Projet - Système de vision basé sur une caméra USB

ELE4205 - Département de génie électrique
Polytechnique Montréal

2023/10/11— 13:20:55

Table des matières

1 Mise en contexte

Dans le cadre de ce projet, vous allez développer un système intégrant une caméra USB 720p qui sera branchée sur votre Odroid-C2. Des livrables hebdomadaires vous seront transmis à chacune des 6 semaines précédant la septième semaine de projet qui servira à la démonstration finale.

Ce document sera mis à jour régulièrement en fonction de l'évolution du projet.

— Tout votre code développé devra être sur le dépôt GIT qui aura été créé pour votre équipe.

— Votre code devra être documenté Doxygen et tous vos programmes devront pouvoir être compilés à l'aide de CMake.

— La racine de votre dépôt devra contenir un fichier `README.md` d'informations sur votre projet (directives de compilation, configuration, usage, ...).

2 Configuration de l'image

Pour la configuration de l'image, vous allez utiliser l'image fourni dans le répertoire `/export/tmp/4205_1/` et le SDK déjà installé. Comme elle a déjà été installé lors du laboratoire 3, aucune étape n'est requise pour l'utiliser et vous pouvez utiliser le SDK comme indiqué dans l'énoncé du laboratoire 2 (ou 3).

2.1 Options additionelles de l'image fournie

L'image fournie contient les librairies pour le traitement des images et les caméra USB. Il a fallu ajouter à l'image des options par ces lignes dans `conf/local.conf` du répertoire `build-oc2` (certaines de ces options étaient déjà incluses avec `fractale` mais nous les listons pour illustrer les requis explicitement).

```
IMAGE_INSTALL_append = " \
    v4l-utils \
    python-modules \
    opencv \
    python-opencv  \
"
```

Avec ces nouveaux outils, voici un exemple de comment appeler CMake pour la compilation croisée (après un `source` du SDK) :

```
cmake -v ..
```

avec les lignes suivantes dans `CMakeLists.txt` pour détection des paths d'OpenCV :

```
find_package(OpenCV REQUIRED)
if(OpenCV_FOUND)
    include_directories(${OpenCV_INCLUDE_DIRS})
    target_link_libraries(monprogramme ${OpenCV_LIBRARIES})
else(OpenCV_FOUND)
    message(FATAL ERROR "Librarie OpenCV introuvable!")
endif(OpenCV_FOUND)
```

2.2 Vos commits GITet les livrables

Utiliser les mêmes répertoires pour tous les livrables. Pas besoin de changer de noms de fichiers ou répertoires pour les différentes versions. Pendant l'évolution de votre projet (les livrables), lorsqu'un livrable est terminé et prêt, vous pouvez utiliser la commande `git tag` pour y mettre un point de référence, ainsi il est simple de retourner à cette version avec un `git checkout` avec le `tag` correspondant (tutoriel).

Pour GIT, vous avez l'option d'utiliser SmartGit ou l'extension VS Code pour GIT, à votre choix.

2.3 Structure de répertoires suggérée

Votre projet va comprendre deux « applications » : un serveur sur l'Odroid-C2 et un client sur l'ordinateur Linux-Centos 7 du labo `vlsi`. Nous vous suggérons la structure de répertoire suivante avec un fichier d'informations à la racine de votre projet :

```
README.md
/client
/commun
/serveur
```

où `commun` est un répertoire comprenant un fichier d'entête pour les informations utilisées par les deux applications. Les répertoires `client` et `serveur` contiennent leur propre `CMakeLists.txt` et les fichiers sources.

3 Livrables

3.1 Livrable 1

Une caméra USB Logitech c270 supportée par Linux vous est fournie. Cette caméra est supportée par les pilotes UVC.

Votre premier livrable consiste à développer un programme simple qui sauvera 5 secondes de capture vidéo dans un fichier nommé `capture-liv1.avi`. Votre programme doit permettre de sélectionner la résolution des images en fonction de la caméra qui est branchée au port USB.

Logitech c270

La résolution pour cette caméra doit être choisie parmi les suivantes :

```
 176 x 144    160 x 120    320 x 176    320 x 240
 352 x 288    432 x 240    800 x 600    864 x 480
 960 x 544    960 x 720   1184 x 656   1280 x 720
1280 x 960
```

et le format des images saisies sera le MJPG. Le nombre d'images par seconde ne peut pas être configuré avec notre environnement et cette caméra, il faudra donc le détecter (voir `boneCVtiming.cpp` de Molloy par exemple).

