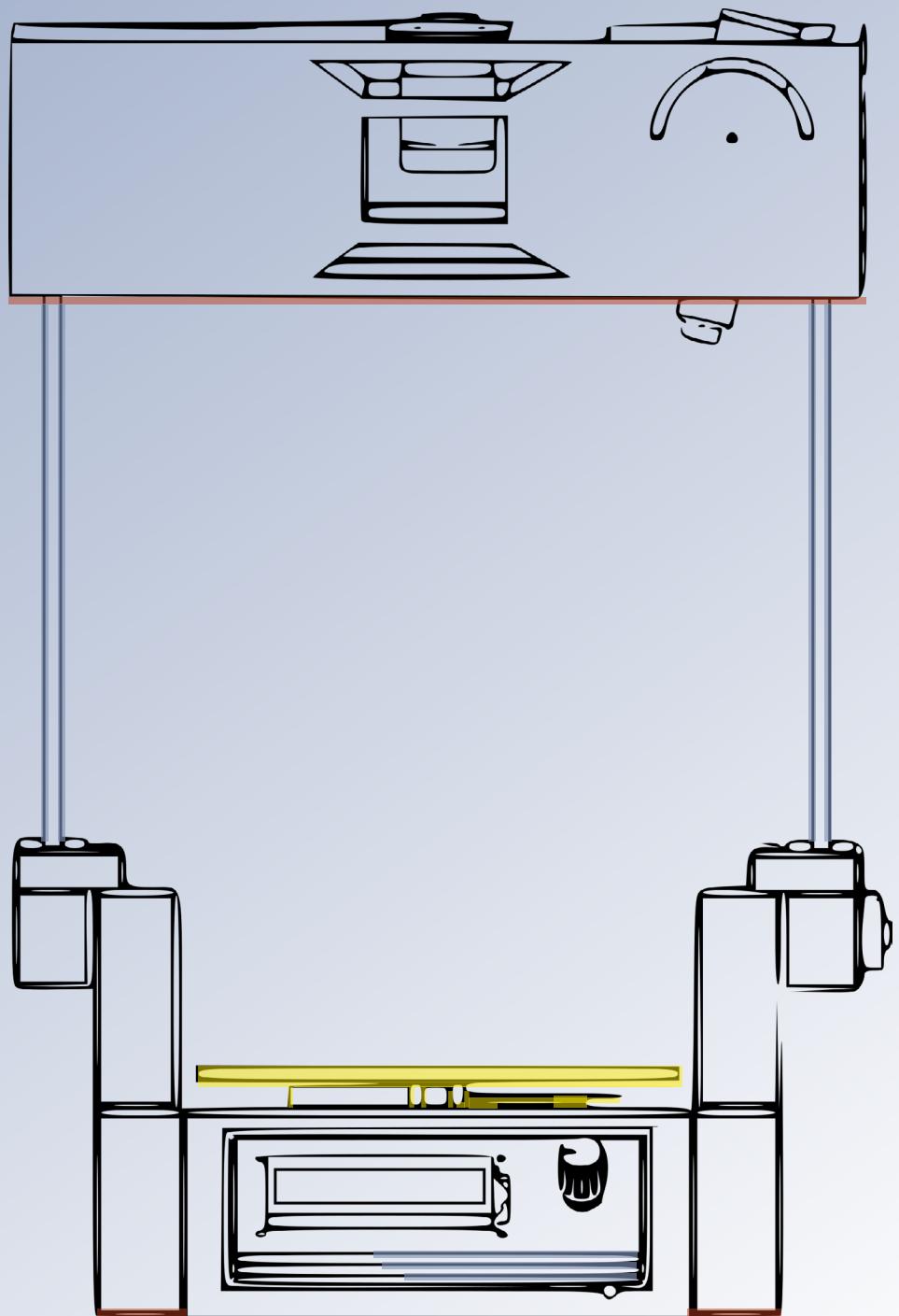


# REMAKE

■ SCANNER DE OBIECTE 3D

■ OANA DATCU & ALEXANDRU OPREA



# REMAKE

ReMake este un scanner 3D de obiecte care va ușura viața oricărui proiectant sau pur și simplu unui om cu un obiect care s-a rupt. Probabil vă gândiți cum?



Doar amplasați obiectul pe patul de scanare, așteptați câteva minute și se va transpune direct pe calculatorul dumneavoastră în format STL fiind gata pentru a îl printa sau modifica într-un soft CAD.

Mai mult decât atât ReMake se diferențiază de orice alt scanner pe care îl puteți găsi pe piață, deoarece dispune de un braț mobil care va putea scana obiectul din fiecare unghi și de un senzor de greutate care vă va ajuta să aflați posibile materiale din care este făcut obiectul.



## CUPRINS

### ReMake

Concept  
Descriere robot  
Utilitate practică

### Piese folosite

Piese alese pentru creare robotului

### CAD

Proiectarea 3D a robotului

### Inginerie și electronică

Cum funcționează  
Design  
Folosirea pieselor alese  
Alimentare

### Programare

Clase  
Organograma apelare functii  
Prezentare generală

### Îmbunătățiri vitoare

Modurile cum vom îmbunătăți robotul

Alexandru Oprea  
clasa a 12-a



Oana Datcu  
clasa a 11-a





Am ales să folosim **PETG** deoarece conferă mai multă rezistență robotului.



**Convertor DC-DC Step-Down Ajustabil** - are eficiență mare, disipa multă putere



**Ecran LCD 1602 (16x2) Verde I2C** - afișează instrucțiunile de utilizare ale robotului



**Motor Pas Cu Pas Stepper NEMA 17** - conferă puterea necesară

### Piese folosite

Acestea sunt piesele pe care le-am ales pentru crearea robotului.



