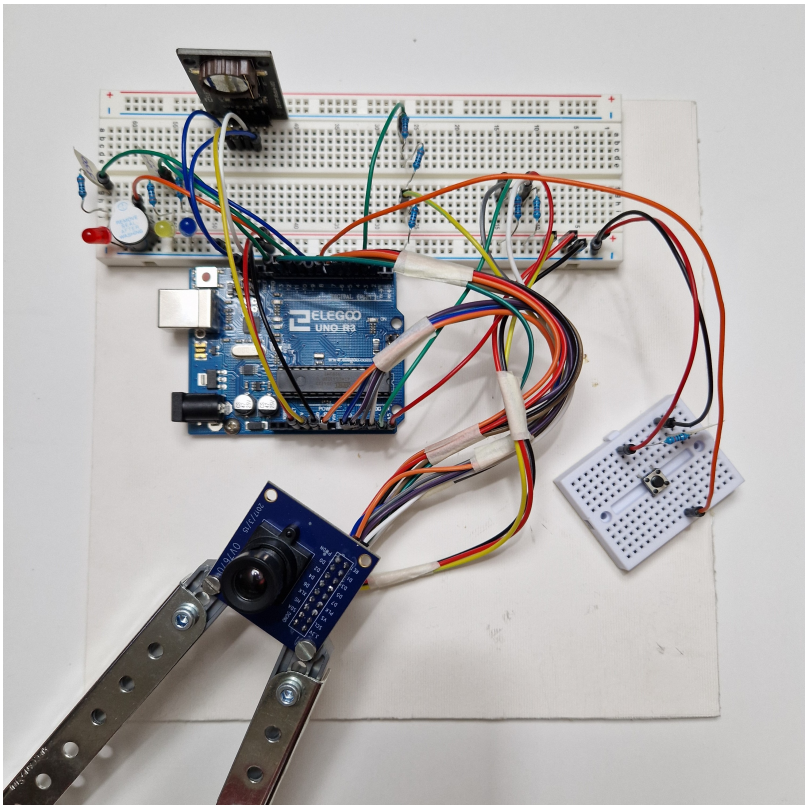
Alma Mater Studiorum - Università di Bologna

**Corso Sistemi Digitali M**
**Anno Accademico 2022-2023**

# DeskSpy

**Riccardo Evangelisti**



# Indice

# Introduzione

DeskSpy è un sistema che monitora il tempo in cui l'utente è posizionato alla scrivania e gestisce l'alternarsi dei momenti di pausa e di lavoro. Il sistema identifica univocamente l'utente attraverso il riconoscimento facciale e lo invoglia a sospendere il lavoro attraverso stimoli visivi e acustici.

Il sistema DeskSpy viene posizionato sopra la scrivania e di fronte all'utente, ad esempio sotto al monitor, con la camera rivolta in direzione del volto.

Il sistema è strutturato in due parti: Arduino e Server. L'Arduino gestisce l'intero programma, cambia gli stati e comanda i dispositivi esterni. Ad esso sono collegati: la camera OV7670, i led, buzzer, pulsanti, RealTimeClock. Il server è connesso via collegamento seriale all'Arduino e svolge il solo compito di effettuare il riconoscimento facciale sulle immagini ricevute e restituire il risultato.

# Protocollo e Stati

## 2.1 Protocollo

La camera OV7670[1] è accessibile dall'Arduino attraverso il protocollo *I2C*. I dati sono mandati in parallelo sui canali D0-D7, ricevuti da Arduino e inviati al Server mediante una connessione seriale. Arduino legge una riga dell'immagine alla volta e la invia al Server prima di proseguire.


