Institut Supérieur de l'Électronique et du Numérique
Tél. : +33 (0)2.98.03.84.00
Fax : +33 (0)2.98.03.84.10
20, rue Cuirassé Bretagne
CS 42807 - 29228 BREST Cedex 2 - **FRANCE**

N4

School Year 2007/2008
Training period

Embedding image processing algorithms into a sensor node for ecological applications

From June 5th, to September 19th

At

Center for Embedded Networked Sensing (CENS)

3563 Boelter Hall
University of California (UCLA)
CA 90015-1596 Los Angeles
+01 310-206-2476

Supervisors: Deborah ESTRIN, Center for Embedded Networked Sensing, 310-206-3923
Teresa KO, Center for Embedded Networked Sensing

Referring teacher: Mrs. Dominique Maratray

SIMOES Vincent

ACKNOWLEDGMENT

First of all, I would like to thank Deborah ESTRIN who gave me the opportunity to realize this internship at the University of California in Los Angeles and Tristan MONTIER, who gave me the contacts to make this travel possible. I would also like to thank David AVERY and Dennis URIE for their help on the administrative side, especially to obtain the visa.

Of course, my thoughts also go to my team of work and specially Teresa KO, Joey DEGGES and Mike WILSON for allowing me to work on such an interesting project, for their assistance when I needed, their time to answer to my questions and their understanding about my bad control of English. I learned a lot with them on very different topics. Thus, I have been able to improve significantly my knowledge of the use of UNIX system. I have also been involved in the redaction of a paper for the CENS Annual Research Review, which is an excellent thing to discover the research's world. Another thank for Ryan NEWTON from MIT (Massachusetts Institute of Technology) for his help with the use of Wavescript.

Then, I would like to thank Dominique MARATRAY for her assistance during the internship and for her help to find scholarship. In this respect, I must also thank the Rotary Club of Brest and the “Conseil Régional de Bretagne” for according me financial helps.

Finally, I would like to thank all the people working in the CENS for their reception and their friendliness. Thanks to them, my time in the laboratory has been a real pleasure and they made me feel good by including me in the team very easily.

TABLE OF CONTENTS

ABSTRACT

The use of imagers as sensors becomes increasingly important as researchers work to collect data from more remote and fragile environments. In regions that are too remote for direct observation, researchers frequently turn to digital cameras as image sensors. But, digital cameras can only work as long as their batteries last and many of the images collected provide little or no information. Thus, it is necessary to find different ways to increase considerably the performances.

In this context, this document propose an interesting solution through the creation of a Portable Imaging Platform (PIP) allowing to detect birds in a natural environment and to have a quick view on the results. This platform is an experimental platform built to ease the development of embedded vision systems through a simple deployment procedure, relatively high speed image capture and storage, and available on-board computational power. We will be focused on the capture of images using the Video for Linux 2 interface since this is the part I have worked on. To realize this system, a camera linked to a card with an ARM processor has been used. Furthermore, a PHP server has been installed on the card to allow an easy communication between the user and the platform. Finally, to increase the flexibility of the system, the software part has been migrated under Wavescript, which is a ML-type language.

Keywords: capture of images, Video for Linux 2 (V4L2), Wavescript, embedded systems, ARM processor.

RESUME

L'utilisation de caméras comme capteurs devient de plus en plus fréquent pour les chercheurs travaillant à collecter des données environnementales. Dans les régions qui sont trop retirées pour une observation directe, les chercheurs utilisent fréquemment de simples caméras numériques comme capteurs. Cependant, ce type d'appareil peut seulement travailler le temps que la batterie le permet et la plupart des images collectées donne peu ou pas d'informations. Il convient alors de trouver différentes possibilités pour améliorer considérablement les performances.

Dans ce contexte, ce document propose une solution intéressante à travers la création d'une plateforme portable permettant de détecter des oiseaux dans la nature et d'avoir une vue rapide sur les résultats. Cette plateforme est faite pour faciliter le développement de systèmes embarqués du même type grâce à un déploiement simple, une rapidité pour capturer, traiter et stocker les images relativement importante et avec des possibilités de traitements multiples. L'accent sera donné sur la partie capture des images utilisant l'interface de Video for Linux 2 puisque c'est celle sur laquelle j'ai principalement travaillé. Pour réaliser un tel système, une caméra reliée à une carte équipée d'un processeur de type ARM a été utilisée. De plus, un server PHP a été installé sur la carte afin de permettre une communication facile entre l'utilisateur et la plateforme. Enfin pour des raisons de flexibilité, la partie software a été retranscrite avec Wavescript qui est un langage de type ML.

Mots Clés : capture d'images, Video for Linux 2 (V4L2), Wavescript, systèmes embarqués, processeur ARM.

GLOSSARY

API: Application Programming Interface

ARM: Advanced RISC Machine, a computer processor architecture

BMP: Bitmap, an image file format

CCD: Charged Couplet Device, an electronic light sensor used in digital cameras

CENS: Center for Embedded Networked Sensing

CPU: Central Processing Unit

Fps: Frame per second

GPIO: General Purpose Input/Output, connection to interface with the outside world

I2C: Inter Integrated Circuit Bus, a serial computer bus

LCD: Liquid Cristal Display

MicroSD: Micro Secure Digital, a format for removable flash memory cards

MIT: Massachusetts Institute of Technology

ML: Metalanguage, a programming language

NSF: National Science Foundation

PIP: Portable Imaging Platform, name of the platform developed during the internship

PhD: Doctor of Philosophy

PHP: Personal Home Page, a computer scripting language

RGB: Red Green Blue

SD: Secure Digital, a format for removable flash memory cards

UCLA: University of California, Los Angeles

USB: Universal Serial Bus

V4L2: Video for Linux 2, a library to communicate with a camera under UNIX system

XGA: Extended Graphics Array, a display standard

LIST OF FIGURES

INTRODUCTION

My training period took place in the Center for Embedded Networked Sensing (CENS), one of the research centers of the University of California in Los Angeles. During approximately four months, from June 5th to September 19th, I worked in a team of seven people for most students (PhD, graduate and undergraduate students). Teresa Ko was in charge of the project by planning the meetings and checking the evolution of the work.

Nowadays, manually monitoring the environment for the presence of animals is limited by the amount of human time that can be devoted to it. Automatically detecting and segmenting out animals from a natural scene would ease the burden of biologists and allow observations to scale beyond what a biologist has time to view. Moreover, embedded video surveillance and monitoring applications generate a significant amount of data and are restricted by the challenge of storage limitations. These limitations become more prevalent in biological sensing when an inability for network communication arises.

Thus, the main goal of the project was to create a flexible and easy-to-use Portable Imaging Platform (PIP) to detect birds in a natural environment. A camera is plugged on a card equipped with an ARM processor more precisely a Gumstix and a large storage space. To have a straightforward communication between the user and the card, we use a PHP server installed on the processor. Each person of the team worked on a part of the project. My main task was to create a video and I/O interface to capture images from the camera using Video for Linux 2. After that, to improve the performances of the overall system, I have worked on Wavescript, which may be used for the software part and gives good results on other systems.

In this report, I will present all the steps of my work. First of all, after a quick presentation of UCLA, CENS and of the overall project, the first part will give a piece of information about the hardware we used on our platform. Afterward, I will explain how to use Video for Linux and how works the video interface from the initialization to the recording of the data. The second part will expose the results obtained with the interface and consider the improvements made to add the support of several cameras and drivers. Finally, I will explain the use of Wavescript and the benefits of such a language.

1 – PRESENTATION

1.1) THE UNIVERSITY OF CALIFORNIA, LOS ANGELES

Figure 1: University of California, Los Angeles

The University of California, Los Angeles (generally known as UCLA) is a public research university located in Westwood, Los Angeles. Established as a branch of the state university in 1919, it is the second oldest general-purpose campus in the University of California system. When UCLA opened, it was composed of four buildings. Today, the campus includes 163 buildings across 1.7 km² and is in a perpetual expansion. It has the largest enrollment of any university in the state. Indeed, with 55,401 applicants for 12,755 accepted for fall 2008 (23%), UCLA has more applicants than any other university in the United States and is the second most selective public university in the country.

In the world, UCLA is one of the most highly regarded schools. Thus, in 2008, it was ranked 11th in North America and 13th in the world by the annual list, *Top 500 World Universities*, published by the Institute of Higher Education at Shanghai Jiao Tong University, China.

The University has a significant impact in the Los Angeles Economy. It is the fourth largest employer in the county, and the seventh largest in the region with more than 30000 employers. In 2005-2006, the university had an operating budget of $3.6 billion. UCLA took the second spot among all universities and the top spot among public universities, for research spending in the sciences and engineering with $773 million spent in more than 300 research centers during the fiscal year 2004, according to a 2006 report by the National Science Foundation.

1.2) CENTER FOR EMBEDDED NETWORKED SENSING (CENS)

UCLA's Center for Embedded Networked Sensing (CENS) is a major research enterprise focused on developing wireless sensing systems and applying this revolutionary technology to critical scientific and societal pursuits. In the same way that the development of the Internet transformed our ability to communicate, the ever decreasing size and cost of computing components is setting the stage for detection, processing, and communication technology to be embedded throughout the physical world. Thereby, these evolutions allow a deeper understanding of the natural environment and, ultimately, enhance our ability to design and control complex systems.

By investigating fundamental properties of embedded networked sensing systems, developing new technologies, and exploring novel scientific and educational applications, CENS is a world leader in unleashing the tremendous potential these systems hold.