**Comutator Cu Senzor De Presiune RFP602** - masurătoare precisă a greutății



**Dioda Laser Fascicul Rosu Tip Linie, 650nm, 5mW** -



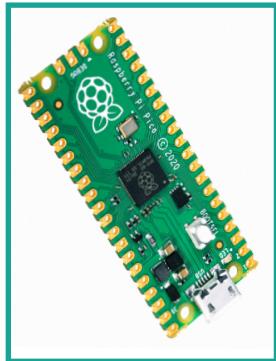
**Dioda Laser Fascicul Rosu Tip Linie, 650nm, 5mW** - informații despre mediul exterior



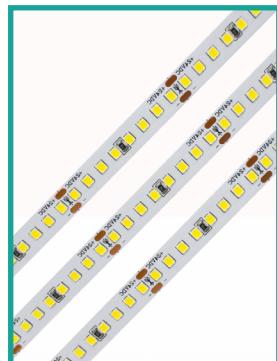
**Raspberry Pi HQ Camera & 6mm camera lens** - realizarea seriei de fotografi



**Raspberry Pi 4**  
computer with 4GB  
**RAM** - controlarea  
camerei și  
procesarea datelor



**Raspberry PI Pico** -  
controlare hardware  
robot



**Bandă led** - ajustare  
lumină pentru a se  
putea face scanarea





# Cum funcționează?

## Cum funcționează?

ReMake este un scanner 3D extrem de ușor de folosit. Doar amplasați obiectul pe patul de printare, urmați instrucțiunile afișate pe ecran, iar ReMake va începe procesul.

Procesul constă în realizarea unei serii de poze la diferite unghiuri în jurul obiectului. Astfel de fiecare data se realizează o poza cu laser pentru informații referitoare la adâncimea obiectului și o poza fără laser pentru informații cu privire la culoarea obiectului în punctele respective. Unghiul pozelor este modificat prin rotirea patului pe care este asezat obiectul, iar în fața acestuia se află camera împreună cu o banda leduri pentru a regla contrastul.

Prin folosirea unor algoritmi de tip computer vision se detectează poziția laserului în imagine, iar apoi prin triangulare și prin legile lentilelor se obține adâncimea obiectului la un anumit unghi. Punctele sunt transformate din sistemul sferic în sistemul cartezian și sunt introduse într-o structură tridimensională. În continuare prin diferiți algoritmi de reconstrucție se determină suprafața obiectului, se netezeste și se salvează în format STL. Transferul de date are loc prin WiFi folosind conexiuni SSH.

## Design

Pentru a asigura o construcție rapidă și sigură a robotului, am proiectat inițial ideile în Autodesk Inventor. După mai multe prototipuri am ajuns la o versiune finală cu un design eficient și optim. Etapa următoare a constat în printarea pieselor cu ajutorul unei imprimante 3D. Timpul total de printare a fost de aproximativ 30h, iar materialul folosit a fost în totalitate PETG.

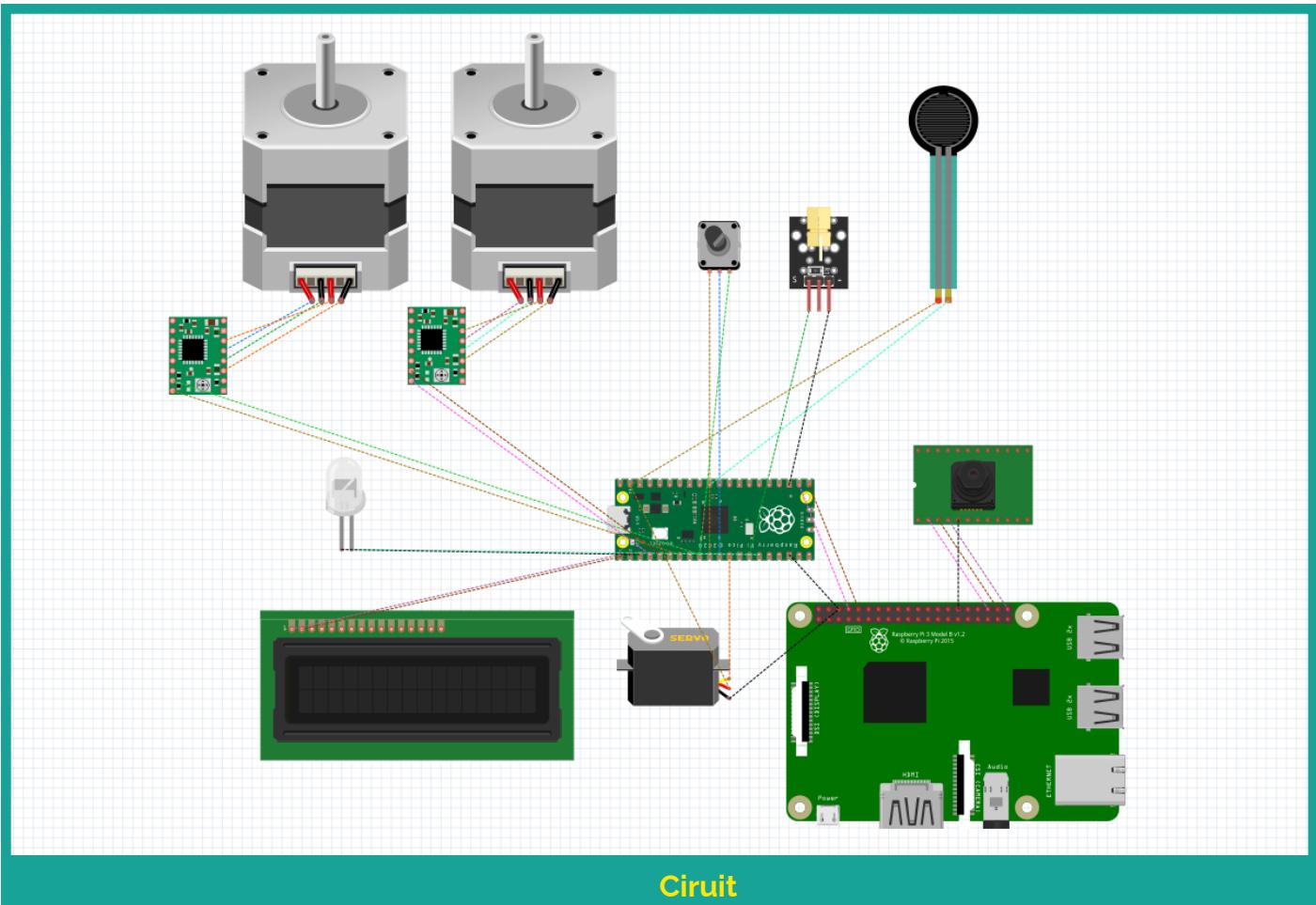
Varianta finală a robotului este una eficientă, ușor de replicat și care poate fi modificată cu ușurință.

