UNIVERSITÀ DI BRESCIA
FACOLTÀ DI INGEGNERIA
Dipartimento di Elettronica per l'Automazione

Laboratorio di Robotica Avanzata
Advanced Robotics Laboratory

Corso di Robotica Mobile
(Prof. Riccardo Cassinis)

Installazione ed adattamento della libreria VisLib su calcolatori Aspire-One (vislib-V1.9.6)

Elaborato di esame di:

Marco Favagrossa, Marco Raggi, Gianpietro Ruggeri

Consegnato il: **10 Giugno 2009**

Sommario

Il nostro lavoro si è concentrato sull'installazione di una Webcam Philips SPC1005NC su un calcolatore Acer Aspire-One con sistema operativo Linux distribuzione Linpus con kernel 2.6.23.9. Lo scopo dell'elaborato era anche quello di installare la libreria VisLib adattata alle API V4L2, su calcolatore Acer Aspire-One, ai fini di consentire la verifica del corretto funzionamento della fotocamera digitale Philips SPC1005NC. Effettuata tale operazione abbiamo eseguito una serie di test per verificare le prestazioni della fotocamera. In ultimo sono stati svolti ulteriori accorgimenti per permettere a VisLib di funzionare sia con fotocamere V4L2 che V4L1, creando la versione 1.9.6.

1. Introduzione

Questa relazione intende descrivere le attività svolte dal nostro gruppo di lavoro nell'ambito del corso di Robotica Mobile e più in particolare l'elaborato assegnatoci dal prof. Cassinis riguardante il funzionamento di una webcam in ambiente Linux. In particolare verranno esposte le varie fasi del lavoro, i problemi riscontrati durante lo svolgimento del progetto e le soluzioni trovate.

1.1. La webcam utilizzata

La webcam sulla quale si è basato il nostro lavoro è una Philips SPC1005NC, della quale riportiamo le caratteristiche principali:

- Sensore CMOS da 1.3 Megapixel
- Frequenza max fotogrammi: 60 fps
- Collegamento pc: USB 2.0
- Dimensioni: 40 x 82 x 88 mm
- Angolo di visuale: 80°
- Sensibilità luminosa: < 5 lux
- Risoluzioni:
  160 x 120
  320 x 240
  352 x 288
  640 x 480
  1280 x 1024

1.2. Il calcolatore a disposizione

Il calcolatore utilizzato è un Aspire-One A110-Ab, un netbook prodotto da Acer e dotato di:

- Processore Intel Atom N270 (1.60 GHz, 533 MHz FSB, 512 KB L2 cache)
- 512 MB di RAM (DDR2 533 MHz SDRAM)
- 8 GB di unità SSD
- Schermo 8.9" CrystalBrite WSVGA (TFT LCD, risoluzione 1024 x 600 pixel - 262,000 colori supportati)
- Sistema operativo Linpus Linux Lite Version 1.0.9.E (kernel: 2.6.23.9)

1.3. Le fasi del lavoro

Il lavoro da noi sviluppato si è articolato in diverse fasi: durante la prima fase la nostra principale necessità è stata quella di risolvere il problema degli shot difettosi generati tramite l'utilizzo di Uvccapture, uno strumento che permette di catturare immagini.

Durante la seconda fase ci siamo dedicati all'installazione della libreria VisLib sull'Acer Aspire-One e ad adattarne le funzionalità per poter sfruttare una fotocamera che si appoggia su API V4L2 piuttosto che V4L1 (come quella su cui si basa il nostro lavoro), usufruendo di codice importato direttamente dai file di LuvcView.

Successivamente abbiamo effettuato dei test basandoci sugli esempi presenti nella cartella Examples di VisLib al fine di verificare il corretto funzionamento dell'intero sistema Aspire-One – Fotocamera SPC1005NC – vislib-V1.9.5.

Infine abbiamo provveduto a effettuare un ulteriore modifica al codice di VisLib, in modo tale da permetterne l'utilizzo con entrambe le versioni di Video for Linux. La nuova versione della libreria è stata chiamata vislib-V1.9.6.

2. Il problema affrontato

I problemi che abbiamo affrontato durante questo elaborato sono molteplici e distinguibili in relazione alla suddivisione delle fasi del lavoro fatta precedentemente.

2.1. Uvccapture

Il primo problema affrontato è stato quello dell'irregolarità degli shot effettuati dalla webcam attraverso l'utilizzo di Uvccapture. Questo accetta, tra i vari possibili, il parametro –t il quale specifica l'intervallo temporale, espresso in secondi, che deve passare tra uno shot ed il successivo. Omettendo tale parametro, o impostandolo al valore 0, otteniamo un unico shot il quale viene salvato come un'immagine jpeg completamente distorta.

2.2. Installazione della VisLib per l'uso con V4L2

Il secondo problema si è focalizzato sull'installazione della libreria VisLib su Acer Aspire-One, adattata alle API V4L2. La sua creazione è stato un lavoro piuttosto complesso, affrontato e studiato dal gruppo Chiodi-Maccarrone-Peli, con i quali è stato obbligatorio collaborare al fine di operare correttamente nella terza e quarta parte relative al nostro lavoro e descritte di seguito.

2.3. Testing

L'intero sistema formato dal netbook Acer Aspire-One, dalla telecamera Philips SPC1005NC e dalla libreria VisLib adattata per V4L2 (vislib-V1.9.5), è stato testato al fine di verificare funzionamento e prestazioni della fotocamera digitale in tale ambiente. Il test si è basato sull'esecuzione di alcuni semplici esempi presenti all'interno della cartella Examples di VisLib e sull'analisi dei risultati ottenuti.

2.4. Adattamento della VisLib per l'uso con V4L2 e V4L1

L'ultimo problema affrontato riguarda la possibilità di utilizzare VisLib sia con le API V4L2 che con V4L1. Questa nuova implementazione aprirebbe l'orizzonte alla possibilità di integrare hardware differenti che si appoggiano su standard diversi utilizzando un'unica libreria, con la comodità di dover interagire con un solo strumento e sintassi comune.