The center is a multidisciplinary collaboration among faculty, staff, and students from a wide spectrum of fields including Computer Science, Electrical Engineering, Civil and Environmental Engineering, Biology, Statistics, Education and Information Sciences, Urban Planning, Theater, Film, and Television. It is one of six National Science Foundation and Technology Centers established in 2002, and is projected to receive $40 million in core funding from the NSF over 10 years. CENS has successfully competed for substantial supplementary funding from both the NSF and other federal agencies to support new research activities generated within the center. A truly interdisciplinary venture, CENS has also received institutional funding to support the activities of the more than 25 UCLA faculty, 20 graduate and 65 undergraduate students from disciplines across campus, as well as faculty and students from UC Merced, UC Riverside, the University of Southern California, California State University, Los Angeles, the James Reserve, the Jet Propulsion Laboratory and Caltech.

CENS technology research draws on a diverse set of researchers within engineering, from distributed system design, to distributed robotics, to wireless communications, to signal processing and low-power multi-modal sensor-technology design. The physically-embedded nature of this technology calls for significant experimentation and exploration in the context of targeted application domains in order to identify the true challenges and opportunities.

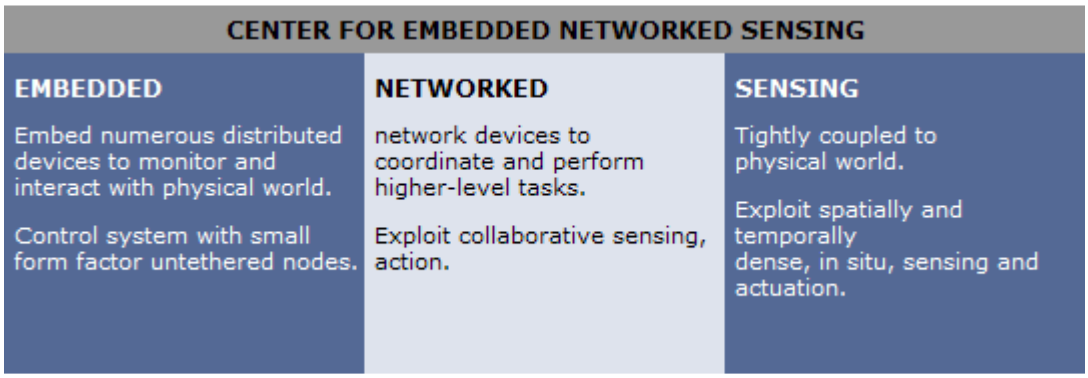


Figure 2: What is Embedded Networked Sensing (ENS)?

The center's current research portfolio encompasses projects across nine technology and applications areas, examples of which include:

- Development and deployment of new measurement tools and techniques to identify the sources and fates of chemical and biological pollutants in natural, urban, and agricultural watersheds and coastal zones.
- Developing cameras and image analysis approaches that assist scientists in making biological observations. Together the camera and analysis systems comprise a new type of biosensor that takes measurements otherwise unobservable to humans.
- Harnessing the technological power of mobile phones and the ubiquitous wireless infrastructure for applications in areas as diverse as public health, environmental protection, urban planning, and cultural expression, each of which is influenced by independent personal behaviors adding up in space in time.

The following figure shows a part of the organization chart concerning my position in the laboratory. Of course, this diagram is simplified compared to the overall chart of CENS.

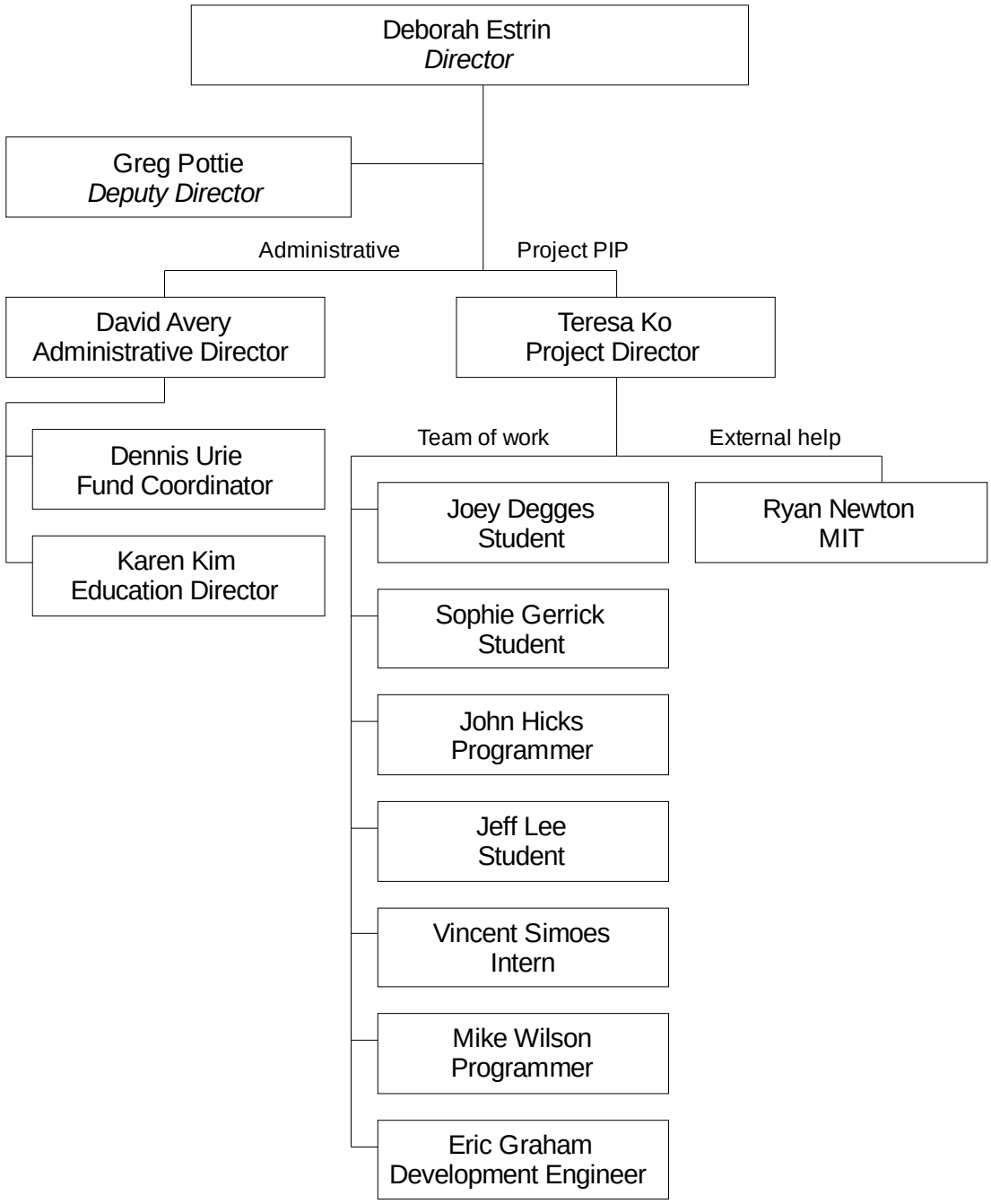


Figure 3: Organization Chart

1.3) PRESENTATION OF THE OVERALL PROJECT

The Portable Imaging Platform (PIP) consists of an embedded node platform and a software platform that supports in-situ image processing algorithms. Figure 4 is a diagram of the system architecture, a framework that allows image processing algorithms to be easily written in C/C++ while taking advantage of a simple live video and image I/O interface. The results of any computation can be archived on the device as well as uploaded to Google's Picasa Web Albums for long-term persistent storage and visualization. A user has the ability to control all aspects of the system through an intuitive web interface which can be accessed

either onsite or remotely via a wireless or wired network. Through the interface, users are able to start and stop data collection, view the current video stream. Moreover, the web interface is used to remotely reconfigure any recording parameters such as white balance and exposure. The current view can be streamed through the web interface before recording to ensure that the camera is properly aligned. When all settings are properly selected, data collection can begin. The benefit of constructing the user interface through a web browser is that there is no need to download and configure any additional software. Each user can simply plug in the Gumstix and then easily navigate to the web interface for direct setup of the sensor. The hardware platform used for the platform is easily reproducible and highly available through standard distribution outlets. All of the software is open source and can be obtained without fee or license. Any minor changes that we have made to existing open source projects have been well-documented so that anyone can replicate the steps taken to build the system.