**Figura 2.1:** Protocollo del sistema

## 2.2 Dispositivi

Il sistema dispone dei seguenti elementi per interagire con l'utente:

- Led Blu (work)
- Led Giallo (break)
- Led Rosso (warning)
- Pulsante (snooze)
- Buzzer attivo

L'arduino deve essere connesso ad un computer tramite cavo USB, che funge da Server. Una volta collegato, Arduino si mette in attesa che venga lanciato il programma di riconoscimento. I cambiamenti di stato sono interamente gestiti dal programma di Arduino.

Tutte le tempistiche riportate di seguito sono valori di esempio, facilmente modificabili in base alle necessità e preferenze. Si faccia riferimento a [6] per un video in tempo reale del funzionamento.

### 2.2.1 Stati

Il programma di riconoscimento sul Server, una volta eseguito, comunica ad Arduino l'inizio dell'esecuzione. Dopo un breve momento di *SETUP*, si ripetono ciclicamente i seguenti stati, come da **Figura 2.2** e **Figura 2.3**:

- *PREPARE_FOR_WORK*. Il led blu lampeggia per 1 minuto. Segnala che la fase di lavoro sta per iniziare. Scaduto il minuto si entra nello stato *WORK*.

- *WORK*. Il led blu rimane acceso continuativamente. Il sistema si aspetta che l'utente sia presente fino allo scadere del tempo di lavoro (1 ora). Ogni 10 secondi la camera scatta una foto e la invia al server il quale esegue il riconoscimento facciale e ritorna l'esito. Se il volto non è presente o non è stato riconosciuto, si entra nello stato *NOT DETECTED*, altrimenti *DETECTED*. Se vi è stato un *early work*, ossia l'utente è tornato dalla pausa prima del tempo, tale valore viene sottratto al tempo di lavoro.

  - *DETECTED*. Si rimane in questo stato finché l'esito rimane positivo. Se l'esito diventa negativo, si entra in *NOT DETECTED*. Se scade il tempo di lavoro, si entra in *PREPARE_FOR_BREAK*.
  - *NOT DETECTED*. L'utente viene segnalato con l'accensione del led rosso (warning) e con un breve segnale acustico del buzzer. La frequenza di scatto della camera, ossia la verifica della presenza dell'utente, diminuisce a 3 secondi. Se l'esito diventa positivo, si entra in *DETECTED* e la frequenza torna a 10 secondi. Se l'esito rimane negativo continuativamente per 5 minuti, si assume che l'utente si è allontanato ed è già entrato in pausa (*early break*), dunque si entra in *BREAK*.

- *PREPARE_FOR_BREAK*. Il led giallo lampeggia per 1 minuto. Segnala che la pausa sta per iniziare. L'utente può premere il pulsante (snooze button) che permette di posticipare la pausa, entrando nello stato *SNOOZE*. Tale operazione è possibile per sole 2 volte, oltre le quali il pulsante viene disabilitato. Scaduto il minuto, si entra nello stato *BREAK*.

- *SNOOZE*. Entrambi i led blu e giallo sono accesi continuativamente, a segnalare che si è in uno stato tra il lavoro e la pausa. Dopo 10 minuti si ritorna automaticamente allo stato *PREPARE_FOR_BREAK*.

- *BREAK*. Il led giallo rimane acceso continuativamente. Tale stato è l'opposto di *WORK*, dunque il sistema si aspetta di non vedere l'utente (o di non riconoscerlo). Il tempo di pausa è di 10 minuti. La frequenza di scatto è di 5 secondi. In base all'esito si entra in *DETECTED* o *NOT DETECTED*. Se vi è stata una *early break*, tale valore viene sottratto al tempo di pausa.

– *DETECTED*. L'utente viene segnalato con l'accensione del led rosso e con un breve segnale acustico del buzzer. La frequenza di scatto diminuisce a 3 secondi. Se l'esito diventa negativo (utente non visto/riconosciuto) si entra in *NOT DETECTED* e la frequenza torna a 5 secondi. Se l'esito rimane positivo continuativamente per 5 minuti, ossia se l'utente è rimasto alla scrivania ignorando i segnali visivi e sonori, si entra in *WORK*.

