UNIVERSITÀ
DI SIENA
1240

Dipartimento di Ingegneria dell'Informazione e
Scienze Matematiche

progetto d'esame:
Laboratorio di Internet of Things

CITOFONO CON CAMERA OV7670 ED ESP8266

A cura di: **Lorenzo Moricone**, **Giovanni Fasino** e **Nicholas Redi**

lorenzo.moricone@student.unisi.it , gfasino@gmail.com , nicholasredimail@gmail.com

Anno Accademico 2018-2019

Indice

Introduzione

Nel realizzare un progetto IoT in un ambiente Smart Home è stato deciso di fare un prototipo di un citofono intelligente. Quest'oggetto ha un'applicazione abbastanza semplice nel mondo reale quanto utile.
L'idea di base è la seguente: la funzione del citofono smart la si attiva quando non si è in casa, in questo modo qualora qualcuno prema il bottone per suonare il citofono, una fotocamera connessa ad un Arduino gli scatterà una foto, che verrà inviata ad un server locale tramite un modulo Wi-Fi.
Lato server poi, attraverso le API Telegram disponibili per PHP, si azionerà un BOT che inoltrerà la foto su un canale di telegram; gli utenti iscritti a questo canale potranno quindi essere notificati (come per un qualsiasi messaggio) e vedere la foto scattata dal proprio smartphone.
L'utilità dell'oggetto risiede nel fatto di poter sapere da chi siamo cercati mentre non siamo in casa.

Hardware e software utilizzato

L'hardware utilizzato per la prototipazione di questo oggetto è composto da i seguenti elementi:

- Jumpers
- Breadboard x1
- Arduino UNO x1
- ESP8266 x1
- Camera OV7670 (no FIFO) x1
- Bottone x1
- Led x1
- Resistore 10 kΩ x2
- Resistore 4.7 kΩ x2
- Resistore 8.2 kΩ x1
- Resistore 220 Ω x1

Mentre per la parte software è necessario avere installato e configurato i seguenti servizi[1]:

- Arduino IDE
- Apache
- php
- php-curl

[1]nota: per la realizzazione dell'intero progetto si suppone di lavorare in ambienti Linux (altri ambienti è possibile utilizzarli con le dovute accortezze)

OV7670 con Arduino UNO

La prima fase della progettazione riguarda la connessione della camera con l'Arduino, affinchè essa sia in grado di trasmettere i dati alla porta sriale. Il chip OV7670 è composto da 18 pin, prevede una tensione di alimentazione di 3.3 V e comunica attraverso il protocollo seriale SCCB (Serial Camera Control Bus), un protocollo seriale proprietario di *OmniVision Technologies Inc.* che è basato sul protollo I2C.
In figura 1 e 2 sono mostrati il chip e la relativa descrizione dei pin.

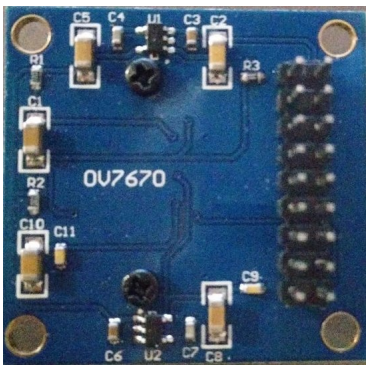

Figura 1: OV7670 chip.

Pin	Type	Description
VDD**	Supply	Power supply
GND	Supply	Ground level
SDIOC	Input	SCCB clock
SDIOD	Input/Output	SCCB data
VSYNC	Output	Vertical synchronization
HREF	Output	Horizontal synchronization
PCLK	Output	Pixel clock
XCLK	Input	System clock
D0-D7	Output	Video parallel output
RESET	Input	Reset (Active low)
PWDN	Input	Power down (Active high)

Figura 2: OV7670 pins description.

Per questo primo interfacciamento si può sfruttare un precedente lavoro[2] pubblicato da "*ComputerNerd*". Grazie ad esso è possibile collegare il modulo OV7670 ad Arduino UNO con uno schema ben preciso, ed andare programmare il microcontrollore ATmega328 di Arduino servendosi di una libreria sviluppata ad-hoc. In particolare, tale libreria permette di resettare e impostare (in accordo al datasheet[3]) i registri della fotocamera per acquisire

[2] `https://github.com/ComputerNerd/ov7670-no-ram-arduino-uno`
[3] `https://www.voti.nl/docs/OV7670.pdf`

immagini con risoluzione QVGA (320x240) e codifica del colore YUV422 (in scala di grigi a 8 bit).

Connettere i pezzi

Lo schematico prevede i **collegamenti mostrati in figura 3**.

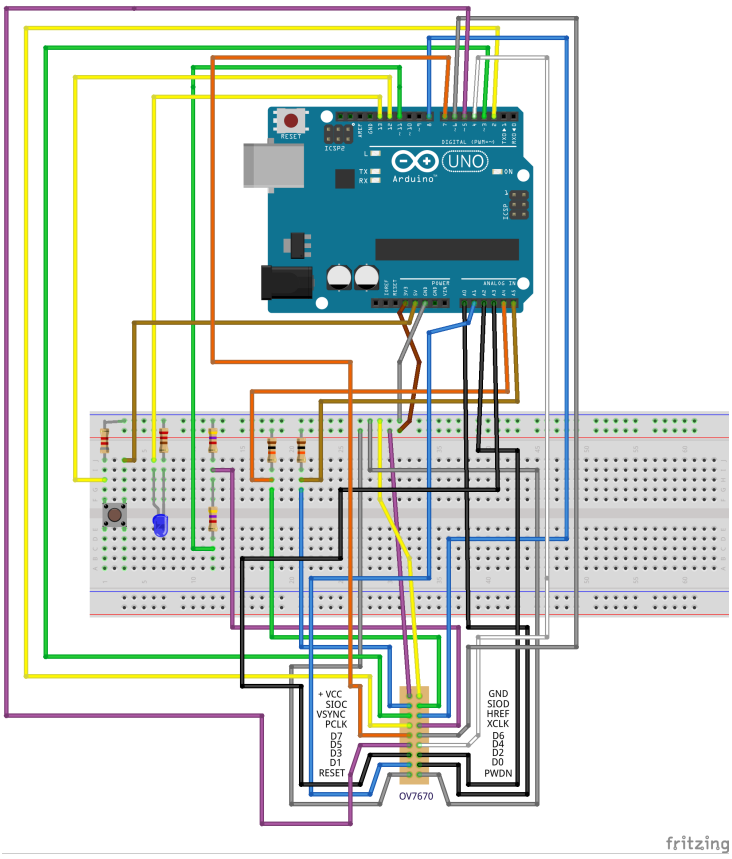

Figura 3: Connettere Arduino UNO alla fotocamera OV7670.

XCLK

Il modulo fotocamera richiede in ingresso un clock di sistema sul pin XCLK. Questo clock è essenziale per il funzionamento dell'interfaccia SCCB e per qualsiasi altra operazione della fotocamera. Secondo il datasheet, la frequenza su XCLK dovrebbe essere compresa tra 10 e 48 MHz, tuttavia la

camera funziona correttamente anche con un clock a 8 MHz.
Infatti, in questa configurazione, tale clock è generato con un segnale PWM da 8 MHz dal pin D11 dell'Arduino e, poichè il chip non sopporta voltaggi superiori ai 3.3 V, è stato inserito un partitore di tensione per assicurarsi che il voltaggio in ingresso al modulo OV7670 non superi i 3 V.

SIOC e SIOD

Ogni comunicazione seriale I2C è costituita da due segnali: SCL (Serial Clock) e SDA (Serial Data). Per la fotocamera questi ruoli sono ricoperti rispettivamente dai pin SIOC e SIOD.
Dato che Arduino lavora a 5 V mentre il modulo OV7670 lavora a 3.3 V, per rendere possibile la comunicazione I2C c'è bisogno di due resistori di pull-up sul più basso dei due voltaggi (3.3 V). In questa maniera, quando non sono attive le linee SCL e SDA (attive basse), il minimo livello di tensione che ci potrà essere sarà 3.3 V, livello che Arduino riconoscerà come HIGH (e quindi non attiverà la linea).