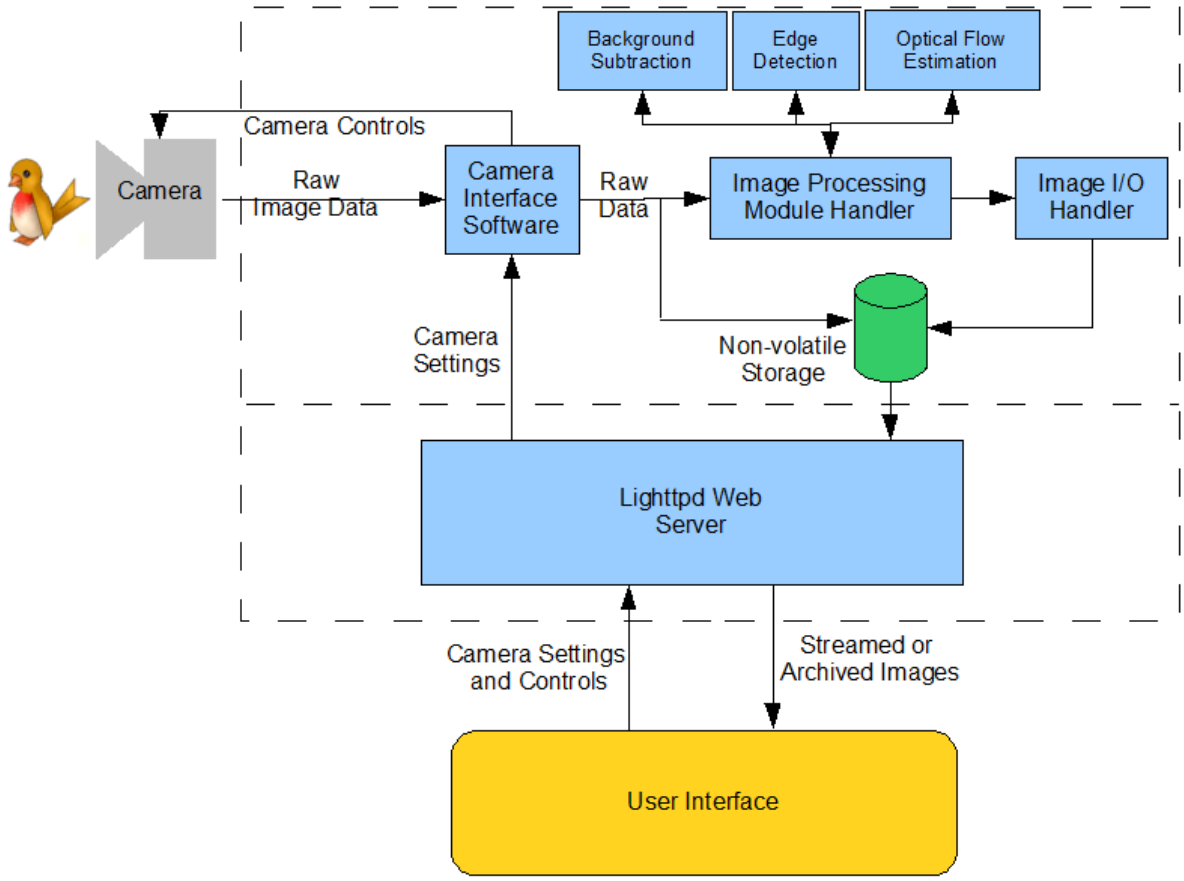


Figure 4: Overall system diagram

Each person of the team worked on a part of the project. I was in charge of the camera interface software and of the conversion of the software part in Wavescript. Joey Degges, Jeff Lee and Sophie Gerrick worked on the web interface with the Lighttpd Web Server. The image processing part (background subtraction, edge detection, etc) has been made by Joey Degges, Mike Wilson and Teresa Ko. Jeff Lee and Sophie Gerrick were only here during July and August.

2 – HARDWARE

2.1) GENERALITIES

The choice of all the components has been made before I arrived in the laboratory. For most of them, they have been chosen because of their performances on systems like ours, for their good support on Internet and because they are widely used by the laboratories working with the CENS. Thus, it was easy to have a quick support when needed. A small Linux distribution is available on the motherboard to simply add drivers or softwares.

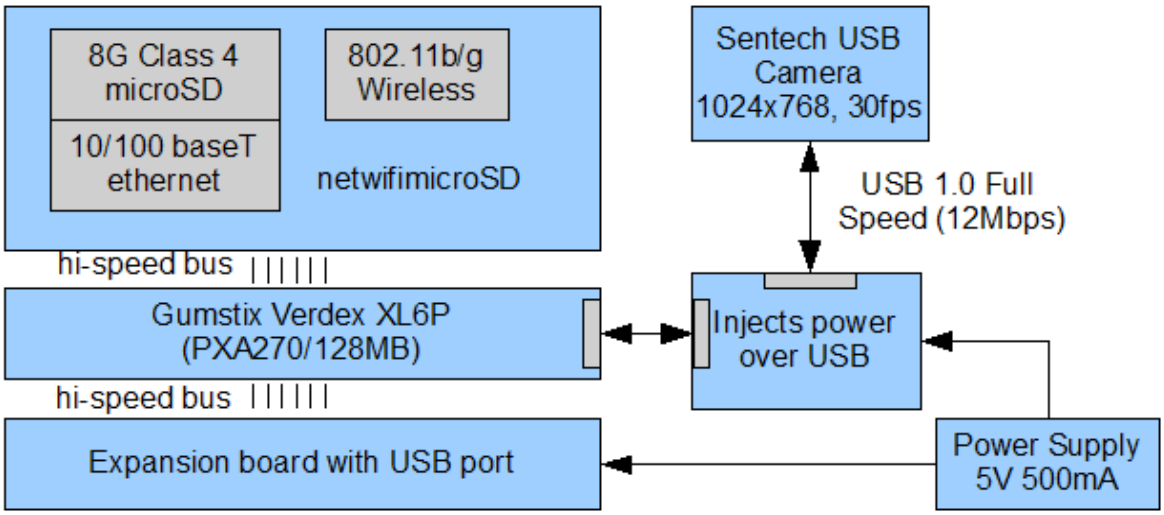


Figure 5: Hardware system diagram

As shown in the Figure 4, there are two important parts, the camera and the motherboard named Gumstix hosting an ARM processor, the storage and the different ports useful to communicate (USB port, Ethernet, Wireless communication). Gumstix are ideal embedded devices for such a platform because they have low power requirements and are about the size of a stick of chewing gum. We use a small card to inject power over USB since the motherboard doesn't provide it. Furthermore, each node is constructed from Commercial off-the-shelf hardware that anyone with basic technical skills can purchase and assemble.

2.2) MOTHERBOARD

The core of the Portable Imaging Platform is a Gumstix Verdex XL6P motherboard with the following specifications:

Feature	Description
Processor	Marvell PXA270 XScale
Speed	600 Mhz
Memory RAM	128 MB SDRAM
Flash	32 MB

Figure 6: Specifications of the Gumstix Verdex X6LP

As it is crucial that the system have the processing capabilities to carry out various image processing tasks, the choice of such motherboard is very interesting. Moreover, the ARM processors are widely used in embedded designs thanks principally to their power saving features.

There are also two expansion slots on the motherboard which hosts Gumstix breakout-vx and netwifimicroSD-FCC expansion boards. The breakout-vx board provides a USB mini-B port with USB 1.0 Full Speed (12Mbps) host signals as well as I2C, GPIO, LCD, and serial ports. The netwifimicroSD board provides 10/100 baseT Ethernet, 802.11b/g wireless communication, and microSD storage capabilities. Thus, we will use an 8GB microSD card to store the images.

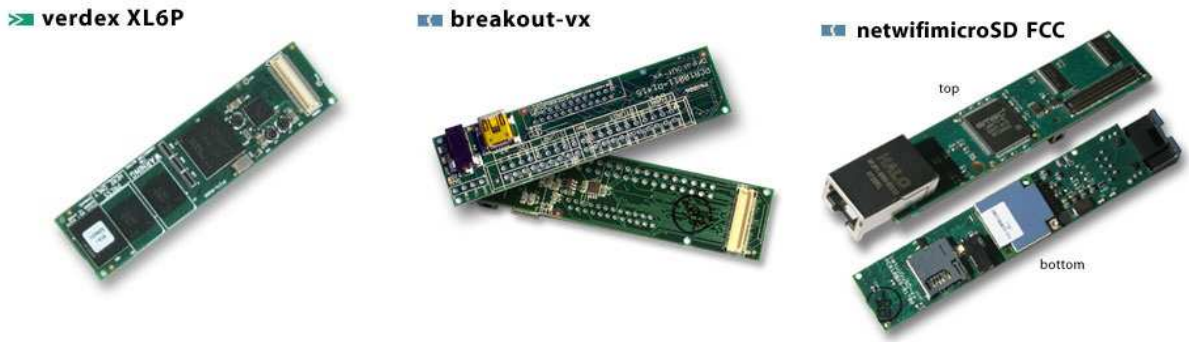


Figure 7: The Gumstix verdex XL6P and expansion boards

2.3) CAMERA

Figure 8: Sentech Camera STC-C83USB-A

A Sentech USB camera (STC-C83USB-A) is attached to the Gumstix via USB and is powered over USB through a custom USB power injector. This camera implements the Video for Linux 2 API thanks to the Sentech's driver, making it compatible with any Linux operating system. It also provides many different controls such as brightness, saturation, and white balance as well as a progressive scanning system to allow all image processing algorithms to have full control over the image data collected with the camera. The maximum supported resolution is 1024x768x3 at 30 frames per second (fps). Although any USB camera that works under Linux can be supported by this system with minimal alterations.

Feature	**Description**
Signal Format	Progressive
Image Type	XGA Color
Image Sensor	1/3” Inter line CCD
Resolution	Full Scan: 1024x768 1/2 Scan: 1024x344 1/4 Scan: 1024x136
Horizontal Drive Frequency	29.5 Mhz
Frame Rate	High speed: 30 fps Middle speed: 15 fps Low speed: 7.5 fps
Dimension (mm)	51(W)x51(H)x44.3(D)
Weight	145g
Power Supply	USB 2.0 Bus Power

Figure 9: Specifications of the Sentech Camera STC-C83USB-A

3 – MY ASSIGNMENT

There are three main components to the overall software system: the video capture and input/output (I/O) interface, the image processing framework, the web interface and control mechanisms. All of these components come together to make the system a flexible platform that supports a multitude of different USB cameras, development of image processing algorithms, efficient encoding, and a simple to use interface. As explained before, a part of my work consisted in creating the video capture and I/O interface.

3.1) VIDEO DEVICE AND I/O INTERFACE

3.1.1) VIDEO FOR LINUX

Video for Linux (V4L) is a video capture API for Linux. Several USB cameras, TV tuners, and other devices are supported. It is closely integrated with the Linux kernel.

V4L is in its second version. The original V4L was introduced late into the 2.1.X development cycle of the Linux kernel. Video for Linux 2 fixed some design bugs and started appearing in the 2.5.X kernels.

So for reasons of performances, I have used the second version of Video for Linux (V4L2) to allow the system to capture images. I have created a very high level interface which provides three main tools to the user: device initialization, image capture, and de-initialization. This interface is a generic interface for a range of cameras while still allowing the user to have full control over specific parameters.

3.1.2) A CUSTOMIZABLE SYSTEM

First of all, it is important to notice that the interface had to be parameterized. To answer this need, the user can choose between two different methods of configuring a video device.