– *NOT DETECTED*. Si rimane in questo stato finché l'esito rimane negativo, ossia il sistema non vede o riconosce l'utente. Se scade il tempo di pausa, si entra in *PREPARE_FOR_WORK*.

| | | LED BLUE | LED YELLOW | LED RED | CAPTURE FREQUENCY (sec) |
|---|---|---|---|---|---|
| | SETUP | OFF | OFF | OFF | |
| | PREPARE_FOR_WORK | BLINK | OFF | OFF | |
| WORK | DETECTED | ON | OFF | OFF | 10 |
| | NOT DETECTED | ON | OFF | BLINK | 3 |
| | PREPARE_FOR_BREAK | OFF | BLINK | OFF | |
| | SNOOZE | ON | ON | OFF | |
| BREAK | DETECTED | OFF | ON | BLINK | 3 |
| | NOT DETECTED | OFF | ON | OFF | 5 |

**Figura 2.2:** Led e frequenza di scatto nei Stati del sistema

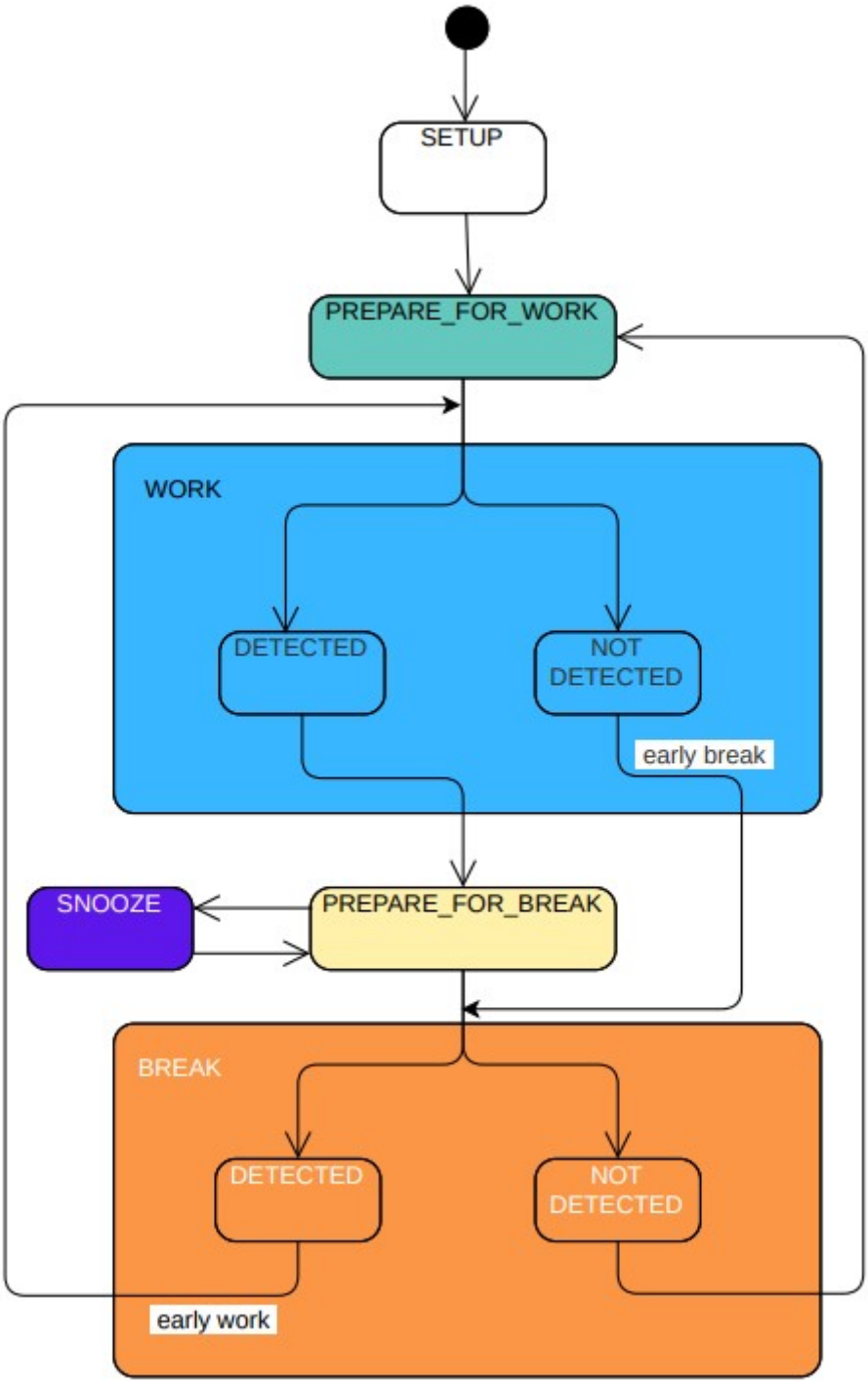



**Figura 2.3:** Stati del sistema

# Parte Arduino

## 3.1 Schema Elettrico

### 3.1.1 Camera OV7670

La camera OV7670 è stata collegata al microcontrollore seguendo lo schema riportato in **Figura 3.1**