3. La soluzione adottata

3.1. Uvccapture

Uvccapture è un semplice tool che permette di ottenere degli screenshot dalla webcam. Si è partiti analizzando il comportamento della webcam a fronte di parametri di ingresso differenti. Per chiamare l'esecuzione di Uvccapture si esegue il seguente codice:

```
# ./uvccapture -ddevice_video [-xvalore_x] [-yvalore_y] -onome_file -
ttime_to_shot
```

Come si può intuire il codice riportato indica la chiamata a Uvccapture, passandogli dei parametri:

-d: indica il device con il quale Uvccapture deve comunicare. Nel caso in cui tale parametro venga omesso il device di default è */dev/video0*.

-x e –y: larghezza e altezza dell'immagine.

-o: nome con cui salvare il file.

-t: è il parametro sul quale dobbiamo focalizzarci. Indica il periodo che intercorre tra uno shot e il successivo.

Nel caso in cui uvccapture venga chiamato con il parametro –t immediatamente seguito da un numero *n*, possono verificarsi due condizioni:

- n > 0: la fotocamera si accende e continua ad acquisire fermi-immagine ogni n secondi fino a quando l'utente non la ferma.
- n = 0: equivale a omettere il parametro; in tal caso la fotocamera si accende il tempo utile per effettuare uno shot e poi spegnersi.

Tale comportamento viene descritto dal codice sottostante, estrapolato direttamente dal file uvccapture.c.

```
while (run)
  {
    if (verbose >= 2)
      fprintf (stderr, "Grabbing frame\n");
    if (uvcGrab (videoIn) < 0)
    {
      fprintf (stderr, "Error grabbing\n");
      close_v4l2 (videoIn);
      free (videoIn);
      exit (1);
    }

    if ((difftime (time (NULL), ref_time) > delay) || delay == 0)
    {
      if (verbose >= 1)
      fprintf (stderr, "Saving image to: %s\n", outputfile);
      file = fopen (outputfile, "wb");
      …
      …
      …
    }
    if (delay == 0) break;
  }
  close_v4l2 (videoIn);
```

```
  free (videoIn);
  return 0;
}
```

Il *while(run)* iniziale indica un ciclo infinito che si interrompe solamente nel caso in cui *delay*=0 (dove *delay* indica il valore associato al parametro t) e ciò è visibile a fine codice.

Il comportamento strano è dato dal fatto che nel caso di valori associati a t > 0 l'immagine che viene salvata è in generale "corretta", mentre nel caso di shot singolo l'errore è più che evidente. Tutto ciò avviene nonostante sia palesemente dimostrato dal codice che nei due casi il programma effettua le medesime operazioni.

Una possibile motivazione di tutto ciò è stata individuata nella mancanza di sincronizzazione tra il flusso video e l'istante di cattura dei frame da parte dello shot, nel caso in cui t sia pari a 0.

Sulla base di queste considerazioni si è deciso di provare ad effettuare una sorta di "Shot a vuoto". In pratica si tratta di far effettuare un' ulteriore iterazione al ciclo *while* nel caso in cui t sia uguale a 0.

Il codice corretto viene mostrato di seguito:

```
int ctrlForDelay0 = 0;
while (run)
  {
    if (verbose >= 2)
      fprintf (stderr, "Grabbing frame\n");
    if (uvcGrab (videoIn) < 0)
    {
      fprintf (stderr, "Error grabbing\n");
      close_v4l2 (videoIn);
      free (videoIn);
      exit (1);
    }

    if ((difftime (time (NULL), ref_time) > delay) || delay == 0)
    {
      if (verbose >= 1)
      fprintf (stderr, "Saving image to: %s\n", outputfile);
      file = fopen (outputfile, "wb");
      …
      …
      …
    }
    if(delay == 0)
    {
      if(ctrlForDelay0 == 0) ctrlForDelay0++;
      else break;
    }
  }
  close_v4l2 (videoIn);
  free (videoIn);
  return 0;
}
```

Attraverso una semplice variabile intera *ctrlForDelay0* inizializzata a 0, modifichiamo la condizione d'uscita dal ciclo *while*. Alla prima iterazione *ctrlForDelay0*=0 e quindi viene incrementato e viene eseguita una seconda iterazione dove, stavolta, *ctrlForDelay0*=1 e quindi attraverso il *break* usciamo dal ciclo.

Usufruendo di tale artifizio lo shot singolo di Uvccapture funziona correttamente permettendo il salvataggio di immagini nitide e prive di errori, a discapito di un allungamento del tempo di registrazione dell'immagine, seppur breve, per effettuare il doppio salvataggio.

3.2. VisLib

Come già detto in precedenza, questo argomento è stato affrontato dal gruppo Chiodi-Maccarrone-Peli con i quali si è svolto un lavoro di collaborazione al fine di ottenere la libreria VisLib adattata alle API V4L2.

Essendo la soluzione di tale problema molto lunga e complessa riportiamo di seguito i passi salienti del relativo procedimento risolutivo, rimandando alla relazione di Chiodi-Maccarrone-Peli per quanto riguarda approfondimenti e spiegazioni ulteriori.

Il lavoro di adattamento alle API V4L2, anziché le già supportate V4L1, si è focalizzato sul file della VisLib grab.c. Tale file, più che modificato, ha subito una totale riscrittura attraverso la sostituzione in toto con il file grabV4L2.c, il quale coopera con altri direttamente importati dal programma Luvcview in VisLib.

I file che sono stati presi da Luvcview (applicazione grafica, compatibile V4L2, molto utile per effettuare debugging e per verificare le funzionalità di streaming e di tracking della webcam) sono distinguibili in 2 gruppi:

- Gli header-file inseriti all'interno della directory vislib/include
  - avlib.h
  - color.h
  - dynctrl-logitech.h
  - huffman.h
  - utils.h
  - uvc_compat.h
  - uvcvideo.h
  - v4l2uvc.h
- I file .c inseriti all'interno della directory vislib/src
  - avlib.c
  - color.c
  - utils.c
  - v4l2uvc.c

Il passo successivo è stato quello di modificare l'header file vlGrabV4L2.h già compreso nella VisLib versione 1.9.