The first one consists of passing a structure to the initialization function containing parameters such as width, height and brightness. This method is able to deal with image processing algorithms that require the need to dynamically alter any camera parameters. On the other hand, this method cannot be used by every user.

→ Structure of parameters (solution 1):

```
typedef struct {
int width;
int height;
int brightness;
int exposure;
...
} list_param;
list_param param;

//FILL param with desired settings
```

An alternative API consists on a configuration file using parameters stored in plain text. Users can either manually edit or use the web interface to push all changes to the configuration file. This gives users who do not have an interest in developing algorithms the ability to easily setup the platform. The file is named “camera.conf” and has for the moment around 10 options. Of course, it is very easy to add a parameter according to the needs. The available options are: width, height, framerate, brightness, contrast, saturation, exposure, white balance. Nevertheless, a part of them is not supported by some cameras depending on the driver used. Thus, with the current version of the Sentech USB driver (1.2.13) provided by Sentech, only width, height, brightness, saturation, exposure are supported. However, framerate, contrast, gain and gamma will certainly be added in the next version of the driver.

→ Part of the file “camera.conf” (solution 2):

```
#List of options used by the program
[...]

#width default : 640
width 640

#height default : 480
height 480

[...]
```

Others options are implemented in the program thanks to the use of “getopt.h”. Indeed, this library is very useful to read parameters given through the command line (argc, argv). All these options will be explained in the section 3.2: Notice, improvements and results.

3.1.3) DEVICE INITIALIZATION

3.1.3.1) Opening and initialization

Once the parameters are loaded, the device needs to be initiated with the right settings in order to start image capture. Hopefully, the V4L2 interface offers a large set of functions through a simple interface called ioctl. To simplify the syntax, I have modified this function to add error handling.

```
//Function in place of iotcl
int xioctl (int fd, int request, void *arg) {
int r;
do r = ioctl (fd, request, arg);
while (-1 == r && EINTR == errno);
return r;
}
```

Where “fd” represents the device, “request” is a request code defined in the V4L2 library, and “argp” a pointer to a function parameter, usually a structure. A specific example will be given bellow.

Prior to using ioctl, the I/O device has to be opened with the appropriate flag (ie: O_RDWR), then the interface allows us to initiate the camera. Most of the time, the video device is known as “/dev/video0”.

```
fd = open(“/dev/video0”, O_RDWR);
if (fd == -1)
errno_exit("OPEN");
```

This initialization is divided in several parts. First, the program requests the device's capabilities which are necessary to know if the camera supports the V4L2 interface and if it is able to capture images. One important step requires setting the capture format by specifying the width and height, pixel output format: YUYV, RGB or YUV420 for example. In our case, BGR24 format was used (Blue, Green, Red with 3 bytes by pixel). The reason of this choice will be given further in this report. Nevertheless, since most drivers works only in YUV format the interface properly detects acceptable camera settings and exposes those only to the user allowing the system to work across various types of cameras.

Example of the use of ioctl (setting the format):

```
struct v4l2_format fmt;

fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
fmt.fmt.pix.width = width;
fmt.fmt.pix.height = height;
fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_BGR24;
fmt.fmt.pix.field = V4L2_FIELD_NONE;

if (-1 == xioctl (fd, VIDIOC_S_FMT, &fmt))
      errno_exit ("VIDIOC_S_FMT");
```

3.1.3.2) Creating the buffers

Once the device is initialized, it is necessary to create several buffers in order to store each image during the capture. As most real-time systems, buffer management is essential for fast and efficient manipulation of data through the platform.

In our situation, memory mapped I/O is used to obtain the actual frames from the camera and to avoid unnecessary copying. Indeed, with this method, only pointers to buffers are exchanged between application and driver, the data itself is not copied. To allocate device buffers, we have to call the VIDIOC_REQBUFS ioctl with the desired number of buffers and buffer type (V4L2_BUF_TYPE_VIDEO_CAPTURE). This ioctl can also be used to change the number of buffers or to free the allocated memory. Prior accessing to the buffers they must be mapped in the memory with the “mmap” function using the good protections and flags as shown in the example below. The location of the buffers in device memory can be

determined with the VIDIOC_QUERYBUF ioctl. The m.offset and length returned in a struct v4l2_buffer are passed as sixth and second parameters to the “mmap()” function. The result of the mmap() function is saved in a pointer “void*”.

```
start = mmap (NULL, buf.length, PROT_READ | PROT_WRITE, MAP_SHARED,
camera.fd, buf.m.offset);

//PROT_READ | PROT_WRITE permitting read and write access to image buffers
//MAP_SHARE allowing applications to share the mapped memory with other
//processes
```

3.1.3.3) Changing controls as brightness, contrast

The API also supports the setting of other important camera parameters, including brightness and exposure. Hopefully, the syntax is the same for all the parameters and the interface can easily detect if an option isn't available on a specific driver using a simple call to the VIDIOC_QUERYCTRL ioctl as shown in the example bellow. Then, the value of the option (brightness in the example) is send to the device thanks to VIDIOC_S_CTRL ioctl through a v4l2_control structure.

```
struct v4l2_queryctrl queryctrl;
struct v4l2_control control;

//brightness
printf("Brightness : %d\n", brightness);
CLEAR(queryctrl);
queryctrl.id = V4L2_CID_BRIGHTNESS;
if (-1 == xioctl (fd, VIDIOC_QUERYCTRL, &queryctrl)) {
if (errno != EINVAL)
      errno_exit ("VIDIOC_QUERYCTR");
else
printf ("V4L2_CID_BRIGHTNESS is not supported\n");
} else if (queryctrl.flags & V4L2_CTRL_FLAG_DISABLED)
printf ("V4L2_CID_BRIGHTNESS is not supported\n");
else {
CLEAR(control);
control.id = V4L2_CID_BRIGHTNESS;
control.value = brightness;
if (-1 == xioctl (fd, VIDIOC_S_CTRL, &control))
      errno_exit ("VIDIOC_S_CTRL");
}
```

3.1.4) CAPTURING IMAGES

Now that the device is ready to capture images, the user can easily use the next tool: Image capture. In this function, the first step is to tell to the camera to start the capture using VIDIOC_STREAMON as below:

```
//Start capturing
enum v4l2_buf_type type;
printf("Start capturing\n\n");
type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
if (-1 == xioctl (camera.fd, VIDIOC_STREAMON, &type))
errno_exit ("VIDIOC_STREAMON");
```

The rest of this tool is incorporated into a loop and consists on several steps. First of all, we have to wait for a change on the device using the function “select”:

```
fd_set fds; //Fixed size buffer
FD_ZERO(&fds); //Clear fds
FD_SET(fd, &fds); //Add a descriptor: here fd corresponding to the camera

tv.tv_sec = 2; //Timeout of 2 secs before leaving the select if no change
r = select (fd + 1, &fds, NULL, NULL, &tv);
```

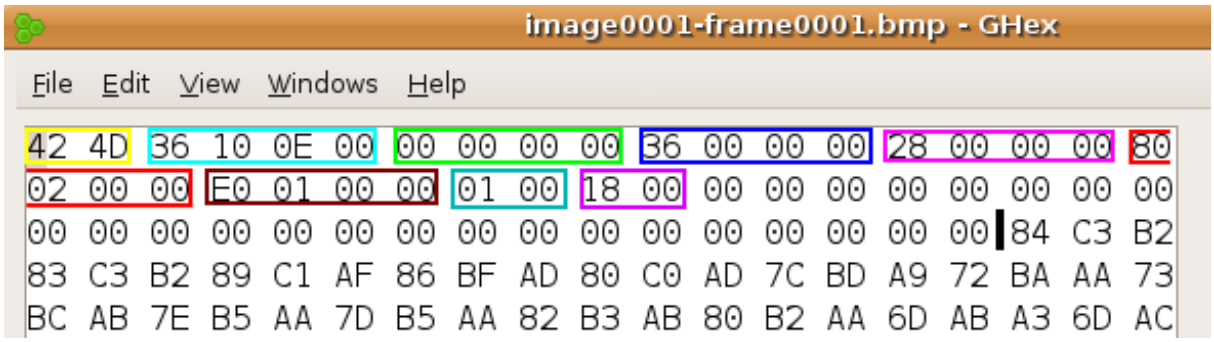
Then, the buffer is dequeued with VIDIOC_DQBUF ioctl. After obtaining a frame, the buffer is either exported to another function to save it in a BMP file or passed on for further processing (background subtraction, for example). The function to save the data in the file will be explained in the next section. Finally, VIDIOC_QBUF ioctl is used to enqueue a new buffer and go for the next frame. During all this work, the video interface counts the number of frame per second thanks to a structure “timeval” containing the time of the frame called timestamp. This element is returned when the buffer is dequeued. Moreover, with the timestamp, we can know if the image is right or not. Indeed, the first images dequeued from the memory match with a previous capture. By comparing the timestamp to the actual time obtained with “gettimeofday()”, it is straightforward to skip these images.

3.1.5) SAVING DATA IN A BMP FILE

As we use BGR24 format to capture images from the camera, the interface saves the data in BMP files. This kind of file is widely used and thus really easy to manipulate.

3.1.5.1) Specification of a BMP file

A BMP file is divided in several parts. The figure 10 shows all these parts and explains their meaning. The first part is the header of the file composed by 4 fields then there is the header of the image with information about it. On the figure, the black marker indicates the beginning of the data.



Hex Value	Real Value	Description
42 4D	BM	Kind of file BM for BMP
36 10 0E 00	3*W*H + 54	Overall size of the file (54 = size of header)
00 00 00 00	0	Reserved field
36 00 00 00	54	Offset to know where the data start
28 00 00 00	40	Size of image header
80 02 00 00	W	Width of image
E0 01 00 00	H	Height of image
01 00	1	Number of planes used
18 00	24	24 bits per pixel
Rest of the header	0	Useless here but can inform about resolution, color used, ...

