

Implementarea low cost a unui sistem flexibil de detectare a imaginilor pentru procese industriale

Candidat: Ioan, Smetanca

Coordonator științific: Asist. ing. Liviu-Dănuț Vitan

Sesiunea: Iunie 2023

CUPRINS

1	INTRODUCERE	3
1.1	INFORMAȚII GENERALE	4
1.2	SCOPUL LUCRĂRII	4
1.3	MOTIVAȚIA ALEGERII TEMEI	4
1.4	AUTOMATIZAREA UNOR ACȚIUNI ÎN CADRUL UNUI JOC	5
1.5	EXPLICAREA CODULUI	6
1.6	SCHEMA REPREZENTATIVĂ PENTRU FUNCȚIONAREA CODULUI ..	10
1.7	AVANTAJE ȘI DEZAVANTAJE A SCRIPTULUI:	11
2	CAMERA O3D200AD PMD 3D SENSOR	13
2.1	CONCEPTELE TEHNOLOGIEI O3D.....	13
2.2	Formate date de ieșire.....	14
2.3	HEARTBEAT	14
2.4	SETĂRILE DE ACHIZIȚIE A IMAGINII.....	15
2.5	TIMPUL DE ACHIZIȚIE A IMAGINII	15
2.6	LIMITA DE TEMPERATURĂ, RATA MAXIMĂ A CADRULUI	16
2.7	FILTRAREA.....	16
2.8	FUNCȚIILE (METODELE) EXISTENTE PE SERVER.....	16
2.9	CERINȚE HARDWARE	19
3.0	EXPLICAREA CODULUI	22
3.1	UTILIZAREA WIRESHARK/PYTHON	29
3.2	CONVERSIA DATELOR DIN HEXAZECIMAL ÎN FLOAT	31
3.3	GRAFICELE ÎN URMA DATELOR PRIMITIVE:.....	32
3.4	INFORMATII DESPRE HEADER	34
4	DETECTAREA UNEI PERSOANE ȘI TRANSMITEREA DATELOR CĂTRE LOGO!SOFT(PLC) PRIN INTERMEDIUL PROTOCOLULUI MODBUS TCP/IP35	
4.1	CVLIB YOLO OBJECT DETECTION.....	40
4.2	DETECTAREA UNUI OBIECT ȘI TRIMITEREA DATELOR CĂTRE RELEUL INTELIGENT SI EXPLICAREA CODULUI.....	41
5.	CONCLUZII.....	46
6	BIBLIOGRAFIE	47
7.	ANEXE.....	48
7.1	COD SURSA PENTRU AUTOMATIZAREA UNUI JOC PE BAZA IMAGINILOR	48
7.2	COD SURSA PENTRU EXTRAGEREA DATELOR	50
7.3	CODUL SURSĂ PENTRU DETECTAREA OBIECTELOR ȘI TRIMITEREA DATELOR CĂTRE PLC.....	54

1 INTRODUCERE

Python a fost creat în 1989 de către un programator olandez Guido van Rossum [1]. Python poate fi folosit în foarte multe domenii și este folosit de către companii precum *Yahoo* sau *Google* pentru programarea aplicațiilor web, desigur este un limbaj multifuncțional și poate fi folosit și în alte aplicații. Acest limbaj de programare este un limbaj interpretabil diferit de celelalte limbaje de programare precum C++. Python pune accentul pe simplitatea codului iar sintaxa le permite dezvoltatorilor să-și exprime ideile într-o manieră mai concisă și mult mai clară față de alte limbaje de programare [1].

Flexibilitatea acestui limbaj ne permite să obținem multe rezultate bune. Astfel Python se poate folosi pentru a scrie programe simple, scripturi în doar câteva linii de cod, dar are și puterea necesară pentru a crea operații destul de complexe folosite de foarte multe companii globale, multinaționale [1].

Automatizarea acțiunilor în cadrul unui joc în browser pe baza detectării unor imagini este din ce în ce mai comună deoarece oferă posibilitatea jucătorilor de a progresa mult mai rapid în cadrul unui joc. Automatizarea se poate face într-un mod mult mai avansat și anume prin recunoașterea imaginilor.

Recunoașterea obiectelor aparține de ramura inteligenței artificiale care se ocupă cu identificarea și clasificarea (etichetarea) obiectelor în imagini sau în fluxuri video. Această tehnologie are la baza algoritmi și modele de învățare automată care permit calculatoarelor să recunoască și să înțeleagă obiectele vizuale la fel cum o fac și oamenii. Identificarea se referă la localizarea și delimitarea obiectelor în imagine, ceea ce înseamnă că sistemul poate determina poziția și conturul fiecărui obiect [2]. Clasificarea se referă la atribuirea unui nume sau unei etichete specifice fiecărui obiect identificat, ceea ce permite sistemului să înțeleagă și să interpreteze conținutul vizual [2].

Recunoașterea obiectelor este folosită la nivel global în multe domenii. Pe partea de securitate, această poate fi folosită pentru detectarea persoanelor și urmărirea persoanelor sau vehiculelor suspecte în timp real, ceea ce previne infracționalitatea. În domeniul autovehiculelor autonome, aceasta joacă un rol important în detectarea obstacolelor ceea ce permite conducerea în siguranță a vehiculelor fără șofer [2].

Există mai multe abordări și tehnici utilizare în recunoașterea obiectelor. Cel mai comune se numără rețele neuronale convoluționale (CNN), care sunt special concepute pentru a extrage caracteristici relevante din imagini și pentru a clasifica obiectele prezentate în acestea. Aceste rețele sunt antrenate pe seturi mari de date etichetate, ceea ce le permite să învețe și să recunoască diverse tipuri de obiecte cu o precizie destul de ridicată [3].

Implementarea acestei automatizării poate fi realizată utilizând diferite limbaje de programare, precum Python în se pot utiliza și biblioteci care permit interacțiunea cu browser

ul precum *Selenium* sau *PyAutoGUI* (care ne permite interacțiunea și cu alte aplicații prin intermediul mouse-ului și tastaturii [4]).

1.1 INFORMAȚII GENERALE

Utilizarea Python în diverse aplicații este din ce în ce mai răspândită în acest moment datorită sintaxei simple și a productivității pe care o oferă dezvoltatorilor doar prin simplul fapt că se apropie cât mai mult de limbajul natural, să nu uităm că acest limbaj este unul interpretat și nu unul compilat [5].

Python este utilizat într-o gamă largă de domenii precum:

- Dezvoltarea web;
- Inteligența artificială și învățarea automată;
- Automatizare și scriptare;
- Dezvoltarea de aplicații desktop și mobile;
- Dezvoltarea de jocuri;
- Administrarea sistemelor și automatizarea infrastructurii;

Python este un limbaj de programare versatil și poate fi adaptat pentru a fi folosit într-o gamă largă de aplicații [5].

1.2 SCOPUL LUCRĂRII

În prima parte a acestei lucrări lucrării este de a colecta imagini dintr-un joc pe browser cu scopul de a crea o automatizare pe baza lor în cadrul jocului, desigur în aceasta parte se poate utiliza și biblioteca *Selenium* din Python care permite interacțiunea cu un browser automat, dar ne rezumăm strict automatizarea pe baza imaginilor respective. În a doua parte a lucrării se utilizează Python pentru a trimite diferite comenzi către o cameră industrială (*O3d200AD*) cu scopul de a realiza o comunicație fără utilizarea implicită a soft-ului industrial. În ultima parte a lucrării este identificarea unei persoane cu scopul de a trimite datele către *LOGO! Soft* (PLC).

1.3 MOTIVAȚIA ALEGERII TEMEI

Tema aleasă a fost selectată cu scopul de a de a explora și a înțelege mult mai în profunzime utilitatea acestui limbaj de programare, atât în domeniul automatizării cât și în interacțiunii cu diverse aplicații și tehnologii. Acest lucru ar permite utilizarea extinsă a acestui limbaj în diverse aplicații în care viteza de transfer a datelor nu reprezintă un factor critic.

1.4 AUTOMATIZAREA UNOR ACȚIUNI ÎN CADRUL UNUI JOC

Automatizarea unor acțiuni în cadrul unui joc pe browser cu ajutorul imaginilor.

În această parte a lucrării înainte de scrierea codului primul pas ar fi colectarea imaginilor pentru a înțelege ordinea în care se propune detectarea imaginilor respective.

Lista imaginilor colectate:






1.  - *cent.png*
2.  - *farmingv.png*
3.  - *collect.png*
4.  - *sselect.png*
5.  - *farm.png* (excepție)



Figura 1 - Centralizator și Farming

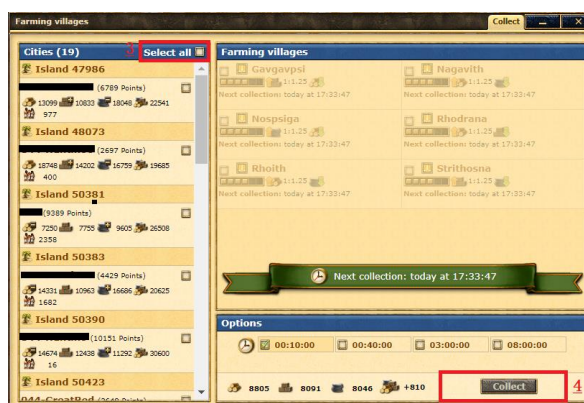


Figura 2 - Selectarea și colectarea resurselor

Se dorește detectarea celor 5 fotografii (*cent.png*, *farmingv.png*, *collect.png*, *sellect.png*, *farm.png*) în următoarea ordine care a fost prezentată în (Figura 1) respectiv (Figura 2). Pentru a face acest lucru am creat un mic script folosind limbajul de programare Python. În cazul în care fereastra din (Figura 2) nu este deschisă, scriptul va încerca să detecteze *cent.png*. În cazul în care *cent.png* nu este detectat pe ecran el va încerca în continuare să îl găsească și ni se va afișa în consola mesajul *Încă caut*.

1.5 EXPLICAREA CODULUI

Explicarea bibliotecilor importate:

```
import random
import time
import tkinter
import tkinter as tk
from threading import Thread
from time import sleep
from tkinter import *
from tkinter import ttk
import pyautogui
```

Figura 3 - Bibliotecile utilizate în Python

import random - ne permite să utilizăm mai multe funcții atât pentru generarea de numere aleatorii cât și pentru a alege un număr aleatoriu dintr-o secvență data sau să amestece în mod aleatoriu elemente, numere dintr-o secvență

import time - importul acestei biblioteci ne ajută să facem pauze în program (*time.sleep*) nu este o metodă tocmai bună, dar este o metodă destul de simplă, și desigur putem utiliza mai multe funcții pentru a afla timpul curent de exemplu.

import tkinter - este o metodă eficientă și simplă în același timp pentru a crea un GUI (grafic user interface) pentru utilizator. Dacă importăm și *tkinter* ca *tk* putem avea acces la diferite funcții pe care le primim.

from threading import thread - este importată clasa Thread din modulul threading ne ajută să controlăm mai bine firele de execuție când executăm mai multe operații în același timp, cu toate acestea putem evita și blocarea interfeței grafice când rulează mai multe operații în paralel.

from time import sleep - importăm *sleep* din biblioteca *time*, așa cum am spus anterior, funcția *sleep* ne ajută să suspendăm temporar execuția codului pentru o anumită perioadă de timp.

*from tkinter import ** - importă clasele și funcțiile din acest modul, iar importarea *** ne ajută să utilizăm aceste clase și funcții direct fără a specifica în mod implicit *tkinter* înaintea fiecărui nume.

from tkinter import ttk - importă clasele și funcțiile din modulul *ttk* după care putem utiliza widget-uri fără a utiliza prefixul *ttk* în față, acestea sunt utilizate mai mult pentru partea vizuală (aspectul) aplicației.

import pyautogui - ne permite să controlăm acțiunile pe ecran (*desktop*), mișcarea cursorului, apăsarea tastelor sau să dăm click-uri.

class Clicker:

```
def __init__(self):
    self.alive = None

def alive_set(self):
    self.alive = True
    print(20 * '-', 'Start farm', '_' * 20)

def alive_clean(self):
    self.alive = False
    print(20 * '-', 'Stop farm', '_' * 20)
```

avem clasa „Clicker” și respectiv constructorul *def __init__(self):* care inițializează atributul *alive* cu valoarea *None*. Mai jos avem 2 funcții (*alive_set*, *alive_clean*), una pentru a permite pornirea programului și cealaltă pentru a permite oprirea programului și a treia funcție *run*, care rulează codul respectiv cu instrucțiunile oferite spre a fi executate.

Funcția *run(self):* [ANEXA 6.1]

Se intră în bucla „*while True*” unde executăm un set de instrucțiuni într-un ciclu infinit. Primul *if self.alive*: verifică starea atributului *alive*, dacă este *True* o să intre în bucla *for*, dacă nu este *True* nu se va întâmpla nimic (dacă apăsăm butonul se trece pe *True*). În bucla *for i in range(600)* sunt efectuate acțiuni repetate (de 600 de ori). După care se verifică periodic dacă atributul „*alive*” este pe *True* și utilizăm următoarea funcție care ne arată vizual progresul barei și la fiecare ciclu progresul barei crește cu *0.16666666666666667*. Dacă înmulțim valoarea *0.16666666666666667* cu 600 de la *range(600)*, deoarece avem 600 de acțiuni repetate, progresul barei va ajunge la 100 ceea ce înseamnă că bara respectivă va fi plină (100%). Urmează o condiție de *elif self.alive is False*: (dacă atributul *alive* este *False* sau mai bine zis dacă se apasă un buton care să treacă atributul pe *False*) atunci progresul barei este oprit și resetat.

Condiția: *if i == 600 and self.alive is True:*

Dacă variabila *i* ajunge la 600 și atributul este încă pe *True* se generează un număr aleatoriu între 0 și 60, numărul generat este salvat în variabila *CEVA*, care la final ne este afișat în consolă și pe urmă programul va fi în așteptare *time.sleep(CEVA)*, durata acestuia va depinde în funcție de numărul care ne este generat. După acest proces se va intra într-o altă buclă *while True* pentru a efectua acțiuni de localizare a unor poze/imagini pe ecran și pentru a da click.

În bucla *while True* avem un bloc de *try* și *except* care ne permite să gestionăm eventualele erori care pot apărea. Se utilizează *time.sleep(0,5)* pentru a permite aplicației să se stabilizeze după care se încearcă localizarea imaginilor pe ecran.

În blocul *try*: se încearcă localizarea pe ecran a imaginii *farmimgv.png*, dacă nu se reușește localizarea acestei imagini o să avem o eroare de tip *TypeError*, fiind într-un block *try-except* putem trata această eroare în *except TypeError*. În *except TypeError*, se încearcă tratarea erorii, iar în consolă se va printa *Încă caut*, după care se încearcă localizarea pe ecran a imaginii *cent.png*, dacă este localizată pe ecran, se va muta cursorul la imaginea localizată și se va aștepta 0.5 secunde între comenzi după care se încearcă localizarea următoarei imaginii *farm.png* după care se va da click și se va deschide o fereastră nouă (Figura 2). Această fereastră rămâne deschisă mereu și se va încerca localizarea imaginilor *farmimgv.png* și *select.png*, după se va da click. Se încearcă localizarea imaginii cu numărul 3 după care se va da click, iar progresul barei se va la 0.