Per prima cosa è stato inserito questo include:

```
# include <linux/videodev.h>
```

necessario per poter sfruttare le due costanti di V4L2, cioè la V4L2_PIX_FMT_YUYV e la V4L2_PIX_FMT_MJPEG, le quali definiscono il formato dell'input.

Un'ulteriore modifica apportata a tale file è stata l'inserimento di "Include guard" utile per evitare che il compilatore includa più volte lo stesso file.

```
# ifndef __GRAB_V4L2_H__

# define __GRAB_V4L2_H__
```

Si tratta di particolari direttive (o macro) che vengono usate nei file header per evitare di avere errori in fase di linking. In alcuni casi è proprio impossibile farne a meno, specialmente se il progetto inizia ad avere molti file sorgenti.

Il lavoro più grosso è stato quello di sostituire il file grab.c con grabV4L2.c.

Il corpo di grabV4L2.c è stato copiato direttamente dal file originale grab.c di VisLib e successivamente modificato. Infatti all'interno del codice le funzioni usavano ancora V4L1 e quindi è stato necessario sostituire le chiamate ai metodi principali, andandoli a prendere direttamente dal file v4l2uvc.c, importato da Luvcview.

I metodi chiamati, appartenenti al file v4l2uvc.c, sono i seguenti:

- init_videoIn( )
- uvcGrab( )
- close_v4l2( )
- v4l2SetControl( )

3.3. Testing

In questa fase del lavoro siamo passati ad effettuare una serie di test, atti a verificare l'effettiva bontà di quanto fatto fino a questo punto e stilare un resoconto prestazionale sulla nuova libreria modificata. A questo proposito abbiamo sfruttato alcuni programmi presenti nella cartella Examples di VisLib, apportando alcune semplici modifiche per adattarli al nuovo contesto.

3.3.1. Acquisizione_V4L2

Il primo esempio che abbiamo preso in considerazione è stato acquisizionePiccola.c. Questo programma permette di acquisire un'immagine dalla webcam, salvandola in formato jpeg all'interno della stessa cartella nella quale si trova l'eseguibile. Il codice è stato leggermente rivisto al fine di permettere il passaggio da linea di comando di parametri come altezza, larghezza dell'immagine e framerate (in modo tale da poter svolgere più test con parametri diversi senza doverlo ricompilare ogni volta, velocizzando le operazioni). Si è rivelato necessario modificare pure la sezione di inizializzazione del grab, in particolare la chiamata al metodo vlGrabInit, specificando opportunamente il device (poiché il calcolatore è fornito di una fotocamera integrata la webcam Philips SPC1005NC nel nostro caso è il device secondario) e il formato dell'input (V4L2_PIX_FMT_MJPEG o V4L2_PIX_FMT_YUYV). Il file è stato infine rinominato in acquisizione_V4L2.c.

```
//Ozioni da linea di comando
  while ((argc > 1) && (argv[1][0] == '-')) {
    switch (argv[1][1]) {
    case 'f':
      fps=atoi(&argv[1][2]);
      argom++;
      break;

    case 'x':
      x = atoi(&argv[1][2]);
      argom++;
      break;

    case 'y':
      y = atoi(&argv[1][2]);
      argom++;
      break;

    default:
      fprintf (stderr, "Unknown option %s \n", argv[1]);
            exit (0);

    }
    ++argv;
```

```
    --argc;
  }

if(argom==0)
{
      printf ("\n \n \nNessuna Opzione Selezionata: la foto verrà
salvata in formato 640X480\n");
      printf ("Il framerate di acquisizione e stato settato a %d
fps\n",fps);
}
…

/* sezione di inizializzazione del grab */
      if (0 > vlGrabInit ("/dev/video1", V4L2_PIX_FMT_MJPEG, x, y, 1))
{
            VL_ERROR ("vlInit: error: Impossibile inizializzare il
framegrabber\n");
      return (-1);            /* failure */
      }
      printf ("vlGrabInit avviato correttamente\n");
```

Dunque, a differenza della versione precedente, il nuovo comando per richiamare questo esempio sarà:

 # ./acquisizione_V4L2 [-xvalore_x] [-yvalore_y] [-fvalore_fps] ove:

valore_x: Numero intero che specifica la larghezza in pixel dello shot. Se questo parametro viene omesso l'immagine viene scattata con una larghezza di default pari a 640 pixel

valore_y: Numero intero che specifica l'altezza in pixel dello shot. Se questo parametro viene omesso l'immagine viene scattata con un'altezza di default pari a 480 pixel

valore_fps: Numero intero che specifica il framerate (frame/secondo) che la fotocamera deve adottare per l'acquisizione dell'immagine. Se questo parametro viene omesso, viene settato un framerate di default a 30fps. Nel caso vengano inseriti framerate non validi o non raggiungibili per quei determinati x ed y specificati, il programma lo notifica a video scegliendo il framerate più vicino al valore richiesto nel parametro.

Ad ogni prova effettuata abbiamo imposto di variare la dimensione dell'immagine e il framerate controllando se tutte le possibili combinazioni fossero accettate.

Dimensione immagine	Framerate supportati (fps)
160x120	5 - 10 - 15 - 20 - 25 - 30
320x240	5 - 10 - 15 - 20 - 25 - 30
352x288	5 - 10 - 15 - 20 - 25 - 30
640x480	5 - 10 - 15 - 20 - 25 - 30
1280x1024	5 - 9

I risultati ottenuti di volta in volta hanno mostrato come, per la maggior parte dei casi, i framerate accettati fossero tutti i multipli di 5 compresi tra 5 e 30. Nel caso vengano inseriti valori di framerate non validi o non raggiungibili per quei determinati x ed y specificati, il programma lo notifica a video

scegliendo il framerate più vicino al valore richiesto nel parametro. Fa eccezione il caso 1280x1024 nel quale non è possibile superare i 9 fps.