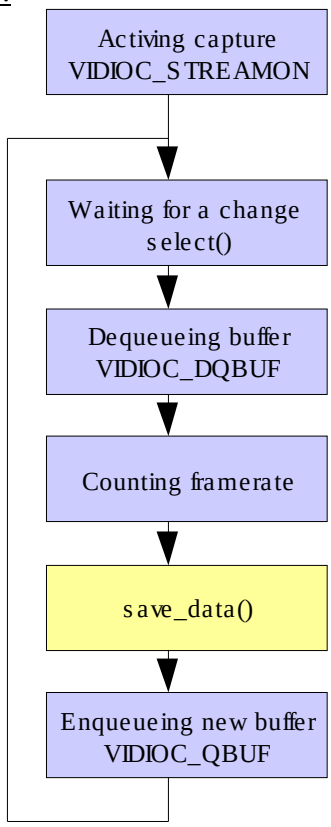
Figure 10: Header of a BMP file using Ghex, a hexadecimal editor

An important thing to notice about a BMP file is that the first pixel matches with the bottom left of the image. Unfortunately, the data coming from the camera start on the top left of the image. The result is that the image has a wrong orientation. Some loops can be used to get the image in the right orientation. However, the time lost by this operation is too important compared to its real benefit.

3.1.5.2) Methods to save the file

To have an interface working quickly (with the best frames per second rate), several methods have been imagined to save the file. The interest of such a work is most for the future if we want to use a camera able to capture with a high framerate, 60 fps for example. The first solution consists in saving the file during the capture by calling a simple function. The second one calls the same function but in a thread to allow the interface to pursue the capture during saving operation. The last solution consists in two threads working separately, one to capture and one to write the file. To test these solutions, another Sentech camera working at 60 fps has been used. Unfortunately, this work did not give interesting results since several problems appear with the memory for example.

Solution 1:



With this solution, the interface is able to capture images at 30 frames per second (fps) whatever the size of the image. This rate can't be improved because of the time spent in the function “save_data()” and it is already well optimized. As this solution is really simple and the results of the other are not really satisfactory, we have worked with it most of the time.

Figure 11: Solution 1 – Saving the file without thread

Solution 2:

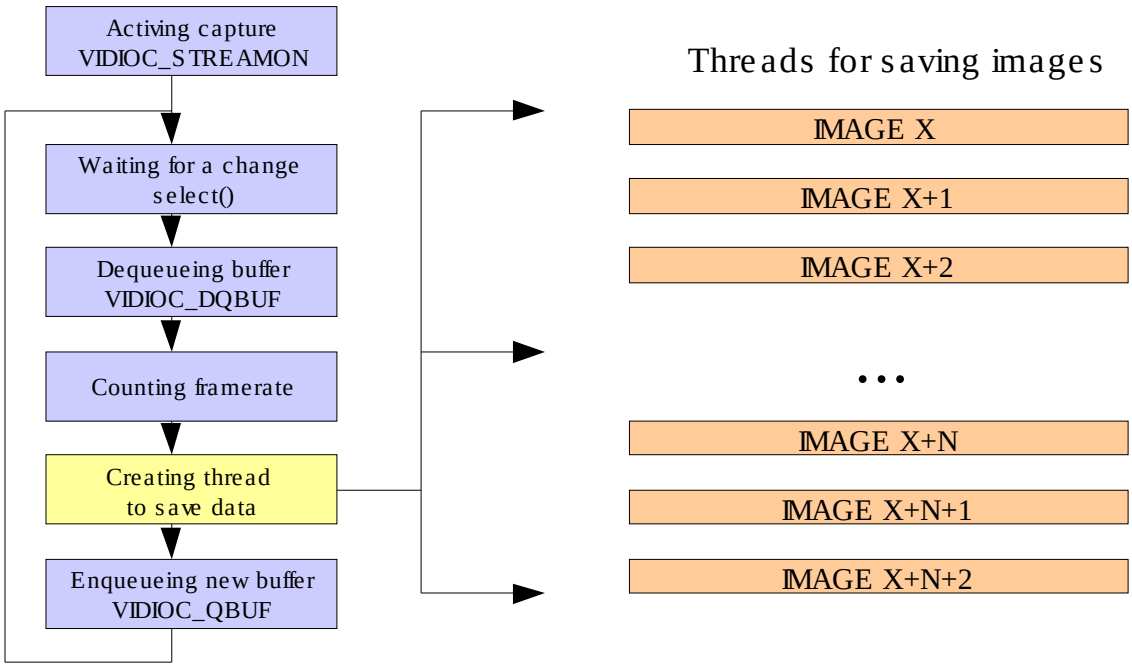


Figure 12: Solution 2 – Saving the file with threads

For this solution, the library “pthread.h” has been used. Indeed, it is very easy-to-use to create and manage threads. For each frame captured, the program copies the corresponding data thanks to “memcpy” and sends it to the thread. Unfortunately, as the time spent in the function save_data() is longer than the time to capture a new frame, the result is not interesting. Indeed, after several frames, the system is divided in too many threads containing the data. This situation increases significantly the CPU charge and memory use especially when the resolution of the image is important.

Solution 3:

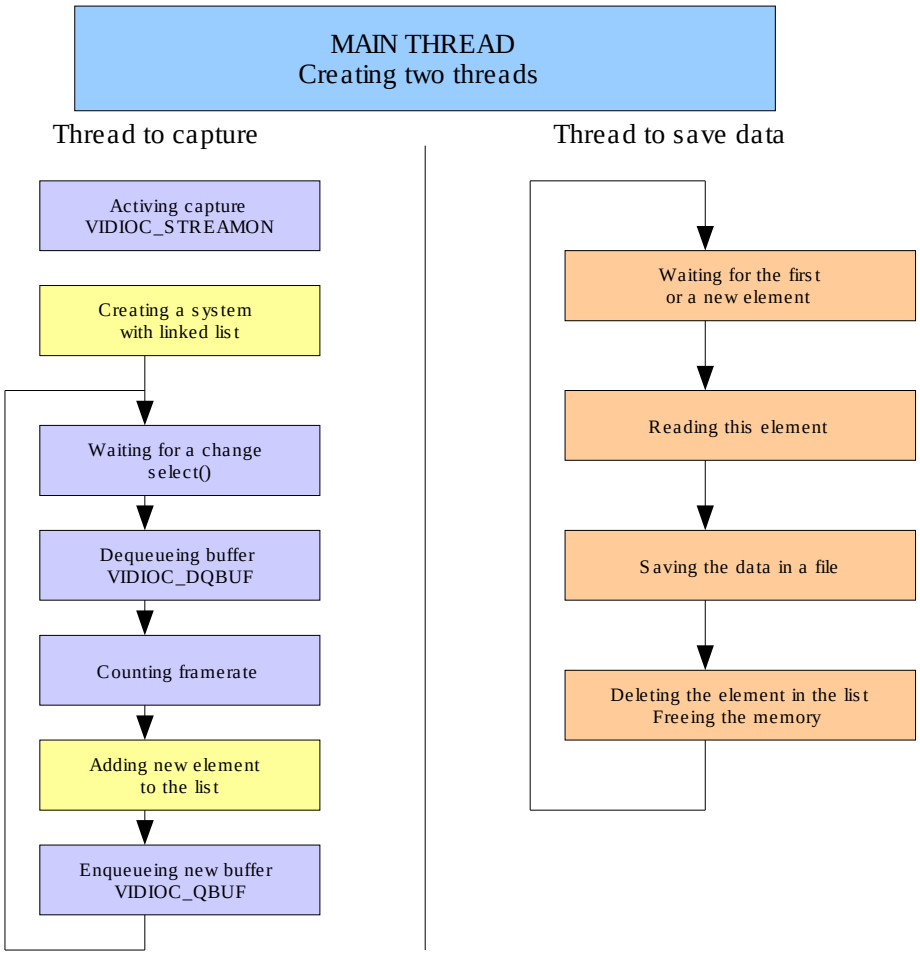


Figure 13: Solution 3 – Saving the file with threads

Unlike the previous solution, there are here two threads working separately. The first one is used to capture frames from the camera and the second one saves the data in the BMP file. To communicate each other, the program works with a double linked list. This list is like a structure but two pointers are added to it, one to the next element and one to the previous element. Thus, the thread to save data is able to read all the elements of the list very easily. At the first sight, this solution worked really nicely but a problem with the memory appeared. Indeed, “save_data” is slower than the capture, and after a lot of frames (around 1000), the gap between the threads is too big. For example at 60 fps, the first thread captures the frame 1000 and during this time the second thread saves the frame 600 in a file. That means that there are 400 frames still in memory and as the size of the data is quite big (depending on the resolution, 900KB for a frame at the size 640*480) the memory usage becomes too important. Thus the interface isn't really usable with this method even with two threads to save data and one to capture images.

3.1.6) DEVICE DE-INITIALIZATION

The last tool provided to the user is the device de_initialization function. First of all, this function stops the capture by calling the VIDIOC_STREAMOFF ioctl:

```
printf("Stop capturing\n\n");
type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
if (xioctl (fd, VIDIOC_STREAMOFF, &type) == -1)
      errno_exit("VIDIOC_STREAMOFF");
```

The next step is to reset all the buffers and to free the memory with the function unmap(). Finally the last thing to do is to close the device.

```
printf("Closing the device\n\n");
close(fd);
```

3.2) NOTICE, IMPROVEMENTS AND RESULTS

3.2.1) NOTICE

With simplicity as a primary design goal, the interface is really easy-to-use by abstracting away the underlying details of V4L2. As explained in the previous section, several means have been implemented to configure the program: with “camera.conf” and with options in the command line.