Se creează o instanță a clasei *Clicker()*.

clicker=Clicker() - este necesar crearea unei instanțe a unei clase pentru a putea accesa și utiliza atributele definite în clasă.

t1=Thread(target=clicker.run) – se creează obiect *Thread* și îi atribuim drept „tintă” metodă *run* a obiectului *clicker*.

t1.start()

t1 este o variabilă care stochează obiectul *Thread* pe care l-am creat anterior și este apelată metoda *start* pentru a porni execuția firului.

```
s = ttk.Style()
s.theme_use('clam')
s.configure("3.Horizontal.TProgressbar", troughcolor='black', background='red')
my_progress = ttk.Progressbar(root, style='3.Horizontal.TProgressbar',
                              orient=HORIZONTAL, length=100, mode='determinate')
my_progress.pack(padx=10, pady=5)

canvas = tk.Canvas(root, height=400, width=400, bg="#000000")
canvas.pack()

b1 = tkinter.Button(root, text="Start farm", command=clicker.alive_set)
b1.pack()
b2 = tk.Button(root, text="Stop farm", command=clicker.alive_clean)
b2.pack()
root.mainloop()
```

Figura 4 - Definirea aspectului vizual al aplicației

În prima linie se definește un stil de elemente pentru interfață grafică a aplicației, la a doua linie tema folosită este *clam*, iar în a treia linie se configurează stilul barei, *troughcolor='black'* - setează culoarea fundalului a barei de progres pe negru.

background='red' - setează culoarea de fundal a barei de progres pe roșu.

A patra linie crează un obiect *ttk.Progressbar* numit *my_progress*, cu diferite caracteristici.

root - este fereastra principală a aplicației, aici se va afișa bara de progres.

La *style* ni se specifica stilul utilizat pentru această bară.

La *orient* ni se specifică cum este orientată bara de progres, în acest caz orizontal.

Lungimea(*length*) ne este specificată lungimea barei de progres aceeași este în pixeli(dacă schimbăm lungimea barei trebuia să modificăm și valoarea de la *my_progress.step(0.1666666666666667)*, deoarece 100/600 ne da 0.1666666666666667, modul *determinat* ne permite să urmărim progresul unei acțiuni la fiecare iterație a buclei

Apelul *my_progress.pack(padx=10, pady=5)* afișează bara de progres pe fereastra principală și are doi parametri *padx* și *pady* care specifică marginea orizontală respectiv marginea verticală.

```
canvas = tk.Canvas(root, height=400, width=400, bg="#000000")
canvas.pack()
```

Figura 5 – Crearea unui obiect de tip *Canvas*

aici este creat un obiect *canvas* în care ne sunt specificate înălțimea(400 de pixeli) respectiv lățimea (400 pixeli) și *bg* negru("#000000") în hexazecimal.

```
b1 = tkinter.Button(root, text="Start farm", command=clicker.alive_set)
b1.pack()
b2 = tk.Button(root, text="Stop farm", command=clicker.alive_clean)
b2.pack()
root.mainloop()
```

Figura 6 – Crearea celor două butoane

Aici avem 2 butoane, primul buton *b1* este creat folosind clasa *Button* din *Tkinter*. La apăsarea acestui buton se va apela metoda *alive_set*. Avem două argumente, *text='Start farm'* care ne va arată pe aplicația noastră butonul sub aceasta denumire, și al doilea argument *command=clicker.alive_set* care la apăsarea acestui buton va apela metoda respectivă, dacă puneam *command=clicker.alive_set()* această metodă ar fi fost apelată fără să mai fie necesară apăsarea butonului, de aceea au fost scoase parantezele. Pe urma butonul *b1* este plasat în *root* folosind metoda *b1.pack()*.

Al doilea buton este creat la fel ca primul buton doar că acest buton este pentru *Stop Farm* și la apăsarea acestuia se va apela metoda *alive_clean*. La fel ca la primul buton avem

două argumente `command=clicker.alive_clean`. Dacă am utiliza parantezele la finalul lui `alive_clean`, de exemplu `command=clicker.alive_clean()` această metodă ar fi apelată fără să mai fie necesară apăsarea butonului. Pe urmă butonul `b2` este plasat în `root` folosind metoda `b2.pack()`.

1.6 SCHEMA REPREZENTATIVĂ PENTRU FUNCȚIONAREA CODULUI

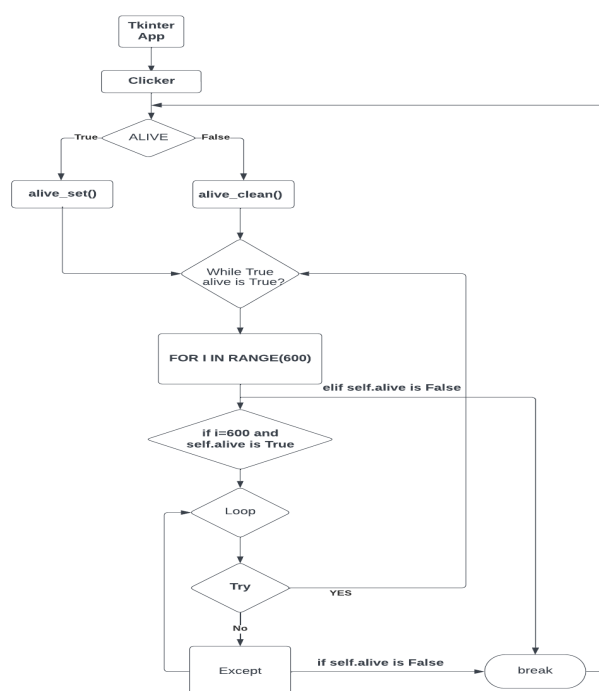


Figura 7 - Schema reprezentativă a aplicației

Explicatia pe baza schemei:

1. După importarea modulelor se creează o instanță (root) a clasei `Tk` pentru a inițializa fereastra principală a aplicației aici vor fi puse și celelalte elemente precum bara de progres, butoanele și obiectul canvas.

2. Pe urmă este definită clasa `Clicker` care conține metodele și atributele respective.

`__init__(self)` - acesta este constructorul clasei care este apelat atunci când este creată o instanță a clasei `Clicker`. Acest constructor inițiază atributul `alive` cu `None`.

3. După ce ajungem la blocul de decizie `ALIVE`, avem de ales între 2 metode:

`alive_set()` ne setează atributul `alive` pe `True` (`self.alive=True`), acest lucru ne indică că procesul a fost pornit și ne este afișat în consolă acest lucru.

`alive_clean()` ne setează atributul `alive` pe `False` (`self.alive=False`), acest lucru ne este arătat în consolă cum că procesul a fost oprit (sau trebuie oprit)

4. Se intră în blocul principal al funcției *run()* aici întregul proces este într-o buclă infinită și verifică de fiecare dată valoarea atributului *alive*. Dacă *alive* este setat pe *True*, se trece mai departe în următorul bloc *for i în range(600)*, unde se va itera un număr de 600 de ori incrementând tot odată bara de progres(*my_progress*) folosind *step()*, la fiecare iterație se va aștepta o secundă *sleep(1)*, de aici rezultă că sunt 600 de secunde. În cazul în care *alive* devine *False* în timpul iterației se oprește bara de progres și iese din buclă. Dacă nu se întâmplă acest lucru și *alive* rămâne pe *True* se trece în următorul bloc.

5. *if i=600 and self.alive is True*, dacă numărul de iterații ajunge la 600 și *alive* rămâne în continuare pe *True* se va genera un număr random(aleatoriu) între 0 și 60, un număr real deoarece am folosit *random.uniform(0,60)* care este folosit pentru generarea de numere reale într-un anumit interval, după ce numărul este generat se va aștepta un interval de timp în funcție de numărul generat (de ex. dacă numărul generat este 5.13, se va aștepta 5.13 secunde după care se va trece mai departe).

6. Se intră într-o buclă infinită(loop) în *try*: unde se caută pe ecran anumite elemente(imagini) cu ajutorul funcției *pyautogui.locateCenterOnScreen()* care primește ca argument calea către o imagine și caută aceea imagine pe ecran. Dacă imaginea respectivă este găsită funcția returnează centrul imaginii găsite. În acest bloc funcția este utilizată pentru a căuta imaginile *farmimgv.png*, *sellelect.png* și *collect.png*. După obținerea coordonatelor ale acestor imagini, sunt efectuate acțiuni de click la acele coordonate. Dacă sunt găsite toate imaginile respective și întreg procesul decurge bine la final avem *break* pentru a ieși din bucla respectivă și va reveni la prima buclă creată. În cazul în care nu se găsesc imagini în blocul *try* cu ajutorul funcției *pyautogui.locateCenterOnScreen()*, atunci se va genera o excepție de tip *TypeError*. În blocul acesta sunt efectuate următoarele instrucțiuni. Dacă atributul *alive* este *False*, bucla este întrerupă utilizând instrucțiunea *break*. În cazul în care atributul *alive* continuă să fie pus pe *True*, se va afișa „Inca caut”, si se caută imaginile respective după care se execută o serie de acțiuni pentru a ajunge la acele imagini (încercăm să găsim cele 2 imagini: *cent.png* și *farm.png*). Astfel în blocul *except* încercăm să găsim alte imagini ca să facem tot posibilul să ajungem să găsim imaginile *farmimgv.png*, *sellelect.png* și *collect.png* din blocul *try*, tratând această excepție ne permite să rulăm în continuare programul fără să fim surprinși de o eroare care ar duce la închiderea programului.

1.7 AVANTAJE ȘI DEZAVANTAJE A SCRIPTULUI:

Avantajele acestui script:

Unul dintre avantajele acestui script este că funcția *random.uniform()* generează valori aleatorii ceea ce ne permite să luăm pauze dintre acțiuni în funcție de valoarea generată implicând o altă funcție și anume *time.sleep()*. Acest lucru ne permite să imităm mai bine comportamentul uman și poate evita detectarea automată a unor acțiuni repetitive.

Utilizând biblioteca *pyautogui* pentru a efectua diferite acțiuni de click pe ecran în mod automat. Acest lucru poate fi util în cazul în care vrem să interacționăm și cu alte aplicații.

Interfața grafică este simplă deoarece se utilizează *tkinter*. Acest lucru permite pornirea și oprirea acțiunilor automate.

Putem urmări progresul vizual al acțiunilor cu ajutorul barei de progress *ttk.ProgressBar*. Acest lucru ne permite să monitorizăm și să urmărim progresul acțiunilor automate.

Automatizarea acțiunilor cu ajutorul acestui script duce la o creștere a eficienței și economisire a timpului. Scriptul poate efectua aceste acțiuni repetitive mult mai rapid decât o persoană și poate funcționa în mod continuu fără să fie necesară supravegherea. Nu în ultimul rând este imposibil să uite sau să omită pașii necesari, asigurând o desfășurare precisă a acțiunilor.

Dezavantajele acestui script:

Utilizarea modului *pyautogui* pentru a automatiza acțiunile pe ecran poate fi ineficientă în cazul în care se produc schimbări de interfață grafică sau se modifică rezoluția browser-ului/ecranului.

Utilizarea buclei infinite în funcția *run* scriptul rulează în mod continuu fără a permite oprirea într-un mod corespunzător. Acest lucru duce la utilizarea excesivă a resurselor sistemului.

Crearea aplicației utilizând modulul *tkinter* este destul de simplă și poate fi destul de dificil în cazul în care se dorește extinderea aplicației sau personalizarea ei.

În cazul în care se modifică textul în cadrul acestui joc, acest lucru nu permite detectarea imaginii respective.

2 CAMERA O3D200AD PMD 3D SENSOR

Camera o3d200AD este o cameră industrială pentru măsurarea distanței și conține o matrice PMD de 64x50 pixeli care oferă imagini de intensitate și distanță pentru fiecare pixel al scenei.

Camera o3d200AD oferă 2 interfețe:

- una este o conexiune ethernet pentru parametrizarea senzorului și obținerea datelor.
- a doua interfață este o conexiune de proces care oferă linii de intrare/ ieșire și o ieșire analogică.

Comunicația cu O3d200 se face prin 2 porturi TCP/IP:

- Primul port este 8080 pentru configurarea setărilor camerei pe baza unui protocol xmlRPC.

- Al doilea port este 50002 este folosit pentru a transfera datele de imagine pe baza unui socket TCP/IP. Transferul pentru construirea imaginii se face prin trimiterea unui singur octet către senzor.

2.1 CONCEPTELE TEHNOLOGIEI O3D

Această tehnologie utilizează un senzor special pentru măsurarea distanței. Acest senzor folosește principiul timpului de zbor pentru a detecta distanța până la obiecte. Elementele de detecție ale senzorului sunt dispuse într-o matrice. Datorită acestui principiu de măsurare a distanței, senzorul poate detecta obiecte până la 6,5 metri, orice obiect aflat la o distanță mai mare de aceasta este văzut ca și cum ar fi la 1.5 metri [5].

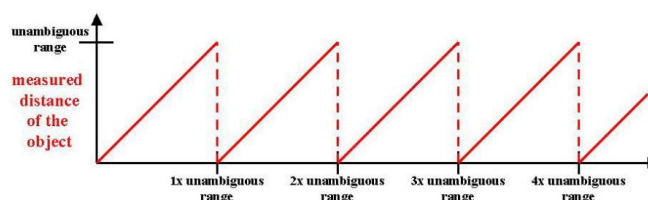


Figura 8 - Măsurarea distanței bazată pe principiul timpului de zbor folosind tehnologia PMD.

Frequency mode	0	1	2	3	4
Frequencies [Mhz]	23	20.4	20.6	23/20.4	23/20.6
Unambiguous Range	6.51m	7.38m	7.40m	45m	48m
Maximal measured distance	6.5m	6.5m	6.5m	6.5m	40m

Figura 9 - Modul cu frecvențe duale pentru extinderea intervalului fără ambiguitate

Pentru a fi extinsă gama în care se pot fi măsurate distanțele fără ambiguitate, se poate folosi modul de frecvență dublă, se fac 2 expuneri la frecvențe diferite și pe urmă se combină rezultatele dar acest lucru reduce rata de cadre la jumătate [5].

Se utilizează o lumina specială invizibilă pentru ochiul uman, numită infraroșu pentru a lumina obiectele dintr-o scenă. Această lumină este emisă de o sursă specială din interiorul camerei numită LED infraroșu [5]. Când această lumină atinge obiectele respective ea se reflectă și se întoarce înapoi spre cameră. Lentila camerei colectează lumina reflectată și o trimite către o matrice specială numită PMD-MATRIX [5]. Această matrice este alcătuită din mici elemente sensibile la lumină care pot măsura distanța până la obiecte. Pentru a calcula distanța camerei trebuie să facă 4 expuneri (4 fotografii) la lumină într-un timp foarte scurt. Aceste expuneri se fac una după alta, după care camera are nevoie de 30 milisecunde pentru a procesa datele pe care le-a obținut [5]. Pe baza datelor primite camera crează o imagine în 3 dimensiuni a scenei care ne permite să vedem și să măsurăm distanțele cu ajutorul ei [5].

2.2 Formate date de ieșire

O3d determină distanța și valoarea tonurilor de gri pentru fiecare pixel. Distanța este măsurată intrinsec într-un sistem de coordonate sferice. Centrul acestui sistem de coordonate este punctul de focalizare teoretic al senzorului, prin urmare senzorul este capabil pentru a furniza o matrice cu valori ale distanței radiale. Folosind distanța focală a lentilei sunt calculate trei matrici ale coordonatelor carteziane X,Y,Z. Fiecare matrice poate fi interogată individual sau în combinație cu altele [5].

2.3 HEARTBEAT

Dacă comunicatia dintre client și server(O3d) este întreruptă și clientul încearcă să se reconecteze aceasta încercare va eșua, deoarece O3D are încă porturile deschise. Pentru a evita această situație *HeartBeat* poate fi activată [5]. Dacă camera nu primește o comandă XML la fiecare 10 secunde aceasta își va închide porturile și clientul se poate reconecta. Desigur dacă pe ecranul camerei ni se afișează *OnLi* înseamnă că conexiunea a fost realizată cu succes și nu există posibilitatea ca un alt client să se poată conecta la cameră cât timp o conexiune este realizată [5].