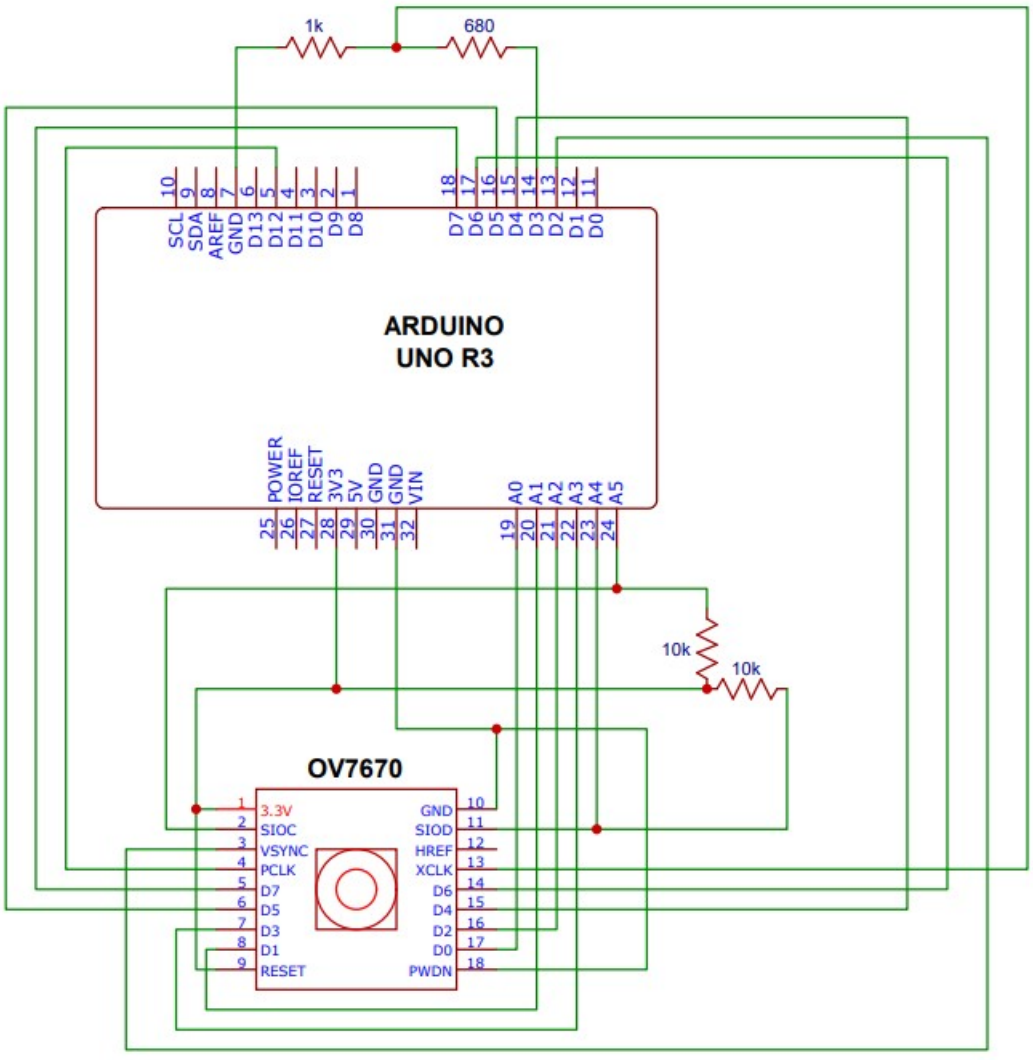



**Figura 3.1:** Collegamenti di OV7670 al microcontrollore

### 3.1.2 Led, Buzzer, Snooze Button

I restanti componenti sono stati collegati usando le porte disponibili (**Figura 3.2**).