3.3.2. framerate_V4L2

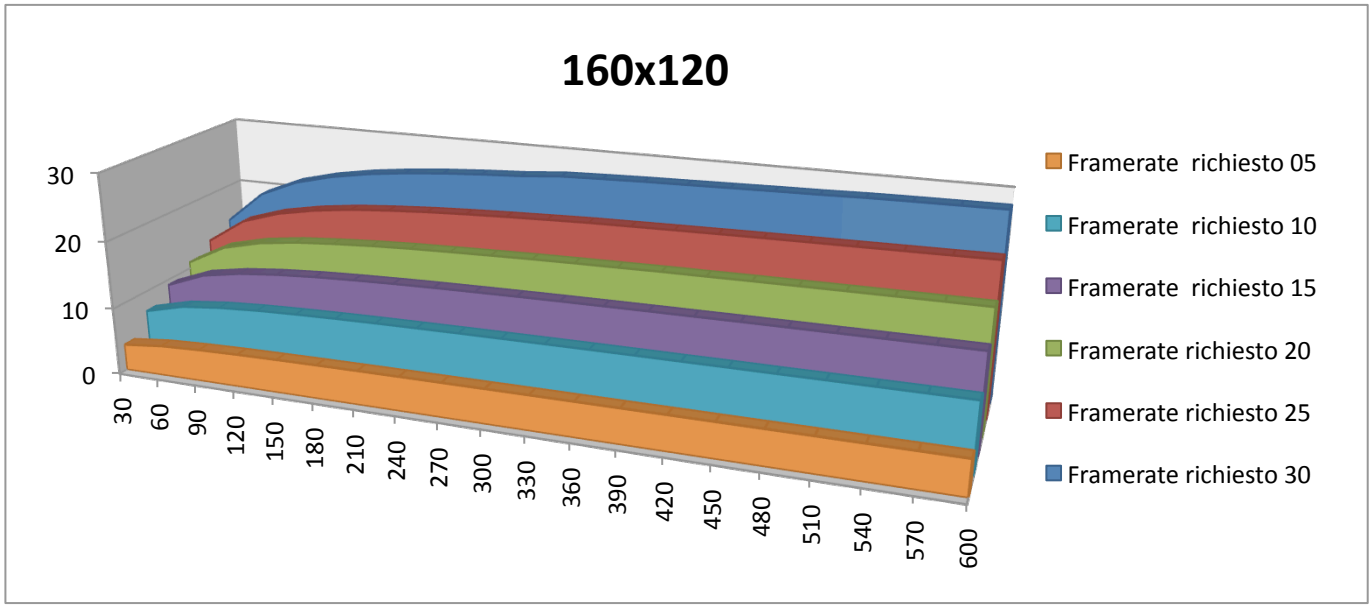
Il secondo programma considerato si chiama frameratepiccolo.c e permette di visualizzare a video il flusso di immagini catturate dalla webcam (il numero di frame al secondo dipende dalla scelta effettuata dall'utente entro i range ed i valori di accettabilità previsti). Inoltre, ad intervalli regolari, viene visualizzato a schermo il framerate reale in modo da poter verificare quanto questo si discosti dal valore richiesto da linea di comando (o di default). Anche in questo caso il primo passo è stato modificare il programma per poter inserire i parametri da linea di comando, esattamente come nel caso precedente (3.3.1), dopodiché si è passati ad effettuare le prove vere e proprie. Il file è stato rinominato in framerate_V4L2.c. Il tutto è stato riportato in 3 tabelle con i relativi grafici, in riferimento a 3 dimensioni di immagine che abbiamo ritenuto particolarmente significative per ottenere una sorta di indice di prestazioni su un campione effettivamente rappresentativo. Sull'asse delle ordinate sono riportati i valori di framerate effettivi, su quello delle ascisse gli intervalli regolari per il campionamento del framerate (30 frame) mentre sull'asse z sono riportati i framerate esplicitamente richiesti dall'operatore.

La sintassi utilizzata per richiamare i tre casi scelti ed i relativi risultati finali sono:

1. ***160x120***

```
# ./framerate_V4L2 [-x160] [-y120] [-fvalore_fps]
```

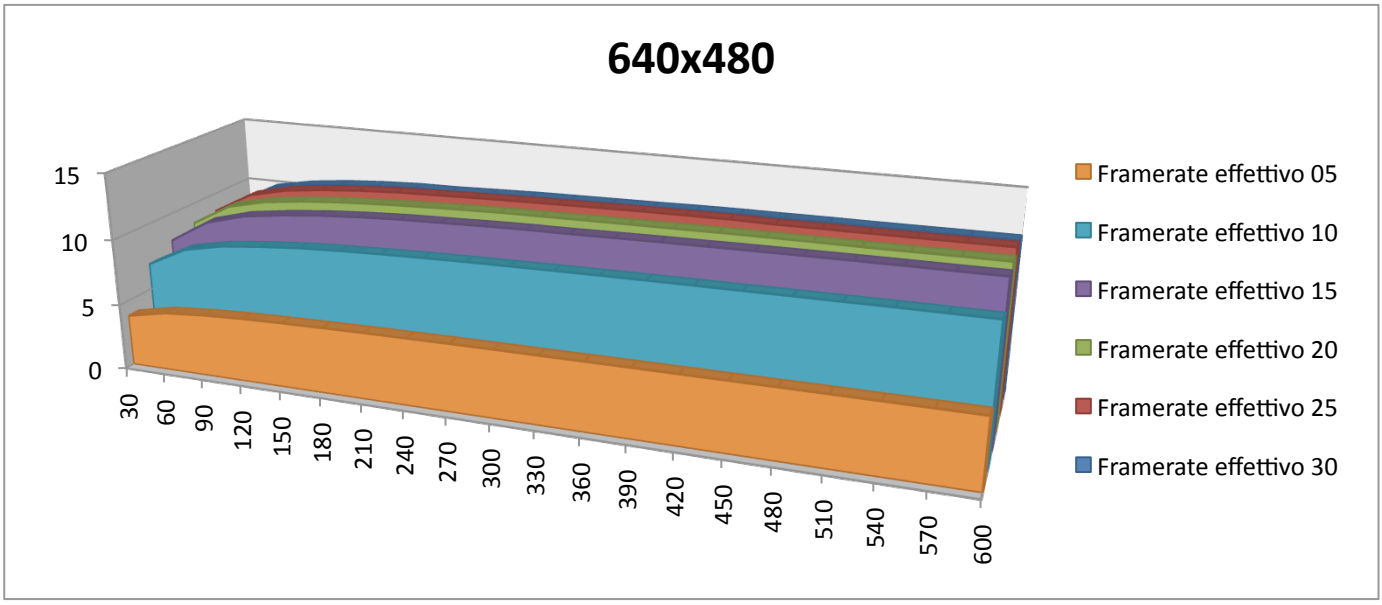
	Framerate effettivi					
Frame effettuati	Framerate richiesto 30	Framerate richiesto 25	Framerate richiesto 20	Framerate richiesto 15	Framerate richiesto 10	Framerate richiesto 05
30	14,42	12,72	10,97	9,19	7,04	3,89
60	19,32	16,51	13,98	11,42	8,49	4,49
90	21,79	18,37	15,39	12,42	9,1	4,73
120	23,27	19,5	16,17	12,99	9,44	4,87
150	24,29	20,26	16,63	13,36	9,65	4,95
180	25	20,75	16,99	13,61	9,8	5,01
210	25,53	21,14	17,25	13,8	9,91	5,05
240	25,94	21,43	17,44	13,95	9,99	5,08
270	26,27	21,67	17,6	14,06	10,06	5,1
300	26,78	21,87	17,75	14,16	10,11	5,12
330	26,97	22,06	17,86	14,24	10,16	5,14
360	27,14	22,17	17,95	14,3	10,2	5,15
390	27,26	22,28	18,05	14,36	10,23	5,16
420	27,39	22,38	18,1	14,4	10,25	5,18
450	27,52	22,47	18,17	14,45	10,28	5,18
480	27,62	22,54	18,21	14,48	10,29	5,19
510	27,78	22,61	18,26	14,51	10,32	5,2
540	27,86	22,68	18,3	14,54	10,33	5,2
570	27,92	22,75	18,34	14,57	10,35	5,21
600	27,98	22,83	18,37	14,59	10,36	5,21



2. 640x480

```
# ./framerate_V4L2 [-x640] [-y480] [-fvalore_fps]
```

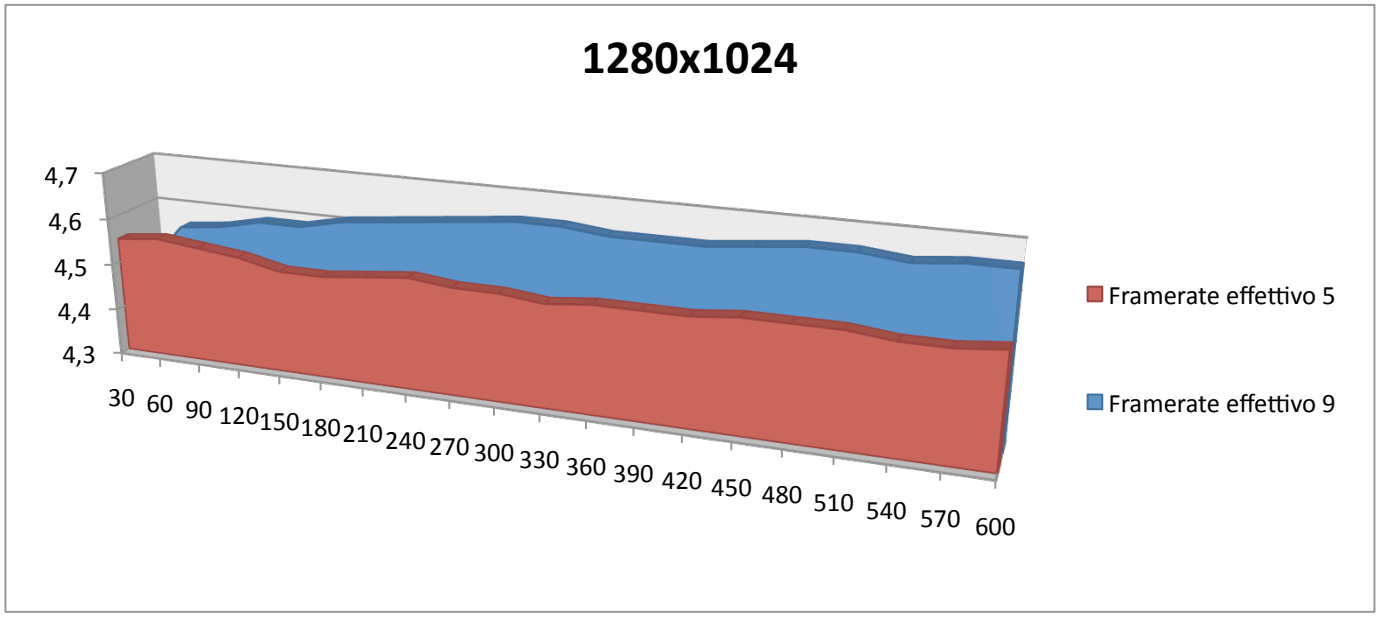
	Framerate effettivi					
Frame effettuati	Framerate richiesto 30	Framerate richiesto 25	Framerate richiesto 20	Framerate richiesto 15	Framerate richiesto 10	Framerate richiesto 05
30	8,47	8,58	8,43	7,91	6,95	3,86
60	9,97	10,2	10,09	9,67	8,43	4,47
90	10,55	10,89	10,78	10,49	9,05	4,71
120	10,89	11,26	11,15	10,9	9,37	4,85
150	11,1	11,5	11,4	11,21	9,62	4,93
180	11,24	11,68	11,57	11,42	9,77	4,99
210	11,27	11,77	11,7	11,57	9,88	5,04
240	11,37	11,86	11,81	11,66	9,97	5,07
270	11,46	11,94	11,89	11,75	10,04	5,1
300	11,51	12	11,93	11,82	10,09	5,12
330	11,56	12,06	11,97	11,85	10,14	5,13
360	11,6	12,11	12,01	11,91	10,18	5,15
390	11,67	12,14	12,04	11,96	10,21	5,16
420	11,7	12,17	12,08	12	10,24	5,17
450	11,71	12,13	12,1	12,04	10,26	5,18
480	11,73	12,15	12,11	12,07	10,29	5,19
510	11,73	12,16	12,11	12,1	10,31	5,2
540	11,75	12,18	12,13	12,12	10,34	5,2
570	11,77	12,2	12,16	12,13	10,35	5,21
600	11,79	12,22	12,17	12,15	10,36	5,21