2.4 SETĂRILE DE ACHIZIȚIE A IMAGINII.

Pentru optimiza performanta camerei există mai multe setări de achiziție a imaginii, un parametru important fiind timpul de expunere care se numește timp de integrare. Dacă timpul de integrare este mai lung cu atât pot fi detectate obiecte multe mai îndepărtate. Se pot vedea obiecte îndepărtate cât și apropiate în același timp dacă și numai dacă se efectueaza 2 măsurători, una folosind timpul de integrare scurt iar cealaltă lung și pentru fiecare pixel sunt selectate valorile corespunzătoare [5]. Dacă se dorește optimizarea timpului de integrare se recomandă să se lucreze în modul de integrare unică fără filtre. Dacă pixeli nu sunt nici saturați nici expuși atunci se găsește cel mai bun timp de integrare, în schimb dacă pixeli sunt. În cazul în care pixelii singulari sunt saturați ei sunt ajutați de cel de-al doilea timp de integrare astfel încât să se păstreze o performanță cât mai bună cu puțință. În cazul în care viteza este prioritară se recomandă funcționarea dispozitivului în modul de eșantionare dublă. Timpul de integrare poate varia de la 1 la 5000 de microsecunde [5]. Totalul timpului de achiziție pentru o imagine este de 4 ori mai mare decât timpul de expunere la care se mai adună și un mic timp de citire. Timpul de integrare de lungă durată este de 2000 de microsecunde aceasta fiind valoarea implicită [5].

2.5 TIMPUL DE ACHIZIȚIE A IMAGINII

O3d poate opera în diferite moduri care definesc când se pot lua imagini și când nu. Sunt moduri precum cel de rulare liberă și modul declanșat. Modul de rulare liberă preia continuu imagini în funcție de modul de achiziție curent „integrare unică/dublă, frecvență unică/dublă, medie a imaginii” [5]. În antetul imaginii ne este furnizat intervalul de timp al modului curent care este denumit „timp de evaluare”.

De fiecare dată datele sunt interogate prin trimiterea unui sir de caractere, de exemplu „xyz”, camera raspunde datelor din ultima perioadă de expunere [5]. Toate tipurile de imagini din interogările(acei sir) derivă din același timp de expunere. În cazul în care intervalul de timp dintre doua interogări este lung, toate imaginile se pierd cu excepția ultimei [5]. În cazul în care intervalul de timp este scurt, aceeași imagine este trimisă de 2 ori. Antetul din al doilea set de date va indica în intrarea „ValidImage” dacă imaginea a fost trimisă. Ambele anteturi vor avea aceeași ora de timp [5].

În modul declanșat camera face imagini doar la un eveniment de declanșare. Declanșatorul poate fi emis atât prin hardware cât și prin software iar interogarea datelor este independenta de evenimentul declanșator și funcționează analogic în modul liber [5]. Dacă trimitem o majusculă (când trimitem comandă) și nu s-a întâmplat nici un eveniment declanșator, nu se vor trimite date [5].

2.6 LIMITA DE TEMPERATURĂ, RATA MAXIMĂ A CADRULUI

Dacă se depășește limita de temperatură senzorul oprește iluminarea și emite o eroare de temperatură, el va funcționa și va raspunde la comenzile XML primite, dar nu va putea achiziționa imagini noi [5]. Când o imagine este interogată se va trimite ultima imagine valabilă. Antetul imaginii este setat la zero, acest lucru ne indică că imaginea este veche, dacă senzorul se răcește se reporneste automat iluminarea [5]. La timpii de integrare ridicată este posibil să apară o eroare de temperatură, pentru a evita deteriorarea iluminării „LED-urile pot fi pornite numai cu un ciclu maxim de 19%” [5]. În modul declanșat continuu hardware, dispozitivul limitează rata maximă a cadrului [5]. Dacă utilizăm modul de declanșare din software, utilizatorul ar trebui să aibă grijă de această limită.

2.7 FILTRAREA

Există doua categorii de filtre:

„Una în domeniul temporal și una performantă în spațiu” [5]. Cele spațiale acționează asupra unei imagini astfel încât fiecare măsurare să fie independenta de predecesorii săi. Filtrul median este filtrul implicit deoarece păstrează imaginea încă rezonabilă și oferă o filtrare rezonabilă. Filtrul mediu este cel care netezește marginile, fiind cel mai puternic [5]. Există de asemenea și un filtru „al mai multor imagini” numit „determinare medie” [5]. Pentru acest filtru, mai multe imagini sunt realizate cu viteză medie și mare înainte de efectuarea procesării datelor. Se recomandă evitarea acestui filtru în cazul în care avem obiecte în mișcare sau distanțe de schimbare rapidă [5].

2.8 FUNCȚIILE (METODELE) EXISTENTE PE SERVER.

2.1. Sensor Connection Settings	
MDAXMLConnectCP	
has to be called at the beginning of a XML-RPC session	
args[0]	IP-address of the client
args[1]	parameter is mandatory. 0=Heartbeat ON, 1=Heartbeat OFF; If ON MDAXMLHeartbeat function has to be called once within 10 seconds.
result[0]	Error code, 0 for no error
result[1]	firmware version
result[2]	sensor type
MDAXMLDisconnectCP	
args[0]	IP-address of the client
result[0]	Error code, 0 for no error
MDAXMLHeartbeat	
args	no arguments
result[0]	Error code, 0 for no error

Figura 10 - Funcția pentru conectare și deconectare a camerei

Trebuie să utilizăm această funcție pentru a ne conecta la cameră (server ul camerei) *MDAXMLConnectCP*, prin protocolul *XML-RPC*. Avem și o funcție pentru deconectare și anume *MDAXMLDisconnectCP*. Și o funcție *MDAXMLHEARTbeat* care ne ajută să menținem sau nu conexiunea.

2.2. Image Acquisition Settings

MDAXMLSetWorkingMode	
args[0]	parameter contains a 1 to turn on the image server and a 0 to turn it off. On O3D201, the live image server is switched on by default and can not be switched off. On O3D200 the live image server must be switched on prior to image data communication
result[0]	Error code, 0 for no error , 3 for invalid parameter
result[1]	TCP Port

Figura 11 - Funcție pentru pornirea server-ului de imagine

Utilizăm funcția *MDAXMLSetWorkingMode* pentru a pornit server ul de imagine, pentru pornirea acestuia argumentul trebuie setat pe 1.

MDAXMLGetFrontendData	
args	no arguments
result[0]	Error code, 0 for no error
result[1]	Integer, always 0
result[2]	Modulation Frequency Mode 0 (23MHz) 1 (20.4MHz) 2 (20.6MHz) 3 (23/20.4Mhz, 6.5m clipping) 4 (23/20.6MHz)
result[3]	Integer, 0 for single sampling an 1 for double sampling mode
result[4]	Integer, always 0
result[5]	Integer value between 1 and 5000 representing the integration time in microseconds for the first integration time (short).
result[6]	Integer value between 1 and 5000 representing the integration time in microseconds for the second integration time (long)
result[7]	Integer, 20 always
result[8]	Actual Inter Frame Mute Time in milliseconds
MDAXMLSetFrontendData	
args[0]	Integer, always 0
args[1]	Modulation Frequency Mode 0 (23MHz) 1 (20.4MHz) 2 (20.6MHz) 3 (23/20.4Mhz, 6.5m clipping) 4 (23/20.6MHz)
args[2]	Integer, 0 for single sampling an 1 for double sampling mode (default 1)
args[3]	Integer, always 0
args[4]	Integer value between 1 and 5000 representing the integration time in microseconds for the first integration time. (default 125) INT1 < INT2
args[5]	Integer value between 1 and 5000 representing the integration time in microseconds for the second integration time (default 200) INT1 < INT2
args[6]	Integer, always 20
args[7]	Inter Frame Mute Time in milliseconds. Controls the Framerate. Integer value between 0 and 10000.
result[0]	Error code, 0 for no error
result[1]	Error hint: bit coded check of parameters
result[2]	Modulation Frequency Mode
result[3]	Integer, 0 for single sampling an 1 for double sampling mode
result[4]	Integer, always 0
result[5]	Integer value between 1 and 5000 representing the integration time in microseconds for the first integration time.
result[6]	Integer value between 1 and 5000 representing the integration time in microseconds for the second integration time
result[7]	Integer, always 20
result[8]	Actual Inter Frame Mute Time in milliseconds.
When sending configuration data: The user may increase the sampling rate, if the mounting of the sensor ensures enough heat dissipation. If the sensor temperature exceeds a critical value, the illumination switches off and the temperature alarm is set. The sensor is still operational and will respond the last valid data, but indicating that the data is not valid in the header. When the sensor cooled off the illumination automatically resumes work.	

Figura 12 - Obținerea și setarea datelor utilizând cele 2 funcții

Se utilizează *MDAXMLGetFrontendData* și *MDAXMLSetFrontendData*:

- Prima functie este obținerea datelor
- A doua functie este pentru setarea datelor (nu toate datele pot fi modificate).

MDAXMLGetTrigger	
args	no arguments
result[0]	Error code, 0 for no error
result[1]	Currently used trigger type, 1 for positive edge, 2 for negative edge and 3 for free running (hardware trigger from the trigger input pin of the sensor is disabled) and 4 for software trigger
MDAXMLSetTrigger	
args[0]	0
args[1]	0
args[2]	Trigger type, 1 for positive edge, 2 for negative edge, 3 for free running (hardware trigger from the trigger input pin of the sensor is disabled) and 4 for software trigger
result[0]	Error code, 0 for no error
MDAXMLTriggerImage	
args	no arguments
result[0]	Error code, 0 for no error
MDAXMLGetDebounceTrigger	
args	no arguments
result[0]	Error code, 0 for no error
result[1]	int, 0 if trigger debouncing is turned off
MDAXMLSetDebounceTrigger	
args[0]	int, 0 for turning trigger debouncing OFF, for any other number ON
result[0]	Error code, 0 for no error
MDAXMLGetAverageDetermination	
args	no arguments
result[0]	Error code, 0 for no error
result[1]	int
MDAXMLSetAverageDetermination	
args[0]	0
args[1]	0
args[2]	Number of images, that are being averaged
result[0]	Error code, 0 for no error

Figura 13 - Gestionarea modului de declanșare

Putem utiliza funcții pentru a primi modul de utilizare curent(cele de declanșare) și la fel putem seta de altfel noi modul al tipului de declanșare.

3. Error Messages

In some communication protocols, the result is filled with an error information. The returned error code can be one of the following:

MEANING	CODE
NO_ERRORS	0
ALREADY_CONNECTED	-102
NO_ACTIVE_PRODUCT	-103
NOT_PRODUCT_LIST	-301
COULD_NOT_COPY_FILE	-900
COULD_NOT_OPEN_FILE	-901
PRODUCT_NOT_FOUND	-902
SEARCHING_FOR_INDEX	-903
PRODUCT_NAME_EXISTS	-904
COULD_NOT_SAVE_XML	-905
ACTIVE_PRODUCT_INVALID	-906
PRODUCT_ID_INCORRECT	-907
SEARCHING_PRODUCT	-908
COULD_NOT_COPY_PRODUCT_DIR	-910
INVALID_TRIGGER_MODE	-1000
INVALID_DEBOUNCED_TRIGGER	-1108
INVALID_MAC_ADDRESS	-1110
INVALID_SW_VERSION	-1111
SAVING_SENSOR_PRODUCT	-1150
SAVING_NETWORK_PARAMS	-1151
SAVING_GLOBAL_SETTINGS	-1152
GENERALDATABASE_XML_NOT_OPENED	-1200
READING_GENERALDATABASE_XML	-1201
INVALID_COMMON_INFO	-1202
INVALID_NETWORK_PARAMS	-1203
OPENING_FILE	-1500
RESULT_ID_NOT_AVAILABLE	-1600
IMAGER_NO_MEM	-1700
IMAGER_OPENING	-1701
IMAGER_NOT_OPENED	-1702
IMAGER_CAPTURING	-1703
IMAGER_GETTING_IMG	-1704
IMAGER_SHARED_MEM_CREATION	-1705
FTW_ERROR	-1800
PATH_ERROR	-1801
ERROR_HANDLING_FLASH	-1802
ERROR_INVALID_SAMPLINGRATE	-2001
ERROR_GENERAL	-4000
ERROR_SHORT_CIRCUIT	-4001
ERROR_OVER_TEMP	-4002
ERROR_HARDWARE	-4003
NO_VALID_IMAGE	-7001
INVALID_PARAM	-7200

Figura 14 - Identificarea erorilor bazata pe coduri

În funcție de codul de eroare pe care îl întâmpinăm ne uităm în poza respectivă pentru a vedea ce eroare avem.

Se utilizează Python pentru a vedea toate funcțiile (metodele) disponibile pe serverul camerei o3d200.

Methods:

```
[ 'ConnectMenu', 'DisconnectMenu', 'GetDisplayStatus', 'GetLockModus', 'MDAXMLCalculateExposureTime', 'MDAXMLConnectCP', 'MDAXMLCopyProduct', 'MDAXMLDeleteProduct',
'MDAXMLDisconnectCP', 'MDAXMLDownloadProduct', 'MDAXMLGetActiveProduct', 'MDAXMLGetAdaptiveMVTThresholdValue', 'MDAXMLGetAdaptiveMeanValueFilterStatus',
'MDAXMLGetAnalogSwitchPoint', 'MDAXMLGetAvailableFilters', 'MDAXMLGetAverageDetermination', 'MDAXMLGetCalibrationType', 'MDAXMLGetCompatibleCPVersions',
'MDAXMLGetCompatibleSensorTypes', 'MDAXMLGetComputedValue', 'MDAXMLGetConfigureOutput', 'MDAXMLGetDHCPMode', 'MDAXMLGetDataFilterSpecificValue',
'MDAXMLGetDebounceTrigger', 'MDAXMLGetDelayTime', 'MDAXMLGetDigitalSwitchPointOut1', 'MDAXMLGetDigitalSwitchPointOut2', 'MDAXMLGetDisplay', 'MDAXMLGetEditMode',
'MDAXMLGetExponentialFilterStatus', 'MDAXMLGetExponentialFilterValue', 'MDAXMLGetExternalProductSwitching', 'MDAXMLGetFokus', 'MDAXMLGetFrequency',
'MDAXMLGetFrontendData', 'MDAXMLGetGatewayAddress', 'MDAXMLGetIOConfigControlMode', 'MDAXMLGetIP', 'MDAXMLGetImageFileName', 'MDAXMLGetImageSource',
'MDAXMLGetLaser', 'MDAXMLGetMacAddress', 'MDAXMLGetMeanFilterStatus', 'MDAXMLGetMedianFilterStatus', 'MDAXMLGetNetworkParams', 'MDAXMLGetProduct',
'MDAXMLGetProductList', 'MDAXMLGetProgram', 'MDAXMLGetPulsLength', 'MDAXMLGetRanges', 'MDAXMLGetReferenceObject', 'MDAXMLGetReferencePlane',
'MDAXMLGetRegionOfInterestCP', 'MDAXMLGetRegionsOfInterest', 'MDAXMLGetReset', 'MDAXMLGetSWVersion', 'MDAXMLGetSegmentOffset', 'MDAXMLGetSegmentRefPlaneFactor',
'MDAXMLGetSegmentSTDFactor', 'MDAXMLGetSensorLocationCP', 'MDAXMLGetSensorNameCP', 'MDAXMLGetSensorTypeCP', 'MDAXMLGetSoftwareVersion',
'MDAXMLGetSoftwareVersionCP', 'MDAXMLGetSubNetmask', 'MDAXMLGetTCPPortCP', 'MDAXMLGetTemperatures', 'MDAXMLGetTrigger', 'MDAXMLGetUnambiguousRange',
'MDAXMLGetUnit', 'MDAXMLGetWorkingMode', 'MDAXMLGetXmlPortCP', 'MDAXMLHeartbeat', 'MDAXMLRenameProduct', 'MDAXMLResetFrontend', 'MDAXMLSensorReset',
'MDAXMLSetAdaptiveMVTThresholdValue', 'MDAXMLSetAdaptiveMeanValueFilterStatus', 'MDAXMLSetAnalogSwitchPoint', 'MDAXMLSetAverageDetermination', 'MDAXMLSetCalibrationType',
'MDAXMLSetConfigureOutput', 'MDAXMLSetDHCPMode', 'MDAXMLSetDataFilterSpecificValue', 'MDAXMLSetDebounceTrigger', 'MDAXMLSetDelayTime', 'MDAXMLSetDigitalSwitchPointOut1',
'MDAXMLSetDigitalSwitchPointOut2', 'MDAXMLSetDisplay', 'MDAXMLSetEditMode', 'MDAXMLSetExponentialFilterStatus', 'MDAXMLSetExponentialFilterValue',
'MDAXMLSetExternalProductSwitching', 'MDAXMLSetFocalDistance', 'MDAXMLSetFokus', 'MDAXMLSetFrontendData', 'MDAXMLSetGatewayAddress', 'MDAXMLSetIOConfigControlMode',
'MDAXMLSetIP', 'MDAXMLSetImageFileName', 'MDAXMLSetImageSource', 'MDAXMLSetLaser', 'MDAXMLSetMeanFilterStatus', 'MDAXMLSetMedianFilterStatus',
'MDAXMLSetMinAmplitudeDiffChannel', 'MDAXMLSetNetworkParams', 'MDAXMLSetProduct', 'MDAXMLSetProductCP', 'MDAXMLSetProgram', 'MDAXMLSetPulsLength',
'MDAXMLSetReferenceObject', 'MDAXMLSetReferencePlane', 'MDAXMLSetRegionOfInterestCP', 'MDAXMLSetRegionsOfInterest', 'MDAXMLSetReset', 'MDAXMLSetSegmentOffset',
'MDAXMLSetSegmentRefPlaneFactor', 'MDAXMLSetSegmentSTDFactor', 'MDAXMLSetSensorLocationCP', 'MDAXMLSetSensorNameCP', 'MDAXMLSetSoftwareVersion',
'MDAXMLSetSubNetmask', 'MDAXMLSetTCPPortCP', 'MDAXMLSetTrigger', 'MDAXMLSetUnit', 'MDAXMLSetWorkingMode', 'MDAXMLSetXmlPortCP', 'MDAXMLTriggerImage',
'MDAXMLUnlockSensor', 'MDAXMLUploadProduct', 'MDAXMLUploadImageFile', 'SetLockModus', 'system.listMethods', 'system.methodHelp', 'xmlAfterSWU', 'xmlGetCompatibleCPVersions',
'xmlGetCompatibleSensorTypes', 'xmlOPS_GetIORegister', 'xmlOPS_ReadIOEeprom', 'xmlOPS_SetIORegister', 'xmlOPS_SetSensorOpsMode', 'xmlOPS_WriteIOEeprom', 'xmlSWU',
'system.multicall']
```