## Piese folosite

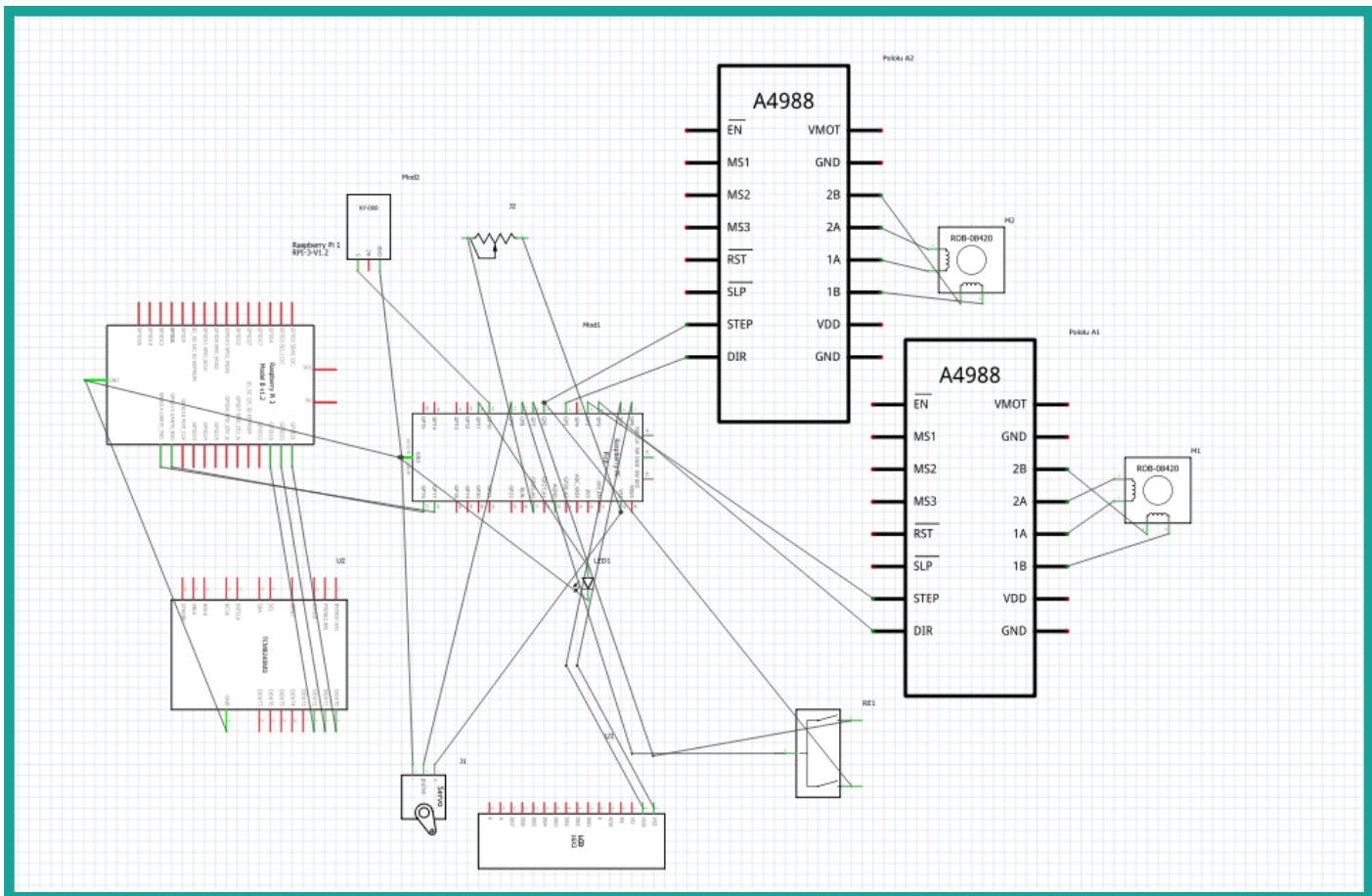
ReMake are 2 motoare care mișcă patul și brațul care ține camera, sistemul de laser (laser+servo motor care conferă rotație laserului), luminile și microcomputer Raspberry Pi 4. Robotul are un senzor de presiune pe patul de scanare pentru a putea afla greutatea obiectului, o funcție care va ajuta la aflarea materialului din care e făcut obiectul. Partea hardware este controlată de un microcontroller Raspberry Pi Pico.