3. 1280x1024

```
# ./framerate_V4L2 [-x1280] [-y1024] [-fvalore_fps]
```

	Framerate effettivi	
Frame effettuati	Framerate richiesto 9	Framerate richiesto 5
30	4,46	4,55
60	4,56	4,56
90	4,57	4,55
120	4,59	4,54
150	4,59	4,52
180	4,61	4,52
210	4,62	4,53
240	4,63	4,54
270	4,64	4,53
300	4,65	4,53
330	4,65	4,52
360	4,64	4,53
390	4,64	4,53
420	4,64	4,53
450	4,65	4,54
480	4,66	4,54
510	4,66	4,54
540	4,65	4,53
570	4,66	4,53
600	4,66	4,54

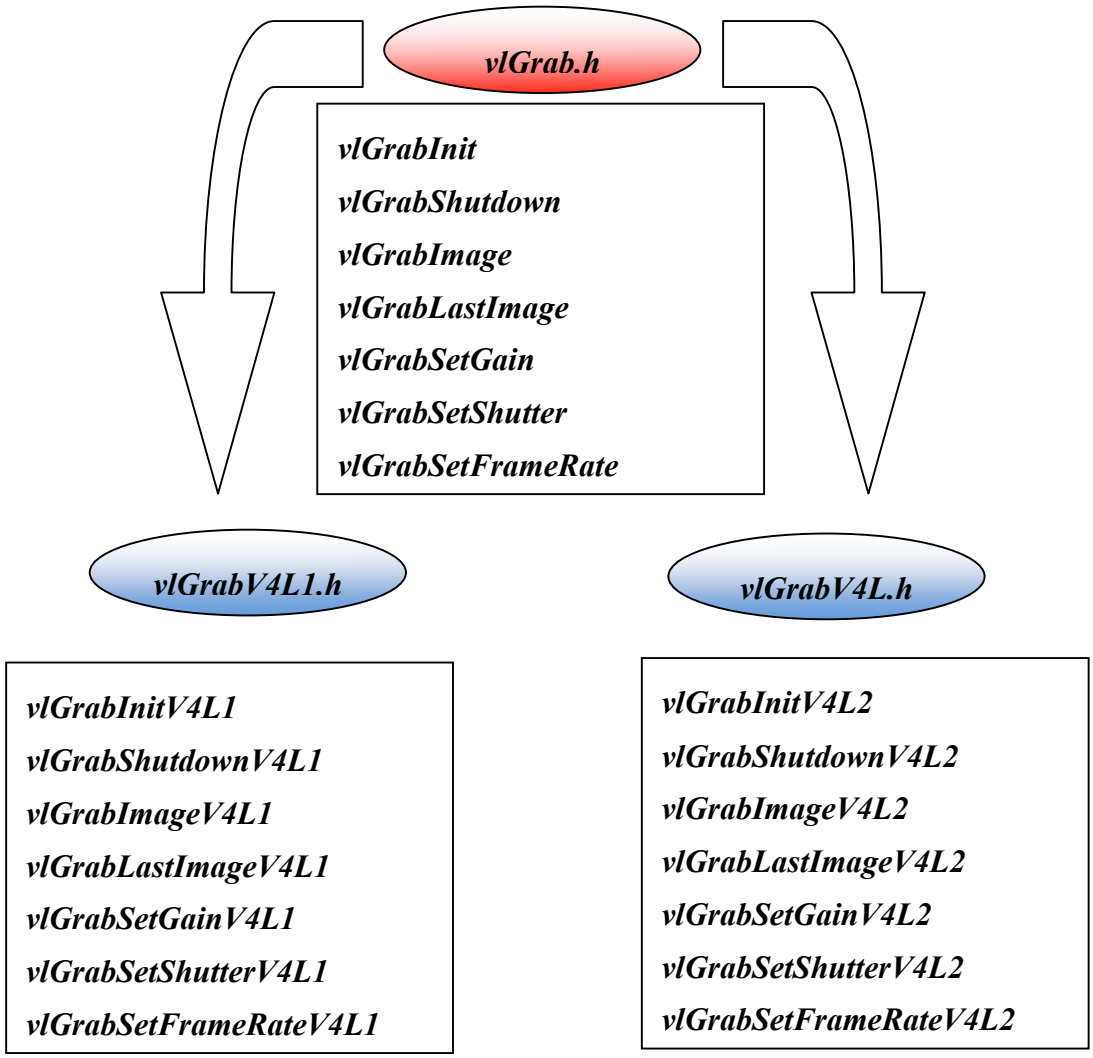


Come si può osservare dai risultati effettivamente ottenuti, nel caso di risoluzioni richieste relativamente limitate, i valori di framerate misurati tendono ad avvicinarsi molto più rapidamente al valore richiesto con framerate piccoli. Quando invece si è in presenza di risoluzioni maggiori o comunque framerate richiesti che si avvicinano a quelli massimi impostabili per questa combinazione librerie-fotocamera-calcolatore, nei valori iniziali rilevati abbiamo framerate molto bassi, che tendono comunque a raggiungere un valore prossimo a quello impostato secondo un andamento verosimilmente logaritmico.