VSYNC, HREF e PCLK

Il chip invia i dati in un formato sincrono parallelo. Dopo che è stato applicato un segnale di clock al pin XCLK, la fotocamera inizierà a pilotare i suoi pin VSYNC, HREF e D0-D7. Il fronte di discesa di VSYNC segnala l'inizio di un frame e il suo fronte di salita segnala la fine. Ogni frame QVGA ha 240 righe e ogni riga ha 320 pixel. Ogni dato di riga deve essere catturato durante lo stato alto di HREF, cioè D0-D7 devono essere campionati solo quando HREF è alto. Il fronte di salita di HREF segnala l'inizio di una riga e il fronte di discesa di HREF segna la fine.
In fugura 4 è mostrata la temporizzazione.

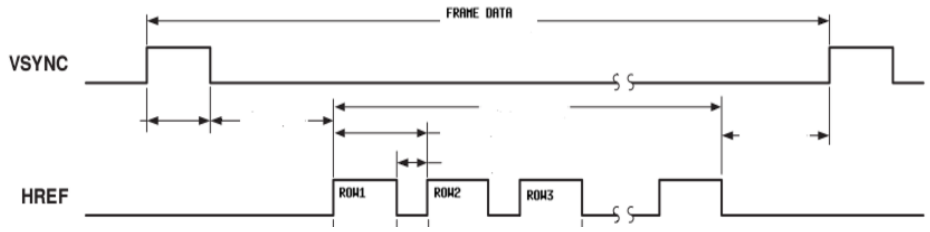

Figura 4: Temporizzazione durante l'acquisizione di un frame.

In particolare, D0-D7 devono essere campionati sul fronte di salita del segnale PCLK. La frequenza del segnale PCLK determinerà quindi, quanto

velocemente vengono acquisiti i byte di una riga (di default PCLK avrà la stessa frequenza di XCLK).
In figura 5 sono mostrate le tempistiche da rispettare.

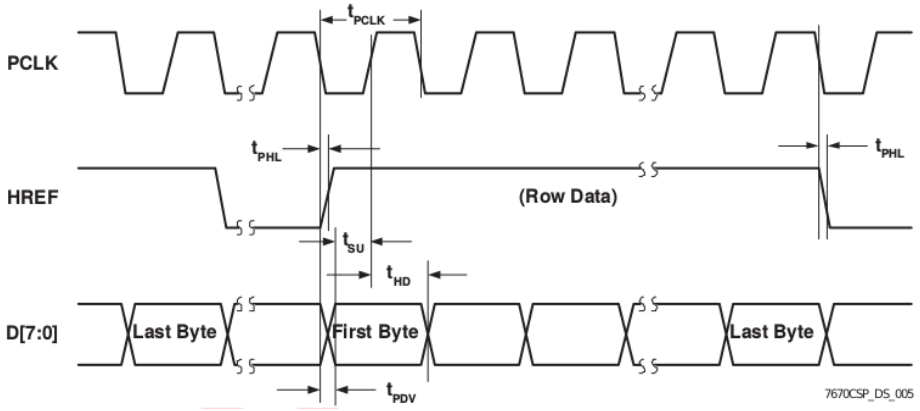

Figura 5: Temporizzazione durante l'acquisizione di una riga.

LED e BOTTONE

La logica che sta dietro il led ed il bottone è molto semplice.
Il bottone simula il tasto per suonare il citofono, quindi la camera inizierà l'acquisizione di un'immagine solo dopo che esso è stato premuto.
Il led, invece, lo si può interpretare come un debug visivo: quando il bottone viene schiacciato, il led starà acceso per 3 secondi, seguirà un cortissimo blink e tornerà di nuovo acceso, questo segnala l'inizio dell'acquisizione effettiva dell'immagine; per tutto il tempo di acquisizione il led rimarrà acceso, dopodichè si spengerà; circa 4 secondi dopo che l'acquisizione è terminata ci sarà un cortissimo blink che segnalerà la possibilità di acquisire una nuova immagine.

Lo sketch

Per quanto riguarda la programmazione del microcontrollore ATmega328 è sufficiente fare l'upload, tramite la IDE Arduino, dello **sketch a pagina 14**.
Si può notare come nello sketch è incluso il file di libreria "ov7670.h" (il cui **codice** è **a pagina 16**), necessario per l'impostazione di tutti i registri della fotocamera.

Analizzando il codice dello sketch secondo la logica di Arduino, in fase di `setup()` si hanno 4 chiamate a funzioni con le seguenti funzionalità:

- `arduinoUnoInit()` : imposta il segnale di clock per XLCK sul pin 11, imposta la comunicazione con protocollo I2C a 100 kHz ed abilita la porta seriale con baud rate 1Mbps;
- `camInit()` : reimposta ai valori predefiniti tutti i registri del chip;
- `setRes()` : imposta la risoluzione con la quale acquisire l'immagine;
- `setColor()` : imposta la codifica dei colori;

successivamente, vengono definiti gli usi dei pin 12 e 13 (rispettivamente per il bottone, INPUT, e per il led, OUTPUT); ed infine si ha un cortissimo blink del led, per segnalare che la fase di setup è terminata e che la fotocamera è pronta per acquisire un'immagine.

Nel `loop()`, semplicemnete, si usano due variabili (`lettura` ed `old_lettura`) per controllare lo stato di pressione del pulsante, ed una volta che esso viene premuto, dopo esserci stati i blink descritti prima, si invoca la funzione `captureImg()`, la quale si occupa della logica di acquisizione dell'immagine.
La logica di acquisizione funziona così: viene inviata una sequenza di 5 caratteri `'*','R','D','Y','*'` che rappresenta il comando con il quale l'ESP8266 sarà in grado di sapere qual'è il primo byte che dovrà memorizzare come dato dell'immagine, e poi si inizia a "ciclare" su 240 righe e 320 colonne con le tempistiche definite da VSYNC, HREF e PCLK. I byte da trasmettere alla porta seriale sono preparati sul registro di Arduino `UDR0`, e poi trasmessi non appena passa il segnale PCLK.
La cosa di rilievo per questa applicazione, è che, siccome l'ESP8266 ha una quantità di RAM utilizzabile che non supera i 50 kB, si deve effettuare un ritaglio dell'immagine, ottenendo un'immagine 160x120. Utilizzando un byte per pixel (scala di grigio da 0 a 255), si avrebbe un peso dell'immagine in formato QVGA pari a $320 \cdot 240\ bytes = 76800\ kB$ e di conseguenza l'ESP non potrebbe ospitarla per intero prima di inviarla, dopo un ritaglio centrale di 160x120 l'immagine peserà $160 \cdot 120\ bytes = 19200\ kB$, così l'ESP sarà in grado di mantenerla in RAM e spezzettarla in più pacchetti per poi inviarla al server.

ESP8266: comunicazione seriale ed invio al server

Dopo che l'Arduino è in grado di acquisire l'immagine e di trasmetterla alla porta seriale correttamente, quello che si deve fare è collegare il pin TX di Arduino al pin RX del chip ESP8266.
In figura 6 è mostrato lo schematico finale.

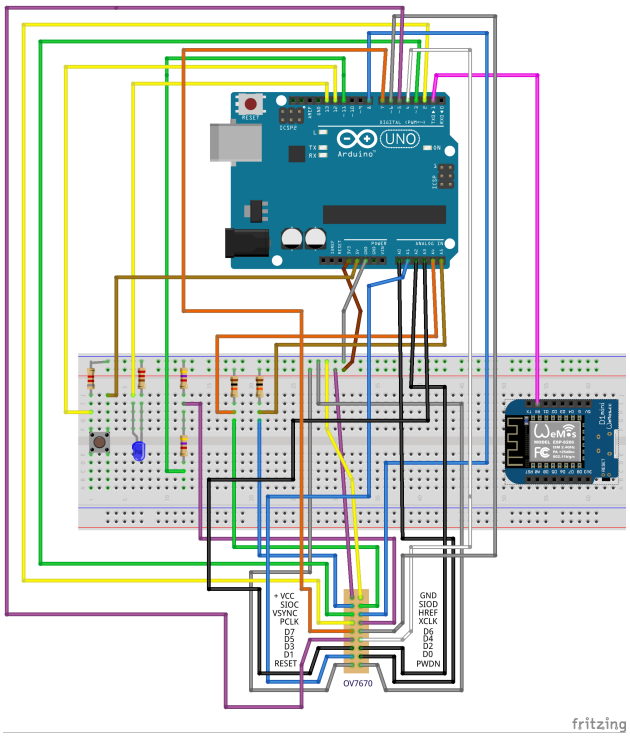

Figura 6: Schema finale.