## Alimentare

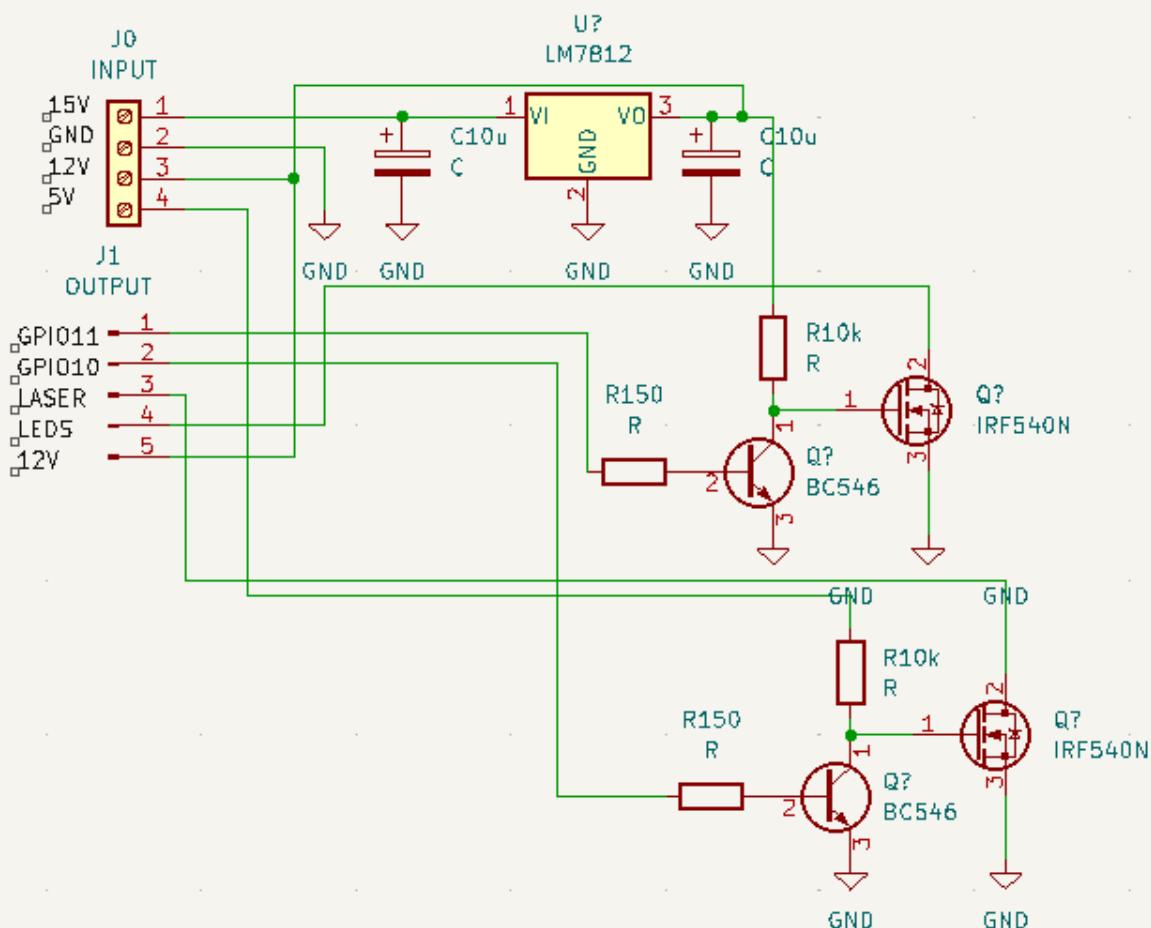
Pentru a reduce dimensiunile robotului și pentru a îi crește portabilitatea acesta are o sursă principală de alimentare externă de 15V și 70W. Elementele logice lucrează la 5V asigurat de un modul STEP-DOWN intern. În plus robotul dispune și de un regulator liniar de 12V pentru controlul benzi de LED.



Circuit



Circuit



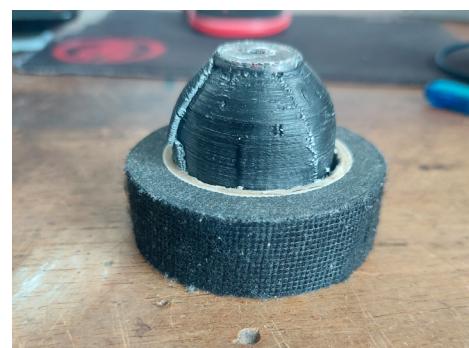
Schemă circuit driver laser și leduri

### TABEL PINI Raspberry PI Pico

#### PIN FUNCTIE TIP

- GPIO0 LCD SDA
- GPIO1 LCD SCL
- GPIO26 Senzor greutate ADC
- GPIO9 Servo PWM
- GPIO7 Encoder DT
- GPIO6 Encoder CLK
- GPIO8 Encoder SW
- GPIO10 Laser Digital
- GPIO11 LEDs Digital
- GPIO3 Motor 1 STEP
- GPIO2 Motor 1 DIR
- GPIO5 Motor 2 STEP
- GPIO4 Motor 2 DIR
- GPIO16 Serial UART\_TX
- GPIO17 Serial UART\_RX

