcourir chaque ligne de l'image. La seconde elle parcourt chaque pixel. Et dans la seconde boucle, les index sont calculés en ajoutant les offset gauche et supérieurs afin de copier dans l'image redimensionnée, uniquement les pixels correspondants de l'image originale.

Exemple

Sur les images ci-dessous, on peut voir à droite, l'image originale prise par la caméra (244x324) et à gauche, l'image redimensionnée (200x200). Pour réaliser cet exemple, nous avons envoyé l'image de gauche sans appliquer le crop et l'image de droite une fois le crop appliqué.

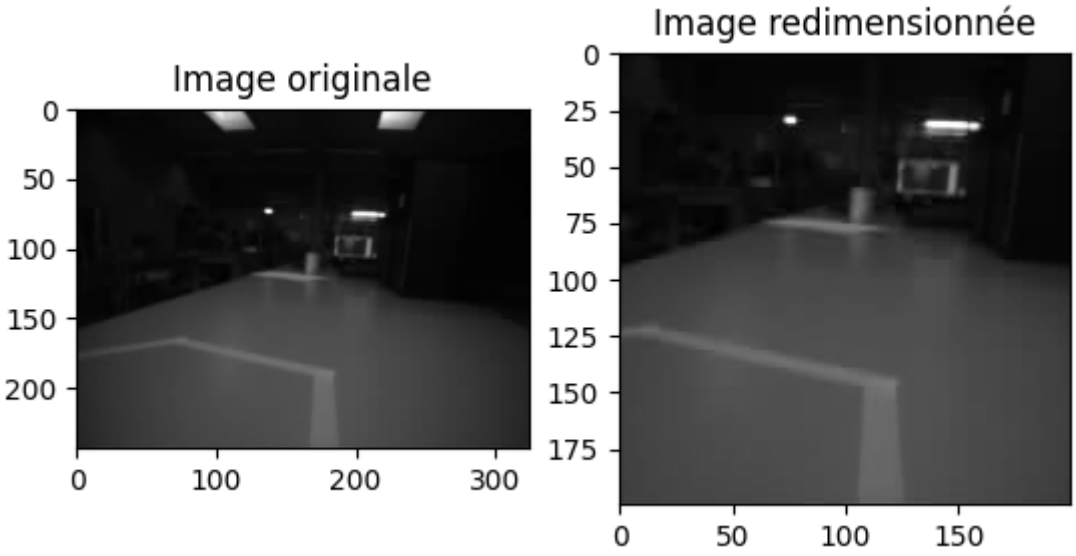


Image originale
0
50
100
150
200
0
100
200
300
Image redimensionnée
0
25
50
75
100
125
150
175
0
50
100
150