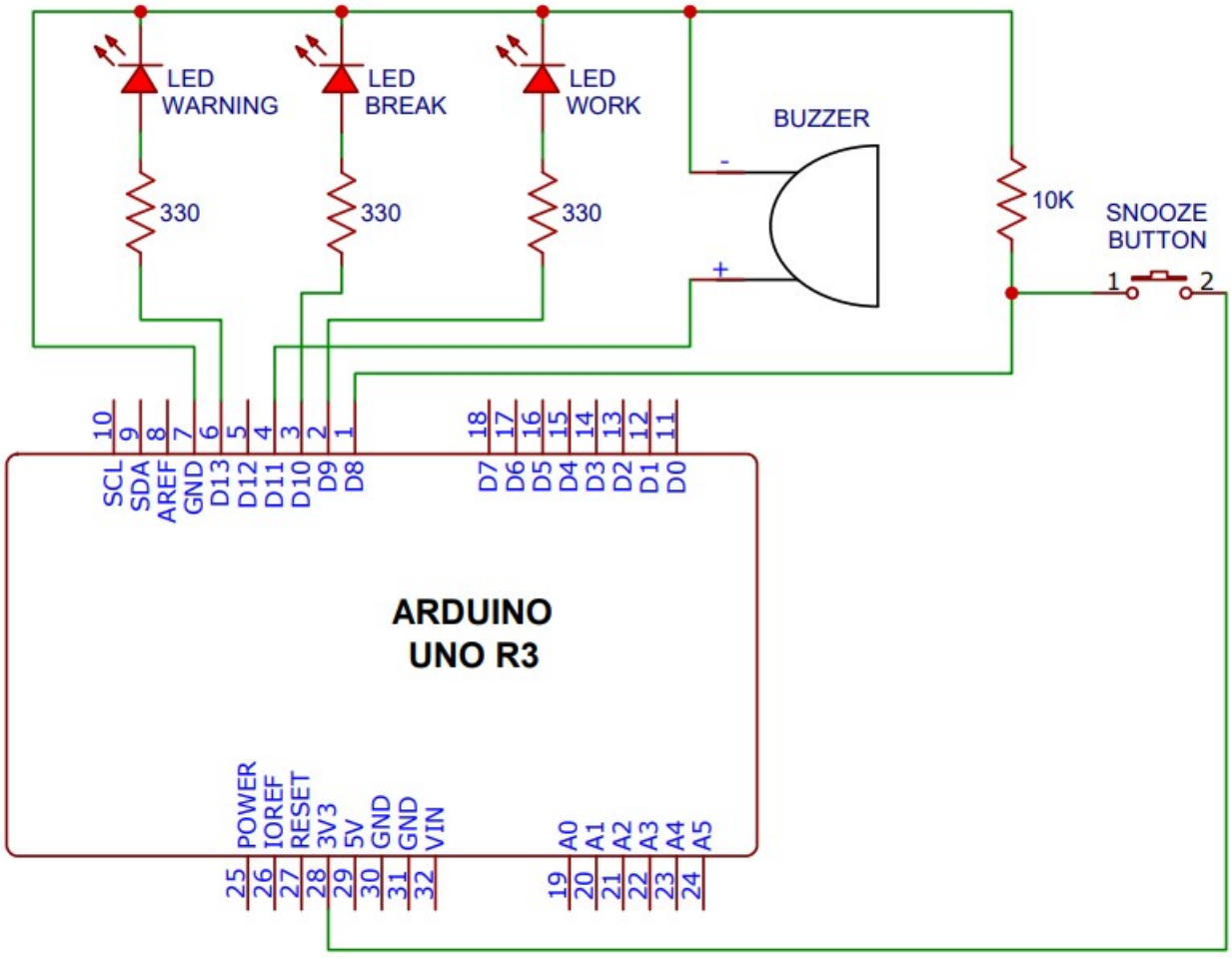



**Figura 3.2:** Collegamenti dei dispositivi al microcontrollore

## 3.2 Software

### 3.2.1 Risoluzione QQVGA

La OV7670 viene impostata per scattare immagini in risoluzione QQVGA (160x120). Sebbene la camera sia in grado di scattare foto a risoluzione VGA (640x480), si è optato per un formato inferiore al fine di alleggerire il flusso di dati e rendere il sistema più veloce. In **Figura 3.3** sono riportate le impostazioni dei registri[2] per la risoluzione QQVGA.

Si inserisce un *vertical-padding* di 2 pixel (4 byte tra un'immagine e la successiva), un *horizontal-padding-left* di 1 byte e un *horizontal-padding-right* di 3 byte (1 byte all'inizio e 3 byte alla fine di ogni riga).

```
// 120 + 2 pixel (4 bytes) for padding.
const uint8_t CameraOV7670Registers::QQVGA_VERTICAL_PADDING = 2;
const uint16_t vstart = 0;
const uint16_t vstop = 120 + CameraOV7670Registers::QQVGA_VERTICAL_PADDING;

// One padding byte from the beginning and three at the end.
const uint16_t hstart = 182;
const uint16_t hstop = 46;

const PROGMEM RegisterData CameraOV7670Registers::regsQQVGA [] = {
    {REG_VSTART,vstart},
    {REG_VSTOP,vstop},
    {REG_VREF,0},