The following help (figure 14) explains how to use the interface and can be obtained with this command: “./capture -h”.

```
./capture => launch the interface with default parameters
./capture -h or –-help => display this help
Syntax: ./capture –-option val or ./capture -o val

-c or --controls => enable function for controls as brightness, ...
-o or --no_conv => disable conversion of data
-x or --x_images <int> => capture x images
-n or --no_save => capture without saving data

If one of the options below is enabled, camera.conf is not read
-w or --width <int> => give value to width
-g or --height <int> => give value to height
-f or --framerate <int> => give value to framerate
-b or --brightness <int> => give value to brightness (0-255)
-t or --contrast <int> => give value to contrast (0-255)
-s or --saturation <int> => give value to saturation (0-255)
-e or --exposure <int> => give value to exposure (0-255)
-i or --white_balance <int> => give value to white_balance (0-255)
```

Figure 14: How to use the interface

It is important to specify some points. If a user calls "./capture -b 100", the file "camera.conf" is not read and as no value is entered for width and height, the interface uses the default values: 640 and 480. Moreover, the interface can't convert the data if the user has asked to capture without saving data.

Example:

```
./capture -x 100 –w 1024 -g 768 -e 255
```

This command will launch the interface to capture 100 images, with 1024*768 as size and 255 as exposure. A user can follow the progress of the interface through a terminal as shown in figure 15.

```
vincent@vincent-desktop:~/vsimoes/source_final/capture$ ./capture -x 100 -w 1024
 -g 768 -e 255
Opening the device: /dev/video0

Driver: SentechUSB
Card: Sensor Technologies USB Camera

Capabilities:
capture
streaming
READ/WRITE

Setting the format:
width: 1024
height: 768


Requesting the buffer

Creating the buffers.

Setting the parameters:
Brightness : 0
Contrast : 0
Saturation : 127
Exposure : 255
Start capturing

---- Fps : 0, Image : 1
---- Fps : 16, Image : 17
---- Fps : 30, Image : 47
---- Fps : 30, Image : 77
---- Fps : 23, Image : 100
Stop capturing

Reseting the buffers

Closing the device

vincent@vincent-desktop:~/vsimoes/source_final/capture$
```

Figure 15: Result of the command in a terminal

3.2.2) IMPROVEMENTS

At this point of the work, the interface works only for the Sentech Camera using Sentech USB driver. As some problems appeared to install this driver on the Gumstix and to increase the flexibility of the platform, support of other cameras and drivers has been added. Indeed, it is really interesting to allow a user to use a simple webcam to make the system running. In this purpose, I have implemented the supports of two other cameras. The first one is a Logitech QuickCam Connect using pwc driver and the second one is a Logitech QuickCam Pro 9000 using uvcvideo driver. These two drivers are used by most of the webcam and work with YCbCr (YUV) format. Unfortunately, to obtain BMP files as output, it is necessary to create functions in order to convert the data format. Thus, the program is able to detect which camera is plugged in and to convert two mains formats: YUV420 and YUYV in BGR24 (3 bytes per pixel).

3.2.2.1) Convert YUV to RGB

Hopefully, it is quite straightforward to convert YUV data into RGB data thanks to these formulas:

$$R = \frac{298(\mathrm{Y}-16) + 409(\mathrm{V}-128) + 128}{256}$$

$$G = \frac{298(\mathrm{Y}-16) - 100(\mathrm{U}-128) - 208(\mathrm{V}-128) + 128}{256}$$

$$B = \frac{298(\mathrm{Y}-16) + 516(\mathrm{U}-128) + 128}{256}$$

As R, G and B need to match with values between 0 and 255, each function checks if the results of these calculations don't overflow using “SATURATE8(x)”:

```
#define SATURATE8(x) ((unsigned int) x <= 255 ? x : (x < 0 ? 0: 255))
```

3.2.2.2) YUV420 format

To understand what the function does to create the BMP file, it is necessary to know how the data are received from the camera. These are planar formats. The three components are separated into three sub-images or planes. The Y plane is first. The Y plane has one byte per pixel. Then, the Cb (U) plane immediately follows the Y plane in memory. The Cb plane is half the width and half the height of the Y plane (and of the image). Each Cb belongs to four pixels, a two-by-two square of the image. For example, Cb0 belongs to Y’00, Y’01, Y’10, and Y’11. Following the Cb plane is the Cr plane, just like the Cb plane.

Byte Order. Each cell is one byte.

start + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}
start + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}
start + 8:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}
start + 12:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}
start + 16:	Cb_{00}	Cb_{01}		
start + 18:	Cb_{10}	Cb_{11}		
start + 20:	Cr_{00}	Cr_{01}		
start + 22:	Cr_{10}	Cr_{11}		

Figure 16: Example for V4L2_PIX_FMT_YUV420 4x4 pixel image

3.2.2.3) YUYV format

In this format each four bytes is two pixels. Each four bytes is two Y's, a Cb and a Cr. Each Y goes to one of the pixels, and the Cb and Cr belong to both pixels. Thus, the Cr and Cb components have half the horizontal resolution of the Y component. V4L2_PIX_FMT_YUYV is known in the Windows environment as YUY2.

Byte Order. Each cell is one byte.

start + 0:	Y'_{00}	Cb_{00}	Y'_{01}	Cr_{00}	Y'_{02}	Cb_{01}	Y'_{03}	Cr_{01}
start + 8:	Y'_{10}	Cb_{10}	Y'_{11}	Cr_{10}	Y'_{12}	Cb_{11}	Y'_{13}	Cr_{11}
start + 16:	Y'_{20}	Cb_{20}	Y'_{21}	Cr_{20}	Y'_{22}	Cb_{21}	Y'_{23}	Cr_{21}
start + 24:	Y'_{30}	Cb_{30}	Y'_{31}	Cr_{30}	Y'_{32}	Cb_{31}	Y'_{33}	Cr_{31}

Figure 17: Example for V4L2_PIX_FMT_YUYV 4x4 pixel image

By using the pointer linked to the buffer containing the data, the BMP file can be built with some loops and calculations.

3.2.3) RESULTS

3.2.3.1) Performances

As explained in the above sections, the interface is very easy-to-use and flexible. It was one of the main requirements to allow a majority of persons to use it. Thus, we will now have a look at the performances of the system specially the CPU charge and the speed of it. These tests have been made in several situations and with three cameras: Sentech USB (Camera 1), Logitech QuickCam Connect (Camera 2) and Logitech QuickCam Pro 9000 (Camera 3). In all of the following tables, some situations are not supported by some cameras, these cases are marked as N.S. => Not Supported. Moreover, it is essentially to notice that all these tests have been made on a computer with a processor of 2 Ghz and 2 Go of RAM since the driver for the Sentech USB driver is still not working on the Gumstix. For this camera, the team is waiting for the new version of the driver to solve this problem. However, comparing with the Logitech cameras on the Gumstix, the results are approximately the same.

First of all, it is important to understand what kind of file the user has as output. The following tables show the size of the file obtained depending on the resolution, the situation and the camera used. The option “conversion” means that the function to convert data is enabled. It concerns the Logitech cameras since they don't provide RGB format to create the BMP files.

With conversion:

Resolution	Size (Mo)
320 x 240	0.22
640 x 480	0.88
1024 x 768	2.3
1600 x 1200	5.5

Without conversion:

Resolution	Camera 2	Camera 3
320 x 240	0.11	0.15
640 x 480	0.44	0.59
1024 x 768	1.15	1.5
1600 x 1200	N.S.	3.7

Figure 18: Size of file (MB)

We will now have a look at the maximum recording length based on the 8 GB micro-SD card available on the system and depending on the framerate. The first table specifies the framerate depending on the resolution and the camera used. The second table gives the recording length.

Resolution	Camera 1	Camera 2	Camera 3
320 x 240	30 fps	10 fps	15 fps
640 x 480	30 fps	10 fps	15 fps
1024 x 768	30 fps	10 fps	10 fps
1600 x 1200	N.S.	N.S.	5 fps

Figure 19: Maximum framerate (depending on the characteristics of the camera)

With conversion:

Resolution	Cam 1	Cam 2	Cam 3
320 x 240	20.68	62.06	41.37
640 x 480	5.17	15.52	10.34
1024 x 768	1.98	5.94	5.94
1600 x 1200	N.S.	N.S.	4.96

Without conversion:

Resolution	Cam 2	Cam 3
320 x 240	124.12	60.68
640 x 480	31.03	15.43
1024 x 768	11.87	9.1
1600 x 1200	N.S.	7.38

Figure 20: Recording length (in minutes)

All these tables show the same thing. The main limitation of the system is the amount of data saved on the platform. Indeed, the recording length decreases when the resolution becomes big. There is no way to solve this problem except convert the data into a compressed format. However, this solution involves a loss of information which could be unsuitable for a further processing on the images. Hopefully, for our system, there is no reason to work with a big resolution, 320x240 is sufficient.

Another important point to look at is the CPU charge and the memory usage illustrated in the following tables. Indeed, to be able to apply some processing algorithms on the image captured, the program has to use least possible resources.