Figura 15 - Funcțiile disponibile pe server al camerei o3d200 utilizand Python

Observăm că în „Figura 15” sunt destul de multe metode disponibile pe server, mult mai multe decât ne oferă documentația actuală. Se încearcă utilizarea metodei *system.methodHelp* pentru a verifica dacă se pot obține anumite informații cu privire la fiecare funcție în parte, fără succes din păcate.

2.9 CERINȚE HARDWARE

- PC cu processor Pentium III sau mai mare, frecvență ceasului min 500 Mhz
- min 128 MB RAM
- min 35 MB memorie disponibilă pe hard disk
- Placă grafică compatibilă XGZ cu rezoluție minimă de 1024x768 pixeli
- placă de rețea pentru 10Base-T/100Base-TX, TCP/IP protocol–
- PC mouse

Partea hardware:

La Process interface (1) se utilizează doar 2 polii (1 și 3) cel de alimentare la 24 de Volți și polul numărul 3 este Null ul.

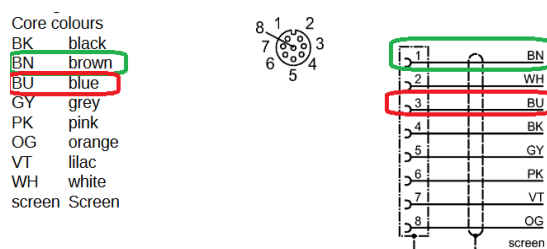


Figura 16 – Culorile firelor pentru pinii respectivi

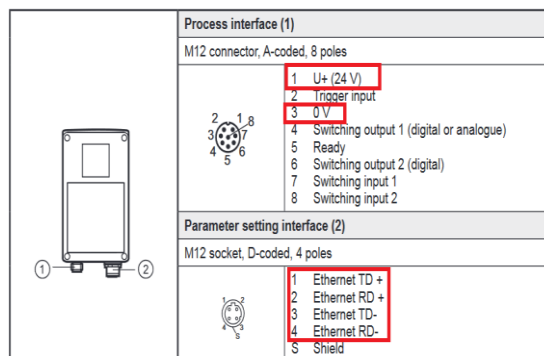


Figura 17 – Cablarea pentru cele 2 interfețe de process

La Parameter setting interface (2) se utilizează toți cei 4 polii (1, 2, 3, 4) aceste sunt folosiți pentru transmiterea și recepționarea datelor.

Pentru a ne conecta la sursă (Process interface(1)) se folosesc doar de 2 fire la pinii respectivi (cel maro la 1 și cel albastru la 3).

În documentație, se specifică că se utilizează un cablu crossover pentru Ethernet.

- Crossover cable for parameter setting connection (Ethernet), M12 connector/RJ45 connector, 4 poles
e.g. article no.: E11898 (2 m)
- Connection cable for supply voltage and process connection, M12 socket, 8 poles
e.g. art. no. E11231 (2 m, wirable cable end)

Figura 18 – Cablu crossover www.ifm.com e.g. O3D200 → Accessories[8]

În cazul în care nu dispunem de un cablu crossover se poate folosi în schimb un cablu de Ethernet este posibil să se realizeze manual un cablu crossover.

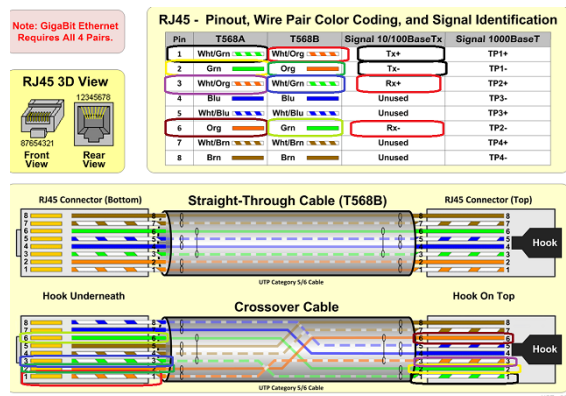


Figura 19 - cablu crossover [9]

Realizarea cablu crossover:

La *Hook On Top* (este partea unde o să conectăm firele la cameră)(1)

Hook Underneath (este partea pe care o conectăm la PC, nu este nevoie să facem modificări aici)

Este necesar observarea pinilor de la *Hook Underneath*, în funcție de pinii respectivi să realizăm cablul crossover așa cum observăm în „Figura 19”.

Cablul crossover și pinii respectivi (Parameter Setting Interface (2)):

1. Verde cu alb este primul pin (TD+)
2. portocaliu cu alb(RD+)
3. Verde(TD-)
4. Portocaliu(RD-)

După realizarea acestei părți, camera a fost poziționată către perete pentru a ne oferi o vizibilitate mult mai bună (în documentație se specifică că poziția camerei să fie în jos) .



Figura 20 - Pozitia camerei

După realizarea părții de hardware și conectării cablului (cel cu 8 poli de tip analogic) de la cameră la sursă și cel cu 4 poli (de Ethernet) la PC, putem încerca să ne conectăm la cameră utilizând Python. Pentru a realiza acest lucru avem nevoie de ip-ul și portul camerei.

5.3 Factory setting

5.3.1 Network setting IP address range

The IP address range of the device and the PC have to match.

	Network address	Station address
efector pmd3d O3D2xx	192.168.0	69
=		≠
PC	192.168.0	e.g. 10

5.3.2 Factory setting parameters

efector pmd3d O3D2xx Parameters	Description	Factory setting
DHCP	Dynamic Host Configuration Protocol	off
IP	IP address	192.168.0.69
nETm	Subnet mask	255.255.255.0
GWIP	Gateway address	192.168.0.201

Figura 21 - configuratia din fabrica ale camerei[10]

Portul a fost specificat la început (8080) pentru protocolul *xmlRPC* și 50002 pentru Ethernet. În acest caz nu se utilizează GWIP(*Gateway address*), deoarece ne conectăm direct de la PC la cameră prin cablul de Ethernet.

Utilizăm ca IDE pycharm-ul pentru a scrie codul respectiv pentru a realiza conexiunea cu camera.

Se creează un nou proiect de lucru prin opțiunea *File>NewProject>Create*

Pentru a ne crea un fișier Python: Click dreapta pe proiectul *Create>New>Python File*



Și alegem numele fișierului respectiv.

Pentru această cameră numele fișierului ales a fost ca: *o3d200xmlrpc.py*

3.0 EXPLICAREA CODULUI

Primul pas este importarea bibliotecilor:

```
import socket
import struct
import time
import xmlrpc.client
import matplotlib.pyplot as plt
import numpy as np
import select
from matplotlib import cm
```

Figura 22 – importarea bibliotecilor necesare.

Se creează 3 variabile pentru a defini server-ul la care ne conectăm și numărului portului.

O variabilă este pentru adresa ip a server-ului, a doua variabilă este pentru portul server-ului care utilizează protocolul *xmlRPC*, iar a treia variabilă este pentru protocolul *TCP/IP*.


```
server_host = "192.168.0.69"  
server_port = 8080  
server_port_socket=50002
```

Se creează o instanță a clasei *ServerProxy*.

```
proxy = xmlrpc.client.ServerProxy(f"http://{server_host}:{server_port}")
```

xmlrpc.client - aici este definită clasa *ServerProxy*, acest modul se utilizează pentru implementarea clientului.

ServerProxy - este o clasă ce ne permite comunicarea cu un server *XML-RPC*. Cu ajutorul acestei clase putem apela funcții (metode) pe care ni le pune la dispoziție server-ul.

Între paranteze se află adresa *URL* a server ului *XML-RPC* către care se va realiza conexiunea. În interiorul acoladelor se specifică adresa ip a server-ului la care se va conecta și respectiv portul. Este necesară utilizarea *f* pentru a se putea formata adresa completă a a acestui server.

Apelăm o funcție care este disponibilă pe server pentru a vedea toate funcțiile care sunt disponibile pe server după care utilizăm *print* pentru a ne afișa toate funcțiile disponibile în consolă.

```
methods = proxy.system.listMethods()  
print("\nMethods:\n", methods, "\n")
```

Apelăm o metodă disponibilă pe server pentru a ne conecta la camera respectivă.

```
Conectare = proxy.MDAXMLConnectCP(server_host, server_port)  
time.sleep(1)  
if Conectare == [0, '4041', '03D200AD']:  
    print('M-AM CONECTAT LA SERVER')  
elif Conectare == [-120]:  
    print("Sunt deja conectat la server")
```

Figura 23 – Apelarea metodei *MDAXMLConnectCP* pentru a ne conecta la server

Apelăm următoarea funcție de mai jos și setăm argumentul pe 1 pentru a activa serverul de imagine.

```
workingmode=proxy.MDAXMLSetWorkingMode(1)  
print("WORKING MODE ACTIVATED!!!!!!",workingmode)
```

Se creează o instanță a clasei *socket*.

```
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

În interiorul parantezelor `socket.AF_INET` specifică că utilizăm o adresă *IPv4*, *AF(Address Family)*, iar `socket.SOCK_STREAM` specifică că folosim un protocol *TCP*.

Se realizează conexiunea cu serverul, în interiorul parantezelor sunt 2 argumente care specifică funcției *connect* adresa ip a server-ului și portul respectiv către cine trebuie realizată conexiunea.

```
client_socket.connect((server_host, server_port_socket))
```

Definim o variabilă și îi atribuim o valoare pe care o să o utilizăm în program.

```
TOTAL_DATA=39528
```

Definim o variabilă și îi atribuim un sir de caractere sau un caracter

```
command='xyz'
```

Trimitem comanda de mai sus către server, dar înainte de a o trimite o codificăm folosind *encode* care convertește șirul de caractere într-o secvență de octeți.

```
client_socket.send(command.encode())
```

Funcția *send* este utilizată pentru trimiterea datelor către server în cazul nostru variabila *command*

Definim o variabilă și îi atribuim o valoare vidă

```
DATE_PRIMITE=b''
```

Utilizăm un block *try* și *except* și în interiorul acestei structuri utilizăm o buclă pentru colectarea datelor, sunt utilizate și câteva printuri pentru a vedea lungimea datelor și pentru a vedea datele în format hexazecimal.

```
try:
    print("Am intrat in blocul try")
    while len(DATA_PRIMITE)<TOTAL_DATA:
        print("Am intrat in bucla while")
        data=client_socket.recv(1460)
        print('Lungimea datelor:',len(data))
        print('DATA IN HEX : ',data.hex())
        if not data:
            break
        DATE_PRIMITE = DATE_PRIMITE + data
except IOError as e:
    print('A aparut o eroare')
    pass
```

Figura 24 – Colectarea datelor și afișarea lungimea acestora

Această structură a codului funcționează în felul următor. Se intră try, pe urmă se intră în bucla *while len(DATA_PRIMITE)<TOTAL_DATA*.

Aceasta bucla *while* are o condiție care ne spune ca: *cat timp* lungimea datelor pe care le-am primit (*DATE_PRIMITE*) este mai mică decât totalul de date (*TOTAL_DATA*) pe care le-am atribuit variabilei *TOTAL_DATA=39528* atunci se intră în interiorul buclei, se afișează că am intrat în bucla respectivă după care colectăm date utilizând *data=client_socket.recv(1460)*, colectăm 1460 de octeți, acești octeți sunt salvați în *data*, după care ne este afișată lungimea datelor și desigur datele în format hexazecimal urmând ca pe urma datele să fie concatenate *DATE_PRIMITE= DATE_PRIMITE + data*, toate datele după fiecare ciclu sunt salvate în *DATE_PRIMITE* până nu mai este îndeplinită condiția *while len(DATA_PRIMITE)*.

Afișarea totalul datelor în hexazecimal cât și totalul octeților:

```
DATE_PRIMITE_HEX=DATE_PRIMITE.hex()
print('TOTAL DATA BYTES: ',DATE_PRIMITE)
print('DATE_PRIMITE_HEX',DATE_PRIMITE_HEX)
```

Acum că avem toți octeții, o parte îi împărțim în HEADER și cealaltă parte în DATA.