Così facendo, Arduino UNO abiliterà la porta seriale in trasmissione verso l'ESP e, ricordando che il baud rate è stato impostato da Arduino a 1Mbps,

la comunicazione seriale da attivare lato ESP, dovrà essere inizializzata alla stessa frequenza.
Il **codice** dello sketch per l'ESP è **a pagina 21**.
Nel `setup()` viene inizializzata la comunicazione seriale a 1 Mbps e creato l'oggetto `WiFiMulti` per la connessione alla rete WiFi.
Nel `loop()` non si procederà oltre il primo controllo, finchè il modulo ESP non risulterà connesso alla rete (può richiedere alcuni secondi). Una volta connesso, l'ESP inizierà a controllare se ci sono dati in arrivo sulla porta seriale, cercando di rilevare la sequenza di comando `'*','R','D','Y','*'`. Quando sono stati letti i 5 bytes consecutivamente, la variabile `photoStarted` diventerà true, e da quel momento, ogni altro byte in arrivo alla seriale sarà un byte che rappresenta un pixel dell'immagine. E' importante che i bytes dell'immagine siano letti dall'ESP il più velocemente possibile, cioè non sono ammesse molte altre operazioni "fattibili" dall'ESP mentre questo sta leggendo in sequenza i bytes, questo perchè, essendo il baud rate molto elevato, ogni altra operazione che farebbe "perdere tempo" all'ESP rischia di essere abbastanza lunga da far intasare il buffer di comunicazione seriale; ciò provocherebbe problemi di sincronia tra i due dispositivi e non sarebbe possibile acquisire tutti i bytes dell'immagine correttamente. Inoltre, c'è da dire che è proprio questo il motivo per cui l'immagine va ritagliata fino a farla entrare nella RAM dell'ESP: si potrebbe pensare di inviare l'immagine a pezzetti proprio nel mentre si stanno acquisendo i dati dall'Arduino, il fatto è, che inviare i dati sul web è un operazione time-consuming, la sincronizzazione sarebbe compromessa. E' quindi fondamentale riuscere a salvare completamente l'immagine prima di inviarla a pezzetti.
Mentre si acquisiscono i dati dell'immagine, si può notare nel codice, che alcuni livelli di grigio acquisiti dalla fotocamera, vengono manipolati (incrimentati di 1 o di 2); questo è necessario affinchè, nell'invio dei pacchetti al server, la stringa che si costruisce come dato del protocollo POST, non sia tagliata involontariamente. Per cui, quando il livello di grigio è pari a uno dei valori tra 0, 9, 10, 13, 37 e 38 si cambierà il valore di 1 (2 nel caso di 9 e 37) e la stringa che rappresenterà il dato del pacchetto da trasmettere, non creerà problemi. Tanto 1 o 2 punti di differenza, in una scala di grigio tra 0 e 255, e, per altro solo in alcuni pixels, non distorceranno la visibilità dell'immagine ad occhio nudo.
Quando il buffer dei dati dell'immagine sarà riempito (`i == BUFFER_SIZE == 19200`), e quindi l'immagine è in quel momento salvata comletamente in RAM (in formato grezzo), si passerà all'invio di essa tramite richieste http al server locale (`http://localhost/PHPIoT/`). Viene diviso il buffer in sotto-buffer di 2400 bytes e inviati 8 pacchetti con il dato incapsulato sotto forma di stringa. Le operazioni in sequenza, eseguite dal momento che

l'immagine è salvata, sono le seguenti:

1. pulizia del file `img.txt` (memorizzato nel server) in caso di foto precedentemente acquisita: chiamata GET al file `initphoto.php` ;

2. invio degli 8 pacchetti da 2400 bytes, appendendoli uno dietro l'altro al file `img.txt`, tramite richieste http POST al file `dato.php` ;

3. concatenazione dell'header BMP (`head160x120.bmp`) al file grezzo `img.txt` per costruire l'immagine finale (apribile con un visualizzatore di foto), ed invio dell'immagine sul canale Telegram: chiamata GET al file `sendphotoAus.php` .

Successivamente a queste tre fasi, l'ESP viene resettato per assicurarsi che l'acquisizione di foto successive non occupi tutte le aree di memoria disponibili. Con il reset, i buffer vengono riallocati nuovamente, ed almeno un'acquisizione potrà sempre andare a buon fine. Il `delay()` di 4 secondi alla fine del `loop()` di Arduino UNO è dovuto proprio a questo tempo, necessario per l'ESP a resettarsi.

Server locale e PHP

I codici PHP utilizzati dal server sono visibili tutti nella sezione "**Codici**" a partire da **pagina 24**.
Come già detto, la fase di invio dell'ESP si può suddividere in 3 diverse fasi: la prima fase, riguarda la pulizia del file che deve contenere i dati della foto; nella seconda fase, ricorrono più chiamate ad uno stesso file che si occupa di costruire il file dell'immagine grezza; e la terza fase, in cui si costruisce l'immagine finale e poi la si invia sul canale Telegram.
La pulizia del file avviene nel file `initphoto.php`, semplicemente eseguendo da PHP un comando di sistema Linux: se il file `img.txt` esiste, viene svuotato (se è pieno), altrimenti viene creato vuoto.
Nel file `dato.php` si estrapola il dato dal metodo POST, ci si assicura che il file `img.txt` esista e sia scrivibile, e poi gli si appende la stringa di bytes.
Il file `sendphotoAus.php` è solamente un file ausiliario che, tramite un comando di sistema, semplicemente richiama il file `sendphoto.php`. Questo è dovuto a ragioni di sicurezza, in quanto è necessario che il file che si occuperà di sfruttare le API Telegram per inviare la foto, dovrà essere chiamato dal server stesso, e non da fonti esterne.

Channel e BOT Telegram

Prima di descrivere il file `sendphoto.php` è necessario spiegare cosa va fatto dall'account Telegram per essere in grado di utilizzare le API in PHP.
Il primo passo è quello di **creare un BOT** sfruttando un'altro BOT messo a disposizione da Telegram con lo username @BotFather. Seguendo le istruzioni di questo BOT sarà possibile creare un proprio BOT personale. Con la creazione, @BotFather ci fornirà ache il **token del BOT** da utilizzare con le API http.
Poi bisognerà **creare un canale** (pubblico) ed **aggiungere** ad esso **il BOT** personalmente creato, nel ruolo di amministratore del canale per far si che sia in grado di inviare messaggi. Per recuperare l'**ID della chat del canale** basta accedere a Telegram Web e cliccare sulla chat del canale, l'ID della chat

è visibile nell'URL del browser dopo il simbolo @. Dopo aver recuperato l'ID della chat del canale è possibile impostare il canale come privato, in modo tale da poter invitare solo le persone interessate.

sendphoto.php

La prima operazione in questo file è quella di finalizzare l'immagine. Lo si fa con due comandi di sistema, che concatenano un header BMP fisso (vedi pagina 25) con il file `img.txt` contenente i valori di ogni pixel.
Dopo si costruiscono un pò di variabili che saranno passate come parametri nel protocollo di invio eseguito tramite il tool *curl*, un tool che incapsula tutti i parametri necessari per la cumunicazione e la esegue in back-end senza interazione dell'utente. Tra le variabili abbiamo la `chat_id` del canale ed il `bot_url` con il token del BOT per usarlo tramite le API di Telegram, poi si costruisce l'URL completo in cui si specifica quale funzione usare (`sendPhoto?chat_id=`), infine, abbiamo un array che associa la foto da inviare al canale.
Con il comando `curl_init()` viene istanziata una risorsa cURL, e poi con i comandi `set_opt` vengono impostate tutte le opzioni per la connessione con i parametri inizializzati.
Alla fine con `curl_exec` si esegue la risorsa costruita e, salvo errori, l'utente iscritto al canale Telegram, dovrebbe ricevere una notifica di messaggio dal canale, con la foto scattata.