### Primul obiect scanat



# Clase

(PC - Windows 10 - Python 3.9)

```
class pointCloud:
```

Module externe necesare:  
math  
numpy  
paramiko  
path  
json  
datetime  
cv2  
open3d  
sys  
trimesh

Clasa care se ocupa cu preluarea datelor de la scaner si prelucrarea acestora pentru obtinerea obiectului 3D. Daca este apelata separat (din terminal) aceasta va genera un obiect pe baza datelor locale (daca exista) sau va descarca automat datele de pe Raspberry PI prin SSH si va genera un obiect 3D pe baza acestora, ulterior fiind salvat ca "demo.stl".

```
def __init__(self):
```

Constructorul clasei in care se initializeaza parametri principali ai obiectului si se seteaza specificatiile scanerului si a camerei.

```
def appendLine(self, line):
```

line (vector de liste) = o linie de date cu coordonate de puncte  
Adauga o linie de date in lista de puncte a obiectului.

```
def debug(self):
```

Afiseaza parametri principali ai obiectului si deseneaza punctele intr-un format 3D.

```
def download_file(self, name, host = "192.168.1.14", port = 22, password = "raspberry", username = "pi", path = '/home/pi/TestCamera/')
```

name (string) = numele fisierului de descarcat  
host (string) = adresa ip a Raspberry PI  
port (int) = portul pentru conexiunea SSH  
password (string) = parola pentru conexiunea SSH  
username (string) = numele utilizatorului pentru conexiunea SSH  
path (string) = locatia de pe Raspberry PI de unde se va descarca fisierului  
Se conecteaza la Raspberry PI prin SSH pentru a descarca fisiere.

```
def load_data(self, name):
```

name (string) = numele fisierului cu datele punctelor

Incarca in memorie datele din fisierul JSON local sau, daca nu exista, mai inainte il va descarca.

## Programare

```
def show_laser(self, index):
    index (int) = index-ul pozei facuta de scanner
    Returneaza: img (matrice de liste) = poza generata
    Returneaza o imagine care contine pozitia laserului si culoarea obiectului in punctul respectiv. Unde nu se afla laser, imaginea va fi neagra.
```

```
def compute_ppm(self):
    Calculeaza valoarea ppm (pixeli pe metru) pentru a converti dimensiunile din pixeli in metri pe baza specificatiilor camerei si a scannerului.
```

```
def set_ppm(self, new_ppm):
    new_ppm (float) = noua valoare pentru ppm
    Seteaza manual valoarea ppm (pixeli pe metru) pentru a converti dimensiunile din pixeli in metri.
```

```
def create_pointcloud(self, viewPC=False):
    viewPC (bool) = activeaza previzualizarea punctelor
    Creeaza in memorie un set de puncte intr-un format 3D si calculeaza caracteristicile acestuia.
```

```
def poisson_reconstruction(self, depth):
    depth (int) = adancimea algoritmului de reconstructie
    Returneaza: mesh (suprafata) = suprafata calculata
    Reconstruieste suprafata obiectului folosind algoritmul Poisson.
```

```
def alpha_reconstruction(self, alpha):
    alpha (int) = parametrul algoritmului de reconstructie
    Returneaza: mesh (suprafata) = suprafata calculata
    Reconstruieste suprafata obiectului folosind algoritmul Alpha Shapes.
```

```
def ball_reconstruction(self, mult):
    mult (int) = marimea bilei de reconstructie
    Returneaza: mesh (suprafata) = suprafata calculata
    Reconstruieste suprafata obiectului folosind algoritmul bilei pivotante.
```

```
def save_mesh(self, mesh, name):
    mesh (suprafata) = suprafata care trebuie salvata
    name (string) = numele fisierului generat
    Salveaza suprafata in formatul STL cu numele specificat.
```

```
def fromImage(self, im):
    im (matrice de liste) = imaginea din care se extrag datele
    Se extrag punctele pornind de la o imagine binara cu pozitia laserului.
```

```
def __linePoint(self, line, seed=-1):
    line (vector de int) = linia din imagine in care se cauta laserul
    seed (int) = pozitia in care s-a gasit laserul in linia anterioara
    Returneaza: (int) = pozitia in care s-a gasit laserul sau -1 daca nu exista
    Cauta pozitia laserului intr-o linie din imagine folosind un algoritm eficient bazat pe Divide et Impera si pe pozitia anterioara gasita.
```

## Programare

(Raspberry PI - Debian - Python 3.10)

```
class imageproc:
```

    Module externe dependente:

        color(turtle)

        cv2

        numpy

    Clasa care se ocupa cu prelucrarea imaginilor de la camera si detectarea laserului si pozitiei acestuia in ele.

```
|def __init__(self, img=None):
```

        img (matrice de liste) = imaginea referinta pentru obiect, daca nu este specificata va fi folosita prima imagine.