```
HEADER_BYTES = DATE_PRIMITIVE[:376]#
print("PRIMUL HEADER: ",len(HEADER_BYTES))
DATA_BYTES = DATE_PRIMITIVE[376:13176]
print("DATA_BYTES 0: ",len(DATA_BYTES))
HEADER_BYTES1=DATE_PRIMITIVE[13176:13552]
print("AL DOILEA HEADER: ",len(HEADER_BYTES1))
DATA_BYTES1=DATE_PRIMITIVE[13552:26352]
print("DATA_BYTES 1: ",len(DATA_BYTES1))
HEADER_BYTES2=DATE_PRIMITIVE[26352:26728]
print("AL TREILEA HEADER: ",len(HEADER_BYTES2))
DATA_BYTES2=DATE_PRIMITIVE[26728:]
print("DATA_BYTES 2 : ",len(DATA_BYTES2))
```

Figura 25 – Împărțirea octeților în HEADER și DATA

În primul *HEADER_BYTES* intră primii 376 de octeți, în al doilea *HEADER_BYTES* intră alte 376 și în al treilea *HEADER_BYTES* tot 376.

În *DATA_BYTES* intră 12800 de octeți, în cel de-al doilea intră tot 12800, iar în cel de-al treilea tot 12800.

Convertirea octeților în float:

```
HEADER_FLOAT=[]
#CONVERSIE IN FLOAT PENTRU HEADER
for i in range(0, len(HEADER_BYTES), 4):
    grup = HEADER_BYTES[i:i+4]
    valoare = struct.unpack('!f', grup)[0]
    HEADER_FLOAT.append(valoare)
print("HEADER FLOAT 0: ", HEADER_FLOAT)

DATA_FLOAT=[]
#CONVERSIE IN FLOAT PENTRU DATA
for i in range(0, len(DATA_BYTES), 4):
    grup = DATA_BYTES[i:i+4]
    valoare = struct.unpack('!f', grup)[0]
    DATA_FLOAT.append(valoare)
print("DATA FLOAT 0: ", DATA_FLOAT)

#AL DOILEA HEADER FLOAT SI DATA
HEADER_FLOAT1=[]
#CONVERSIE IN FLOAT PENTRU HEADER
for i in range(0, len(HEADER_BYTES1), 4):
    grup = HEADER_BYTES1[i:i+4]
    valoare = struct.unpack('!f', grup)[0]
    HEADER_FLOAT1.append(valoare)
print("HEADER FLOAT 1: ", HEADER_FLOAT1)

DATA_FLOAT1=[]
#CONVERSIE IN FLOAT PENTRU DATA
for i in range(0, len(DATA_BYTES1), 4):
    grup = DATA_BYTES1[i:i+4]
    valoare = struct.unpack('!f', grup)[0]
    DATA_FLOAT1.append(valoare)
print("DATA FLOAT 1: ", DATA_FLOAT1)

#AL TREILEA HEDER FLOAT SI DATA
HEADER_FLOAT2=[]
#CONVERSIE IN FLOAT PENTRU HEADER
for i in range(0, len(HEADER_BYTES2), 4):
    grup = HEADER_BYTES2[i:i+4]
    valoare = struct.unpack('!f', grup)[0]
    HEADER_FLOAT2.append(valoare)
print("HEADER FLOAT 2: ", HEADER_FLOAT2)

DATA_FLOAT2=[]
#CONVERSIE IN FLOAT PENTRU DATA
for i in range(0, len(DATA_BYTES2), 4):
    grup = DATA_BYTES2[i:i+4]
    valoare = struct.unpack('!f', grup)[0]
    DATA_FLOAT2.append(valoare)
print("DATA FLOAT 2: ", DATA_FLOAT2)
```

Figura 26 – Împărțirea octeților în float

Se definește o listă goală pentru *HEADER_FLOAT* cât și pentru *DATA_FLOAT*. În *for i in range(0,len(HEADER_BYTES),4)* se va itera variabila în acel range, de la 0 până la

lungimea *Headerului* (376 la primul *Header*), pasul fiind de 4 (iar o să fie interat din 4 în 4 (0,4,8...376) până la 376 pentru primul *HEADER*).

La *grup=HEADER_BYTES1[i:i:4]*, împărțim octeții în grupuri de câte 4, după împărțirea octeților convertim grupul respectiv într-o valoare reală(float), *valoare=struct.unpack('!f',grup)[0]* valoarea convertită se va adăuga în lista pe care am definit-o la început *HEADER_FLOAT.append(valoare)*, urmând ca la final să ni afișeze toate valorile din lista respectivă. Acest lucru se aplică și la celelalte conversi (*DATA_FLOAT*, *HEADER_FLOAT1*, *DATA_FLOAT1*, *HEADER_FLOAT2*, *DATA_FLOAT2*).

În acest pas o să se creeze 3 matrice (X,Y,Z).

```
'''
#MATRICE CU NUMPY!

# PRIMA MATRICE X

print(('Lungimea datelor pentru prima matrice :',len(DATA_FLOAT)))
MATRICE1X=np.array(DATA_FLOAT)
MATRICE2X=MATRICE1X.reshape(50,64)
MATRICE3X=MATRICE2X.transpose()
print("A TREIA MATRICE X: ",MATRICE3X)

# #A DOUA MATRICE Y
print(('Lungimea datelor pentru a doua matrice :',len(DATA_FLOAT1)))
MATRICE1Y=np.array(DATA_FLOAT1)
MATRICE2Y=MATRICE1Y.reshape(50,64)
MATRICE3Y=MATRICE2Y.transpose()
print("A DOUA MATRICE Y:",MATRICE3Y)

#
# #A TREIA MATRICE Z
print(('Lungimea datelor pentru a treia matrice :',len(DATA_FLOAT2)))
MATRICE1Z=np.array(DATA_FLOAT2)
MATRICE2Z=MATRICE1Z.reshape(50,64)
MATRICE3Z=MATRICE2Z.transpose()
print("A TREIA MATRICE Z: ",MATRICE3Z)
```

Figura 27 – Crearea celor 3 matrice de dimensiunea(64x50

Se creează o matrice unidimensională dintr-o listă folosind numpy:

MATRICE1X=np.array(DATA_FLOAT). (1)

Se creează o matrice de dimensiunea (50 de linii,64 de coloane) modificând forma matricei *MATRICE1X*.

MATRICE2X=MATRICE1X.reshape(50,64) (2)

Se creează o noua matrice folosind *transpose* pe matricea anterioară(*MATRICE2X*) astfel încât liniile să devină coloane,iar coloanele să devină linii.

$MATRICE3X = MATRICE2X.transpose()$ (3)

În final se obține $MATRICE3X$ de dimensiunea (64 de linii, 50 de coloane).

Același procedeu se aplică și pentru $MATRICEA Y$ respectiv $MATRICEA Z$. (1)(2)(3)

Următorul pas după obținerea matricelor se realizează atât grafice 2d cât și grafice 3d folosind biblioteca *matplotlib*

GRAFICE 3D pentru cele 3 matrici: $MATRICE3X$, $MATRICE3Y$, $MATRICE3Z$.

```
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
```

```
surf = ax.plot_surface(MATRICE3X, MATRICE3Y, MATRICE3Z, cmap=cm.coolwarm,  
                      linewidth=0, antialiased=False)
```

GRAFICE 2D pentru cele trei matrici: $MATRICE3X$, $MATRICE3Y$, $MATRICE3Z$.

```
plt.imshow(MATRICE3X, cmap='viridis')  
plt.colorbar()  
plt.show(block=True)
```

```
plt.imshow(MATRICE3Y, cmap='viridis')  
plt.colorbar()  
plt.show(block=True)
```

```
plt.imshow(MATRICE3Z, cmap='viridis')  
plt.colorbar()  
plt.show(block=True)
```

3.1 UTILIZAREA WIRESHARK/PYTHON

Se utilizează wireshark pentru a verifica lungimea datelor pe care le primim de la cameră utilizând doar o singură comandă, de exemplu `command='x'`.

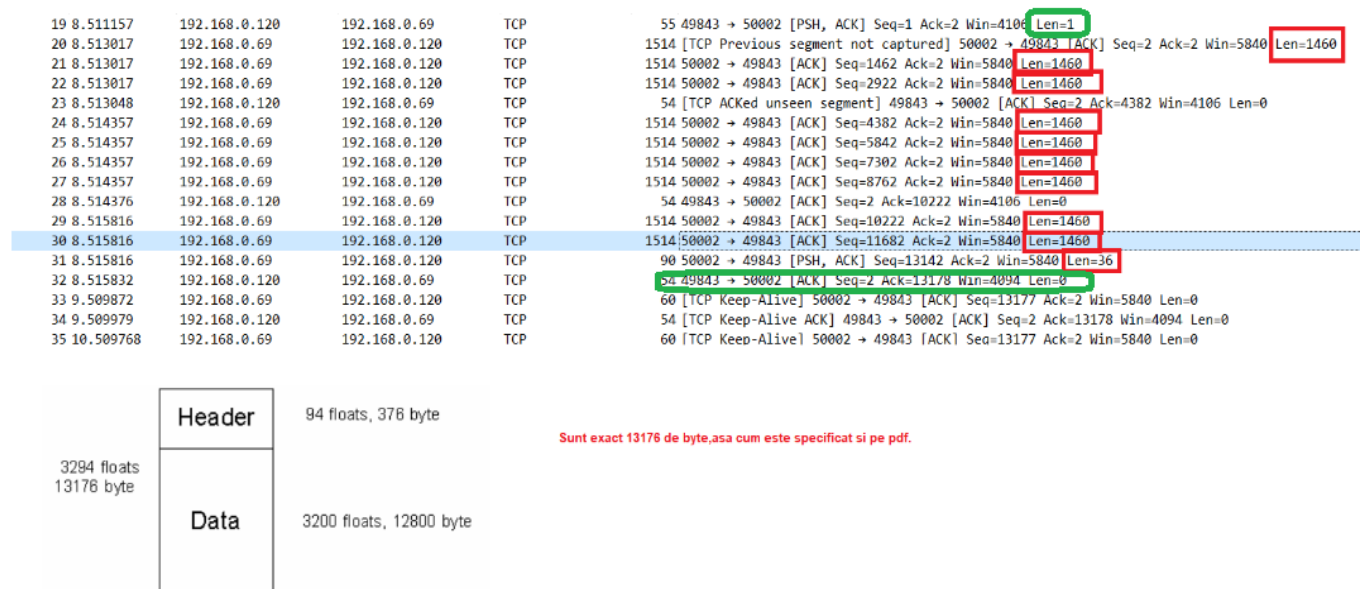


Figura 28 - Verificarea lungimii datelor primite de la cameră utilizând wireshark

După cum se poate observa în wireshark se trimite un singur octet. Iar camera trimite 3 pachete, fiecare pachet având 1460 octeți (mai puțin ultimul). După aceea se confirmă primirea celor 3 pachete, camera raspunde și trimite încă 4 pachete, se confirmă primirea celor 4 pachete, și trimite încă 3 pachete și la final se specifică că nu mai are ce să ne trimită și că se dorește încheierea transmiterii datelor. Se confirmă încheierea transmiterii datelor după care se trimit semnale periodice pentru menținerea conexiunii active (*TCP Keep-Alive*)

Sunt 9 pachete a câte 1460 de octeți și unul de 36 de octeți în total sunt 13176 de octeți, 376 sunt în HEADER și 12800 sunt în DATA.

Aceste date sunt afișate atât în wireshark cât și în Python.

Figura 29 - Verificarea lungimii datelor primite de la cameră utilizând Python

8 8 8 8 8 8 8 8 8 8

3f64a7003f64a7003f5455903f6218203f5821903f4012483f11de003e3ce7003ed138403e3ecf003e3edad0

Pentru a ne putea verifica „3f64a700” îl punem în „HEXADECIMAL REPRESENTATION(FIG14).



- 30 -

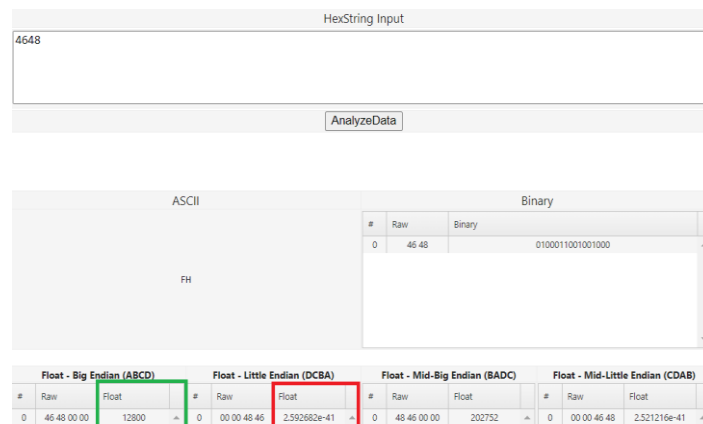


Figura 34 - Menținerea ordinii octeților în conversia datelor utilizând „f”

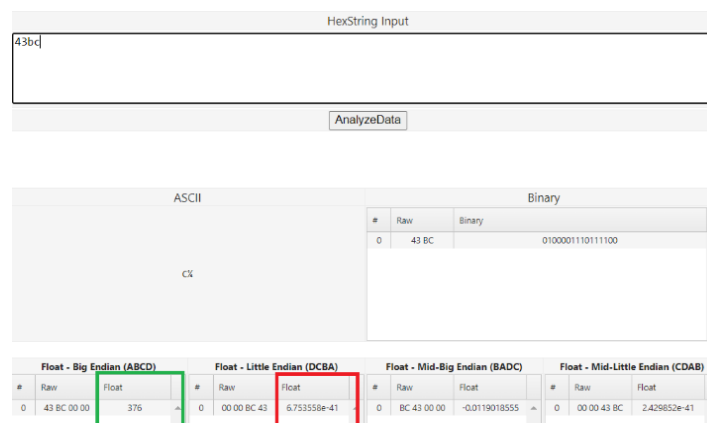


Figura 35 - Inversarea ordinii octeților utilizând „f”

Se utilizează „f” în codul nostru [26]

3.3 GRAFICELE ÎN URMA DATELOR PRIMITE:

Pentru comenzile x,y,z cu obiect în față 2d

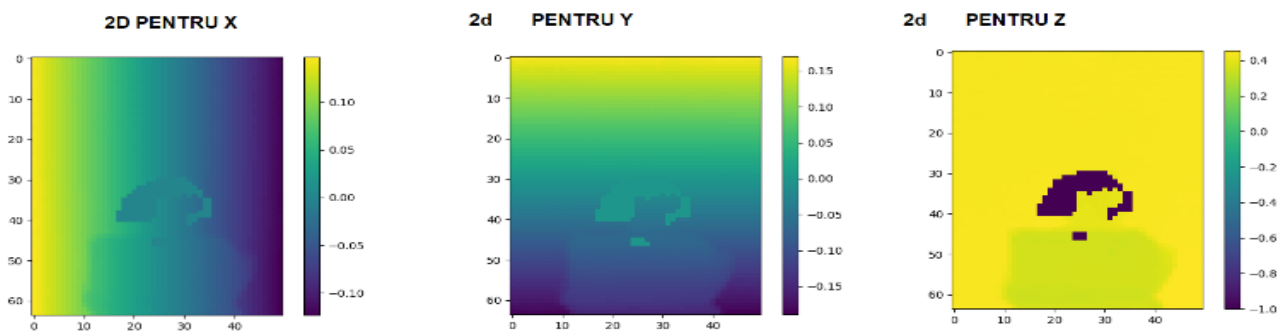


Figura 36 - Grafice pentru comanda x, y, z

Pentru comenzile e,f,g 2d

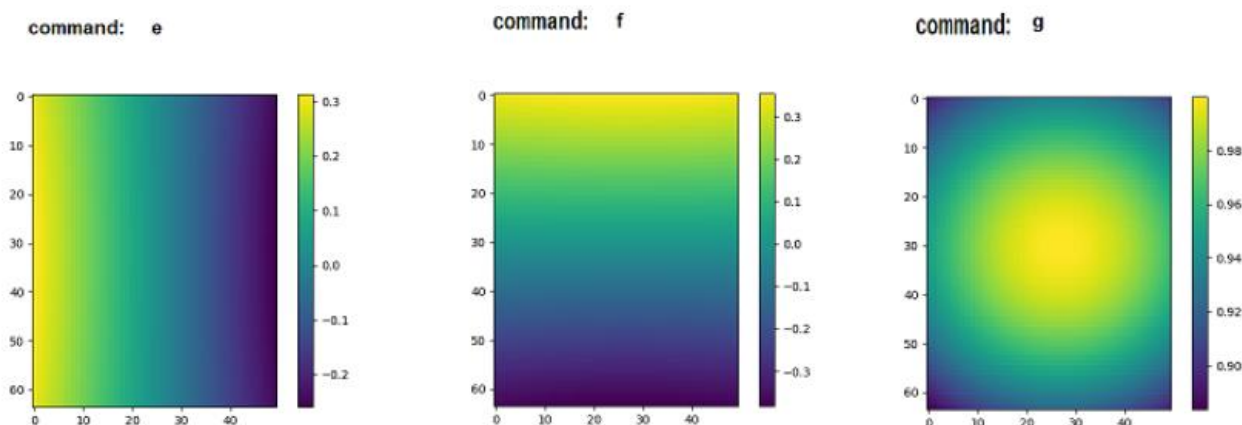


Figura 37 - Cele 3 grafice pentru comenzile e,f,g

Grafice 3d pentru comanda „xyz” văzută de sus pentru un alt obiect(o cutie)

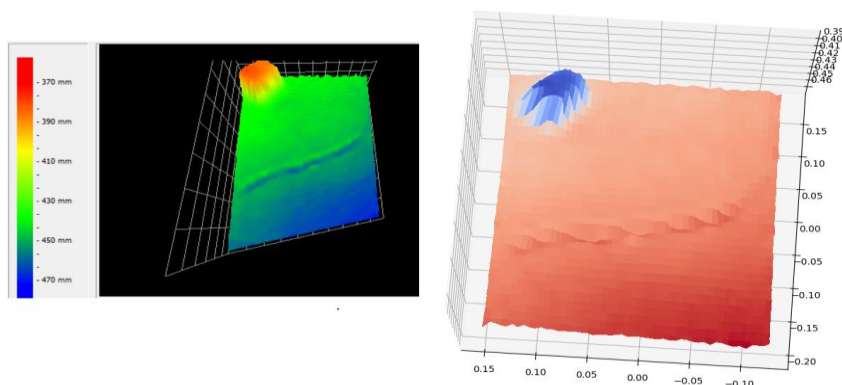


Figura 38 - Vizualizarea 3d a unui obiect, vedere de sus.

Toate aceste comenzi se află în documentație:

Imagetype	Request Byte	ID	Unit	Range	Too Far/ Dark	To Close/ Bright
radial distance	d/D	1	m	0-40	-2	-1
grey image	i/I	3	-	0-1800	-2	-1
x coordinate of line of sight vector	e/E	5	-	0-1	-	-
y coordinate of line of sight vector	f/F	6	-	0-1	-	-
z coordinate of line of sight vector	g/G	7	-	0-1	-	-
cartesian x coordinates	x/X	8	m	0-40	0	0
cartesian y coordinates	y/Y	9	m	0-40	0	0
cartesian z coordinates	z/Z	10	m	0-40	-2	-1
stop the socket connection	q					

Figura 39 - Caractere specifice in achizitia de imagini

Fiecare caracter care se va utiliza o sa trimită un răspuns cu datele respective în functie de caracterul utilizat.

Pentru sincronizarea cu senzorul(modul de rulare liberă) se recomandă începerea cu o literă majusculă, litera majuscula indicând că, datele trebuie să provină din achiziția de imagini care nu a mai fost trimisă până acum. Se poate utiliza și un sir de interogare consecutivă de exemplu „XYZ”. Acest mecanism se poate utiliza pentru sincronizarea cu camerele declanșate. Putem considera comenzile cu majuscule ca având rolul de „blocare”.Comanda utilizată pentru închiderea conexiuni este „q”.

3.4 INFORMATII DESPRE HEADER

Struktura „ImageHeaderInformation”