Conclusioni e possibili sviluppi

Nel realizzare questo progetto ci siamo serviti di componenti estremamente economiche. Andando a valutare il costo complessivo dell'oggetto (10 $ circa, se al posto dell'Arduino UNO sostituiamo la versione standalone: microcontrollore ATmega328, resistore, due condensatori e oscillatore al quarzo), si può affermare che un prototipo del genere è più che adeguato allo scopo citato nell'introduzione. Se l'obbiettivo è riconoscere la persona che ha suonato il citofono, questo lo si fa anche con un'immagine, non di eccellente qualità, come la seguente (acquisita direttamente dal prototipo finalizzato).

Figura 7: Immagine acquisita dall'oggetto creato.

Sarebbe possibile ottenere dei miglioramenti significativi andando a sostituire la fotocamera OV7670 non FIFO con un'altra fotocamera dotata di buffer interno (FIFO). Questo permetterebbe di rimuovere Arduino UNO, che in questo caso funziona come driver della fotocamera, ed il modello FIFO potrebbe essere collegato direttamente sul chip ESP8266; si avrebbe così una semplificazione nell'architettura.
Altra considerazione riguarda la parte del server: si è usato qui un server esterno per inviare la foto ai server di Telegram, il motivo di ciò è dovuto al fatto che, il protocollo HTTP MultipartData (necessario per l'invio segmentato di dati superiori ai 2 kB) non è implementato nell'ESP8266. Si potrebbe quindi, semplificare ulteriormente l'architettura con un chip ESP più potente e che abbia implementato questo protocollo. Così facendo, sarebbe rimossa la parte del server esterno e la foto sarebbe inviata su Telegram direttamente dal chip.

Codici

Arduino UNO

```
#include <stdint.h>
#include <avr/io.h>
#include <util/twi.h>
#include <util/delay.h>
#include <avr/pgmspace.h>
#include "ov7670.h"

void setColor(void) {
  wrSensorRegs8_8(yuv422_ov7670);
}

void setRes(void) {
  wrReg(REG_COM3, 4); // REG_COM3 enable scaling
  wrSensorRegs8_8(qvga_ov7670);
  wrReg(0x11, 11); // PCLK prescaler
}

void camInit(void) {
  wrReg(0x12, 0x80);
  _delay_ms(100);
  wrSensorRegs8_8(ov7670_default_regs);
  wrReg(REG_COM10, 32);//PCLK does not toggle on HBLANK.
}

void arduinoUnoInit(void) {
  cli();//disable interrupts

  /* Setup the 8mhz PWM clock
    This will be on pin 11*/
  DDRB |= (1 << 3);//pin 11
  ASSR &= ~(_BV(EXCLK) | _BV(AS2));
  TCCR2A = (1 << COM2A0) | (1 << WGM21) | (1 << WGM20);
  TCCR2B = (1 << WGM22) | (1 << CS20);
  OCR2A = 0;//(F_CPU)/(2*(X+1))
  DDRC &= ~15;//low d0-d3 camera
  DDRD &= ~252;//d7-d4 and interrupt pins
  _delay_ms(3000);

  //set up twi for 100khz
  TWSR &= ~3;//disable prescaler for TWI
  TWBR = 72;//set to 100khz

  //enable serial
  UBRR0H = 0;
  UBRR0L = 1;//0 = 2M baud rate. 1 = 1M baud. 3 = 0.5M. 7 = 250k 207 is 9600 baud rate.
  UCSR0A |= 2;//double speed aysnc
  UCSR0B = (1 << RXEN0) | (1 << TXEN0);//Enable receiver and transmitter
  UCSR0C = 6;//async 1 stop bit 8bit char no parity bits
}


void StringPgm(const char * str) {
  do {
    while (!(UCSR0A & (1 << UDRE0)));//wait for byte to transmit
    UDR0 = pgm_read_byte_near(str);
    while (!(UCSR0A & (1 << UDRE0)));//wait for byte to transmit
  } while (pgm_read_byte_near(++str));
}

static void captureImg(){
```

```
  uint16_t y, x;

  StringPgm(PSTR("*RDY*"));

  while (!(PIND & 8));//wait for high
  while ((PIND & 8));//wait for low

    y = 240;
  while (y--){
        x = 320;
      //while (!(PIND & 256));//wait for high
    while (x--){
      while ((PIND & 4));//wait for low
      if( (y >= 60) && ( y < 180 ) && (x >= 80) && ( x < 240 ) ){
          UDR0 = (PINC & 15) | (PIND & 240);
          while (!(UCSR0A & (1 << UDRE0)));//wait for byte to transmit
      }
      while (!(PIND & 4));//wait for high
      while ((PIND & 4));//wait for low
      while (!(PIND & 4));//wait for high
    }
    //  while ((PIND & 256));//wait for low
  }
    _delay_ms(100);
}

void setup(){
  arduinoUnoInit();
  camInit();
  setRes();
  setColor();

  pinMode(13, OUTPUT); // led for visual debugging
  pinMode(12, INPUT); // simulate the door phone button

  // a small flash after which the interphone can be played
  digitalWrite(13, HIGH);
  _delay_ms(200);
  digitalWrite(13, LOW);
}


int lettura = 0, old_lettura = 0;

void loop(){

  lettura = digitalRead(12); // read button status

  // capture the image only if the button come pressed and not if it was remained pressed
  if (old_lettura != lettura && old_lettura == 0) {

    digitalWrite(13, HIGH); // high when the button come pressed, wait 3 seconds
    _delay_ms(3000);
    digitalWrite(13, LOW); // little blink when the photo starts
    _delay_ms(200);

    digitalWrite(13, HIGH); // led high while capturing
    captureImg();
    digitalWrite(13, LOW); // led low when finished

    _delay_ms(4000); // wait 4 seconds before strat a new capturing
    digitalWrite(13, HIGH); // little blink to signal that it is possible to start a new capturing
    _delay_ms(200);
    digitalWrite(13, LOW);
  }

  old_lettura = lettura; // old_lettura remains high until the next time that lettura come back high, double
        read avoided if the button didn't come released
  delay(200);
}
```

