HEIG VD | REDS Institut Reconfigurable and Embedded Digital Systems

Prof. Yann Thoma,
Marina Zapater,
Assist. Mehdi Akeddar,
Guillaume Chacun,
Thomas Rieder

Artificial Intelligence for Autonomous Systems

Spring 2024

Lab 3 - Drone communications

This lab will be graded.

Please answer the questions in a separate text file. The questions are marked in blue.

We provide a pdf file as an Annex to this lab with more detailed information and guidelines about the code. Also, please refer to the '*cpx.h*' for examples of code usage.

Available time : **3 lab sessions**.

Please upload your code and text file with your answers on cyberlearn before the next lab session scheduled for **May 7th**.

1 Lab objectives

In this lab, you will get familiar with the edge device used in this course. You will implement different functionalities on the GAP8, STM32, as well as a connection to a PC.

2 Transfer information from UART to STM32

We ask you to send the Fabric Controller (FC) frequency value from GAP8 to STM32. An FC core is used for control, communications, and security functions. More information can be found here. To complete this stage, follow these steps :

— Complete the function `sendToSTM32` in the GAP8. The code doesn't need to be generic for all types of data.

— Adapt the STM32 application if necessary and change the variable `CRAZYFLIE_BASE` in your Makefile to the absolute path to the crazyflie-firmware.

— What is the value of the FC frequency ?

3 Create a WiFi Task

Complete the task called `rx_wifi_task` that outputs 1 when a PC is connected and 0 if not. This task is necessary to know whether a PC has been connected to the drone because sending packets from the drone without a connected PC could cause errors or unwanted blocking.

— In the CfClient console do you only see your prints ? If not, explain the other prints.

— What part of the data flow is responsible for these ?

— What happens when you connect 2 clients to the GAP8 ? Is it an expected behavior ?

4 Image acquisition

Complete the task called `camera_task`. Please refer to the section "Task acquisition image de la camera" from the README in the annex. Complete also the callback called `capture_done_cb`.

5 Transfer information from GAP8 to PC via WiFi

Complete the function called `send_image_via_wifi` to send an image from the GAP8 to a local PC. This function implies the conception of a simple communication protocol to deal with the size of the image. From the task `camera_task` send the images taken by the camera of your drone using your new `send_image_via_wifi` function. Images should only be sent if a PC is currently connected to the drone.

Write a Python script that connects to the WiFi of the drone and receive the images. Your script has to follow the same communication protocol as the one used on the drone. The example you used in Lab 1 can serve as an inspiration.

— What is the max packet size ?

— Explain your implementation of the communication protocol.

6 Image processing

As you may observe, the images sent initially in stage 4 are quite big. We ask you to implement a new function to reshape the images on the GAP8 before sending them via WiFi. This function takes an original image and returns a [1x200x200] image.

— Provide a plot of one original image and one of the cropped images.

— Explain your cropping algorithm.

— What is the shape of the original image ?

APP Bootstrap

- Fonction main
  - initialiser bsp avec `pi_bsp_init`
  - Set fréquence/voltage si nécessaire
  - appeler votre fonction d'init à l'aide de `pmsis_kickoff(func);`
- fonction d'init
  - init CPX à l'aide de `cpxInit()`
  - activer les fonction de cpx (qui nécessitent une queue comme **CPX_F_WIFI_CTRL**) à l'aide de `cpxEnableFunction(FUNC)`
  - Attendre un certain délai afin de s'assurer que le drone ait le temps de boot correctement à l'aide de `vTaskDelay(2000);`
  - Possibilité de faire différentes initalisations
  - initialiser les tasks qui pourront être bloquante
  - finir par une boucle infinie

    ```
    while(1) {
        pi_yield();
    }
    ```

  - Pour le bon fonctionnement des différentes tasks, il est important que cette fonction rentre dans cette boucle, elle ne doit donc pas faire d'appel bloquant avant.

transfert UART vers stm32

- Créer un `CPXPacket_t`
- initialiser sa route à l'aide de la fonction suivante (tout se trouve dans *lib/cpx/inc/cpx.h*) `void cpxInitRoute(const CPXTarget_t source, const CPXTarget_t destination, const CPXFunction_t function, CPXRouting_t *route)`
  - source: GAP8
  - destination: STM32
  - function: APP
  - route: `&packet.route`
- Copier les données dans `packet.data`
- Set la taille des données dans `packet.dataLength`
- Envoyer le packet à l'aide de la fonction `cpxSendPacketBlocking(&packet)`

Task gestion wifi

Cette tâche est nécessaire afin de savoir si un PC s'est connecté au drone. Car envoyer des paquet depuis le drone sans qu'un PC soit connecter pourrait causer des erreur ou un blocage non souhaité.

