

Projet - Système de vision basé sur une caméra USB

ELE4205 - Département de génie électrique
Polytechnique Montréal

2019/10/28— 19:20:56

Table des matières

1	Mise en contexte	2
2	Configuration du noyau	2
2.1	Compilation d'une nouvelle image	2
2.2	Mise à jour carte SD	3
3	Livrables	4
3.1	Livrable 1	4
3.2	Livrable 2	5
3.3	Livrable 3	6
3.4	Livrable 4	6
3.5	Livrable 5	7
3.6	Livrable 6	7
4	Présentation finale	9
5	Évaluation	9
6	Références	10

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 du noyau

Pour la configuration du noyau, vous avez deux options :

1. Utiliser l'image et le SDK fournis dans le répertoire `/export/tmp/4205_1/` et passer à la section [2.2](#);
2. Compiler votre image comme spécifié à la section [2.1](#).

2.1 Compilation d'une nouvelle image

Pour réaliser ce projet, vous allez devoir modifier votre image pour le Odroid-C2 pour s'assurer que toutes les composantes requises sont disponibles.

- L'image doit contenir les bibliothèques pour le traitement des images et les caméra USB. Il faut donc ajouter à l'image des options par ces lignes dans `conf/local.conf` de votre `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 \
"
```

Dans votre dossier `poky`,

```
$ source oe-init-build-env build-oc2
```

Compilez maintenant la distribution et générer votre SDK. Cela devrait prendre quelques minutes.

```
$ umask a+rx u+rwX
$ nice bitbake core-image-base
$ nice bitbake core-image-base -c populate_sdk
$ sh ./tmp/deploy/sdk/poky-glibc-x86_64-core-image-base-aarch64-toolchain-2.1.3.sh
Poky (Yocto Project Reference Distro) SDK installer version 2.1.3
=====
Enter target directory for SDK (default: /opt/poky/2.1.3): /export/tmp/4205_nn/opt/poky
The directory "/export/tmp/4205_nn/opt/poky" already contains a SDK for this architecture.
If you continue, existing files will be overwritten! Proceed[y/N]? Y
Extracting SDK.....done
Setting it up...done
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to source the environment setup s
$ . /export/tmp/4205_nn/opt/poky/environment-setup-aarch64-poky-linux
```

2.2 Mise à jour carte SD

Quelque soit la méthode choisie, mettez à jour votre carte SD. Démarrer votre Odroid-C2 et vérifier la connectivité série et SSH (section 7.1, laboratoire 1, page 19, édition du fichier `/etc/network/interfaces`).

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 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)
```

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 **MJPEG**. 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 | grep Logitech
```

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 entre 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 champ `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.

3.2 Livrable 2

Maintenant, que vous savez comment *capturer* une image de la caméra USB, au lieu de la sauvegarder 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 Scientific Linux. 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 transfère 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*. La difficulté réside dans le transfert d’un objet `cv::Mat` dont la dimension et le type d’éléments ne sont pas connus à priori. Un objet `cv::Mat` a une *entête de dimension fixe*. Cette entête contient, entre autres, les informations concernant le format, la dimension de l’image de même qu’un pointeur vers les données de l’image. Comme les images ont différents formats, le type des éléments du vecteur contenant les « pixels » est spécifié dans l’entête (voir [aussi](#)).

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.

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

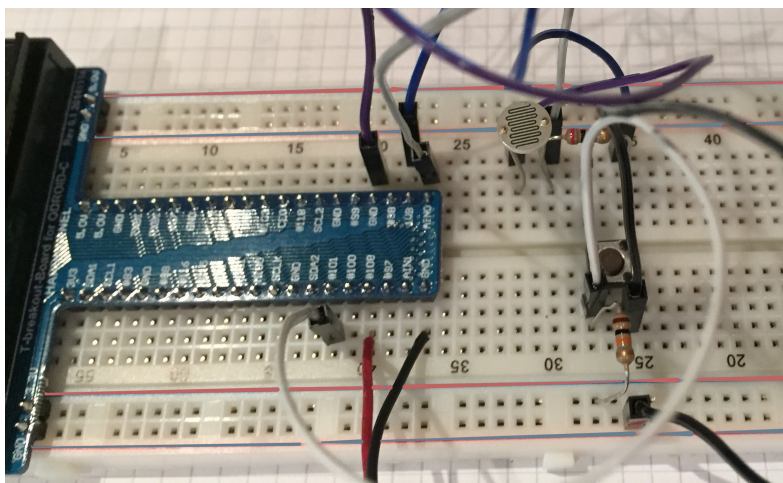
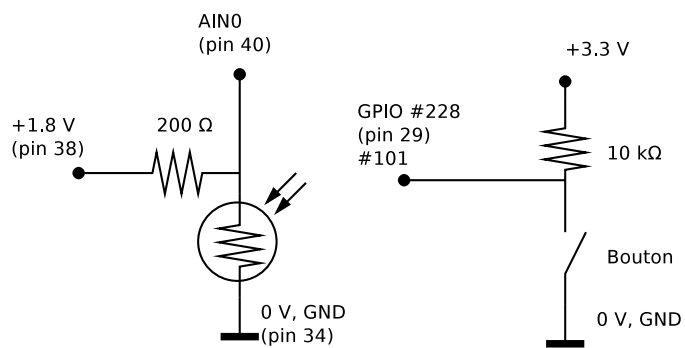
Dans ce livrable, vous allez traiter l'image capturée lors de la pression du bouton. Vous allez la traiter avec **OpenCV** pour faire une détection de visages. Ce traitement doit se faire dans le `fork()` qui sauvegardait l'image sur le disque. Une recherche sur **Google** avec les mots clé **opencv 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 trouvés et sauver cette image modifiée.

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.



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	SV0				
	205	I2CA_SDA	SDA1	3	4	SV0				
	206	I2CA_SCL	SCL1	5	6	GND				
7	249	GPIOX.BIT21	#249	7	8	TXD1		113		
			GND	9	10	RXD1		114		
0	247	GPIOX.BIT19	#247	11	12	GND	GPIOY.BIT10	238	1	
2	239	GPIOX.BIT11	#239	13	14	GND				
3	237	GPIOX.BIT9	#237	15	16	GND	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.BIT6	#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

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 reconnaître 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 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*.

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 Présentation finale

À la dernière séance de laboratoire (26 ou 28 novembre 2019), une version finale avec documentation complète doit être démontrée. Cette version (le dernier `commit` avant 2019-11-26-15h35 ou 2019-11-28-16h35) doit être complètement fonctionnelle et ne doit plus avoir de messages de débogage. Seuls les messages d'interaction avec l'utilisateur ou de terminaison de processus non-prévue (erreurs) sont permis.

5 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**.

Barème final

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

6 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](#)
- [Streaming Video using RTP on the Beaglebone Black](#)

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
- [C++ OpenCV image sending through socket](#)