With conversion:

Resolution	Cam 1	Cam 2	Cam 3
320 x 240	2.2	3	2.6
640 x 480	6.4	12.5	10
1024 x 768	11.8	24.1	16.5
1600 x 1200	N.S.	N.S.	21

Without conversion:

Resolution	Cam 2	Cam 3
320 x 240	1.4	1
640 x 480	7	2
1024 x 768	8.2	3.2
1600 x 1200	N.S.	5

Figure 21: CPU Usage (%)

With conversion:

Resolution	Cam 1	Cam 2	Cam 3
320 x 240	0.1	0.1	0.1
640 x 480	0.2	0.1	0.1
1024 x 768	0.5	0.1	0.3
1600 x 1200	N.S.	N.S.	0.9

Without conversion:

Resolution	Cam 2	Cam 3
320 x 240	0	0
640 x 480	0	0.1
1024 x 768	0	0.3
1600 x 1200	N.S.	0.6

Figure 22: Memory Usage (%)

With these results, we can assume now that the program consumes few resources for a basic use. Nevertheless, the CPU charge increases significantly when the data are converted and with big resolutions. Unfortunately we can't avoid it for Logitech cameras if we want a BMP file as output. Indeed this function calculates the values R, G, B for each pixel. With a resolution of 1600x1200, the amount of calculations is huge. Anyway to reduce the charge on the CPU, the only solution is to find another way to convert the data, certainly using optimized libraries. About the use of the memory by the program, the library “valgrind” allow us to know how many bytes have been used, if errors have been detected as shown in the figure 23.

```
==16666==
==16666== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 11 from 1)
==16666== malloc/free: in use at exit: 0 bytes in 0 blocks.
==16666== malloc/free: 102 allocs, 102 frees, 235,929,864 bytes allocated.
==16666== For counts of detected errors, rerun with: -v
==16666== All heap blocks were freed -- no leaks are possible.
vincent@vincent-laptop:~/vsimoes/source_final/capture$
```

Figure 23: Result of the command “valgrind ./capture –x 100”

3.2.3.2) Results on images:

In this section, the following images show how the controls as brightness, saturation change the image. An important thing to notice is that all the controls are not supported by all the cameras depending on the driver used. Thus, the Sentech USB camera can only change the brightness, the saturation and the exposure on the images.

The image on the right is the normal image obtained when capturing without changing the controls.

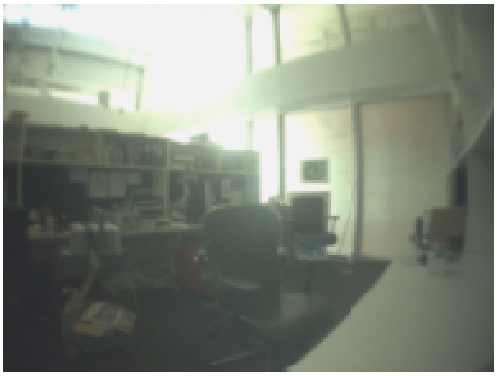
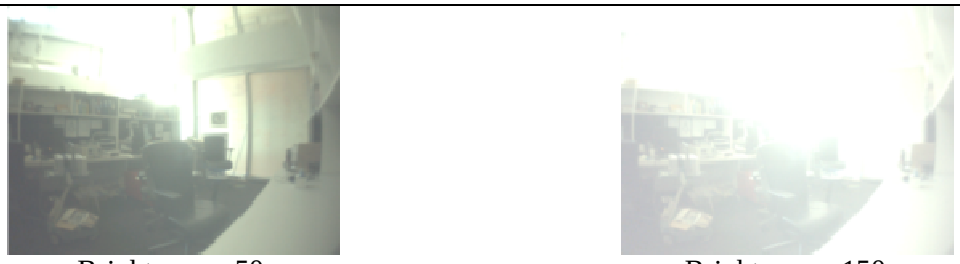


Figure 24: Normal Image (without controls)



Brightness = 50 Brightness = 150

Figure 25: Control of brightness

For the brightness, the minimum value is 0 and is the default value. The maximum value is 255 corresponding in a white image.

Saturation = 0 Saturation = 255

Figure 26: Control of saturation

When saturation is 0, the result is a blue image. For 255, the image is yellow-green. 127 is the default value.

Exposure = 0 Exposure = 255

Figure 27: Control of exposure

For this control, the value 0 gives a black image and 255 is the default value.

3.3) WAVESCRIPT

In order to improve the overall performances of the system, the team has decided to use a specific language for the software part. This language is Wavescript, which seems to give good results on system like ours. Indeed some works have been made by the developers as a platform to detect marmots in a natural environment. The main problem with Wavescript is that it is still in development and nobody in the team had already worked on this subject. Thus, I was in charge to discover the language with the help of one of the developers: Ryan Newton from the MIT. For the moment, there aren't a lot of supports for Wavescript since only few people use it. Nevertheless, I have translated the video device and I/O interface and Ryan Newton has worked on the image processing algorithms. Unfortunately, we haven't been able to compare the performances of both systems in C and in Wavescript since the translation is not finished yet. In this section, I will present the language from the beginning with the installation and a simple "Hello Word". Then, I will explain the main part of the translation of the video interface.

3.3.1) PRESENTATION OF WAVESCRIPT

Wavescript is a ML-like functional language developed by the MIT with special support for stream-processing. Although it employs a C-like syntax, Wavescript provides type inference, polymorphism, and high level functions. A Wavescript program is structured as a set of communicating stream operators and functions are considered as value. In Wavescript, however, rather than directly define operators and their connections, the programmer writes a declarative program that manipulates named, first-class streams and stream transformers: functions that take and produce streams. Those stream transformers do the work of assembling the stream operators that makes up the nodes in an executable stream graph.

Using Wavescript might increase considerably the performances of the system. Indeed, this language is high level, this usually means that the source should be shorter. Wavescript has proven to be more efficient compared to a software written in C. It is stream-oriented. That allows a lot of optimizations depending on the application. Moreover, according to the developers, it enables distributed and parallel execution with very little effort. The last point is that it can be easily transportable on a lot of different devices as mobile phone since it can delete the dependence on external libraries.

3.3.2) INSTALLATION ON A COMPUTER

The first part of the work has been to find a straightforward way to install Wavescript on a computer. Indeed, there wasn't a good explanation to install it in the documentation since the language is still in development. An important thing to notice is that Wavescript is only available on UNIX system for the moment.

There are two ways to install Wavescript: with subversion and with aptitude.

3.3.2.1) Installation with subversion

A repository is available to get the latest revision of Wavescript using subversion. Since Wavescript is still in development, it can be useful to install a previous revision to avoid bugs or errors.

First of all, some softwares and libraries are necessary:

- Ikarus 0.0.3+ (not the last official release):

For this one, build-essential, GNU Multiple Precision Arithmetic Library (GMP), bzrtools, autoconf and automake need to be installed. On Linux Ubuntu 8.04 LTS, we can use “aptitude” to install these softwares.

Then, the last version of Ikarus has to be downloaded using:

```
bzr checkout --lightweight http://www.cs.indiana.edu/~aghuloum/ikarus.dev
```

The following steps install Ikarus on the computer. The CFLAGS and LDFLAGS repertories need to match with the repertories where the library GMP is installed.

```
- ./configure CFLAGS=-I/usr/local/include LDFLAGS=-L/usr/local/lib
- make
- make install (root privilege may be required)
```

If the installation works, the command “ikarus” will give this result in an UNIX console:

```
Ikarus Scheme version 0.0.3+ (revision 1533, build 2008-07-07)
Copyright (c) 2006-2008 Abdulaziz Ghuloum
>
```

- PLT Scheme (>3.7): A shell script can be downloaded on the website of PLT Scheme.

- Subversion : It can be installed using “aptitude”

The installation of Wavescript can now begin:

```
- svn co https://svn.csail.mit.edu/wavescript/branches/wavescope Wavescript

- cd Wavescript
- source install_environment_vars
- cd src
- make
```

3.3.2.2) Installation with aptitude

At the end of my internship, the team working on Wavescript has added a new way to install it with aptitude. Unfortunately, the repository is not updated for each revision. Thus, the previous method is better even if longer.

The following repository has to be added in the “source.list” of aptitude.

```
deb http://regiment.us/ubuntu hardy main
deb http://regiment.us/ubuntu hardy multiverse
```

3.3.3) How to use Wavescript

3.3.3.1) Command ws

Running the command “`ws mysource.ws`” will compile the source file, and execute the resultant dataflow graph directly within Scheme (another language from the MIT), without generating code for another platform. The dataflow graph is converted to a Scheme program which is compiled to native code and executed. Compile time is low, because we need not call a separate compiler. However, performance is not as good as the other possibilities. Further, this method does not support external (real time) data sources, only queries running in virtual time using data stored in files.

3.3.3.2) Command wsc2

This command will be the most used for the translation. It has three purposes. First, it generates the most efficient code. Second, it serves as a platform to build custom garbage collectors (it simply uses Boost smart pointers library) and experiment with other aspects of the runtime. Third, it generates code with minimal dependencies (not requiring a runtime scheduler).

After a simple compilation of the program, the command creates two files: “query.c” and “query.exe”. The first file contains all sources rewritten without the use of external libraries. Thus, the program can be easily used on several devices as mobile phone. The second file is an executable file to launch the program.

Some options are available as “wsc2 file.ws --dot” which creates a file “query.png” to see the dataflow graph of the program. An example of such a graph will be given below for the video device and I/O interface.

3.3.3.3) Basics of the language

Wavescript is quite different compared to another language as C. It needs a different point of view since it works with streams most of the time. Thus, I will now explain the basics of it with the simplest programs possible.

```
main = timer(10.0)
```

This creates a timer that fires at 10 Hz. The return value of the timer function is a stream of empty-tuples (events carrying no information). The return value of the whole program is, by convention, the stream named “main”. The type of the above program is Stream (), where () designates the empty-tuple.

```
main = iterate x in timer(10.0) {
          emit "Hello world!";
        }
```

In this program, the “iterate” keyword provides a special syntax for accessing every element in a stream, running arbitrary code using that element's value and producing a new stream as output. In the above example, “iterate” ignores the values in the timer stream: “x”' and produces one string value on the output stream on each invocation. Thus, it prints “Hello world!” at 10 Hz. The type of the program is Stream String.