ov7670.h

```
/* Registers */
#define REG_GAIN    0x00  /* Gain lower 8 bits (rest in vref) */
#define REG_BLUE    0x01  /* blue gain */
#define REG_RED       0x02  /* red gain */
#define REG_VREF    0x03  /* Pieces of GAIN, VSTART, VSTOP */
#define REG_COM1    0x04  /* Control 1 */
#define COM1_CCIR656  0x40    /* CCIR656 enable */
#define REG_BAVE    0x05  /* U/B Average level */
#define REG_GbAVE   0x06  /* Y/Gb Average level */
#define REG_AECHH   0x07  /* AEC MS 5 bits */
#define REG_RAVE    0x08  /* V/R Average level */
#define REG_COM2    0x09  /* Control 2 */
#define COM2_SSLEEP         0x10  /* Soft sleep mode */
#define REG_PID           0x0a  /* Product ID MSB */
#define REG_VER           0x0b  /* Product ID LSB */
#define REG_COM3    0x0c  /* Control 3 */
#define COM3_SWAP         0x40  /* Byte swap */
#define COM3_SCALEEN          0x08  /* Enable scaling */
#define COM3_DCWEN          0x04  /* Enable downsamp/crop/window */
#define REG_COM4    0x0d  /* Control 4 */
#define REG_COM5    0x0e  /* All "reserved" */
#define REG_COM6    0x0f  /* Control 6 */
#define REG_AECH    0x10  /* More bits of AEC value */
#define REG_CLKRC   0x11  /* Clocl control */
#define CLK_EXT           0x40  /* Use external clock directly */
#define CLK_SCALE   0x3f  /* Mask for internal clock scale */
#define REG_COM7    0x12  /* Control 7 */ //REG mean address.
#define COM7_RESET          0x80  /* Register reset */
#define COM7_FMT_MASK         0x38
#define COM7_FMT_VGA          0x00
#define COM7_FMT_CIF          0x20  /* CIF format */
#define COM7_FMT_QVGA         0x10  /* QVGA format */
#define COM7_FMT_QCIF         0x08  /* QCIF format */
#define COM7_RGB          0x04  /* bits 0 and 2 - RGB format */
#define COM7_YUV          0x00  /* YUV */
#define COM7_BAYER          0x01  /* Bayer format */
#define COM7_PBAYER         0x05  /* "Processed bayer" */
#define REG_COM8    0x13  /* Control 8 */
#define COM8_FASTAEC          0x80  /* Enable fast AGC/AEC */
#define COM8_AECSTEP          0x40  /* Unlimited AEC step size */
#define COM8_BFILT    0x20  /* Band filter enable */
#define COM8_AGC    0x04  /* Auto gain enable */
#define COM8_AWB    0x02  /* White balance enable */
#define COM8_AEC    0x01  /* Auto exposure enable */
#define REG_COM9    0x14  /* Control 9- gain ceiling */
#define REG_COM10   0x15  /* Control 10 */
#define COM10_HSYNC         0x40  /* HSYNC instead of HREF */
#define COM10_PCLK_HB         0x20  /* Suppress PCLK on horiz blank */
#define COM10_HREF_REV          0x08  /* Reverse HREF */
#define COM10_VS_LEAD         0x04  /* VSYNC on clock leading edge */
#define COM10_VS_NEG          0x02  /* VSYNC negative */
#define COM10_HS_NEG          0x01  /* HSYNC negative */
#define REG_HSTART    0x17  /* Horiz start high bits */
#define REG_HSTOP   0x18  /* Horiz stop high bits */
#define REG_VSTART    0x19  /* Vert start high bits */
#define REG_VSTOP   0x1a  /* Vert stop high bits */
#define REG_PSHFT   0x1b  /* Pixel delay after HREF */
#define REG_MIDH    0x1c  /* Manuf. ID high */
#define REG_MIDL    0x1d  /* Manuf. ID low */
#define REG_MVFP    0x1e  /* Mirror / vflip */
#define MVFP_MIRROR         0x20  /* Mirror image */
#define MVFP_FLIP   0x10  /* Vertical flip */
#define REG_AEW           0x24  /* AGC upper limit */
#define REG_AEB           0x25    /* AGC lower limit */
#define REG_VPT           0x26  /* AGC/AEC fast mode op region */
#define REG_HSYST   0x30  /* HSYNC rising edge delay */
#define REG_HSYEN   0x31  /* HSYNC falling edge delay */
#define REG_HREF    0x32  /* HREF pieces */
#define REG_TSLB    0x3a  /* lots of stuff */
#define TSLB_YLAST    0x04  /* UYVY or VYUY - see com13 */
#define REG_COM11   0x3b  /* Control 11 */
#define COM11_NIGHT         0x80  /* NIght mode enable */
#define COM11_NMFR          0x60  /* Two bit NM frame rate */
#define COM11_HZAUTO          0x10  /* Auto detect 50/60 Hz */
#define COM11_50HZ          0x08  /* Manual 50Hz select */
#define COM11_EXP   0x02
#define REG_COM12   0x3c  /* Control 12 */
#define COM12_HREF          0x80  /* HREF always */
#define REG_COM13   0x3d  /* Control 13 */
#define COM13_GAMMA         0x80  /* Gamma enable */
#define COM13_UVSAT         0x40  /* UV saturation auto adjustment */
```

```
#define COM13_UVSWAP          0x01  /* V before U - w/TSLB */
#define REG_COM14   0x3e  /* Control 14 */
#define COM14_DCWEN         0x10  /* DCW/PCLK-scale enable */
#define REG_EDGE    0x3f  /* Edge enhancement factor */
#define REG_COM15   0x40  /* Control 15 */
#define COM15_R10F0         0x00  /* Data range 10 to F0 */
#define COM15_R01FE         0x80  /*      01 to FE */
#define COM15_R00FF         0xc0  /*      00 to FF */
#define COM15_RGB565          0x10  /* RGB565 output */
#define COM15_RGB555          0x30  /* RGB555 output */
#define REG_COM16   0x41  /* Control 16 */
#define COM16_AWBGAIN         0x08  /* AWB gain enable */
#define REG_COM17   0x42  /* Control 17 */
#define COM17_AECWIN          0xc0  /* AEC window - must match COM4 */
#define COM17_CBAR          0x08  /* DSP Color bar */
/*
  This matrix defines how the colors are generated, must be
  tweaked to adjust hue and saturation.

  Order: v-red, v-green, v-blue, u-red, u-green, u-blue
  They are nine-bit signed quantities, with the sign bit
  stored in0x58.Sign for v-red is bit 0, and up from there.
*/
#define REG_CMATRIX_BASE  0x4f
#define CMATRIX_LEN           6
#define REG_CMATRIX_SIGN  0x58
#define REG_BRIGHT    0x55  /* Brightness */
#define REG_CONTRAS         0x56  /* Contrast control */
#define REG_GFIX    0x69  /* Fix gain control */
#define REG_REG76   0x76  /* OV's name */
#define R76_BLKPCOR         0x80  /* Black pixel correction enable */
#define R76_WHTPCOR         0x40  /* White pixel correction enable */
#define REG_RGB444          0x8c  /* RGB 444 control */
#define R444_ENABLE         0x02  /* Turn on RGB444, overrides 5x5 */
#define R444_RGBX   0x01  /* Empty nibble at end */
#define REG_BD50MAX         0xa5  /* 50hz banding step limit */
#define REG_BD60MAX         0xab  /* 60hz banding step limit */
#define REG_RED           0x02  /* red gain */
#define COM1_CCIR656          0x40  /* CCIR656 enable */
#define COM7_RGB    0x04  /* bits 0 and 2 - RGB format */
#define COM7_YUV    0x00  /* YUV */
#define COM10_VS_LEAD           0x04  /* VSYNC on clock leading edge */
#define REG_AEB           0x25  /* AGC lower limit */
#define CMATRIX_LEN             6
#define REG_HAECC1    0x9f  /* Hist AEC/AGC control 1 */
#define REG_HAECC2    0xa0  /* Hist AEC/AGC control 2 */
#define REG_HAECC3    0xa6  /* Hist AEC/AGC control 3 */
#define REG_HAECC4    0xa7  /* Hist AEC/AGC control 4 */
#define REG_HAECC5    0xa8  /* Hist AEC/AGC control 5 */
#define REG_HAECC6    0xa9  /* Hist AEC/AGC control 6 */
#define REG_HAECC7    0xaa  /* Hist AEC/AGC control 7 */
#define MTX1            0x4f  /* Matrix Coefficient 1 */
#define MTX2            0x50  /* Matrix Coefficient 2 */
#define MTX3            0x51  /* Matrix Coefficient 3 */
#define MTX4            0x52  /* Matrix Coefficient 4 */
#define MTX5            0x53  /* Matrix Coefficient 5 */
#define MTX6            0x54  /* Matrix Coefficient 6 */