    Constructorul clasei in care se initializeaza parametri de detectie a laserului si zona de interes din imagine.

```
|def undist(self, path) -> None:
```

        path (string) = locatia fisierului cu parametrii de distorsiune a camerei

        Aplica imaginii pentru adancime transformarea corespunzatoare parametriilor de distorsiune a camerei pentru a obtine o imagine nedistorsionata si taie din imagine zonele care nu sunt de interes.

```
|def undistC(self, path) -> None:
```

        path (string) = locatia fisierului cu parametrii de distorsiune a camerei

        Aplica imaginii pentru culoare transformarea corespunzatoare parametriilor de distorsiune a camerei pentru a obtine o imagine nedistorsionata si taie din imagine zonele care nu sunt de interes.

```
|def changePerspective(self) -> None:
```

        Schimba perspectiva camerei pentru a alinia planul focal cu planul obiectului.

```
|def localMaxTolmg(self, mask, dh):
```

        mask (matrice de liste) = masca unde se va cauta laserul

        dh (int) = diferența maxima de nuanta admisa

        Returneaza: img (matrice de liste) = poza binara cu pozitiile laserului

        Cauta pozitia laserului in imagine pe baza unui algoritm elementar si a intensitatii culorii si returneaza o imagine binara cu pozitiile gasite.

```
|def localMax(self, mask, dh):
```

        mask (matrice de liste) = masca unde se va cauta laserul

        dh (int) = diferența maxima de nuanta admisa

        Returneaza: points (vector de int) = pozitiile laserului in imagine

        Cauta pozitia laserului in imagine pe baza unui algoritm elementar si a intensitatii culorii si returneaza o lista cu pozitiile gasite.

```
|def localMaxDIVIDEIMPERA(self, mask):
```

        mask (matrice de liste) = masca unde se va cauta laserul

        Returneaza: points (vector de int) = pozitiile laserului in imagine

        Cauta pozitia laserului in imagine pe baza unui algoritm Divide et Impera si a intensitatii

## Programare

cilorii si returneaza o lista cu pozitiile gasite. Comportamentul functiei este similar cu cel al unei functii Greedy cu privire la rezultate.

**def localMaxQUICK(self, mask):**

mask (matrice de liste) = masca unde se va cauta laserul

Returneaza: points (vector de int) = pozitiile laserului in imagine

Cauta pozitia laserului in imagine pe baza unui algoritm de sortare QuickSort si a intensitatii colorii si returneaza o lista cu pozitiile gasite. Este cel mai eficient algoritm cu rezultate optime.

**def combineFilter(self, dilate = 2):**

dilate (int) = gradul de suprapunere a filtrelelor

Returneaza: mask (matrice de bool) = masca filtru generata

Combina cele doua filtre generate pentru o acuratete mai ridicata folosind blur de tip Gaussian. Astfel se anuleaza zgomotul fiecarui filtru.

**def dilate(self,img,size):**

img (matrice de liste) = imaginea la care se aplica transformarea de tip dilatare

size (int) = gradul de dilatare

Returneaza: (matrice de liste) = imaginea rezultata in urma transformarii

Aplica imaginii transformarea de tip dilatare.

**def colorFilter(self, ch=0):**

ch (int) = canalul colorii de referinta

Returneaza: (matrice de bool) = masca filtru generata

Genereaza un filtru pe baza colorilor din imagine.

**def hsvFilter(self):**

Returneaza: mask (matrice de bool) = masca filtru generata

Genereaza un filtru pe baza saturatiei si a componentelor HSV din imagine.

**def threshold\_image(self, channel) -> None:**

channel (string): canalul HSV asupra caruia se aplica operatia

Aplica canalului corespunzator operatia de tip limitare de prag.

**def getImg(self):**

Returneaza: img (matrice de liste) = imaginea de adancime

**def getRes(self):**

Returneaza: (lista) = rezolutia imaginii de adancime.

**def setImg(self, img) -> None:**

img (matrice de liste) = imaginea de adancime

Seteaza imaginea de adancime.

**def setColor(self, img) -> None:**

img (matrice de liste) = imaginea de culoare

Seteaza imaginea de culoare.

**def getMask(self, mask):**

Returneaza: (matrice de bool) = imaginea binara corespunzatoare filtrului folosit.

## Programare

(Raspberry PI - Debian - Python 3.10)

class writer:

Module externe dependente:

json

datetime

Clasa care se ocupa cu memorarea datelor generate de scanner. Datele sunt salvate intr-un fisier JSON sub forma unor liste cu pozitiile laserului si culoarea obiectului in acel punct. La inceputul fisierului se regasesc data scanarii, rezolutia folosita si greutatea obiectului scanat.

|def \_\_init\_\_(self, fname) -> None:

    fname (string) = numele fisierului in care se vor salva datele

    Constructorul clasei in care se initializeaza parametri esentiali.

|def setHeader(self, res, weight=0) -> None:

    res (lista) = rezolutia folosita la scanare

    weight (float) = greutatea obiectului scanat

    Seteaza datele din capul fisierului.

|def addData(self, line, angle) -> None:

    line (vector de int) = pozitiile laserului determinate

    angle (float) = unghiul patului relativ la pozitia initiala

    Adauga datele determinate in memoria obiectului.

|def save(self) -> None:

    Creeaza structura fisierului si sterilizeaza si salveaza datele in fisier.

(Raspberry PI Pico - MicroPython)

class Nema:

Module externe dependente:

Pin(machine)

sleep(time)

Clasa care se ocupa cu controlul motoarelor pas cu pas de tip NEMA17 alimentate de drivere A4988 sau echivalente. Scopul final este de a traduce comenzi de tipul servo, si anume pozitii unghiulare, in pulsuri corespunzatoare pasilor motorului.

|def \_\_init\_\_(self, st, dr, ust = 16):

    st (int) = pinul STEP al driverului

    dr (int) = pinul DIR al driverului

    ust (int) = gradul de micropasi folosit

    Constructorul clasei prin care se precizeaza pinii corespunzatori motorului si se initializeaza parametrii esentiali.

|def hold(self):

    Opreste motorul si pastreaza pozitia acestuia.

|def go(self, angle, path = 0):

    angle (float) = unghiul la care motorul trebuie sa ajunga

    path (int) = sensul de rotatie al motorului

## Programare

Calculeaza numarul de pasi necesari pentru a ajunge la unghiul transmis si comanda motorul conform acestora.

```
def step(self):
```

Comanda motorul pentru a executa un pas.

```
def getSteps(self):
```

Returneaza: (int) = numarul de pasi executati relativ la pozitia initaiala.

```
def getAng(self):
```

Returneaza: (float) = unghiul curent relativ la pozitia initaiala.