Ressources

Comme nous avons inclus Video4Linux et OpenCV dans notre image, ces outils sont disponibles pour votre développement. Comme dans un projet réel, vous allez devoir rechercher les informations pour intégrer votre caméra.

Sous Linux lorsque l'on branche un périphérique, les messages concernant l'initialisation de celui-ci pourront être consultés à l'aide de la commande `dmesg`. De plus la commande, `lsusb` permet de lister les périphériques USB connectés au système Linux. Par exemple :

```
# lsusb
```

Conseils

— Avant d'écrire votre programme qui sauve le fichier vidéo, vous devriez modifier le `boneCVtiming.cpp` de Molloy en faisant une boucle sur les n résolutions disponibles pour votre caméra pour avoir les *timings* pour les différentes résolutions.

Lire 2 *frames* après un changement de résolution avant la boucle de *timing* pour mesurer seulement le temps entres les *frames* afin de ne pas inclure le temps d'initialisation d'une nouvelle résolution.

— Utilisez un tableau de structures de données ou objets C++ (`int resX, int ResY, double fps`) qui sera utilisé pour les caractéristiques de la caméra (le champs `fps` aura été déterminé avec votre programme de *timings*).

— La caméra `c270` a l'identifiant USB suivant `ID 046d:0825`.

— Pensez programmation modulaire, car le livrable 2 va utiliser les fonctions de votre livrable 1.

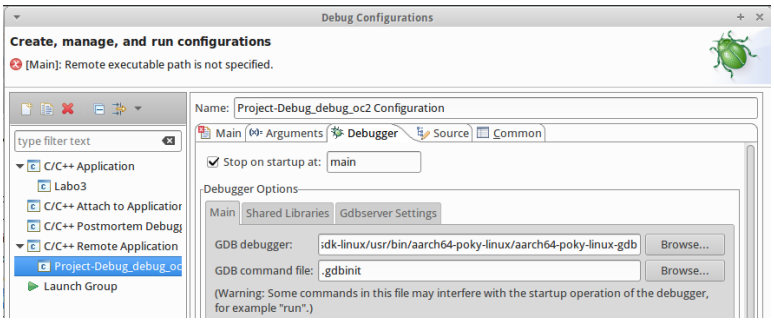
Le démarrage de GDB en *remote* dans Eclipse peut être lent à cause du transmission des librairies dynamiques d'OpenCV et autres du OC2 vers Eclipse.

Voici la solution à ce problème :

1. créer un fichier `gdboc2` dans le HOME de votre compte Linux (`/users/Cours/ele4205/nn`) qui contient la ligne suivante avec un **return** à la fin : important.

```
set sysroot /usr/local/opt/poky/2.1.3/sysroots/aarch64-poky-linux/
```

2. ajouter ce fichier à la configuration de remote debugging : *GDB command file* (image ci-dessous) à la place du `.gdbint` avec le Browse et Apply. Ça devrait être plus rapide après cela.



Dans VS Code, on peut faire l'équivalent avec l'ajout du `sysroot` dans la section `"setupCommands"` du fichier `launch.json` pour avoir :

```
"setupCommands": [{
    "description": "Enable pretty-printing for gdb",
    "text": "-enable-pretty-printing",
    "ignoreFailures": true
},
{
    "description": "Set sysroot ",
```

```
        "text": "set sysroot /usr/local/opt/poky/2.1.3/sysroots/aarch64-poky-linux/"
    }
    ],
```

3.2 Livrable 2

Maintenant, que vous savez comment *capturer* une image de la caméra USB, au lieu de la sauver sur le « disque » local, nous allons la transmettre via TCP/IP vers l'ordinateur et nous allons l'afficher avec OpenCV. Nous allons donc avoir deux applications :

— un « serveur d'images » sur le Odroid-C2 en compilation croisée et

— un programme « client » sur le poste Centos 7 du laboratoire L–5904. La compilation avec OpenCV se fera par un CMake similaire à

```
% cmake -DCMAKE_BUILD_TYPE=Release  ..
```

Voici une brève description de comment le système fonctionne :

— Le serveur démarre et ouvre un socket TCP sur le port 4099. Et attend une requête d'un client (`listen` puis `accept`).