#define MTXS            0x58  /* Matrix Coefficient Sign */
#define AWBC7           0x59  /* AWB Control 7 */
#define AWBC8           0x5a  /* AWB Control 8 */
#define AWBC9           0x5b  /* AWB Control 9 */
#define AWBC10            0x5c  /* AWB Control 10 */
#define AWBC11            0x5d  /* AWB Control 11 */
#define AWBC12            0x5e  /* AWB Control 12 */
#define REG_GFI           0x69  /* Fix gain control */
#define GGAIN           0x6a  /* G Channel AWB Gain */
#define DBLV            0x6b
#define AWBCTR3           0x6c  /* AWB Control 3 */
#define AWBCTR2           0x6d  /* AWB Control 2 */
#define AWBCTR1           0x6e  /* AWB Control 1 */
#define AWBCTR0           0x6f  /* AWB Control 0 */

struct regval_list {
  uint8_t reg_num;
  uint16_t value;
};

const struct regval_list qvga_ov7670[] PROGMEM = {
  { REG_COM14, 0x19 },
  { 0x72, 0x11 },
  { 0x73, 0xf1 },

  { REG_HSTART, 0x16 },
```

```
  { REG_HSTOP, 0x04 },
  { REG_HREF, 0xa4 },
  { REG_VSTART, 0x02 },
  { REG_VSTOP, 0x7a },
  { REG_VREF, 0x0a },


/*  { REG_HSTART, 0x16 },
  { REG_HSTOP, 0x04 },
  { REG_HREF, 0x24 },
  { REG_VSTART, 0x02 },
  { REG_VSTOP, 0x7a },
  { REG_VREF, 0x0a },*/
  { 0xff, 0xff }, /* END MARKER */
};

const struct regval_list yuv422_ov7670[] PROGMEM = {
  { REG_COM7, 0x0 },  /* Selects YUV mode */
  { REG_RGB444, 0 },  /* No RGB444 please */
  { REG_COM1, 0 },
  { REG_COM15, COM15_R00FF },
  { REG_COM9, 0x6A }, /* 128x gain ceiling; 0x8 is reserved bit */
  { 0x4f, 0x80 },   /* "matrix coefficient 1" */
  { 0x50, 0x80 },   /* "matrix coefficient 2" */
  { 0x51, 0 },    /* vb */
  { 0x52, 0x22 },   /* "matrix coefficient 4" */
  { 0x53, 0x5e },   /* "matrix coefficient 5" */
  { 0x54, 0x80 },   /* "matrix coefficient 6" */
  { REG_COM13, COM13_UVSAT },
  { 0xff, 0xff },   /* END MARKER */
};

const struct regval_list ov7670_default_regs[] PROGMEM = {//from the linux driver
  { REG_COM7, COM7_RESET },
  { REG_TSLB, 0x04 }, /* OV */
  { REG_COM7, 0 },  /* VGA */
  /*
    Set the hardware window.  These values from OV don't entirely
    make sense - hstop is less than hstart.  But they work...
  */
  { REG_HSTART, 0x13 }, { REG_HSTOP, 0x01 },
  { REG_HREF, 0xb6 }, { REG_VSTART, 0x02 },
  { REG_VSTOP, 0x7a }, { REG_VREF, 0x0a },

  { REG_COM3, 0 }, { REG_COM14, 0 },
  /* Mystery scaling numbers */
  { 0x70, 0x3a }, { 0x71, 0x35 },
  { 0x72, 0x11 }, { 0x73, 0xf0 },
  { 0xa2,/* 0x02 changed to 1*/1 }, { REG_COM10, 0x0 },
  /* Gamma curve values */
  { 0x7a, 0x20 }, { 0x7b, 0x10 },
  { 0x7c, 0x1e }, { 0x7d, 0x35 },
  { 0x7e, 0x5a }, { 0x7f, 0x69 },
  { 0x80, 0x76 }, { 0x81, 0x80 },
  { 0x82, 0x88 }, { 0x83, 0x8f },
  { 0x84, 0x96 }, { 0x85, 0xa3 },
  { 0x86, 0xaf }, { 0x87, 0xc4 },
  { 0x88, 0xd7 }, { 0x89, 0xe8 },
  /* AGC and AEC parameters.  Note we start by disabling those features,
    then turn them only after tweaking the values. */
  { REG_COM8, COM8_FASTAEC | COM8_AECSTEP },
  { REG_GAIN, 0 }, { REG_AECH, 0 },
  { REG_COM4, 0x40 }, /* magic reserved bit */
  { REG_COM9, 0x18 }, /* 4x gain + magic rsvd bit */
  { REG_BD50MAX, 0x05 }, { REG_BD60MAX, 0x07 },
  { REG_AEW, 0x95 }, { REG_AEB, 0x33 },
  { REG_VPT, 0xe3 }, { REG_HAECC1, 0x78 },
  { REG_HAECC2, 0x68 }, { 0xa1, 0x03 }, /* magic */
  { REG_HAECC3, 0xd8 }, { REG_HAECC4, 0xd8 },
  { REG_HAECC5, 0xf0 }, { REG_HAECC6, 0x90 },
  { REG_HAECC7, 0x94 },
  { REG_COM8, COM8_FASTAEC | COM8_AECSTEP | COM8_AGC | COM8_AEC },
  { 0x30, 0 }, { 0x31, 0 },//disable some delays
  /* Almost all of these are magic "reserved" values.  */
  { REG_COM5, 0x61 }, { REG_COM6, 0x4b },
  { 0x16, 0x02 }, { REG_MVFP, 0x07 },
  { 0x21, 0x02 }, { 0x22, 0x91 },
  { 0x29, 0x07 }, { 0x33, 0x0b },
  { 0x35, 0x0b }, { 0x37, 0x1d },
  { 0x38, 0x71 }, { 0x39, 0x2a },
  { REG_COM12, 0x78 }, { 0x4d, 0x40 },
  { 0x4e, 0x20 }, { REG_GFIX, 0 },
  /*{0x6b, 0x4a},*/{ 0x74, 0x10 },
```

```
  { 0x8d, 0x4f }, { 0x8e, 0 },
  { 0x8f, 0 }, { 0x90, 0 },
  { 0x91, 0 }, { 0x96, 0 },
  { 0x9a, 0 }, { 0xb0, 0x84 },
  { 0xb1, 0x0c }, { 0xb2, 0x0e },
  { 0xb3, 0x82 }, { 0xb8, 0x0a },

  /* More reserved magic, some of which tweaks white balance */
  { 0x43, 0x0a }, { 0x44, 0xf0 },
  { 0x45, 0x34 }, { 0x46, 0x58 },
  { 0x47, 0x28 }, { 0x48, 0x3a },
  { 0x59, 0x88 }, { 0x5a, 0x88 },
  { 0x5b, 0x44 }, { 0x5c, 0x67 },
  { 0x5d, 0x49 }, { 0x5e, 0x0e },
  { 0x6c, 0x0a }, { 0x6d, 0x55 },
  { 0x6e, 0x11 }, { 0x6f, 0x9e }, /* it was 0x9F "9e for advance AWB" */
  { 0x6a, 0x40 }, { REG_BLUE, 0x40 },
  { REG_RED, 0x60 },
  { REG_COM8, COM8_FASTAEC | COM8_AECSTEP | COM8_AGC | COM8_AEC | COM8_AWB },

  /* Matrix coefficients */
  { 0x4f, 0x80 }, { 0x50, 0x80 },
  { 0x51, 0 },    { 0x52, 0x22 },
  { 0x53, 0x5e }, { 0x54, 0x80 },
  { 0x58, 0x9e },

  { REG_COM16, COM16_AWBGAIN }, { REG_EDGE, 0 },
  { 0x75, 0x05 }, { REG_REG76, 0xe1 },
  { 0x4c, 0 },     { 0x77, 0x01 },
  { REG_COM13, /*0xc3*/0x48 }, { 0x4b, 0x09 },
  { 0xc9, 0x60 },   /*{REG_COM16, 0x38},*/
  { 0x56, 0x40 },

  { 0x34, 0x11 }, { REG_COM11, COM11_EXP | COM11_HZAUTO },
  { 0xa4, 0x82/*Was 0x88*/ }, { 0x96, 0 },
  { 0x97, 0x30 }, { 0x98, 0x20 },
  { 0x99, 0x30 }, { 0x9a, 0x84 },
  { 0x9b, 0x29 }, { 0x9c, 0x03 },
  { 0x9d, 0x4c }, { 0x9e, 0x3f },
  { 0x78, 0x04 },

  /* Extra-weird stuff.  Some sort of multiplexor register */
  { 0x79, 0x01 }, { 0xc8, 0xf0 },
  { 0x79, 0x0f }, { 0xc8, 0x00 },
  { 0x79, 0x10 }, { 0xc8, 0x7e },
  { 0x79, 0x0a }, { 0xc8, 0x80 },
  { 0x79, 0x0b }, { 0xc8, 0x01 },
  { 0x79, 0x0c }, { 0xc8, 0x0f },
  { 0x79, 0x0d }, { 0xc8, 0x20 },
  { 0x79, 0x09 }, { 0xc8, 0x80 },
  { 0x79, 0x02 }, { 0xc8, 0xc0 },
  { 0x79, 0x03 }, { 0xc8, 0x40 },
  { 0x79, 0x05 }, { 0xc8, 0x30 },
  { 0x79, 0x26 },
  { 0xff, 0xff }, /* END MARKER */
};


void error_led(void) {
  DDRB |= 32;//make sure led is output
  while (1) { //wait for reset
    PORTB ^= 32;// toggle led
    _delay_ms(100);
  }
}

void twiStart(void) {
  TWCR = _BV(TWINT) | _BV(TWSTA) | _BV(TWEN);//send start
  while (!(TWCR & (1 << TWINT)));//wait for start to be transmitted
  if ((TWSR & 0xF8) != TW_START)
    error_led();
}

void twiWriteByte(uint8_t DATA, uint8_t type) {
  TWDR = DATA;
  TWCR = _BV(TWINT) | _BV(TWEN);
  while (!(TWCR & (1 << TWINT))) {}
  if ((TWSR & 0xF8) != type)
    error_led();
}

void twiAddr(uint8_t addr, uint8_t typeTWI) {
  TWDR = addr;//send address
```

```
331   TWCR = _BV(TWINT) | _BV(TWEN);    /* clear interrupt to start transmission */
332   while ((TWCR & _BV(TWINT)) == 0); /* wait for transmission */
333   if ((TWSR & 0xF8) != typeTWI)
334     error_led();
335 }
336
337 void wrReg(uint8_t reg, uint8_t dat) {
338   //send start condition
339   twiStart();
340   twiAddr(0x42, TW_MT_SLA_ACK);
341   twiWriteByte(reg, TW_MT_DATA_ACK);
342   twiWriteByte(dat, TW_MT_DATA_ACK);
343   TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);//send stop
344   _delay_ms(1);
345 }
346
347 void wrSensorRegs8_8(const struct regval_list reglist[]) {
348   uint8_t reg_addr, reg_val;
349   const struct regval_list *next = reglist;
350   while ((reg_addr != 0xff) | (reg_val != 0xff)) {
351     reg_addr = pgm_read_byte(&next->reg_num);
352     reg_val = pgm_read_byte(&next->value);
353     wrReg(reg_addr, reg_val);
354     next++;
355   }
356 }
```