Of course, most of the time, a program is composed of several streams working together. The above example executes the body of the “iterate” every time data is received on the stream S1. The resulting stream is bound to the variable S2.

```
S2 = iterate x in S1 {
         …
       }
```

3.3.3.4) Types of variables

An important thing to know before the translation of the video device and I/O interface is the type of variables supported by the language. Unfortunately, as shown in the figure 28, only few types are implemented.

Wavescript	C	Precision
Int	int	Native ints have a system-dependent length, note that in the emulator Wavescript Ints may have less precision than C ints
Float	float	Wavescript floats are single-precision
Bool	bool	
String	char*	Pointer to null-terminated string
Array T	T*	Pointer to C representation of type T
Pointer	void*	Type for handling C-pointers. Only good for passing back to C

Figure 28: Comparison between C types and Wavescript types

3.3.3.5) Wavescript libraries

To facilitate the use of Wavescript, the developers have worked on several libraries implementing in the language basics functions as “fopen”, “fwrite” or “printf”. The main libraries are “unix.ws” and “stdlib.ws”. Thus, it is easier to create programs with Wavescript.

3.3.4) TRANSLATION OF THE VIDEO DEVICE AND I/O INTERFACE

3.3.4.1) Initial problems and solutions

After some tests of the language, the translation of the video device and I/O interface can start. Unfortunately, some types are not available in the language as the structure. Moreover, it is not possible to use pointer arithmetic. If we have a pointer called “data”, we can’t do a simple operation as “data+1”. These two elements are widely used in the program. I have been obliged to use different tools provided by the language to solve these problems. As the library Video for Linux 2 uses most of the time structures to communicate with the camera, the only solution has been to keep a part of the program in C. Indeed, the Wavescript compiler provides a facility for calling external foreign functions written in C or C++ and a tool to communicate between the parts written in C/C++ and the parts written in Wavescript. Of course, the earning of performances is lesser when external functions are used. Thus, it is really important to translate a maximum of functions.

3.3.4.2) Foreign interface and communication between C and Wavescript

As explained in the previous section, a big part of the program can’t be translated and stay in C using the foreign interface. This part will show how works this interface.

First of all, it is essential to notice that there are two tools to use C code in a Wavescript program. The first one is quite simple and consists on calling a function written in C thanks to the function “foreign” as shown below. In this example, the function declared in “foo.c” received and returned a variable Int. It can be called with “c_foo” in a Wavescript program.

```
c_foo :: Int -> Int = foreign("foo", ["foo.c"])
```

The second tool is more complicated and is used to communicate between the part in C and the part in Wavescript. A call to register a foreign source has the same form as for a foreign function: “foreign_source(function-name, file-list)”. The call to foreign_source will return a stream of incoming values. The function exported to C is called an entry point.

```
Wavescript:
wsentry1 :: Stream Array Int = foreign_source("wsentry1", ["capture.h"])

C:
void wsentry1(int*);
```

This example shows the declarations of the foreign_source and of the entry point in the C file. It is now possible to send data from the C part to the Wavescript part as shown below. Each time that a call to “wsentry1(value) “ is done in the file in C, the stream “strm_process1” received the corresponding value thanks to “emit value”.

```
strm_process1 = iterate value in wsentry1 {
      emit value;
}
```

Because the main thread of control belongs to the foreign C code, there is another convention that must be followed. The user must implement three functions that Wavescript uses to initialize, start up the system, and handle errors respectively.

```
void wsinit(int argc, char** argv)
void wsmain(int argc, char** argv)
void wserror(const char*)
```

“wsinit” is called at startup, before any Wavescript code runs. “wsmain” is called when the Wavescript dataflow graph is finished initialing and is ready to receive data. In this function, we can call other functions of the program as “init()”, “de_init()”, “capture()”. These functions use Video for Linux 2, thus the translation of them is not possible. “wserror” is called when Wavescript reaches an error.

It is now interesting to know how to communicate in the other direction, i.e. send data from Wavescript to C. After the following declarations, a call to the function “rcvr(value)” in a Wavescript program will send the corresponding value in the part in C.

```
Wavescript:
rcvr :: Array Int -> () = foreign("wssink", ["capture.c"])

C:
void wssink(int *value){
      VALUE1=value[0];
      VALUE2=value[1];
}
```

3.3.4.3) Functions and streams involved in the translation

Function	Explanation
`ptrIsNull(pointer_name)`	Returns true if the pointer is NULL
`Array:make(128, '_')`	Creates an array of string with 128 ‘_’
`String:fromArray`	Converts the array of string into one string
`Array:toList(array_name)`	Converts an array in a list
`List:ref(line,0)`	Extracts the element 0 of the list “line”
`String:implode(list_name)`	Converts a list in a string
`List_name := val:::list_name`	Adds val to the head of the list
`List:reverse(list_name);`	Reverses all elements of a list
`List:append(list1,list2);`	Assembles two lists together

Figure 29: Main Wavescript functions used in the translation

The figure 29 shows the main Wavescript functions used in the translation. Hopefully, a lot of Wavescript functions are easy to understand thanks to their name as “intToChar()”. Nevertheless, the majority of these functions can be found at the end of the Wavescript documentation with their names and their type signatures.

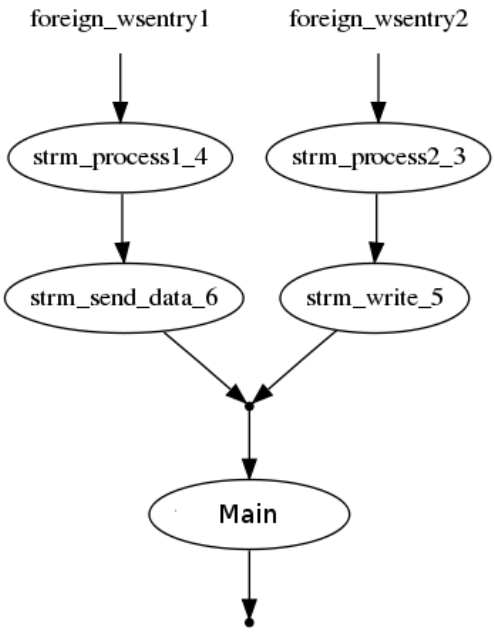


Figure 30: Dataflow graph showing the links between the streams

This figure has been obtained by compiling the program with the option “--dot” as explained in a previous section. It allows us to see the main streams involved in the program. First, “strm_process1” and “strm_process2” are used to receive data from the C part. Then, “strm_send_data” sends the information extracted from the file “camera.conf” to the C part. Finally, “strm_write” is used to read the data returned during the capture and to save it in a BMP file.

3.3.5) CONCLUSION

As it is still in development, Wavescript is not perfect and the support was relatively poor at the beginning. Thus, I spent a lot of time to understand this language. This work has been really useful for the team and for the developers. Indeed, as I came from a “C world”, the developers have been able to improve significantly their support for beginner user depending on my questions and they have implemented interesting functions to facilitate the use of it during the translation of the video device and I/O interface. Concerning the team, they have earned a lot of time with this work and can now use the language more easily.

Unfortunately, the translation of the interface hasn’t been tested on the Gumstix and compared to the version in C. Indeed, as a big part of the translation is still in C because of the use of structures by Video for Linux, the only translation of this specific part hasn’t a real interest. The benefits will be made when all the software part of the platform especially the image processing algorithms will be translated. Nevertheless, the program translated seems to be able to capture images from the camera on a computer with a similar speed as the program in C.

CONCLUSION

To conclude after all this work, the portable imaging platform is now equipped with an easy-to-use and flexible tool to communicate with cameras through the video device and I/O interface using Video for Linux 2. Thanks to good results clarified in a previous section, this interface has proved to be efficient and successful for a basic use. Nevertheless, some improvements can certainly be made and concern future works on the platform especially for the conversion data and Wavescript parts. Ryan Newton is currently working on the translation of the image processing software with Wavescript. After that, the platform will be portable to other devices such as cell phones and will certainly be quicker and, more generally, more efficient. There is also room for improvement in the hardware platform since, for the moment, the Gumstix boards use a USB1.0 connection. With further improvements by the Gumstix developers, the system should be updated with USB 2.0, which will significantly increase the performance. It might also be possible to port the platform to a FPGA, which would allow more multi-processing work.

Nevertheless, the team is currently concentrated on the deployment of the platform in the nature for real tests, on the pursuit of the translation in Wavescript, on the power management to allow a more important record length and on the improvement of the paper for a future publication.

BIBLIOGRAPHY

BOOKS:

Degges, Joey; Estrin, Deborah; Gerrick, Sophie; Hicks, John; Ko, Teresa; Lee, Jeff; Simoes, Vincent; Wilson, Mike - "PIP: Portable Imaging Platform with Context Based Encoding" - CENS Annual Research Review 2008, October 2008.

Ko, T.; Deborah E.; Stefano S. - "Background Subtraction on Distributions" http://vision.ucla.edu/~tko/summaries/cvpr08.pdf

Ahmadian S. Ko, T.; Coe, S.; M P. Hamilton; Mohammad Rahimi; Stefano Soatto; and D Estrin - "Heartbeat of a Nest: Using Imagers as Biological Sensors" - November 7, 2007 - Center for Embedded Network Sensing - Technical Reports. http://repositories.cdlib.org/cens/techrep/1

INTERNET :

CENS : http://research.cens.ucla.edu/

Conversion format: http://www.fourcc.org/fccyvrgb.php

Gumstix: http://www.gumstix.com/

Sentech camera: http://www.sentechamerica.com/pages/cameras/STC-TC83USB-A.htm

Threads: http://franckh.developpez.com/tutoriels/posix/pthreads/

UNIX commands and C functions: www.linux-kheops.com/doc/man/manfr/man-html-0.9/

UNIX and Linux: http://www.ubuntu-fr.org

Video For Linux 2: http://linuxtv.org/v4lwiki/index.php/Development:_Video4Linux_APIs

Wavescript: http://wavescope.csail.mit.edu/doku.php