    {REG_HSTART,hstart >> 3},
    {REG_HSTOP,hstop >> 3},
    {REG_HREF,0b00000000 | (hstart & 0b111) | ((hstop & 0b111) << 3)},

    {REG_COM3, COM3_DCWEN}, // enable downsamp/crop/window
    {REG_COM14, 0x1a},        // divide by 4
    {SCALING_DCWCTR, 0x22},   // downsample by 4
    {SCALING_PCLK_DIV, 0xf2}, // divide by 4

    {0xff, 0xff}, /* END MARKER */
};
```

**Figura 3.3:** Impostazioni dei registri di OV7670 per la risoluzione QQVGA

### 3.2.2 Codifica YUV422

Si è scelta la codifica YUV422 in modo da poter facilmente scartare le componenti "U" e "V" e prelevare solo l'immagine in scala di grigi. Ogni pixel viene così identificato da un solo byte ("Y") e lo stream di dati per immagine diventa effettivamente 160*120 byte. Tale scelta è giustificata dal fatto che il programma di riconoscimento sul Server converte l'immagine ricevuta in scala di grigi, rendendo inutile l'eventuale passaggio di byte che identificano i colori. In **Figura 3.4** sono riportate le impostazioni dei registri[3] per la codifica YUV422.

```
const PROGMEM RegisterData CameraOV7670Registers::regsYUV422 [] = {
    {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_GAMMA|*/COM13_UVSAT},
    {0xff, 0xff},
};
```