- Cette Task soit être init dans votre fonction d'init de la manière suivante :

```
BaseType_t xTask;
xTask = xTaskCreate(rx_wifi_task, "rx_wifi_task", configMINIMAL_STACK_SIZE *
2,
                    NULL, tskIDLE_PRIORITY + 1, NULL);
```

- Elle doit boucler à l'infini afin de lire des packet de fonction `CPX_F_WIFI_CTRL`
- Afin de lire des paquet en précisant sa fonction, il est nécessaire d'activer une queue pour cette fonction dans la fonction d'init de la manière suivante: `cpxEnableFunction(CPX_F_WIFI_CTRL);`
- Il est ensuite possible de lire un paquet ayant cette fonction grâce à :
  - `cpxReceivePacketBlocking(CPX_F_WIFI_CTRL, &rxp);`
  - rxp étant de type `CPXPacket_t`
- Une fois un paquet de ce type reçu, les données sont dans un format définit dans la structure `WiFiCTRLPacket_t` se trouvant dans *wifi.h* (possible de cast `rxp.data` en `WiFiCTRLPacket_t`)
- La commande dans cette structure peut correspondre à différente actions (voir enum `WiFiCTRLType` dans *wifi.h*), la cmd représentant un client se connectant au drone est `WIFI_CTRL_STATUS_CLIENT_CONNECTED` définit dans *wifi.h*
- Lorsque un paquet de ce type est reçu est que la cmd vaut `WIFI_CTRL_STATUS_CLIENT_CONNECTED` il est possible de mettre une variable global a jour (par exemple) permettant ainsi aux autres tâches de savoir si un client est connecté.

transfert WIFI GAP8 to PC

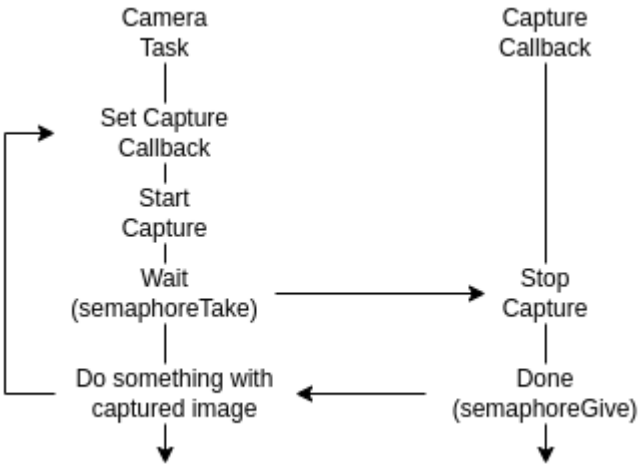
Afin d'envoyer de grosse données par WIFI, il est nécessaire de concevoir un petit "protocole" de communication afin d'informer combien de donnée sont transmises et de quel types il s'agit (d'une image, d'une commande ou autre) .

Voici une façon de faire:

- Toujours envoyer un header avant les données.
- Ce header va permettre d'informer combien de donnée le client connecté va devoir réceptionner.
- Dans notre cas (pour une image **RAW**) ce header peut contenir un **type** précisant qu'il s'agit d'une image, de la **width**, **height** et **profondeur** de l'image. Si l'image subit une compression alors la taille != width*height*depth il est donc nécessaire d'ajouter un champ **size**.
- Afin de simplifier la lecture de ce header par le récepteur, il est pratique de définir ce header comme une structure packed à l'aide de `__attribute__((packed))`. Cela permet d'éviter d'aligner les différents champs de la structure et de les mettre directement à la suite l'un de l'autre dans la mémoire. (Cela réduit aussi la taille en mémoire de la structure)
- Il est donc nécessaire d'envoyer un paquet CPX contenant comme donnée ce header avant l'image qui elle, sera split en plusieurs paquets.
- Pour envoyer un buffer conséquent, il existe la fonction `sendBufferViaCPXBlocking` se trouvant dans *wifi.h* qui s'occupe de séparer le buffer en différent paquet CPX et de les envoyer.

Task acquisition image de la camera

la librairie d'acquisition d'image fournie par FreeRTOS fonctionne à l'aide d'un callback voici une solution possible d'implémentation.



- Avant tout il est nécessaire d'initialiser un device correspondant à la caméra en prenant exemple sur la fonction `open_pi_camera_himax` se trouvant dans l'exemple du lab01 **wifi-img-streamer**.
- Inclure le fichier `bsp/camera/himax.h`
- Allouer puis Initialiser un buffer pouvant contenir l'image à l'aide des fonctions suivates :

```
#include "bsp/buffer.h"


static pi_buffer_t buffer;
...
    imgBuff = (unsigned char *)pmsis_l2_malloc(CAM_WIDTH*CAM_HEIGHT);
    pi_buffer_init(&buffer, PI_BUFFER_TYPE_L2, imgBuff);
    pi_buffer_set_format(&buffer, CAM_WIDTH, CAM_HEIGHT, 1,
PI_BUFFER_FORMAT_GRAY);
...
```

- Créer une task qui s'occupera de la capture, cette task doit :
  - Set le callback qui sera appelé lorsque la capture est terminée avec `pi_camera_capture_async`
  - démarre la capture avec `pi_camera_control(&camera, PI_CAMERA_CMD_START, 0);`
  - attendre à l'aide d'un semaphore (ou autre)
  - puis faire ses traitements/envoie de donnée sur l'image capturée.
- Le callback lui n'est responsable que de :
  - Stopper la capture à l'aide de `pi_camera_control(&camera, PI_CAMERA_CMD_STOP, 0);`
  - puis de libérer le sémaphore.

Semaphore binaire sur FreeRTOS

- Définition: `SemaphoreHandle_t capture_sem;`
- Initialisation: `capture_sem = xSemaphoreCreateBinary();`
- Take (sans timeout): `xSemaphoreTake(capture_sem, portMAX_DELAY);`
- Give : `xSemaphoreGive(capture_sem);`