```
typedef struct ImageHeaderInformation {
    float DataSize; // Image data size in Bytes without header
    float HeaderSize; // Size of the header
    float ImageType; // type of image (ID)
    float Version; // version number of data format
    float SamplingMode; // single or double integration
    float ZlibMode; // zlib status 0,1,2,3 bit coded
    float FrequencyMode; // frequency mode
    float UnambiguousRange; // unambiguous range of current frequency
    float EvaluationTime; // time needed by image evaluation [ms]
    float IntegrationTime_Exp0; // first int. time single sampling mode [us]
    float IntegrationTime_Exp1; // second int. time double sampling mode [ms]
    float Second; // seconds of time stamp
    float Useconds; // microseconds of time stamp
    float MedianFilter; // median filter status
    float MeanFilter; // mean filter status
    float Internal_a[4];
    float ValidImage; // 1: new data, 0: data was queried before
    float ErrorCode;
    float Internal_b[3];
    float CurrentTriggerMode; // configured trigger mode
    float Internal_c[2];
    float MinProcessValue; // smallest of all process values (03D200)
    float MaxProcessValue; // highest of all process values (03D200)
    float IfTime; // Inter Frame Mute time [ms]
    float Internal_d[28];
    float ProcessValues[32]; // value of first 32 ROI's of 03D200
    float Internal_e[4];
    float ImageHeaderSize;
};
```

Figura 40 - Informatii despre *Header*

Această structură a headerului este compusă din 94 de *float* iar ordinea octeților este în *big-indian*.

Matricea de 64x50 are 3200 de float, 376 de octeți sunt reprezentați de *HEADER* iar 12800 sunt reprezentați de *DATA*.

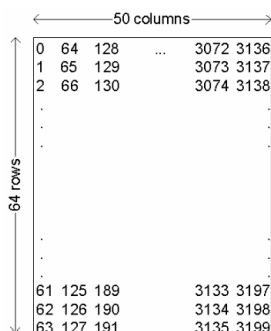


Figura 41 - Matricea cu cele 3200 de valori reale

4 DETECTAREA UNEI PERSOANE ȘI TRANSMITEREA DATELOR CĂTRE *LOGO!SOFT(PLC)* PRIN INTERMEDIUL PROTOCOLULUI *MODBUS TCP/IP*

Ce este Modbus?

Modbus este un protocol de comunicație utilizat în general în automatizarea industrială pentru a permite schimbul de date/informații între dispozitive [13]. A fost dezvoltat de către Modicon în 1979 și a devenit esențial pe partea industrială [13].

Prima dată a fost realizată o conexiune serială pe urmă s-a trecut pe ethernet.

Există câteva variante principale ale protocolului Modbus cum ar fi: **Modbus RTU**, *ASCII*, *TCP/IP* fiecare dintre acestea având caracteristici și modalități de transmitere specifice [15].

Modbus RTU - această variantă utilizează transmisia în format binar (*RTU*-remote terminal unit) [15]. Datele sunt transmise sub formă unor secvențe de biți și pot fi utilizate atât în comunicația pe portul serial *RS-232* cât și *RS-485*. **Modbus RTU** este cunoscut pentru eficiența sa în transmiterea datelor și este utilizat destul de frecvent în aplicațiile industriale.

Modbus ASCII - această variantă utilizează formatul *ASCII* pentru transmiterea datelor. În cazul acesta datele sunt reprezentate sub forma unor caractere *ASCII* și utilizează același port de comunicare ca *Modbus RTU* și anume *RS-232*. Acest **Modbus** este mai puțin eficient în transmiterea datelor decât **Modbus RTU** [15].

Modbus TCP/IP-această variantă utilizează rețele *TCP/IP*(Ethernet) pentru transmiterea datelor. **Modbus TCP/IP** permite comunicarea rapidă și eficientă între dispozitivele conectate la o rețea Ethernet [15]. Prin utilizarea **Modbus TCP/IP** dispozitivele pot fi integrate în sisteme de control și pot fi accesate și monitorizate de la distanță prin intermediul rețelei.



Figura 42 - MODELUL OSI [11]

În domeniul rețelelor de calculatoare, termenul de „STIVĂ OSI” se referă la o stivă de protocoale OSI(Open Systems Interconnection) [11]. Acest model descrie modul în care comunicațiile ar trebui să aibă loc.

1. Nivelul fizic se ocupă cu transmiterea și recepționarea semnalelor între dispozitivele de rețea, fără a le modifica în vreun fel. Acest strat se concentrează atât pe transmiterea electrică optică sau radio prin diferite medii fizice(cablurile de cupru, fibră optică, unde radio) [12].

Principalele funcționalități ale stratului fizic:

- Caracteristici fizice ale interfeței
- Transmiterea biților
- Sincronizare și Multiplexarea
- Detectarea și corectarea erorilor[12]

La nivelul fizic biții sunt reprezentați ca semnale electrice iar subnivelul PLS(Physical Signaling Sublayer) asigura codificarea si decodificarea acestora(la trimiterea datelor bitii sunt codificati in semnale electrice iar la destinatie sunt decodificati datorita acestui *PLS*). Stratul fizic asigură transmiterea datelor între dispozitivele de rețea dar nu oferă și control asupra acestora.

Caracteristici fizice ale interfeței:

-este necesar să știm specificațiile mecanice ale conectorilor și porturilor utilizate referim la formă, dimensiune și mecanisme de fixare a conectorilor.

Transmiterea biților prin intermediul mediilor fizice.Acest proces presupune convertirea datelor digitale în semnale electrice optice sau radio care pot fi transmise mai departe prin fibră optică, cabluri de cupru sau unde radio[12].Conversia datelor este necesară deoarece altfel nu am putea transmite datele prin medii fizice și desigur pentru a putea fi interpretata de către un receptor.Pentru realizarea acestor conversii se folosesc diverse tehnici în funcție de modul prin care alegem să transmitem datele.

Dacă transmisia se face utilizând cabluri de cupru datele digitale pot fi convertite în semnale electrice, iar nivelul de tensiune este utilizat pentru a reprezenta biții în 0 și 1[12]. Aici avem diverse tehnologii de modulație:

- modulația în amplitudine(AM)
- modulația în frecvență(FM)

-modulația în fază(PM)

Modulația de amplitudine se modifică amplitudinea semnalului în funcție de valorile biților transmiși, o amplitudine mai mare poate reprezenta un bit 1 și o valoare a a amplitudinii poate reprezenta un bit 0 [12].

Modulația de frecvență implică modificarea semnalului electric în funcție de valorile biților, o anumită frecvență poate reprezenta un bit,iar o altă frecvență poate reprezenta un bit 0 [12].

Modulatia în fază - se modifică faza semnalului electric în funcție de valorile biților, o anumită fază poate reprezenta un bit 1, o alta fază poate reprezenta un bit 0 [12].

Sincronizare și Multiplexare

Prin sincronizare semnalele sunt transmise și recepționate într-un mod coordonat ceea ce permite o comunicare eficientă între dispozitive

Multiplexarea este procesul de transmitere a mai multor semnale simultan prin intermediul unui singur canal fizic, ceea ce permite o utilizare eficientă a lățimii de bandă și a resurselor disponibile dintr-o rețea [12]. Există mai multe tipuri de semnale(Multiplexarea în frecvență, Multiplexarea în timp, multiplexarea în diviziune de lungime de unda) [12].

Atât multiplexarea cât și sincronizarea sunt necesare în gestionarea transmisiei de date în rețele de comunicații asigurând o comunicare corectă, eficientă și fiabilă între dispozitivele implicate [12].

Detectarea și corectarea erorilor este o etapă importantă atunci când vine de transmiterea datelor [12]. Această implică verificarea integritatea datelor, iar în cazul în care avem erori acestea să fie corectate.

Există mai multe metode pentru a verifica dacă datele au fost afectate de erori.

- Coduri de detecție a erorilor
- Coduri de corecție a erorilor
- Reincercarea transmisiei
- TTL

Nivelul 2: Legătura de date

Are rolul de a facilita transmiterea datelor între nodurile rețelei. Principala funcție a sa constă în asigurarea unei transiteri fără erori a informațiilor de la un nod la altul, prin intermediul nivelului fizic al rețelei [12]. Când pachetul de date ajunge într-o rețea, acest

nivel preia controlul asupra acestuia și se asigură că este transmis corect către destinatar utilizând MAC(Media acces Control) [12]. Astfel, acest nivel este important pentru a ne asigura că transferul este unul eficient și sigur al informațiilor între diferite dispozitive conectate la rețea.

Legătura de date este împărțită în 2 subnivele:

- *Logical link Control(LLC)*
- *Media Acces Control(MAC)*

Controlul legăturii logice se asigură că datele sunt transmise într-un mod corect și fără erori. Tot LLC se ocupă și cu gestionarea fluxului de date, cantitatea de date transmisă pentru a evita suprasolicitarea rețelei [12].

Control Media Acces(MAC) este responsabil cu accesul la mediul fizic al rețelei. MAC se ocupă cu transmiterea și împărțirea resurselor de comunicație pentru a evita coliziunile și să se asigure o transmitere eficientă [12]. Atunci când un packet este primit la nivelul 2(Legătura de date) acesta este împărțit în cadre. Putem spune că adresa MAC este o adresă unică la nivel mondial(fiecare dispozitiv are o adresă unică).

Nivelul 3: Rețea

Dacă nivelul 1 și nivelul 2 se ocupă mai mult pe partea fizică, nivelul 3 se ocupă cu rutarea și gestionarea traficului de date în cadrul unei rețele. Acest nivel utilizează IP ul pentru a determina cel mai eficient traseu pentru transmiterea pachetelor de date între rețele [16].

Serviciile oferite de acest protocol sunt următoarele:

- Împachetarea
- Rutarea
- Redirectionarea

Împachetarea este un proces în care datele pe care le trimitem prin rețea sunt ambalate în pachete speciale [16]. Să ne imaginăm că avem niste scrisori pe care dorim să le trimitem prin poștă. Pentru a le proteja și pentru a le trimite într-un mod organizat, vom împacheta scrisorile în plicuri. La fel se întâmplă și aici, când trimitem date prin rețea, acestea sunt împachetate în pachete de rețea, adică pachetul are un antet special. Acest antet conține informații importante, cum ar fi adresa sursă(de unde provin datele) și adresa destinație(unde să ajungă datele) [16]. Apoi pachetul este trimis prin rețea până ajunge la destinație. Pachetul este deschis și datele sunt extrase din el. Astfel, datele pot fi utilizate de către program sau aplicația care le așteaptă.

Rutarea este procesul de a muta datele de la un dispozitiv la altul [16]. Să ne imaginăm că dorești să trimiți o scrisoare unui prieten care locuiește în alt oraș și dorești ca această scrisoare să ajungă cât mai repede cu putință. Cum se întâmplă asta?

Într-o rețea sunt mai multe rute disponibile, la fel ca atunci când mergi cu mașina și dorești să mergi în alt oraș, alegi ruta cea mai scurtă și mai puțin costisitoare [12]. Procesul de rutare își propune să găsească cea mai bună rută pentru a trimite datele la destinație.

Redirecționarea înseamnă punerea pachetului pe ruta potrivită pentru a ajunge la destinația finală. Când un router primește un pachet el se uita în tabela(de rutare)pentru a vedea unde trimite mai departe pachetul, ce ruta trebuie sa aleagă pentru a ajunge la destinația finală.

Nivelul 4: Transport

Preia datele de la nivelul superior(cel de aplicație) și apoi le împarte în segmente de dimensiuni mai mici și pe urmă le duce la stratul inferior(cel de rețea) pentru a le livra. Acest strat se asigură că datele care au fost trimise respectă aceeași secvență în care au fost primite. Acest nivel se împarte în 2 categorii [17]:

-TCP

-UDP

Transmission Control Protocol(TCP) este utilizat pentru transmiterea de date și singura diferență între acest protocol și cel UDP este că TCP confirmă dacă pachetul de date a fost primit sau nu și cere retransmiterea pachetului.

-UDP acest protocol este exact opusul TCP, nu confirmă dacă pachetul a fost primit sau nu și nu cere retransmiterea pachetului, ceea ce înseamnă că se pot pierde.

Nivelul 5: Sesiune realizează canale de comunicare, sesiuni, între dispozitive [18]. Acest nivel este responsabil cu deschiderea sesiunilor și se asigură că acestea rămân funcționale și deschise până când datele sunt transferate după care se închide sesiunea când nu se mai realizează comunicarea, dacă se întâmpla ca sesiunea să fie întrerupă dispozitivele pot relua transferul numai în cazul în care la acest nivel este setat un punct de control, pentru a se putea relua transferul de la acel punct de control.

Nivelul 6: Prezentare

La acest nivel datele sunt pregătite pentru următorul nivel cel de aplicație. Aici este definit modul în care două dispozitive codifica cripteaza și comprimă datele, astfel încât datele să fie primite într-un mod corect la celalalt capăt. La acest nivel datele sunt preluate și transmise la nivelul de aplicație și mai departe datele sunt pregătite pentru nivelul inferior cel de sesiune [19].

Nivelul 7: Aplicație

La acest nivel se facilitează comunicarea și informațiile care sunt transferate între aplicațiile pe care le utilizăm pe serverele respective la acele servicii(servere,site-uri web) [20].

4.1 CVLIB YOLO OBJECT DETECTION

Biblioteca CVLIB este simplă și utilizat open-source pentru computer vision, atât pentru detectarea imaginilor cât și clasificarea lor.

Utilizăm cvlib din următoarele motive:

- Ușurință în utilizare
- Extensibilitate și modularitate
- Simplitate

Biblioteca mai are încă 2 biblioteci pe care ar trebui să le utilizăm pentru a putea folosi această bibliotecă fără să avem probleme [21].

- *OpenCV*
- *Yolo*

biblioteca OpenCV este o open-source folosită în general pentru procesarea imaginilor viziunea computerizată, învățarea automată [22]. OpenCV acceptă mai multe limbaje de programare precum C++, Java, Python. Această bibliotecă se utilizează pentru identificarea obiecte, fețe sau chiar scrisul de mână a unui om. Se poate integra cu alte biblioteci cum ar fi Numpy(se utilizează în general pentru operațiuni numerice). Pe scurt, operațiunile pe care le facem cu Numpy pot fi combinate cu biblioteca OpenCV.

Yolo utilizează rețelele neuronale pentru detectarea obiectelor în timp real, fiind popular pentru viteza și precizia sa, fiind folosit în diverse aplicații pentru a detecta oamenii, animalele și semnalele din trafic [23]. Acest algoritm funcționează într-un mod diferit, se uită la toată imaginea o singură dată și trece prin rețea tot o singură dată după care detectează obiecte, de aceea a devenit destul de popular.

Prin intermediul limbajului avansat de programare Python se pot crea aplicații complexe utilizând echipamente cu preț accesibil deoarece se preiau datele din echipamentul de comandă se procesează în Python și se transmite direct rezultatul final pe baza căruia releul inteligent poate comanda actuatori sau alte echipamente de câmp. Aplicațiile se pot extinde utilizând avantajele unui PC (utilizarea camerei web, baze de date, periferice, etc).

Se trimite date către logosoft(PLC) cu ajutorul Python utilizând protocolul de comunicare *Modbus TCP/IP* doar în momentul în care camera noastră web a detectat că există o persoană

4.2 DETECTAREA UNUI OBIECT ȘI TRIMITEREA DATELOR CĂTRE RELEUL INTELIGENT ȘI EXPLICAREA CODULUI

Importarea bibliotecilor