3.4. Adattamento della VisLib per l'uso con V4L2 e V4L1

Per riuscire ad adattare la nuova versione della libreria VisLib anche alle API V4L1, ovvero per creare una sorta di retro compatibilità con l'hardware che ne faceva uso, abbiamo deciso di creare una soluzione che consentisse un riconoscimento automatico del tipo di funzioni da utilizzare. Questo è stato realizzato sdoppiando le chiamate interne a più file appartenenti alla VisLib stessa. Quando un applicativo ne richiede un particolare metodo, viene dapprima testata, in un'apposita struttura condizionale, la compatibilità con V4L2 inizializzando la relativa variabile globale; se la chiamata va a buon fine significa che l'oggetto sta utilizzando lo standard corretto, altrimenti vengono richiamate automaticamente le API V4L1 modificando di conseguenza il valore della variabile globale. Questa infatti, ci servirà all'interno del corpo dei metodi per discriminare tutte le successive funzioni che dovranno essere utilizzate per far funzionare correttamente l'hardware.

Vediamo ora di seguito le modifiche sostanziali apportate a *vislib/include* e *vislib/src.*



Per quanto riguarda gli header file ci siamo limitati alla rinominazione dei metodi, mentre per i file sorgenti il lavoro è stato più complesso.
Infatti, a fronte di questi cambiamenti, i metodi all'interno del file vlGrab.c sono stati modificati come mostrato di seguito:

- **vlGrabInit**

```
int vlGrabInit (char *device_name, int input,
int cols, int rows, int norm)
{
version=2;
if(vlGrabInitV4L2(device_name,input,cols,rows,norm)==-1)
{
      version=1;
      return vlGrabInitV4L1(device_name,input,cols,rows,norm);
}
return 0;
}
```

Come si può vedere, questo metodo richiama subito vlGrabInitV4L2 e verifica che il ritorno di tale metodo sia diverso da "-1", cioè che ci sia una compatibiltià con le API V4L2. Nel caso in cui ciò non avvenga viene richiamato il metodo associato alle API V4L1. Alla variabile globale *version* viene assegnato un preciso valore a seconda della compatibilità soddisfatta. Tale variabile verrà poi utilizzata all'initerno degli altri metodi come mostrato nel seguente esempio.

- **vlGrabImage**

```
int vlGrabImage (vlImage *pic)
{
if(version==1)
      vlGrabImageV4L1(pic);
else
      vlGrabImageV4L2(pic);
}
```

Per quanto riguarda i restanti metodi del file vlGrab.c, il corpo di questi sarà strutturalmente identico a quello proposto come esempio del vlGrabImage. Tale comportamento comporterà obbligatoriamente l'inclusione, all'interno del file vlGrab.c, dei file vlGrabV4L1.h e vlGrabV4L2.h.

Successivamente è stato necessario includere, oltre a vlGrab.h, i due nuovi file generati vlGrabV4L1.h e vlGrabV4L2.h all'interno dell'header vislib.h

```
//*******PER FAR FUNZIONARE SIA CON V4L2 CHE CON V4L1**********
#include "vlGrab.h"
#include "vlGrabV4L2.h"
#include "vlGrabV4L1.h"
//************************************************************
```

Come ulteriore accorgimento è stato necessario modificare il Makefile presente all'interno della cartella src al fine di inlcudere i file grabV4l1.cc e grabV4l2.c nelle operazioni di compilazione.

```
SOURCES  = v4l2uvc.c          \
         …
         grabV4L1.c           \
         grabV4L2.c           \
         grab.c               \
         …
```

Inoltre è necessaria l'installazione di altri tre pacchetti al fine di ottenere un corretto funzionamento della libreria modificata:

libx11-devel – 1.1.3-4.fc8

libxt-devel – 1.0.4-3.fc8.i386

libxext-devel – 1.0.1-4.fc8.i386

NOTA: Nel caso si utilizzasse un sistema operativo Linux distribuzione Linpus con kernel 2.6.23.9, come nel caso dell'Acer Aspire-One, in fase di compilazione si verifica un errore di mancato rilevamento di un file. Tale errore fa riferimento all'inclusione presente all'interno del file grabV4L1.c (vecchio grab.c) del file "pwc-ioctl.h".
Per risolvere tale problema è necessario installare il pacchetto Kernel-devel – 2.6.21.1-42.fc8-i686 che fornisce i kernel headers necessari all'interno dei quali poter trovare il file mancante. Una volta installato il codice da inserire in grabV4L1.c è il seguente:

```
#include "/usr/src/linux/include/media/pwc-ioctl.h"
```

dove “linux” è un link simbolico al quale è associato il path “kernels/2.6.23.1-42.fc8-i686”.

NOTA: Per creare l’etichetta “linux” il comando è il seguente:

```
sudo ln –s kernels/2.6.23.1-42.fc8-i686 linux
```

In conclusione a questo lavoro si è dato un nome al lavoro svolto, creando una nuova versione di VisLib denominata vislib-V1.9.6.

4. Modalità operative

4.1. Componenti necessari

Il sistema presentato nei capitoli precedenti necessita dei componenti hardware e software elencati di seguito.

Componenti hardware:

- Webcam USB Philips SPC1005NC.
- NetBook Acer ApireOne A110-Ab.

Componenti software:

- Sistema operativo Linpus Linux Lite Version 1.0.9.E (kernel: 2.6.23.9).
- Compilatore C: per la compilazione del codice sorgente dei driver e del software applicativo (consigliati: gcc e g++).
- Driver uvc-video: sono driver preinstallati sulla piattaforma utilizzata, risultati compatibili con la webcam in questione.
- Luvcview: applicativo per lo streaming delle immagini acquisite con la webcam e che dà la possibilità di controllarne i movimenti direttamente dalla sua interfaccia grafica. E’ lo strumento consigliato dalla Logitech.
- Uvccapture: applicativo per acquisire le immagini dalla webcam da linea di comando.
- VisLib: libreria necessaria al funzionamento della fotocamera, versione adattata alla compatibilità V4L2.