(Raspberry PI - Debian - Python 3.10)

```
class pico:
```

Module externe dependente:

time

serial

netifaces

Clasa care se ocupa cu comunicarea prin serial (UART) cu microcontollerul Raspberry PI Pico. Contine metodele principale pentru a controla elementele hardware ale scannerului impreuna cu verificarea handshake a comunicarii si retrimiterea mesajelor in cazul pierderii bitilor in timpul comunicarii.

```
def __init__(self, dr=0):
```

dr (int) = directia de rotire normala a patului de scanat

Constructorul clasei in care se initializeaza comunicarea serial pe portul /dev/ttyS0 la o taxa de 115200.

```
def sendM(self, mes):
```

mes (string) = mesajul care trebuie transmis

Incearca sa trimita prin serial un mesaj. Daca confirmarea receptionarii nu este primita, se apeleaza recursiv functia cu acelasi mesaj. In cazul erorilor de decodare sau a altor tipuri, acestea nu intrerup programul fiind prinse si afisate in consola.

```
def sendM_wc(self, mes):
```

mes (string) = mesajul care trebuie transmis

Incearca sa trimita prin serial un mesaj fara a verifica confirmarea receptionarii acestuia. In cazul erorilor de decodare sau a altor tipuri, acestea nu intrerup programul fiind prinse si afisate in consola.

```
def readM(self):
```

Returneaza: (string) = mesajul receptionat

Incearca sa citeasca mesajul primit si sa il decodeze. In cazul erorilor de decodare sau a altor tipuri, acestea nu intrerup programul fiind prinse si afisate in consola.

```
def get_interfaces(self):
```

Returneaza: out\_interfaces (interfata) = interfetele de networking disponibile alturi de adresele ip.

## Programare

```
|def sendIp(self):
```

Trimite comanda "i{ip}\n" specificand adresa ip a conexiunii WiFi a microcomputerului.

```
|def rotateBed(self, angle):
```

Trimite comanda "r{directie}\_{unghi}\n" pentru a roti patul la un anumit unghi.

```
|def servo(self, angle):
```

Trimite comanda "k{unghi}\n" pentru a roti servoul la un anumit unghi.

```
|def laserOn(self):
```

Trimite comanda "l0\n" pentru a activa laserul.

```
|def laserOff(self):
```

Trimite comanda "l1\n" pentru a dezactiva laserul.

```
|def ledsOn(self):
```

Trimite comanda "e0\n" pentru a activa LED-urile.

```
|def ledsOff(self):
```

Trimite comanda "e1\n" pentru adezactiva LED-urile.

```
|def LCDOn(self):
```

Trimite comanda "x1\n" pentru a activa lumina de fundal a ecranului.

```
|def LCDOff(self):
```

Trimite comanda "x0\n" pentru a dezactiva lumina de fundal a ecranului.

```
|def getWeigt(self):
```

Trimite comanda "w\n" pentru a cere greutatea obiectului si asteapta primirea aceliei.

```
|def ck(self):
```

Trimite comanda "c\n" pentru a verifica conexiunea.

```
|def start(self):
```

Returneaza: (bool) = True daca se primeste raspunsul "start\n", False in caz contrar

Trimite comanda "s\n" pentru a cere statusul scannerului obiectului si asteapta primirea acelui.