```
import time  
import cv2  
import cvlib as cv  
from cvlib.object_detection import draw_bbox  
from pymodbus.client import ModbusTcpClient
```

Definirea variabilei *IP_Address*:

```
IP_Address = '192.168.0.12'
```

Definirea variabilei *port*:

```
port=502
```

Crearea unei instanțe a *ModbusTcpClient*, între paranteze definim variabilele de mai sus *IP_Address* și *port*.

```
client = ModbusTcpClient(IP_Address,port)
```

Apelăm funcția *connect* pentru a ne conecta la server:

```
client.connect()
```

Verificăm dacă conexiunea este realizată sau nu.

if client.connect() is False:

```
print(f"Nu s-a putut realiza conexiunea la adresa {IP_Address} ")
```

else:

```
print(f"Conectat la adresa {IP_Address}!")
```

Se creează un obiect VideoCapture(0) pentru a utiliza prima cameră disponibilă pe sistem.

```
video = cv2.VideoCapture(0)
```

Se creează o listă goală:

```
lista = []
```

Se creează o buclă infinită pentru citirea într-un mod continuu a cadrelor dintr-un flux video.

```
while True:
    ret, frame = video.read()
    bbox, label, conf = cv.detect_common_objects(frame)
    output_image = draw_bbox(frame, bbox, label, conf)
    cv2.imshow("Obiecte detectate: ", output_image)
    time.sleep(0.1)

    for item in label:
        if item not in lista:
            lista.append(item)

    if 'person' in lista:
        print(client.write_registers(2, 101))
        print("Persoana in apropiere")
        lista.remove('person')
    else:
        print(client.write_registers(2, 200))
        print("Persoana nu este in apropiere")
    if cv2.waitKey(1) & 0xFF == ord("q"):
        break
    time.sleep(0.01)
```

Figura 43 – Detectarea obiectelor comune și condiția pentru detectarea unei persoane

Se intră într-o buclă infinită care va rula până când apăsăm tasta „q”.

```
ret, frame = video.read()
```

ret ne indică dacă citirea a fost realizată cu succes, iar *frame* este cadrul citit.

```
bbox, label, conf = cv.detect_common_objects(frame)
```

Rezultatele detecției sunt stocate în *bbox*(dreptunghiuri delimitatoare ale obiectelor), *label* este pentru etichetarea obiectelor
conf este pentru conformitatea detecției.

Se utilizează funcția *detect_common_objects* pentru a detecta obiectele din cadru

```
output_image = draw_bbox(frame, bbox, label, conf)
```

Se utilizează `draw_bbox` pentru desenarea dreptunghiurilor delimitatoare în jurul obiectelor detectate pe cadru citit (*frame*). Imaginea fiind stocată în variabila `output_image`

```
cv2.imshow("Obiecte detectate: ", output_image)
```

Se afiseaza imaginea rezultată.

```
for item in label:  
    if item not in lista:  
        lista.append(item)
```

Se parcurge lista cu etichetele pe care le regăsim în *label* și se adaugă doar dacă nu există acolo.

Se verifică:

```
if 'person' in lista:  
    print(client.write_registers(2, 101))  
    print("Persoana in apropiere")  
    lista.remove('person')  
else:  
    print(client.write_registers(2, 200))  
    print("Persoana nu este in apropiere")  
[27]
```

În acest cod se verifică dacă „persoana” detectată pe cameră web se află în lista respectivă, În cazul în care persoana este în lista(dacă este detectată pe camera web) atunci la registrul 2 se pune valoarea 101) se afișează că persoana este în apropiere după care se elimină din listă *person*. Dacă persoana nu este detectată pe camera web atunci la registrul 2 se schimbă valoarea pe 200, și se afișează că persoana nu este în apropiere.

```
if cv2.waitKey(1) & 0xFF == ord("q"):  
    break
```

În cazul în care se dorește ieșirea din buclă și închiderea camerei se apasă tasta „q”.

Schemă LOGO! Soft:

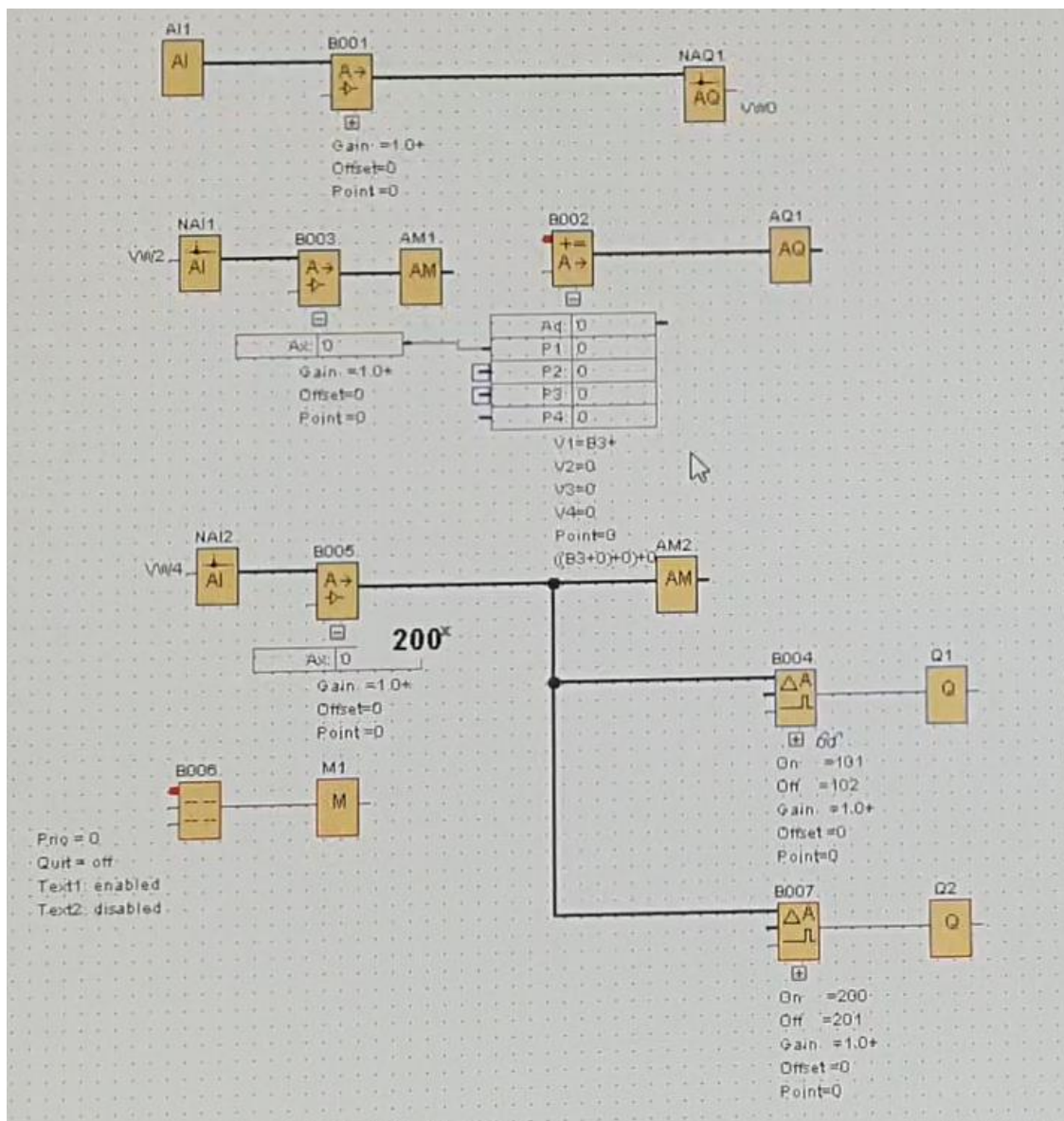


Figura 44 - Schemă LOGO! Soft

Când persoana este detectată se va trimite pe registrul 2 valoarea 101 și se va aprinde Q1, în cazul în care nu este detectată o persoană se va trimite tot pe registrul 2 valoarea 200 și se va aprinde Q2, acesta este doar un exemplu, dar cu acest limbaj de programare Python se pot si citii date din LOGO! Soft.

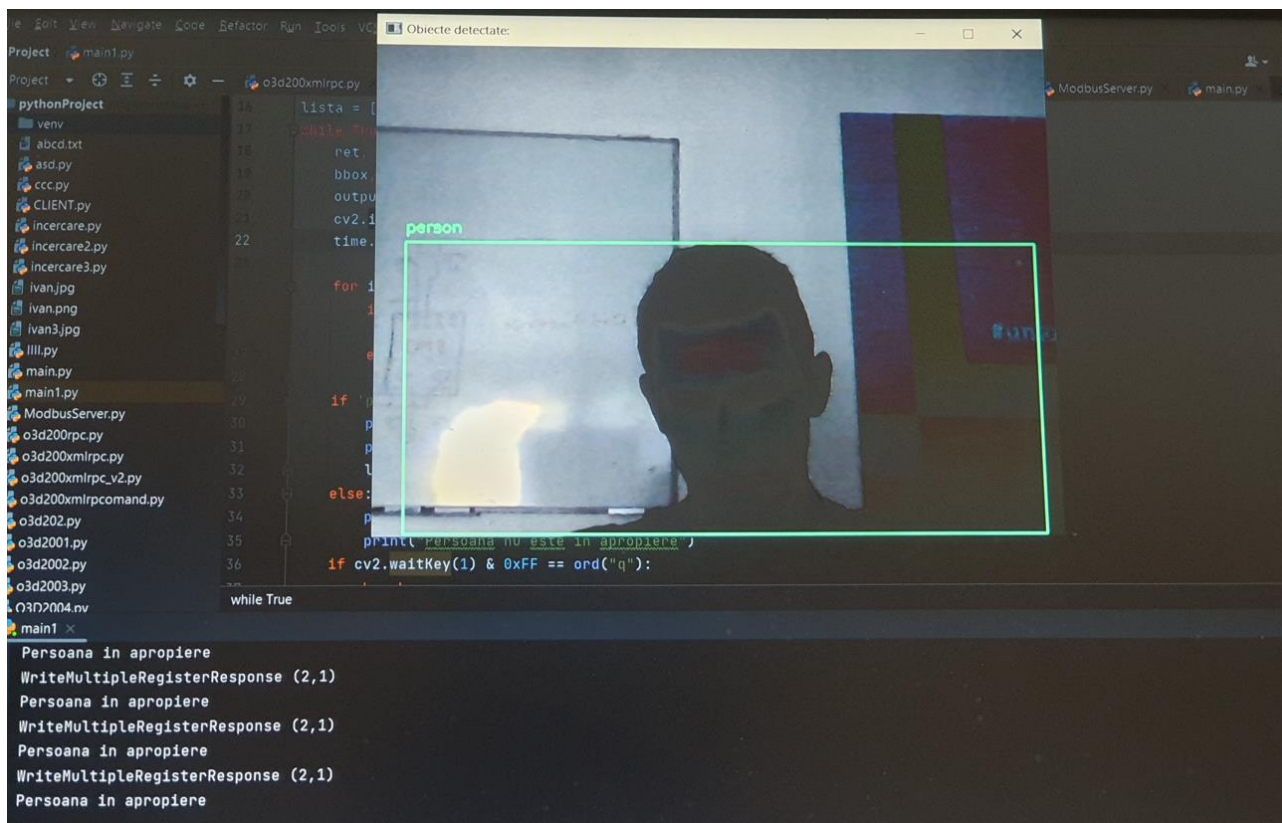


FIGURA 45 – Persoana detectata

Se aprinde ledul roșu când nu este nici o persoană detectată

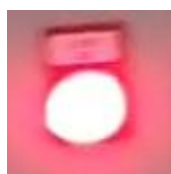


Figura 46 - Led Rosu(Q2)

Se aprinde Ledul verde când este detectata o persoană:



Figura 48 - Led Verde(Q1)

5. CONCLUZII

Scopul prezentei lucrări de licență a fost de a explora ariile limbajului de programare atât pe partea de web cât și pe partea de hardware, în ceea ce privește rețelistica. Lucrarea este împărțită în trei părți fiecare având la bază acest limbaj de programare.

În prima parte se propune să se automatizeze anumite task-uri care se efectuează zilnic în mod repetitiv și care se rezumă strict la partea web, desigur acest lucru se poate dezvolta astfel încât, pe baza rezultatelor de pe interfața web să se interacționeze cu dispozitive fizice.

În a doua parte a lucrării se propune comunicarea cu o cameră O3d200 cu scopul de a vedea dacă cu ajutorul acestui limbaj de programare există posibilitatea de a prelua date și pe baza rezultatelor să fie transmise mai departe. Acest lucru este destul de complicat deoarece camera are 2 protocoale de comunicație precum xmlRPC și este destul de dificil de realizat pentru cineva care este începător.

În a treia parte, s-au utilizat câteva dintre capabilitățile inteligenței artificiale (în cazul acesta, rețele convoluționare CNN) pentru a crea și trimite date în timp real către controlerul logic programabil SIEMENS LOGO!, cu scopul de a dezvolta aplicații diverse care necesită aceste date vizuale cu rată de actualizare rapidă. Pentru obținerea acestor aplicații, pe lângă rețelele CNN și PLC este necesară utilizarea limbajului de programare Python, care este foarte capabil în ceea ce privește prelucrarea de date la o viteză satisfăcătoare.

În urma rezultatelor dobândite s-a cristalizat faptul că se pot realiza anumite „circuite” care pot operaționaliza sarcini automate. Pe parcursul obținerii dovezilor concrete și valide s-a concretizat ideea că se poate maximiza nivelul de eficiența a unor sarcini simple de zi cu zi neacordând un efort considerabil efectuării acestora. Plecând de la această afirmație se pot transforma sarcinile obișnuite în procese automatizate, lăsând la o parte energia necesară și focalizarea pe acelea care necesită o analiză mai riguroasă. În urma demonstrării practice s-a concluzionat că se pot trimite și prelua date de la LOGO! Soft prin intermediul limbajului avansat de programare Python, date care pot fi manipulate cu acest limbaj de programare în funcție de aplicația care se dorește a fi realizată.

Prin urmare, lucrarea Implementarea low cost a unui sistem flexibil de detectare a imaginilor pentru procese industriale s-a dovedit a fi o experiență foarte vastă de învățare, atât pe partea de programare în Python, cât și partea de rețelistică și protocoale speciale de comunicație, de asemenea fiind întâmpinate multiple provocări legate de aspecte și configurație hardware a întregii aplicații practice.

6 BIBLIOGRAFIE

- [1] *** [PYTHON - WIKIWAND](#)
- [2] *** [\(PDF\) DETECȚIA ȘI RECUNOAȘTEREA DE OBIECTE & VOCE PRIN INTERMEDIUL REȚELOR CONVOLUȚIONALE ADÂNCI PE PLATFORME RASPBERRY PI 4 \(RESEARCHGATE.NET\)](#)
[RECUNOAȘTEREA FORMELOR - FRWIKI.WIKI](#)
- [3] *** [WHAT ARE CONVOLUTIONAL NEURAL NETWORKS \(CNNs\)? - YOUTUBE](#)
- [4] *** [SELENIUM WEBDRIVER WITH PYTHON TUTORIAL - JAVATPOINT](#)
- *** [WELCOME TO PYAUTOGUI'S DOCUMENTATION! — PYAUTOGUI DOCUMENTATION](#)
- [5] "O3D2XX PROGRAMMERS GUIDE"
- [5] *** [TOPUL CELOR MAI CUNOSUTE APLICATII CREATE IN PYTHON \(BURDAMEDIA.RO\)](#)
- [6] *** <https://docs.rs-online.com/9cf9/0900766b8147765c.pdf>
- [7] *** <https://www.ifm.com/mounting/704538uk.pdf> ,pg 13
- [8] *** [O3D200 - 3D SENSOR - IFM](#)
- [9] *** https://wiki.networksecuritytoolkit.org/index.php/lan_ethernet_network_cable
- [10] *** [DATASHEET \(RS-ONLINE.COM\)](#)
- [11] *** [MODELUL DE REFERINȚĂ OSI | SECURITATEA REȚELOR \(SECURITATEA-REȚELOR.RO\)](#)
- [12] *** [CCNA1-ITNV7 - MODULE 04 - PHYSICAL LAYER - YOUTUBE](#)
- [13] *** [HTTPS://WWW.AUTOMATIZARI-SCADA.RO/FAQ/CE-ESTE-MODBUS/](https://www.automatizari-scada.ro/faq/ce-este-modbus/)
- [14] *** [HTTPS://WWW.AUTOMATIZARI-SCADA.RO/CONVERTOARE-PROTOCOL-MODBUS/CE-ESTE-MODBUS/](https://www.automatizari-scada.ro/convertoare-protocol-modbus/ce-este-modbus/)
- [15] *** [HTTPS://THEAUTOMIZATION.COM/MODBUS-ASCII-VS-MODBUS-RTU-VS-MODBUS-TCPIP/](https://theautomation.com/modbus-ascii-vs-modbus-rtu-vs-modbus-tcpip/)
- [16] *** [NETWORK LAYER SERVICES- PACKETIZING, ROUTING AND FORWARDING - GEEKSFORGEEKS](#)
- [17] *** [TRANSPORT LAYER RESPONSIBILITIES - GEEKSFORGEEKS](#)
- [18] *** [SESSION LAYER IN OSI MODEL - GEEKSFORGEEKS](#)
- [19] *** [PRESENTATION LAYER IN OSI MODEL - GEEKSFORGEEKS](#)
- [20] *** [APPLICATION LAYER IN OSI MODEL - GEEKSFORGEEKS](#)
- [21] *** [CVLIB : YOLO OBJECT DETECTION IN SECONDS! - ANALYTICS VIDHYA](#)
- [22] *** [ABOUT - OPENCV](#)
- [23] *** [HTTPS://WWW.V7LABS.COM/BLOG/YOLO-OBJECT-DETECTION](https://www.v7labs.com/blog/yolo-object-detection)
- [24] *** [OBJECT DETECTION USING OPENCV PYTHON IN 15 MINUTES! CODING TUTORIAL FOR BEGINNERS COURSE! - YOUTUBE](#)
- [25] *** [IEEE-754 FLOATING POINT CONVERTER \(H-SCHMIDT.NET\)](#)
- [26] *** [ONLINE HEX CONVERTER - BYTES, INTS, FLOATS, SIGNIFICANCE, ENDIANS - SCADACORE](#)
- [27] *** [001 | READ & WRITE HOLDING REGISTER OF MODBUS TCP DEVICE USING PYTHON | PYMODBUS | - YOUTUBE](#)
- [28] *** [HTTPS://PJREDDIE.COM/DARKNET/YOLO/](https://pjreddie.com/darknet/yolo/)

7. ANEXE

7.1 COD SURSA PENTRU AUTOMATIZAREA UNUI JOC PE BAZA IMAGINILOR

