OV7670 e filtro convoluzionale su Pynq-Z2

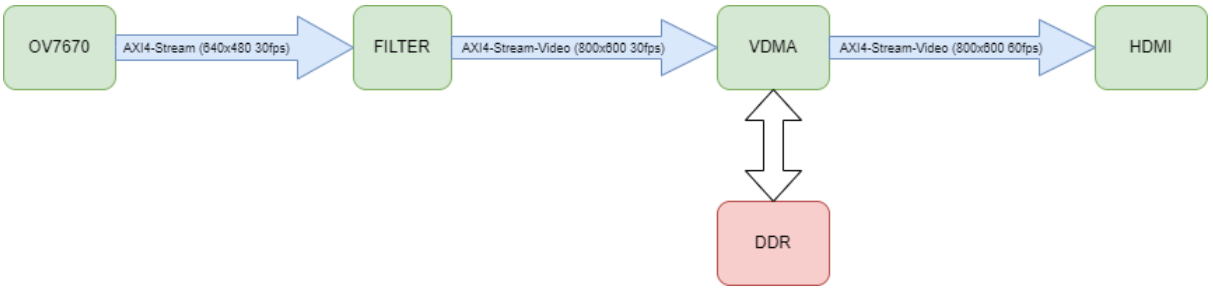
Alma Mater Studiorum - Università di Bologna
Laurea Magistrale in Ingegneria Informatica
Esame di **Sistemi Digitali M**
a.a. 2020/2021

Mazzoli Mattia - *mattia.mazzoli@studio.unibo.it*
Di Lorenzi Matteo - *matteo.dilorenzi@studio.unibo.it*

Introduzione

Il progetto implementato consiste nel porting su Pynq-Z2 di un precedente progetto[1] realizzato per Zedboard. Il progetto consisteva nell'acquisizione di uno stream video dal sensore OV7670 a cui veniva poi applicato un filtro convoluzionale, configurabile da PC attraverso interfaccia Ethernet, mostrando infine a schermo il video risultante utilizzando l'interfaccia VGA.
L'obiettivo del nostro progetto è effettuare il porting del progetto già esistente sulla board Pynq-Z2 sfruttando il più possibile le opportunità offerte dal framework Pynq. Per fare questo, alcuni IP Core come quelli per l'acquisizione delle immagini dal sensore sono stati ripresi dal progetto per Zedboard (e per questo non verranno qui descritti nel dettaglio), mentre altre parti sono state realizzate utilizzando nuovi IP Core, alcuni dei quali realizzati su Vivado HLS.
Il progetto finale prevede tutte le funzionalità presenti nel progetto originale con alcune differenze:

- Lo stream video viene acquisito dal sensore OV7670 utilizzando gli IP Core presenti nel progetto originale
- Alle immagini viene applicato un filtro convoluzionale realizzato su FPGA il quale è configurabile mediante Jupyter Notebook con Python
- Lo stream video delle immagini elaborate viene mostrato a video tramite l'interfaccia HDMI.



Lo schema qui sopra rappresenta quelli che sono i macro blocchi all'interno del progetto e come essi siano collegati tra di loro. Il blocco OV7670 si occupa di acquisire le immagini dal sensore e di trasformarle in un AXI4-Stream, il quale verrà elaborato dal blocco FILTER. Questo ha l'obiettivo di applicare il filtro convoluzionale allo stream in ingresso e di rendere lo stream compatibile per essere mostrato a video, eseguendo l'upscaling e la trasformazione da scala di grigi a RGB. Il blocco successivo, VDMA, si occupa di sincronizzare lo stream in ingresso e lo stream di uscita utilizzando un buffer di 3 frame sulla memoria DDR. Questo permette di avere in ingresso al blocco HDMI, il quale ha il compito di gestire l'uscita HDMI presente sulla board, uno stream compatibile con i blocchi al suo interno.

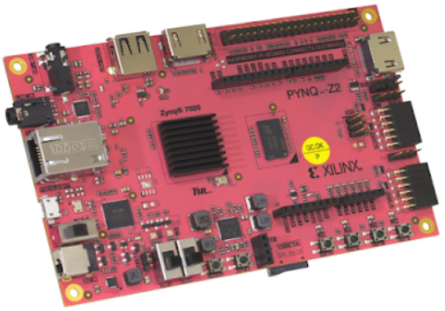
[1] https://github.com/stefano-mattoccia/SmartCamera

Strumentazione utilizzata

Dispositivi hardware

Pynq-Z2

TUL Pynq-Z2[2] è una scheda basata su Xilinx Zynq SoC che supporta il framework Pynq[3] (Python Productivity for Zynq), il quale è un progetto open-source che permette di utilizzare Python e le sue librerie unendolo ai benefici di una logica programmabile.