ESP8266

```
#include <ESP8266WiFiMulti.h> // for connect to Wi-Fi
#include <ESP8266HTTPClient.h> // for the http requests

// the object through which the ESP connect its to Wi-Fi
ESP8266WiFiMulti WiFiMulti;

// buffer and sub-buffer
#define BUFFER_SIZE 19200
#define SUB_BUFFER_SIZE 2400
#define N_PACKETS BUFFER_SIZE/SUB_BUFFER_SIZE
char buf[BUFFER_SIZE];
char subBuf[SUB_BUFFER_SIZE + 1];

int i = 0; // index of buffer

HTTPClient http; // the object for the http requests

bool photoStarted = false; // become true while capturing the photo

// constants Wi-Fi connection
const char server[] = "10.101.130.120";
const char ssid[] = "UnisiConference";
const char psw[] = "squatuests";

// function to check the success of the http requests
bool http_code_ok(int httpcode) {
  if (httpcode > 0) {
    if (httpcode != HTTP_CODE_OK) {
      Serial.write("httpcode != HTTP_CODE_OK");
      Serial.write('\n');
      return false;
    }
  }
  else {
    Serial.write("httpcode <= 0 : ERROR");
    Serial.write('\n');
    return false;
  }
  return true;
}

void setup() {
  Serial.begin(1000000); // setup the baud rate to 1Mbps
  WiFiMulti.addAP(ssid, psw); // initialize the Wi-Fi object and wait for connection
  Serial.write("connecting...");
}

void loop() {
  // do nothing while the ESP is not connected
  if ((WiFiMulti.run() != WL_CONNECTED)) {
    Serial.write('.');
    return;
  }

  // wait for bytes into the serial lines
  while (Serial.available() > 0) {
     if (!photoStarted) { // wait for the command sequence
      if (Serial.read() == '*') {
        Serial.write("*");
        if (Serial.read() == 'R') {
          Serial.write("R");
          if (Serial.read() == 'D') {
            Serial.write("D");
            if (Serial.read() == 'Y') {
              Serial.write("Y");
              if (Serial.read() == '*') {
                photoStarted = true;
                Serial.write("*\n");
                Serial.write("Capturing started...");
                Serial.write('\n');
              }
            }
          }
        }
      }
    }
    else { // capturing started, buffer filling
      buf[i] = Serial.read();
      // some bytes give problem while converting into String objects for the http requests, then some pixel
            values will be increased by 1 or 2 (doesn't compromise the image quality)
      if (buf[i] == 0 || buf[i] == 9 || buf[i] == 13 || buf[i] == 37)
```

```
81          buf[i] = buf[i] + 1;
82        if (buf[i] == 10 || buf[i] == 38)
83          buf[i] = buf[i] + 1;
84        i++; // increase index
85      }
86    }
87
88    // once the buffer is full, start trasmission
89    if (i == BUFFER_SIZE) {
90      Serial.write("Capturing finished.");
91      Serial.write('\n');
92
93      // clear the file test.txt
94        http.begin("http://" + String(server) + "/PHPIoT/initphoto.php");
95        if (!http_code_ok(http.GET())) {
96          i = 0;
97          Serial.write("Photo not uploaded.");
98          Serial.write('\n');
99          Serial.write("error during the GET to initphoto.php");
100         Serial.write('\n');
101         http.end();
102         return;
103       }
104       http.end();
105
106     // start to uploading splitting the buffer in packets of 300 bytes
107     Serial.write("Uploading to localhost...");
108     Serial.write('\n');
109     for (int j = 0; j < N_PACKETS; j++) {
110       // extrapolate the packet with memcpy
111       memcpy(subBuf, buf + j * SUB_BUFFER_SIZE, SUB_BUFFER_SIZE);
112       subBuf[SUB_BUFFER_SIZE] = '\0'; // end of String
113       String message = "dato=" + String(subBuf);
114       http.begin("http://" + String(server) + "/PHPIoT/dato.php");
115       http.addHeader("Content-Type", "application/x-www-form-urlencoded");
116       if (!http_code_ok(http.POST(message))) {
117         i = 0;
118         Serial.write("Photo not uploaded.");
119         Serial.write('\n');
120         Serial.write("error during the POST of packet ");
121         Serial.print(j + 1);
122         Serial.write('\n');
123         http.end();
124         return;
125       }
126       http.end();
127     }
128     Serial.write("uploaded.");
129     Serial.write('\n');
130
131     // after the photo is been uploaded, send it with Telegram BOT
132     http.begin("http://" + String(server) + "/PHPIoT/sendphotoAus.php");
133     if (!http_code_ok(http.GET())) {
134       i = 0;
135       Serial.write("Photo not sent.");
136       Serial.write('\n');
137       Serial.write("error during the GET to sendphotoAus.php");
138       Serial.write('\n');
139       http.end();
140       return;
141     }
142     http.end();
143
144     // after the photo is been sent it is possible to start a new capturing
145     Serial.write("Press the button to start a new capturing...");
146     Serial.write('\n');
147     Serial.write('\n');
148     photoStarted = false;
149     i = 0;
150     ESP.reset();
151   }
152 }
```