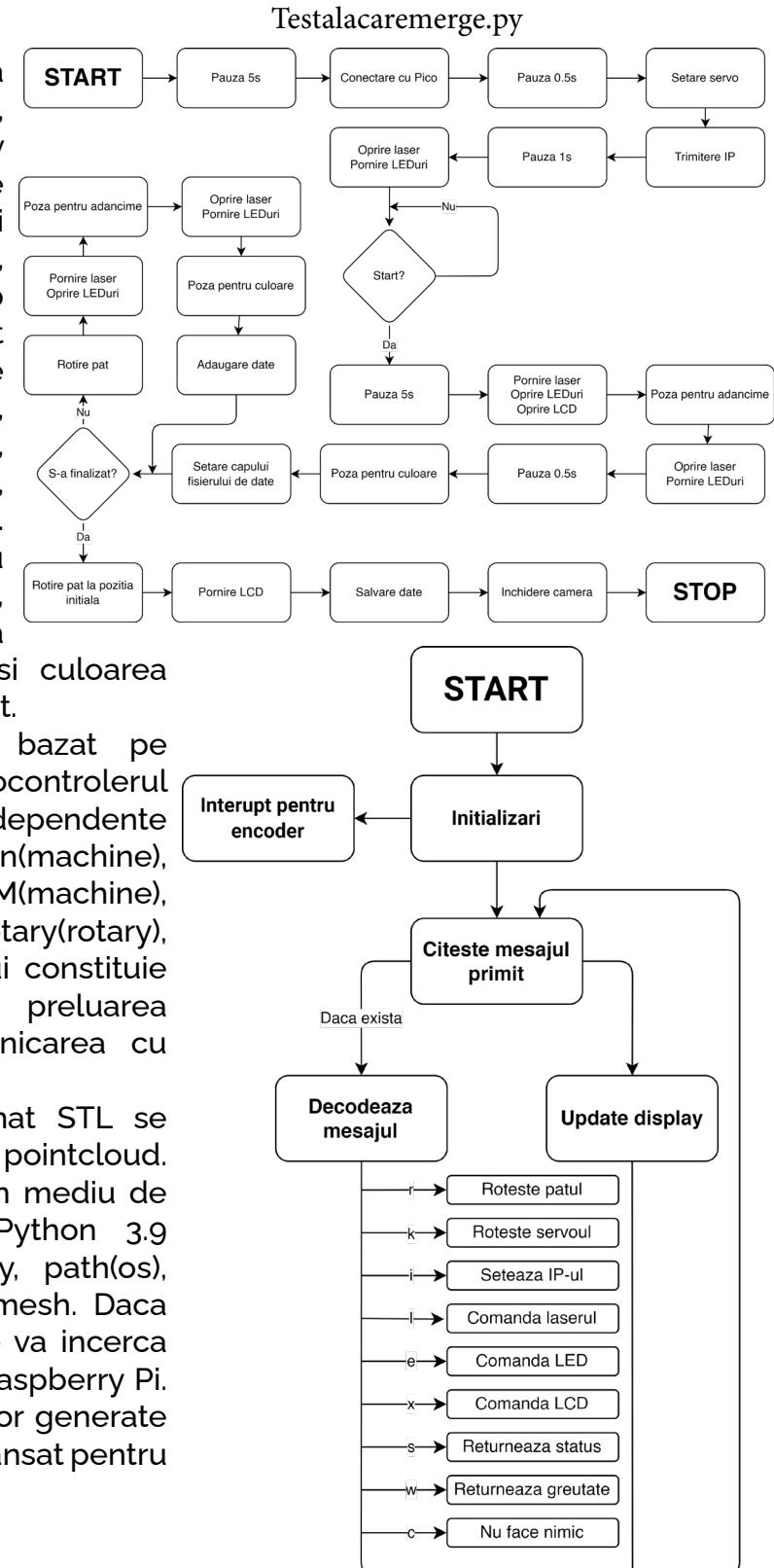
## Programare

### Diagramă funcțională a codului

Scannerul are la baza două scripturi principale scrise în Python3, Testalacaremerge.py și respectiv main.py, reprezentate prin diagramele alăturate. Primul rulează pe Raspberry Pi și este apelat la start printr-un script shell, launcher.sh, folosind serviciul crontab disponibil în linux. Pentru rulare sunt necesare urmatoarele module externe dependente: BytesIO(io), sleep(time), PiCamera(picamera), Image(PIL), cv2, numpy, yaml, imageproc(dataHandler), writer(dataHandler), pico(Pico). Acestea realizează conexiunea cu microcontrolerul Raspberry Pi Pico, înregistrează datele de la camera și senzori, și calculează adâncimea și culoarea punctelor care constituie obiectul scanat.

Cel de-al doilea scrip este bazat pe MicroPython și rulează la start pe microcontrolerul Raspberry Pi Pico. Modulele externe dependente pentru acest script sunt: I2C(machine), Pin(machine), UART(machine), ADC(machine), PWM(machine), sleep(time), I2cLcd(pico\_i2c\_lcd), Rotary(rotary), Nema(nema). Funcționalitatea scriptului constituie controlul elementelor hardware, preluarea comenziilor de la utilizator și comunicarea cu microcomputerul Raspberry Pi.

Generarea obiectului 3D în format STL se realizează pe computer prin rularea clasei pointcloud.py din terminalul Anaconda folosind un mediu de programare(environment) preștabilit Python 3.9 cu modulele necesare: math, numpy, path(os), json, datetime, cv2, open3d, sys și trimesh. Dacă nu se gasesc local datele generate se va încerca descarcarea acestora prin SSH de pe Raspberry Pi. Motivul procesării pe computer a datelor generate este necesitatea unui procesor grafic avansat pentru calculele de suprafață efectuate



main.py

# ReMake în vîtor

Desigur ca ne dorim va ReMake sa devine din ce in ce mai bun, asa ca ne propunem ca pana la urmatoarea etapa sa aducem o serie de imbunatatiri.

## Mișcare braț

În momentul de față brațul nu se mișcă, dar ne propunem să îl mișcăm circular în jurul obiectului pentru a putea fotografia și din mai multe unghiuri, lucru care ar duce la o scanare mai precisă.

Acest lucru este fezabil de realizat într-un timp scurt deoarece în momentul proiectării acesta îmbunătățire a fost luată în calcul și totul este pregatit pentru realizare.

## Creștem rezoluția scanări

In momentul de fata, pozele realizate de camera sunt la o rezolutie de 1280 pe 720 si sunt mai apoi taiate pentru a pastra doar regiunea de interes. In viitor dorim sa folosim rezolutia maxima a camerei, 4056 pe 3040 pentru a creste densitatea punctelor. Si rezolutia unghiulara va fi imbunatatita prin cresterea eficientei algoritmului de detectie a laserului, lucru care va permite realizarea mai multor poze fara a creste timpul de scanare. Perspectiva camerei va fi ajustata pentru a asigura ca planul focal este paralel cu planul obiectului.

## Creștem sensibilitatea detecției laserului

Detectarea laserului va fi imbunatatita prin ajustarea parametrilor de detectie. Vom implementa functii de ajustare automata pe baza luminozitatii exterioare si vom regla intensitatea laserului si a LED-urilor in mod corespunzator pentru a asigura un contrast ridicat. Si algoritmul prelucrare a imaginilor va fi optimizat prin evitarea conversiilor inutile intre tipurile de date.