La board prevede una serie di periferiche che permettono di sviluppare su di essa una vasta gamma di progetti. Per il nostro progetto andremo ad utilizzare le due porte Pmod per connettere il sensore di immagine, l'uscita HDMI per la visualizzazione delle immagini, la porta Ethernet per la programmazione mediante Jupyter Notebook ed alcuni led e switch.

Sensore OV7670

Omnivision OV7670[4] è un sensore d'immagine dai costi molto ridotti. Il sensore è composto da una singola camera, capace di produrre uno stream video con risoluzione massima di 640x480 pixel a 30 fps. Il sensore supporta diversi formati di output (YUV, RGB, Raw Data) e presenta una serie di registri grazie ai quali è possibile configurare diversi parametri di acquisizione tra i quali esposizione, bilanciamento del bianco, saturazione e luminosità.

In questo progetto, il sensore è configurato per acquisire uno stream video a 30fps con risoluzione di 640x480 in formato YUV. Questo stream verrà poi elaborato e visualizzato a schermo mediante la porta HDMI presente sulla Pynq.

[2] https://www.tulembedded.com/FPGA/ProductsPYNQ-Z2.html
[3] http://www.pynq.io/home.html
[4] http://web.mit.edu/6.111/www/f2016/tools/OV7670_2006.pdf

Ambienti di sviluppo

Xilinx Vivado e Vivado HLS 2019.1

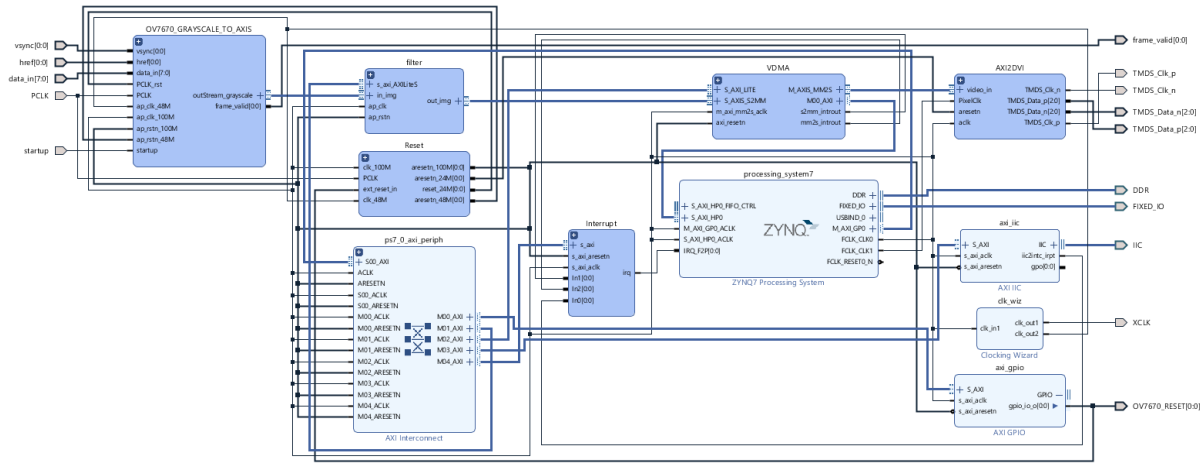
Per la realizzazione della parte software ci siamo avvalsi dei tool di sviluppo messi a disposizione da Xilinx. In particolare è stata utilizzata la versione 2019.1 di questi tool. Con Vivado HLS sono stati realizzati gli IP Core descritti in seguito, i quali sono poi stati inseriti nel progetto all'interno di Vivado. Con quest'ultimo è stato possibile connettere i vari IP in modo da creare il progetto vero e proprio, assegnare i vari pin di ingresso e uscita della board ed infine generare il bitstream necessario al Jupyter Notebook.

Jupyter Notebook

Connettendo la scheda tramite un cavo Ethernet al router della rete e collegandosi all'indirizzo http://pynq:9090 è possibile collegarsi alla piattaforma Jupyter presente sulla Pynq. Questa permette di creare documenti interattivi (testo e codice) in Python per poi eseguirli sulla board. In questo modo è possibile utilizzare il bitstream generato da Vivado per creare un overlay, il quale permetterà poi di interagire con gli IP Core presenti al suo interno e quindi di avere un'astrazione di alto livello sulla logica programmabile.