**Figura 3.4:** Impostazioni dei registri di OV7670 per la codifica YUV422

### 3.2.3 Scatto immagine

La camera scatta la foto durante VSYNC a LOW e mette a disposizione i dati sui pin D0-D7 in parallelo. Il programma di Arduino che preleva e invia l'immagine è strutturato come segue (**Figura 3.5**):

1. Il programma invia sulla comunicazione seriale il comando di inizio di un nuovo frame (0x00).
2. Attende che la camera sia in stato di cattura (VSYNC a LOW).
3. Scarta il vertical-padding.
4. Per ogni riga dell'immagine:
   - 4.1 Scarta il horizontal-padding-left.
   - 4.2 Per ogni colonna, legge solo i due byte a scala di grigi, ignorando gli altri. L'ordine con cui i byte sono disposti per 2 pixel è il seguente: Y, U, Y, V.
   - 4.3 Scarta il horizontal-padding-right.
   - 4.4 Invia la riga sulla comunicazione seriale. Ogni byte è inviato uno per volta.[4]

```
void takeAndSendFrame() {
  waitUntilPreviousUartByteIsSent();
  UDR0 = 0x00; // START FRAME COMMAND

  noInterrupts();
  camera.waitForVsync();
  camera.ignoreVerticalPadding();

  for (uint16_t y = 0; y < lineCount; y++) {
    camera.ignoreHorizontalPaddingLeft();

    uint16_t x = 0;
    while (x < lineLength) {
      camera.waitForPixelClockRisingEdge(); // YUV422 grayscale byte: Y1
      camera.readPixelByte(lineBuffer[x]);

      camera.waitForPixelClockRisingEdge(); // YUV422 color byte. Ignore: U
      x++;

      camera.waitForPixelClockRisingEdge(); // YUV422 grayscale byte: Y2
      camera.readPixelByte(lineBuffer[x]);

      camera.waitForPixelClockRisingEdge(); // YUV422 color byte. Ignore: V
      x++;
    }
    camera.ignoreHorizontalPaddingRight();

    // Send the line
    for (uint16_t i = 0; i < x; i++) {
      waitUntilPreviousUartByteIsSent();
      if (lineBuffer[i] == 0b00000000) {
        lineBuffer[i] = 0b00000001; // Because 0x00 is the START FRAME COMMAND
      }
      UDR0 = lineBuffer[i];
    }
  }
  interrupts();
  waitUntilPreviousUartByteIsSent();
}
```

**Figura 3.5:** Funzione che scatta e invia l'immagine

# Parte Server

## 4.1 Funzionamento

Il programma lato Server è scritto in linguaggio Python, per la facilità con cui è possibile sviluppare il riconoscimento facciale.

Il programma apre la connessione seriale, si sincronizza con l'Arduino e ciclicamente legge i frame che vengono inviati. La ricezione del byte 0x00 identifica l'inizio di un nuovo frame. Per ogni immagine ricevuta, effettua il riconoscimento facciale e restituisce l'esito. Arduino attende la ricezione dell'esito prima di inviare un nuovo frame.

I dati ricevuti vengono letti 1 byte alla volta e trasformati in immagine di risoluzione QQVGA:

```
# Read one frame
for i in range(WIDTH * HEIGHT):
    byte = ser.read(1)
    buffer.append(int.from_bytes(byte, byteorder="big"))
# Reshape buffer into array[HEIGHT][WIDTH]
img_array = np.array(buffer).reshape(HEIGHT, WIDTH)
# Convert the array to a grayscale image
img_array = np.array(Image.fromarray(img_array, "L"))
```