```
import random
import time
import tkinter
import tkinter as tk
from threading import Thread
from time import sleep
from tkinter import *
from tkinter import ttk
import pyautogui

root = Tk()

class Clicker:
    def __init__(self):
        self.alive = None

    def alive_set(self):
        self.alive = True
        print(20 * '-', 'Start farm', '-' * 20)

    def alive_clean(self):
        self.alive = False
        print(20 * '-', 'Stop farm', '-' * 20)

    def run(self):
        i = 0
        while True:
            if self.alive:
                for i in range(600):
                    if self.alive:
                        my_progress.step(0.1666666666666667)
                        i += 1
                        sleep(1)

                elif self.alive is False:
                    my_progress.stop()
                    my_progress.step(0)
                    break

            if i == 600 and self.alive is True:
                CEVA = random.uniform(0, 60)
                print("FARM: ", CEVA)
                sleep(CEVA)
                while True:
                    try:
                        time.sleep(0.5)
                        k, l = pyautogui.locateCenterOnScreen('farmingv.png')
```

```

        sleep(0.5)
        x, y = pyautogui.locateCenterOnScreen('select.png')
        pyautogui.leftClick(x + random.uniform(43, 48), y + random.uniform(-3, +3),
                            interval=random.uniform(0.2, 0.9),
                            duration=random.uniform(0.2, 0.9))

        sleep(0.5)
        t, r = pyautogui.locateCenterOnScreen('collect.png')
        sleep(0.5)
        pyautogui.leftClick(t + random.uniform(-30, +30), r + random.uniform(-5, +5),
                            interval=random.uniform(0.2, 0.9),
                            duration=random.uniform(0.2, 0.9))
        my_progress.step(0)
        break
    except TypeError:
        if self.alive is False:
            break
        print("Inca caut")
        cent = pyautogui.locateCenterOnScreen("cent.png")
        time.sleep(0.5)
        pyautogui.moveTo(cent)
        time.sleep(0.5)
        farm = pyautogui.locateCenterOnScreen("farm.png")
        time.sleep(0.5)
        pyautogui.leftClick(farm)
        sleep(2)

clicker = Clicker()
t1 = Thread(target=clicker.run)
t1.start()
s = ttk.Style()
s.theme_use('clam')
s.configure("3.Horizontal.TProgressbar", troughcolor='black', background='red')
my_progress = ttk.Progressbar(root, style='3.Horizontal.TProgressbar',
                              orient=HORIZONTAL, length=100, mode='determinate')
my_progress.pack(padx=10, pady=5)

canvas = tk.Canvas(root, height=400, width=400, bg="#000000")
canvas.pack()

b1 = tkinter.Button(root, text="Start farm", command=clicker.alive_set)
b1.pack()
b2 = tk.Button(root, text="Stop farm", command=clicker.alive_clean)
b2.pack()
root.mainloop()

```


7.2 COD SURSA PENTRU EXTRAGEREA DATELOR

```
import socket
import struct
import time
import xmlrpc.client
import matplotlib.pyplot as plt
import numpy as np
import select
from matplotlib import cm

server_host = "192.168.0.69"
server_port = 8080
server_port_socket=50002
proxy = xmlrpc.client.ServerProxy(f"http://{server_host}:{server_port}")
time.sleep(1)
methods = proxy.system.listMethods()
print("\nMethods:\n", methods, "\n")
Conectare = proxy.MDAXMLConnectCP(server_host, server_port)
time.sleep(1)
if Conectare == [0, '4041', 'O3D200AD']:
    print('M-AM CONECTAT LA SERVER')
elif Conectare == [-120]:
    print("Sunt deja conectat la server")
#activez server-ul de imagine apeland functia MDAXMLSetWorkingMode si punem
argumentul pe 1 pentru activare
workingmode=proxy.MDAXMLSetWorkingMode(1)
print("WORKING MODE ACTIVATED!!!!!!",workingmode)
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((server_host, server_port_socket))#conectare la server-ul TCP/IP

TOTAL_DATA=39528

command='xyz'

client_socket.send(command.encode()) #comanda trimisa catre server.
time.sleep(0.1)
DATE_PRIMITE=b"
try:
    print("Am intrat in blocul try")
```



```
while len(DATE_PRIMITIVE)<TOTAL_DATA:
    print("Am intrat in bucla while")
    data=client_socket.recv(1460)
    print('Lungimea datelor:',len(data))
    print('DATA IN HEX : ',data.hex())
    if not data:
        break
    DATE_PRIMITIVE = DATE_PRIMITIVE + data
except IOError as e:
    print('Complicata situatie')
    pass
DATE_PRIMITIVE_HEX=DATE_PRIMITIVE.hex()
print('TOTAL DATA BYTES: ',DATE_PRIMITIVE)
print('DATE_PRIMITIVE_HEX',DATE_PRIMITIVE_HEX)

HEADER_BYTES = DATE_PRIMITIVE[:376]#
print("PRIMUL HEADER: ",len(HEADER_BYTES))

DATA_BYTES = DATE_PRIMITIVE[376:13176]
print("DATA BYTES 0: ",len(DATA_BYTES))

HEADER_BYTES1=DATE_PRIMITIVE[13176:13552]
print("AL DOILEA HEADER: ",len(HEADER_BYTES1))

DATA_BYTES1=DATE_PRIMITIVE[13552:26352]
print("DATA BYTES 1: ",len(DATA_BYTES1))

HEADER_BYTES2=DATE_PRIMITIVE[26352:26728]
print("AL TREILEA HEADER: ",len(HEADER_BYTES2))

DATA_BYTES2=DATE_PRIMITIVE[26728:]
print("DATA BYTES 2 : ",len(DATA_BYTES2))

#PRIMUL HEADER FLOAT SI DATA
HEADER_FLOAT=[]
#CONVERSIE IN FLOAT PENTRU HEADER
for i in range(0, len(HEADER_BYTES), 4):
    grup = HEADER_BYTES[i:i+4]
    valoare = struct.unpack('!f', grup)[0]
    HEADER_FLOAT.append(valoare)
print("HEADER FLOAT 0: ", HEADER_FLOAT)
```

```
DATA_FLOAT=[]
#CONVERSIE IN FLOAT PENTRU DATA
for i in range(0, len(DATA_BYTES), 4):
    grup = DATA_BYTES[i:i+4]
    valoare = struct.unpack('!f', grup)[0]
    DATA_FLOAT.append(valoare)
print("DATA FLOAT 0: ", DATA_FLOAT)

#AL DOILEA HEADER FLOAT SI DATA

HEADER_FLOAT1=[]
#CONVERSIE IN FLOAT PENTRU HEADER
for i in range(0, len(HEADER_BYTES1), 4):
    grup = HEADER_BYTES1[i:i+4]
    valoare = struct.unpack('!f', grup)[0]
    HEADER_FLOAT1.append(valoare)
print("HEADER FLOAT 1: ", HEADER_FLOAT1)

DATA_FLOAT1=[]
#CONVERSIE IN FLOAT PENTRU DATA
#DATA_LISTA[1]
for i in range(0, len(DATA_BYTES1), 4):
    grup = DATA_BYTES1[i:i+4]
    valoare = struct.unpack('!f', grup)[0]
    DATA_FLOAT1.append(valoare)
print("DATA FLOAT 1: ", DATA_FLOAT1)

#AL TREILEA HEADER FLOAT SI DATA
HEADER_FLOAT2=[]
for i in range(0, len(HEADER_BYTES2), 4):
    grup = HEADER_BYTES2[i:i+4]
    valoare = struct.unpack('!f', grup)[0]
    HEADER_FLOAT2.append(valoare)
print("HEADER FLOAT 2: ", HEADER_FLOAT2)

DATA_FLOAT2=[]
#CONVERSIE IN FLOAT PENTRU DATA
#DATA_LISTA[2]
for i in range(0, len(DATA_BYTES2), 4):
    grup = DATA_BYTES2[i:i+4]
    valoare = struct.unpack('!f', grup)[0]
```

```
DATA_FLOAT2.append(valoare)
print("DATA FLOAT 2: ", DATA_FLOAT2)
```

```
#MATRICE CU NUMPY!
```

```
# #PRIMA MATRICE X
```

```
print(('Lungimea datelor pentru prima matrice :', len(DATA_FLOAT)))
MATRICE1X=np.array(DATA_FLOAT)
MATRICE2X=MATRICE1X.reshape(50,64)
MATRICE3X=MATRICE2X.transpose()
print("A TREIA MATRICE X: ", MATRICE3X)
```

```
# #A DOUA MATRICE Y
```

```
print(('Lungimea datelor pentru a doua matrice :', len(DATA_FLOAT1)))
MATRICE1Y=np.array(DATA_FLOAT1)#
MATRICE2Y=MATRICE1Y.reshape(50,64)
MATRICE3Y=MATRICE2Y.transpose()
print("A DOUA MATRICE Y:", MATRICE3Y)
```

```
#
```

```
# #A TREIA MATRICE Z
```

```
print(('Lungimea datelor pentru a treia matrice :', len(DATA_FLOAT2)))
MATRICE1Z=np.array(DATA_FLOAT2)
MATRICE2Z=MATRICE1Z.reshape(50,64)
MATRICE3Z=MATRICE2Z.transpose()
print("A TREIA MATRICE Z: ", MATRICE3Z)
```

```
#PENTRU 3D
```

```
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
surf = ax.plot_surface(MATRICE3X, MATRICE3Y, MATRICE3Z, cmap=cm.coolwarm,
                        linewidth=0, antialiased=False)
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
plt.colorbar(surf)
plt.show(block=True)
```

```
# PENTRU 2D
```

```
plt.imshow(MATRICE3X, cmap='viridis')
plt.colorbar()
```

```
plt.show(block=True)
plt.imshow(MATRICE3Y, cmap='viridis')
plt.colorbar()
plt.show(block=True)
plt.imshow(MATRICE3Z, cmap='viridis')
plt.colorbar()
plt.show(block=True)
```

7.3 CODUL SURSĂ PENTRU DETECTAREA OBIECTELOR ȘI TRIMITEREA DATELOR CĂTRE PLC

```
import time
import cv2
import cvlib as cv
from cvlib.object_detection import draw_bbox
from pymodbus.client import ModbusTcpClient

IP_Address = '192.168.0.12'
port=502
client = ModbusTcpClient(IP_Address,port)
client.connect()
if client.connect() is False:
    print(f"Nu s-a putut realiza conexiunea la adresa {IP_Address} ")
else:
    print(f"Conectat la adresa {IP_Address}!")

video = cv2.VideoCapture(0)
lista = []
while True:
    ret, frame = video.read()
    bbox, label, conf = cv.detect_common_objects(frame)
    output_image = draw_bbox(frame, bbox, label, conf)
    cv2.imshow("Obiecte detectate: ", output_image)
    time.sleep(0.1)

    for item in label:
        if item not in lista:
            lista.append(item)

    if 'person' in lista:
        print(client.write_registers(2, 101))
        print("Persoana in apropiere")
        lista.remove('person')
    else:
        print(client.write_registers(2, 200))
        print("Persoana nu este in apropiere")
    if cv2.waitKey(1) & 0xFF == ord("q"):
        break
    time.sleep(0.01)
```