Progettazione ed implementazione

Top Level Design



Come specificato nella parte di introduzione, lo scopo del lavoro è il porting di un progetto sviluppato per Zedboard su una scheda FPGA diversa, ovvero Pynq-Z2. Essendo le due schede diverse dal punto di vista delle periferiche offerte, sono state necessarie alcune modifiche per permettere il funzionamento dell'intero progetto.
L'input dell'intera elaborazione è rappresentato dallo stream video del sensore OV7670 a risoluzione 640x480 e 30fps. Lo stream video catturato dalla camera del sensore viene elaborato applicando ad esso un filtro convoluzionale 7x7 configurabile dall'utente come nel progetto originale utilizzando però il Jupyter Notebook. Una volta applicato il filtro, lo stream video che deve essere riproposto in uscita dalla scheda è ancora in un formato non supportato dall'IP Core che gestisce l'uscita HDMI, avviene quindi uno scaling dell'immagine che viene portata alla dimensione minima supportata di 800x600 pixel. Il blocco VDMA si occupa infine di sincronizzare lo stream elaborato del sensore e lo stream in ingresso all'IP che si occupa di gestire l'uscita HDMI (*AXI2DVI*).

Nella sezione successiva saranno descritti in dettaglio solo gli IP Core implementati appositamente per questo progetto, vale a dire quelli relativi allo scaling dell'immagine (*scaleImage*), alla conversione dello stream da AXI4-Stream ad AXI4-Stream-Video (*axistream2axivideo*) e della conversione da scala di grigi in formato RGB (*gray2rgb*). La descrizione dettagliata degli altri blocchi è reperibile nella documentazione del progetto originale.

Vivado HLS

scaleImage

Come introdotto nella parte iniziale della sezione, per il corretto funzionamento del progetto è stato necessario eseguire uno scaling delle immagini generate dal sensore OV7670. Se nel progetto di partenza questa operazione non era necessaria in quanto il formato video del sensore (640x480 a 30fps) era supportato dallo standard VGA della Zedboard, la Pynq-Z2 comprende tra le sue periferiche solo HDMI, la cui risoluzione minima supportata è di 800x600.

```
1   #include "resize_image.h"
2
3   void scaleImage(AXI_STREAM& inStream, AXI_STREAM& outStream){
4       #pragma HLS INTERFACE axis port=outStream
5       #pragma HLS INTERFACE axis port=inStream
6
7       SOURCE_IMAGE inImage(HEIGHT_SOURCE, WIDTH_SOURCE);
8       OUTPUT_IMAGE outImage(HEIGHT_OUTPUT, WIDTH_OUTPUT);
9
10      #pragma HLS dataflow
11      hls::AXIvideo2Mat(inStream, inImage);
12      hls::Resize(inImage, outImage);
13      hls::Mat2AXIvideo(outImage, outStream);
14  }
```

Per eseguire lo scaling sono state utilizzate delle funzioni fornite da Vivado HLS, in particolare:

- **AXIvideo2Mat** → converte un *AXI Video* nel formato *Mat* utilizzato per rappresentare un'immagine.
- **Resize** → esegue il resizing dell'immagine nella sua rappresentazione *Mat*.
- **Mat2AXIvideo** → converte un'immagine in formato *Mat* nel rispettivo formato *AXI Video*.

Le costanti utilizzate nel codice (*HEIGHT_SOURCE*, *WIDTH_SOURCE*, *HEIGHT_OUTPUT*, *WIDTH_OUTPUT*) contengono le dimensioni rispettivamente delle immagini di input e di output. Utilizzando questa configurazione, l'IP Core potrà facilmente essere utilizzato anche in altri progetti futuri, occorrerà infatti solamente modificare il valore di queste variabili ed esportare nuovamente l'IP.