### 4.1.1 Rilevamento facciale

Il rilevamento del volto (*face detection*) viene svolto utilizzando la libreria *cv2* e *Haar Cascade* specifico per il volto[5]:

```
face_detector = cv2.CascadeClassifier(
    "haarcascade_frontalface_default.xml")
faces = face_detector.detectMultiScale(
    img_array, scaleFactor=1.05, minNeighbors=3)
```

### 4.1.2 Riconoscimento facciale

Il riconoscimento del volto (*face recognition*) avviene utilizzando un oggetto *LBPH-FaceRecognizer* della libreria cv2 che si basa su un modello di dati sviluppato con un *trainer*.

```
for (x, y, w, h) in faces:
    ID, conf = face_recognizer.predict(
                        img_array[y:y+h,x:x+w])
    if conf >= 20 and conf <= 115:
        recognized = True
```

### 4.1.3 Trainer

Il programma di training viene eseguito in precedenza e mette a disposizione il modello di dati necessario per il riconoscimento facciale. Si utilizza la libreria cv2 che produce il modello a partire da una serie di immagini scattate direttamente con la camera del sistema.

Per aumentare la precisione, le immagini fornite sono state scattate sotto diversi punti di illuminazione, alternando luce diurna e luci elettriche.

```
face_recognizer.train(faces_array, np.array(ids_array))
face_recognizer.save("trainer.yml")
```

# Conclusioni

Questo progetto ha avuto come obiettivo lo sviluppo di un sistema che aiutasse un utente ad alternare momenti di lavoro e di pausa quando seduto alla scrivania.

DeskSpy si posiziona di fronte all'utente, ad esempio sotto lo schermo del monitor, con la camera rivolta in direzione del volto.

Funziona solo con l'utente registrato, riconoscendone il volto e distinguendolo da altri utenti alla scrivania.

Ha integrate delle funzionalità che rendono flessibile l'alternasi pausa/lavoro, come l'early break, l'early work e break snooze.

Nello sviluppo si sono tenuti conto di diversi requisiti, che sono stati rispettati:

- Alta velocità di elaborazione
- Basso impatto sulle prestazioni del server
- Basso costo
- Portabilità e piccole dimensioni

Il funzionamento in tempo reale di DeskSpy è disponibile [6] insieme al codice sorgente[7].

### Sviluppi futuri

È possibile migliorare DeskSpy aumentando la difficoltà nell'aggirare il sistema. Al momento basta coprire la camera per ingannare DeskSpy e fargli credere che si è andati in pausa. Per ovviare a ciò, si può verificare che vi sia un minimo di movimento, o un cambiamento nell'immagine, o verificare la presenza di un oggetto che deve rimanere sempre visibile.

# Bibliografia


[1] Documentazione di OV7670: https://www.haoyuelectronics.com/Attachment/OV7670%20+%20AL422B(FIFO)%20Camera%20Module(V2.0)/OV7670%20Implementation%20Guide%20(V1.0).pdf Datasheet di OV7670: http://web.mit.edu/6.111/www/f2016/tools/OV7670_2006.pdf

[2] Impostazioni registri camera OV7670 per formato QQVGA: https://github.com/indrekluuk/LiveOV7670/blob/master/src/lib/LiveOV7670Library/CameraOV7670RegistersQQVGA.cpp.

[3] Impostazioni registri camera OV7670 per formato YUV422: https://github.com/ComputerNerd/ov7670-no-ram-arduino-uno/blob/master/ov7670.c.

[4] Registri USART: https://onlinedocs.microchip.com/pr/GUID-85E9B23E-BE9C-47EA-AED9-E8719EA2F7AF-en-US-2/index.html?GUID-6BB9ECEA-6542-4810-89C4-99006D9156A6

[5] CV2 Haar Cascade: https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_default.xml.

[6] Video in tempo reale del funzionamento: https://www.youtube.com/watch?v=nMDWoYeMBm0.

[7] Codice sorgente: https://github.com/RiccardoEvangelisti/DeskSpy.git.