— Sur une requête du client (`connect`), le serveur démarre la caméra, capture une image et la transfert au client.

— Le client reçoit l'image, l'affiche (`cv::imshow`), attend une entrée pendant 30 ms (`waitKey`). Si la touche est `ESC`, le client retourne `QUIT` au serveur, ferme le *socket* et termine, sinon il retourne `OK` et attend nouvelle image.

Le `QUIT` et le `OK` sont transmis dans un bit d'un entier non-signé `uint32_t`. Dans les livrables suivants, nous allons utiliser d'autres bits de ce `uint32_t` pour transmettre des informations supplémentaires (cet entier agira comme un registre de contrôle). On peut utiliser des directives (`#define`) dans un fichier d'entête pour définir ces différents « états ».

```
#include  <stdint.h>
...

#define ELE4205_OK       0b1

...

    uint32_t messages;
```

— Le serveur arrête la caméra et ferme le *socket* sur `QUIT`, sinon sur `OK`, il capture une nouvelle image puis l'envoie au client et le processus recommence.

— En tout temps, le client ou le serveur termine en cas d'erreur sur les *sockets*.

C'est un système client/serveur similaire à ce que l'on retrouve dans des livres ou tutoriels sur la programmation des *sockets*.

Mais avant d'envoyer l'image, nous allons la compresser avec `cv::imencode` et du côté client nous allons la décompresser avec `cv::imdecode`. Pour plus de détails sur ces fonctions et OpenCV, voir la documentation en-ligne (liens sur le site moodle du cours). Doc 2.4.5 et 3.1.0 de `encode/decode`.

Deux formats de compression pourront être utilisés (PNG et JPEG).

Pour le débogage sous VS Code, vous allez avoir besoin de deux terminaux à partir de la racine de votre projet :

— un en `bash` avec compilation croisée

```
% cd serveur
% bash
$ source /usr/local/opt/poky/2.1.3/environment-setup-aarch64-poky-linux
$ code --extensions-dir /export/tmp/${USER}/vscode-ext .
```

— un en `tcsh` avec compilation en natif

```
% cd client
% code --extensions-dir /export/tmp/${USER}/vscode-ext .
```

Vous obtiendrez ainsi deux fenêtres VS Code disponibles pour votre travail.

3.3 Livrable 3

Dans le cadre du livrable 3, vous allez choisir 4 résolutions vidéo qui ne vous causent pas de problème. En plus du `OK` pour continuer, le client va retourner la résolution : un choix entre `RES01`, `RES02`, `RES03` et `RES04` (à mettre dans les bits de votre choix dans `messages`). Ceci peut demander un changement de résolution. Le programme client (à l'aide d'un « menu » de votre choix) permet à l'usager de demander une nouvelle résolution.

Choisir des résolutions de « basses » à « hautes ».

De plus, l'usager peu choisir à partir du même menu le type de compression (PNG ou JPEG à mettre dans un bit de votre choix : `FORMAT_PNG` et `FORMAT_JPEG`).

3.4 Livrable 4

Avant de retourner une image, le serveur va envoyer un `uint32_t` contenant un message. Il y a trois messages possibles :

READY : il y a de la lumière et on peut transférer une image. Le client demande l'image et l'affiche comme dans le livrable 3 ;

IDOWN : il n'y a pas de lumière et pas d'image disponible, le client ne demande pas d'image ;

PUSHB : il y a de la lumière, le bouton pression est enfoncé et une image est disponible, le client fait un `fork()` : le *parent process* affiche l'image comme dans le livrable 3 et le *child process* sauve l'image (format PNG) sur disque avec un numéro séquentiel puis termine.

Sur le Odroid-C2, on utilise l'ADC pour lire la tension du circuit du « capteur de lumière » et la valeur 0 ou 1 en logique inverse du bouton pression (attention sur front descendant du bouton, une seule image par pression, et on vérifie l'état du bouton après avoir vérifié la luminosité (au même rythme que les « frames »).

Le schéma de la figure 1 illustre le branchement des composantes.