AxiStream2AxiVideo

Il blocco *AXI4-Stream to Video Out*[5] necessita che lo stream in ingresso, oltre ai segnali *data*, *valid* e *ready*, contenga anche i due segnali *last* e *user*, necessari rispettivamente per identificare il fine linea e l'inizio di un nuovo frame. Questi due segnali però non sono generati dal blocco che implementa il filtro convoluzionale, per cui si è resa necessaria la realizzazione del blocco **axistream2axivideo**.

```
if(!inStream.empty()){
    pixels++;
    inStream.read(in);
    out.data = in.data;
    if(lines == 0 && pixels == 1){
        out.user = 1;
        out.last = 0;
    }
    else if(pixels < WIDTH){
        out.user = 0;
        out.last = 0;
    } else if(pixels == WIDTH){
        out.user = 0;
        out.last = 1;
        pixels = 0;
        lines++;
        if(lines >= HEIGHT){
            lines = 0;
        }
    }

    outStream.write(out);
}
```

Il codice sopra mostrato ricava il valore dei due segnali mancanti contando semplicemente il numero di pixel arrivati. Nel momento dell'arrivo del primo pixel del nuovo frame il segnale *user* viene impostato ad 1, mentre il segnale *last* viene asserito quando viene raggiunta la fine di una linea.

gray2rgb

L'IP Core **gray2rgb** si occupa della conversione delle immagini che sono appena state elaborate dal filtro convoluzionale, andandole a portare nel formato RGB per poter essere trasmesse in uscita tramite l'HDMI della Pynq-Z2.

```
void gray2rgb(IN_STREAM &inStream, OUT_STREAM &outStream){
    #pragma HLS INTERFACE axis port=inStream
    #pragma HLS INTERFACE axis port=outStream


    IN_IMAGE inImage(HEIGHT, WIDTH);
    OUT_IMAGE outImage(HEIGHT, WIDTH);

    #pragma HLS dataflow
    hls::AXIvideo2Mat(inStream, inImage);
    hls::CvtColor<HLS_GRAY2RGB, HLS_8UC1, HLS_8UC3>(inImage, outImage);
    hls::Mat2AXIvideo(outImage, outStream);

}
```

[5] https://www.xilinx.com/support/documents/ip_documentation/v_axi4s_vid_out/v4_0/pg044_v_axis_vid_out.pdf

Oltre alle due funzioni **AXIVideo2Mat** e **Mat2AXIVideo** già illustrate nel funzionamento dell'IP Core *scaleImage*, viene utilizzata la funzione **CvtColor** fornita da Xilinx, la quale richiede come parametri la conversione da eseguire, nel nostro caso ***HLS_GRAY2RGB***, ed i formati dell'immagine di input e di output (rispettivamente 8 bit per canale con singolo canale ed 8 bit per canale con 3 canali).

Jupyter Notebook

Una volta implementato il design tramite Vivado 2019.1 e Vivado HLS, Pynq-Z2 permette la gestione tramite overlay dei blocchi utilizzati grazie alla piattaforma Jupyter. Nel progetto implementato sono stati configurati 3 blocchi:

- filtro convoluzionale
- VDMA - Video Direct Memory Access
- sensore OV7670

Per configurare tali componenti si utilizza l'overlay del progetto, il quale necessita, per essere caricato ed utilizzato, di due file particolari:

- file di bitstream con estensione *.bit* (./*nome_progetto*.runs/impl_1)
- file di hardware handoff con estensione *.hwh* (./*nome_progetto*.srcs/sources_1/bd/*nome_block_design*/hw_handoff)

Il file con estensione *.hwh* viene generato automaticamente insieme al file *.bit* da Vivado e contiene informazioni riguardanti la mappatura dei registri di memoria della scheda utilizzata per il progetto ed altre informazioni riguardanti i clock impostati.

Una volta caricati i due file richiesti e rinominati con lo stesso nome, si potrà effettivamente caricare l'overlay in memoria tramite l'apposito metodo ***Overlay*** fornito dalla libreria ***pynq***.

```
from pynq import Overlay

overlay = Overlay("design_1.bit")
```

Una volta eseguito questo passaggio, sarà possibile fin da subito visualizzare tutti i componenti presenti nel progetto e configurabili tramite Python con l'istruzione ***overlay?***.

```
IP Blocks
----------
axi_gpio             : pynq.lib.axigpio.AxiGPIO
axi_iic              : pynq.lib.iic.AxiIIC
Interrupt/axi_intc   : pynq.overlay.DefaultIP
VDMA/axi_vdma        : pynq.lib.video.dma.AxiVDMA
filter/convolution_filter : pynq.overlay.DefaultIP
```

Configurazione del filtro convoluzionale

Il filtro convoluzionale utilizza un kernel di dimensione 7x7 e può essere configurato e modificato in real time tramite Python.
Per permettere queste due operazioni è stata implementata la classe ***Convolution_Filter***, la quale prevede i seguenti metodi:

- **__init__** → inizializza le informazioni riguardanti i vari registri utilizzati dal filtro convoluzionale.
- **update_filter** → permette di applicare il filtro specificato dall'utente come parametro allo stream proveniente dal sensore. Il filtro deve rispettare una particolare formattazione, la cui descrizione dettagliata è contenuta nel progetto originale per la Zedboard.
- **print_filter** → permette, tramite la lettura dei registri utilizzati, di stampare il filtro attualmente in uso.