initphoto.php

```
<?php
   echo system(" > img.txt", $res);
?>
```

dato.php

```
<?php
$filename = 'img.txt';
$dato = $_POST['dato'];

set_time_limit(5*60);
// Let's make sure the file exists and is writable first.
if (is_writable($filename)) {
    if (!$handle = fopen($filename, 'a')) {
         echo "Cannot open file ($filename)";
         exit;
    }

    // Write $dato to our opened file.
    if (fwrite($handle, $dato) === FALSE) {
        echo "Cannot write to file ($filename)";
        exit;
    }

    fclose($handle);

} else {
    echo "The file $filename is not writable";
}
?>
```

sendphotoAus.php

```
<?php
   system("php sendphoto.php", $res);
?>
```

sendphoto.php

```
<?php
#create final bmp
system("cat head160x120.bmp > final160x120.bmp", $res);
```

```php
system("cat img.txt >> final160x120.bmp", $res);

$chat_id="@ov7670_ricciolino";
$bot_url = "https://api.telegram.org/bot" . $your_bot_token . "/";
$url = $bot_url . "sendPhoto?chat_id=" . $chat_id;
$post_fields = array(
            'chat_id'    => $chat_id ,
            'photo' => new CURLFile(realpath("final160x120.bmp"))
                    );

$ch = curl_init();
curl_setopt($ch, CURLOPT_HTTPHEADER,
                array( "Content-Type:multipart/form-data" )
            );
curl_setopt($ch, CURLOPT_URL, $url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, $post_fields);
$output = curl_exec($ch);
?>
```

head160x120.bmp

Per quanto riguarda il file header da concatenare al file `img.txt`, basta incollare i seguenti byte in un editor esadecimale (per esempio Bless Hex Editor) e salvare il file come head160x120.bmp :

```
    42 4D 7A 4F 00 00 00 00 00 00 7A 04 00 00 6C 00 00 00 A0 00 00 00 78 00 00 00 01 00 08 00 00 00 00 00 00
4B 00 00 13 0B 00 00 13 0B 00 00 00 01 00 00 00 01 00 00 42 47 52 73 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 02 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 01 01 00 02 02 02 00 03 03 03 00 04 04 04 00 05 05 05 00
06 06 06 00 07 07 07 00 08 08 08 00 09 09 09 00 0A 0A 0A 00 0B 0B 0B 00 0C 0C 0C 00 0D 0D 0D 00 0E 0E 0E 00 0F
0F 0F 00 10 10 10 00 11 11 11 00 12 12 12 00 13 13 13 00 14 14 14 00 15 15 15 00 16 16 16 00 17 17 17 00 18 18
18 00 19 19 19 00 1A 1A 1A 00 1B 1B 1B 00 1C 1C 1C 00 1D 1D 1D 00 1E 1E 1E 00 1F 1F 1F 00 20 20 20 00 21 21 21
00 22 22 22 00 23 23 23 00 24 24 24 00 25 25 25 00 26 26 26 00 27 27 27 00 28 28 28 00 29 29 29 00 2A 2A 2A 00
2B 2B 2B 00 2C 2C 2C 00 2D 2D 2D 00 2E 2E 2E 00 2F 2F 2F 00 30 30 30 00 31 31 31 00 32 32 32 00 33 33 33 00 34
34 34 00 35 35 35 00 36 36 36 00 37 37 37 00 38 38 38 00 39 39 39 00 3A 3A 3A 00 3B 3B 3B 00 3C 3C 3C 00 3D 3D
3D 00 3E 3E 3E 00 3F 3F 3F 00 40 40 40 00 41 41 41 00 42 42 42 00 43 43 43 00 44 44 44 00 45 45 45 00 46 46 46
00 47 47 47 00 48 48 48 00 49 49 49 00 4A 4A 4A 00 4B 4B 4B 00 4C 4C 4C 00 4D 4D 4D 00 4E 4E 4E 00 4F 4F 4F 00
50 50 50 00 51 51 51 00 52 52 52 00 53 53 53 00 54 54 54 00 55 55 55 00 56 56 56 00 57 57 57 00 58 58 58 00 59
59 59 00 5A 5A 5A 00 5B 5B 5B 00 5C 5C 5C 00 5D 5D 5D 00 5E 5E 5E 00 5F 5F 5F 00 60 60 60 00 61 61 61 00 62 62
62 00 63 63 63 00 64 64 64 00 65 65 65 00 66 66 66 00 67 67 67 00 68 68 68 00 69 69 69 00 6A 6A 6A 00 6B 6B 6B
00 6C 6C 6C 00 6D 6D 6D 00 6E 6E 6E 00 6F 6F 6F 00 70 70 70 00 71 71 71 00 72 72 72 00 73 73 73 00 74 74 74 00
75 75 75 00 76 76 76 00 77 77 77 00 78 78 78 00 79 79 79 00 7A 7A 7A 00 7B 7B 7B 00 7C 7C 7C 00 7D 7D 7D 00 7E
7E 7E 00 7F 7F 7F 00 80 80 80 00 81 81 81 00 82 82 82 00 83 83 83 00 84 84 84 00 85 85 85 00 86 86 86 00 87 87
87 00 88 88 88 00 89 89 89 00 8A 8A 8A 00 8B 8B 8B 00 8C 8C 8C 00 8D 8D 8D 00 8E 8E 8E 00 8F 8F 8F 00 90 90 90
00 91 91 91 00 92 92 92 00 93 93 93 00 94 94 94 00 95 95 95 00 96 96 96 00 97 97 97 00 98 98 98 00 99 99 99 00
9A 9A 9A 00 9B 9B 9B 00 9C 9C 9C 00 9D 9D 9D 00 9E 9E 9E 00 9F 9F 9F 00 A0 A0 A0 00 A1 A1 A1 00 A2 A2 A2 00 A3
A3 A3 00 A4 A4 A4 00 A5 A5 A5 00 A6 A6 A6 00 A7 A7 A7 00 A8 A8 A8 00 A9 A9 A9 00 AA AA AA 00 AB AB AB 00 AC AC
AC 00 AD AD AD 00 AE AE AE 00 AF AF AF 00 B0 B0 B0 00 B1 B1 B1 00 B2 B2 B2 00 B3 B3 B3 00 B4 B4 B4 00 B5 B5 B5
00 B6 B6 B6 00 B7 B7 B7 00 B8 B8 B8 00 B9 B9 B9 00 BA BA BA 00 BB BB BB 00 BC BC BC 00 BD BD BD 00 BE BE BE 00
BF BF BF 00 C0 C0 C0 00 C1 C1 C1 00 C2 C2 C2 00 C3 C3 C3 00 C4 C4 C4 00 C5 C5 C5 00 C6 C6 C6 00 C7 C7 C7 00 C8
C8 C8 00 C9 C9 C9 00 CA CA CA 00 CB CB CB 00 CC CC CC 00 CD CD CD 00 CE CE CE 00 CF CF CF 00 D0 D0 D0 00 D1 D1
D1 00 D2 D2 D2 00 D3 D3 D3 00 D4 D4 D4 00 D5 D5 D5 00 D6 D6 D6 00 D7 D7 D7 00 D8 D8 D8 00 D9 D9 D9 00 DA DA DA
00 DB DB DB 00 DC DC DC 00 DD DD DD 00 DE DE DE 00 DF DF DF 00 E0 E0 E0 00 E1 E1 E1 00 E2 E2 E2 00 E3 E3 E3 00
E4 E4 E4 00 E5 E5 E5 00 E6 E6 E6 00 E7 E7 E7 00 E8 E8 E8 00 E9 E9 E9 00 EA EA EA 00 EB EB EB 00 EC EC EC 00 ED
ED ED 00 EE EE EE 00 EF EF EF 00 F0 F0 F0 00 F1 F1 F1 00 F2 F2 F2 00 F3 F3 F3 00 F4 F4 F4 00 F5 F5 F5 00 F6 F6
F6 00 F7 F7 F7 00 F8 F8 F8 00 F9 F9 F9 00 FA FA FA 00 FB FB FB 00 FC FC FC 00 FD FD FD 00 FE FE FE 00 FF FF FF
00
```

MEME moments

Per maggiori dettagli vedere **allegato1**.