La tension du circuit avec la resistance photosensible peut être lue à partir du fichier `ch0` dans le répertoire

```
/sys/class/saradc
```

Le bouton connecté avec la pin `#101` sur le *breadboard* (entrée 0/1, GPIO 228) est configuré avec

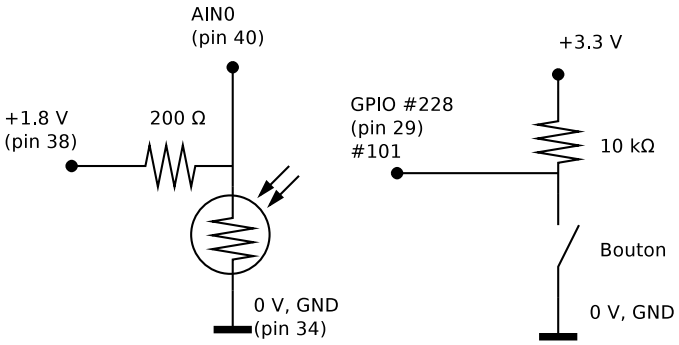
```
echo 228 > /sys/class/gpio/export
echo in > /sys/class/gpio/gpio228/direction
```

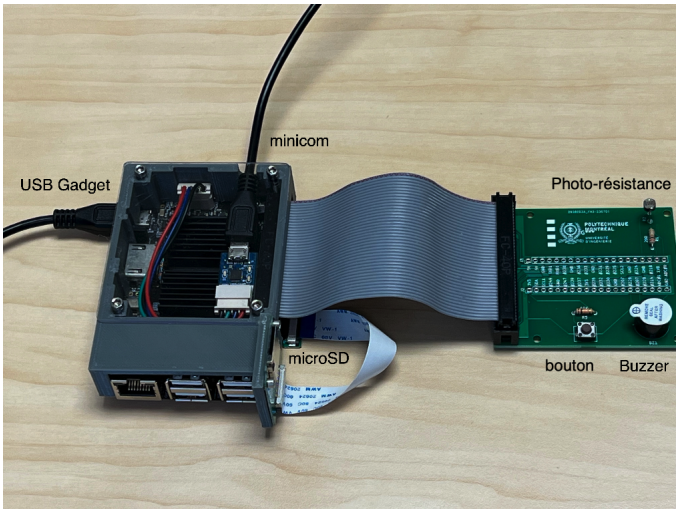
puis on le lit à partir du fichier

```
/sys/class/gpio/gpio228/value
```

3.5 Livrable 5

Dans ce livrable, vous allez traiter l'image capturée lors de la pression du bouton. Vous allez la traiter avec OpenCV pour faire une détection de visages. Ce traitement doit se faire dans le `fork()` qui sauvegardait l'image sur le disque. Une recherche sur Google avec les mots clé `opencv 2.4 face detection` (ou autres) devrait vous permettre de trouver de la documentation et des exemples similaires à votre problème. En cas de succès, vous devez « encadrer » tous les visages trouvées et sauver cette image modifiée.





ODROID-C2 40pin Layout

Power Pin
Special Function
GPIO/Special Function

WiringPi GPIO#	Export GPIO#	ODROID-C2 PIN	Label	HEADER		Label	ODROID-C2 PIN	Export GPIO#	WiringPi GPIO#
			3V3	1	2	5V0			
	205	I2CA_SDA	SDA1	3	4	5V0			
	206	I2CA_SCL	SCL1	5	6	GND			
7	249	GPIOX.BIT21	#249	7	8	TXD1	TXD_B	113	
			GND	9	10	RXD1	RXD_B	114	
0	247	GPIOX.BIT19	#247	11	12	#238	GPIOY.BIT10	238	1
2	239	GPIOX.BIT11	#239	13	14	GND			
3	237	GPIOX.BIT9	#237	15	16	#236	GPIOX.BIT8	236	4
			3V3	17	18	#233	GPIOX.BIT5	233	5
12	235	GPIOX.BIT7	#235	19	20	GND			
13	232	GPIOX.BIT4	#232	21	22	#231	GPIOX.BIT3	231	6
14	230	GPIOX.BIT2	#230	23	24	#229	GPIOX.BIT1	229	10
			GND	25	26	#225	GPIOY.BIT14	225	11
	207	I2CB_SDA	SDA2	27	28	SCL2	I2CB_SCL	77	
21	228	GPIOX.BIT0	#228	29	30	GND			
22	219	GPIOY.BIT8	#219	31	32	#224	GPIOY.BIT13	224	26
23	234	GPIOX.BIT6	#234	33	34	GND			
24	214	GPIOY.BIT3	#214	35	36	#218	GPIOY.BIT7	218	27
		ADC.AIN1	AIN1	37	38	1V8	1V8		
			GND	39	40	AIN0	ADC.AIN0		

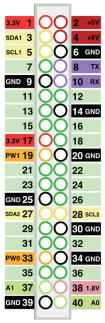


FIGURE 1 – Circuits bouton et cellule photosensible

3.6 Livrable 6

Dans ce dernier livrable, vous allez ajouter 2 options dans le menu du client : apprentissage et reconnaissance.