In seguito viene riportato un esempio di utilizzo della classe.

```
vertical_filter = [
    -1, -2, -4, 0, 4, 2, 1,
    -1, -2, -4, 0, 4, 2, 1,
    -2, -4, -6, 0, 6, 4, 2,
    -4, -6, -8, 0, 8, 6, 4,
    -2, -4, -6, 0, 6, 4, 2,
    -1, -2, -4, 0, 4, 2, 1,
    -1, -2, -4, 0, 4, 2, 1,
    120, 127]

fil = Convolution_Filter(overlay)
fil.update_filter(vertical_filter)
```

Configurazione del VDMA

Il VDMA[6] - *Video Direct Memory Access* permette di sincronizzare due AXI4-Stream-Video utilizzando la memoria DDR come buffer. Questo è molto utile nei casi in cui si renda necessario un cambio di frame rate o il frame rate in ingresso non sia stabile.
Nel nostro progetto viene utilizzato per eseguire l'aumento del frame rate dello stream video da 30fps a 60fps. Infatti, il sensore OV7670 acquisisce immagini a massimo 30 fps nel formato VGA, mentre lo standard HDMI richiede almeno il doppio del frame rate, cioè 60 fps. Nella configurazione qui utilizzata, i frame in arrivo vengono scritti all'interno di un'area di memoria della DDR e vengono successivamente letti dal canale in uscita. In questo modo, essendo il frame rate di ingresso minore di quello di uscita, l'ultimo frame scritto in memoria verrà letto più volte fino a quando un nuovo frame non sarà presente in memoria. Nel caso di passaggio da 30 a 60 fps ciascun frame verrà quindi letto due volte, salvo eventuali ritardi di arrivo dei frame in ingresso.

[6] https://docs.xilinx.com/v/u/en-US/pg020_axi_vdma

Nel Jupyter Notebook il VDMA necessita della configurazione dei due canali di lettura e scrittura, ai quali devono essere specificate le dimensioni dei frame ed il numero di bit necessari a rappresentare ogni pixel.

```
vdma = overlay.VDMA.axi_vdma

vdma.readchannel.reset()
vdma.readchannel.mode = VideoMode(800, 600, 24)
vdma.readchannel.start()

vdma.writechannel.reset()
vdma.writechannel.mode = VideoMode(800, 600, 24)
vdma.writechannel.start()

frame = vdma.readchannel.readframe() # Needed because first frame is always black

vdma.readchannel.tie(vdma.writechannel) # Connect input directly to output of vdma
```

Nella figura viene mostrata la configurazione del VDMA utilizzata nel progetto. Da notare come il primo frame venga scartato in quanto consiste sempre in un'immagine totalmente nera.

Configurazione del sensore

Per la configurazione del sensore OV7670 si è utilizzato il protocollo I^2C, che permette la comunicazione tra la board ed il sensore utilizzando solamente due fili, uno per i dati ed uno per il clock necessario alla comunicazione.
Come nel caso del filtro convoluzionale, per la configurazione e la gestione del sensore è stata creata la classe apposita ***OV7670***, la quale mette a disposizione i seguenti metodi:

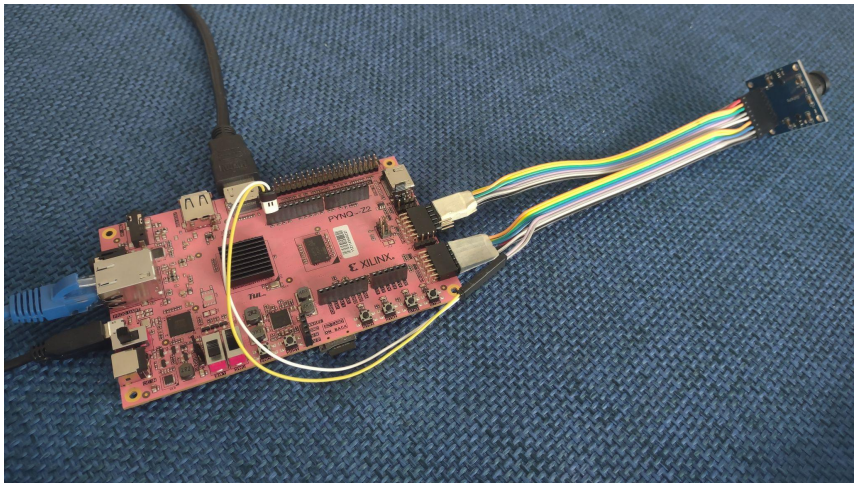
- **__init__** → inizializza i registri ed i buffer necessari per la scrittura e la lettura di dati delle comunicazioni I^2C tra la board ed il sensore.
- **write_register** → permette di scrivere un particolare dato in un determinato registro, entrambi forniti come parametri.
- **read_register** → permette di leggere il registro fornito come parametro.
- **default_setup** → contiene una configurazione di default del sensore ripresa dal progetto originale.

In seguito viene mostrato un esempio dell'utilizzo della classe, che configura il sensore con la sua configurazione di base.

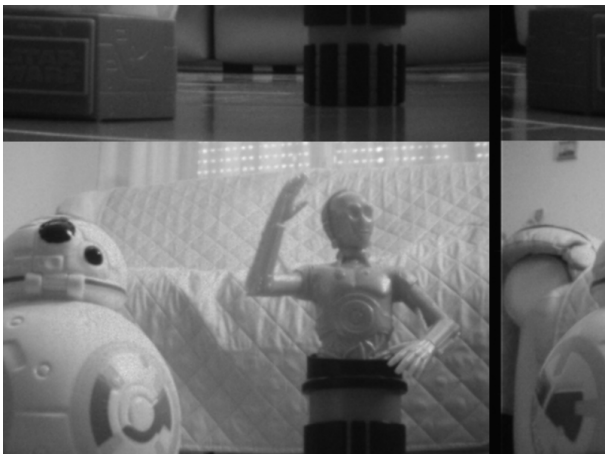
```
iic = overlay.axi_iic
ov7670 = OV7670(iic)
ov7670.default_setup()
```

Risultati ottenuti

Andremo ora ad analizzare quelli che sono stati i risultati ottenuti testando il progetto sopra descritto. Il setup utilizzato e le connessioni necessarie al funzionamento del progetto sono quelle già descritte in precedenza e mostrate qui sotto in figura.



Siamo riusciti a riprodurre il funzionamento del progetto originale per la Zedboard, raggiungendo quindi l'obiettivo prefissato. Inizialmente abbiamo riscontrato alcuni problemi con la sincronizzazione delle immagini mostrate a video, le quali apparivano a volte mal formattate con bande verticali ed orizzontali all'interno dello schermo, come mostrato dalla seguente immagine.



Questi problemi erano dovuti alla mancanza di sincronizzazione nel momento in cui, tramite il Jupyter Notebook, si andava a configurare il sensore OV7670. La configurazione causava infatti la perdita di sincronizzazione del sensore con i successivi blocchi di elaborazione.
Per risolvere tale problema abbiamo deciso di utilizzare uno switch sulla board. Inizialmente tale switch è impostato nella posizione di OFF e, solo dopo l'avvenuta configurazione, viene impostato su ON permettendo l'avvio, tramite il segnale di *start* a cui esso è collegato, del blocco relativo all'interfaccia dell'OV7670. Quest'ultimo comincia a quel punto la generazione dello stream video e la relativa sincronizzazione con i blocchi successivi.

Una volta risolto il problema abbiamo acquisito alcune immagini di test, ciascuna delle quali mostra l'applicazione di un diverso filtro convoluzionale.

La figura qui a destra mostra l'immagine originale acquisita dal sensore a cui è stato applicato un filtro convoluzionale neutro. L'immagine è poi stata utilizzata come reference per quelle mostrate in seguito.

Questa figura rappresenta l'applicazione di un filtro convoluzionale che va a sfocare l'immagine. Infatti è possibile notare come rispetto a quella precedente l'intera immagine risulti più mossa seppur acquisita nelle medesime condizioni.

La figura a fianco mostra invece l'applicazione di un filtro convoluzionale che permette di evidenziare i bordi verticali presenti all'interno dell'immagine. Anche in questo caso il risultato è coerente con quanto ci si aspettasse.