4.2. Modalità di installazione

Le modalità di installazione del sistema operativo Linpus e del kernel Linux esulano dagli scopi di questa trattazione e non verranno affrontate in quanto già installati sul calcolatore Acer Aspire-One.

Per lo stesso motivo non saranno approfondite le installazioni del compilatore C e dei driver uvc-video, anch’essi già presenti sul calcolatore usato, in quanto nativi del kernel 2.6.23.9.

4.2.1. Collegamento fisico

Innanzitutto è necessario connettere il cavo della webcam ad una porta USB del calcolatore e controllare che il dispositivo venga riconosciuto dal sistema operativo. Per verificarlo è sufficiente che appaiano i dati della webcam nell’output del comando:

```
# dmesg
# uvcvideo: Found UVC 1.00 device Philips SPC1000NC Webcam (0471:0332)
```

Il sistema assegna alla webcam un device denominato *video* seguito da un numero, ad esempio *video0, video1,* etc. Nel nostro caso alla webcam Philips SPC1005NC viene sempre assegnato il device */dev/video1* a causa della presenza di una webcam integrata nell’AspireOne con etichetta di device */dev/video0.*

4.2.2. Installazione di uvccapture

Uvccapture è un semplice tool che permette di ottenere degli screenshot dalla webcam.

Scaricare il pacchetto http://staticwave.ca/source/uvccapture/uvccapture-0.5.tar.bz2 e installarlo tramite i comandi:

```
# bzip2 –d uvccapture-0.5.tar.bz2
# tar –xvf uvccapture-0.5.tar
# cd uvccapture-0.5
# make
# make install
```

4.2.3. Installazione di VisLib

Affrontiamo ora l'installazione della libreria VisLib su Linpus kernel 2.6.23.9, differenziando diverse versioni di VisLib.

- vislib-V1.9.4.tgz:
  La versione 1.9.4 della libreria, reperibile sul sito del corso di Robotica Mobile, è stata scaricata e installata sulla macchina attraverso la seguente procedura:

  Scaricamento dei seguenti pacchetti tramite l'applicazione "Package Manager" presente sul sistema operativo Linpus:
  libx11-devel – 1.1.3-4.fc8
  libxt-devel – 1.0.4-3.fc8.i386
  libxext-devel – 1.0.1-4.fc8.i386
  Kernel-devel – 2.6.21.1-42.fc8-i686

  Posizionarsi nella cartella dove è stata scaricata la VisLib
  ```
  # tar –xvf vislib-V1.9.4.tgz
  # cd vislib
  # make
  # make install
  ```

- vislib-V1.9.5.tgz (versione Chiodi-Maccarrone-Peli):
  Essendo tale libreria completamente adattata al solo supporto delle API V4L2, la sua installazione non necessita dell'inclusione dei pacchetti sopra citati. Per quanto riguarda le operazioni di installazione, vedere sopra.

5. Conclusioni e sviluppi futuri

Le prime fasi del nostro lavoro, ovvero l'installazione di uvccapture, la sua modifica per la risoluzione del problema relativo allo shot singolo della webcam, sono risultate abbastanza scorrevoli. Più ostica invece è stata la fase di adattamento della libreria VisLib. La nostra scelta progettuale si è orientata sull'ottenere uno strumento che fosse il più generale possibile e permettesse di mantenere la compatibilità con le versioni V4L2 e V4L1; questo ha introdotto delle potenziali inefficienze in quanto una persona che utilizzasse il nostro lavoro per far funzionare una webcam basata su V4L2, sarebbe costretto anche ad installarsi i pacchetti necessari esclusivamente al supporto V4L1.

Un possibile sviluppo futuro sarà quindi quello di realizzare un'applicazione che preveda la stessa copertura funzionale, ma con la possibilità di effettuare una sorta di compilazione selettiva che permetta di scegliere a quale formato la fotocamera che si sta utilizzando si riferisce.

5.1. Verifica del funzionamento della fotocamera PWC

A seguito della richiesta effettuata dal prof. Cassinis, abbiamo provato l'utilizzo della fotocamera Philips PWC sul sistema composto da Acer Aspire One e vislib-V1.9.6.

Alla chiamata di uno degli esempi presenti nella cartella Examples (modificando opportunamente i parametri passati al metodo vlGrabInit()) è stato visualizzato un errore di Permission Denied restituito dal metodo grabInit() durante l'operazione specifica di "open device". Abbiamo quindi effettuato la chiamata attraverso il comando sudo: in questo caso l'operazione di "open device" è stata superata correttamente, ma subito dopo è stata riscontrata una incompatibilità con i driver, più precisamente è stato segnalato il seguente errore: "Inappropriate ioctl for device".

Forti della struttura della versione 1.9.6 di VisLib, abbiamo subito deciso di verificare il funzionamento della fotocamera in questione usufruendo della libreria vislib-V1.9.4 già presente sul sito del corso e sulla quale era già stata testata la Philips PWC.

Gli errori elencati precedentemente sono stati nuovamente riscontrati anche su questa vecchia versione della libreria VisLib. Sulla base della consultazione della relazione "WebCam USB" datata luglio 2005, abbiamo ipotizzato la possibilità di dover installare su Acer Aspire One i driver necessari al corretto funzionamento. Al fine di verificare tutto ciò rimandiamo quindi alla relazione sopra citata per tentare di risolvere il problema.

Il nostro consiglio rimane comunque quello di ricompilare il kernel di Linpus inserendo i driver a supporto della Philips PWC.

Bibliografia

[1] Bollati, Gabrielli, Donati: "Scrivere un programma per la gestione del meccanismo pan/tilt della webcam Logitech Quickcam Sphere MP sotto Linux", luglio 2008

[2] Cisamolo, Givannozzi, Nobile; Rossi: "WebCam USB", luglio 2005

[3] Dettagli tecnici della fotocamere SPC1005NC: http://www.p4c.philips.com/files/s/spc1005nc_00/spc1005nc_00_pss_ita.pdf

[4] Tutorial presente nella documentazione ufficiale scaricata insieme alla vislib-V1.9.4

Indice