Apprentissage

Dans ce mode, il doit y avoir un seul visage dans l'image. Pour cette image, on demande le nom de la personne ou on sélectionne un nom pour lequel on a déjà une image. On peut donc faire l'apprentissage pour plusieurs personnes.

Reconnaissance

Dans ce mode, il peut y avoir un ou plusieurs visages dans l'image et l'on utilise la « base de données » de visages du mode précédent pour reconnaitre la ou les visages.

Ces traitements se font toujours dans le `fork()` qui sauvegardait l'image sur le disque. Une recherche sur Google avec les mots clé `opencv 2.4 face recognition` (ou autres) devrait vous permettre de trouver de la documentation et des exemples similaires à votre problème. Vous pouvez adapter du code d'un projet existant en citant bien votre source dans les commentaires de votre code et en respectant le *copyright*.

Dans ce dernier livrable, vous allez aussi ajouter un programme sur l'Odroid-C2. Ce programme sera démarré lorsque le bouton est pressé (par un `fork()` suivi d'un `exec()`). Ce programme et ouvre un socket TCP sur le port 4100 et attend une requête d'un client (`listen` puis `accept`).

Le fork du client qui analyse l'image, après avoir fait le travail du livrable, se connecte au « nouveau » programme (en client sur le port 4100), envoie un des trois messages suivants encodé dans un entier :

LEARN : en mode apprentissage des visages ;

KNOWN : un visage reconnu et authentifié ;

UNKNOW : aucun visage connu ou pas de visage.

Le « nouveau » programme va recevoir cet entier va faire l'une des trois choses suivantes selon le cas

LEARN : rien ;

KNOWN : active le buzzer 3 secondes à la fréquence de 440hz ;

UNKNOW : trois Bip de 1 seconde à 660hz intercalés de silences de 0.5 seconde.

Après avoir terminé le traitement du message, le programme « serveur de buzzer » se termine.

Indices

- Capturer des images du visage seulement (il faudra faire un « crop » pour avoir seulement la zone d'intérêt) ;
- Il faut environ 20 captures de visage par personne de l'équipe ;
- Toutes les images ainsi capturées doivent être de la même taille ;
- Lister les images pour l'entraînement avec un code numérique pour la personne voulue ;
- Trouver une façon d'afficher la détection sur l'image.

4 Évaluation

Au début des séances spécifiées, le livrable de la semaine précédente sera évalué. Pour obtenir tous vos points, le livrable devra être complet, documenté sur GIT dans votre *repo* BitBucket.

Voici les modalités d'évaluation des livrables (voir la dernière page du plan de cours pour les dates des différentes sections) :

- Livrable i terminé au début de la séance du livrable $i + 1 = 100\%$
- Livrable i terminé à la fin de la séance du livrable $i + 1 = 75\%$
- Livrable i terminé au début de la séance du livrable $i + 2 = 50\%$

Barème final

Item	**Points**
Livrables (6x2.5/liv.)	15
`README.md`	1
GIT	1
Doxygen	1
Code « propre » et modulaire	2
Total	**20**

Votre projet complet doit être dans BitBucket (*commit* final) avant la fin de votre dernière séance de laboratoire (séance d'évaluation finale).

5 Références

Les liens suivants fournissent des instructions et des exemples d'utilisation pertinents pour vos livrables.

OpenCV Tutorial C++ :

— OpenCV Tutorial C++

Site Beaglebone de Derek Molloy :

— Beaglebone: Video Capture and Image Processing on Embedded Linux using OpenCV

Video for linux 2 (V4L2) : utilitaire de contrôle de paramètres de webcam :

— Beaglebone Images, Video and OpenCV

— Video for Linux Two API Specification

TCP/IP :

— TCP/IP Sockets in C (Second Edition) disponible en-ligne à Polytechnique