# PriviBot

## Autor

Laurentiu Mihalcea, 333CB

## Introducere

Proiectul este un simplu sistem care, folosindu-se de două matrice de LED-uri ce reprezintă ochii acestuia, dă impresia că urmărește cu privirea individul aflat în fața camerei web. Scopul proiectului este în principiu acela de a înțelege mai bine termenii prezentați în cadrul laboratoarelor de PM și de a îmbina partea de embedded programming cu partea de ML/FaceRecognition.

## Descriere generală

## Schemă bloc

## Mod funcționare

Folosind OpenCV și Python serial, calculatorul află poziția utilizatorului față de camera web și o trimite plăcuței Arduino. În funcție de poziția utilizatorului aceasta va trimite celor două matrice de LED-uri modul în care ele trebuie să se aprindă astfel încât să rezulte efectul de mișcare a ochilor. În plus, aceasta mișcare a ochilor se va face doar dacă distanta obținută de senzorul ultrasonic este mai mică decât o valoare fixă. Efectul de clipire al ochilor va fi obținut prin intermediul unui timer intern plăcuței Arduino.

## Hardware Design

## Lista piese

- 1 x Arduino UNO/NANO
- 2 x matrice LED-uri
- 1 x Senzor ultrasonic (HC-SR04)
- 5 x Butoane
- 1 x camera web

## Schemă electrică

## Software Design

## Calculator

Mediu de dezvoltare: PyCharm
Librării folosite: OpenCV (detectare față), Python Serial (comunicare cu Arduino)

## Workflow

1. Se inițializează constantele și obiectele folosite pe parcursul programului.

```
# constants used when deciding if the face is in W, N, S or E
y_dif_offset = 50
x_dif_offset = 50

# constants used to define the frame's dimensions
frame_width = 800
frame_height = 400

# constants used to define the centered (reference) rectangle's dimensions
ref_width = 50
ref_height = 50

# bottom left corner for centered (reference) rectangle
ref_btm = (int(frame_width / 2 - ref_width / 2), int(frame_height / 2 -
ref_height / 2))
```

```
# top right corner for centered (reference) rectangle
ref_top = (int(frame_width / 2 + ref_width / 2), int(frame_height / 2 +
ref_height / 2))

# coordinates of the reference rectangle
ref_center = (frame_width / 2, frame_height / 2)

# initialize classifier
classifier = cv2.CascadeClassifier(cv2.data.haarcascades
                                   + 'haarcascade_frontalface_default.xml')
# initialize camera, use laptop's web camera
camera = cv2.VideoCapture(0)

# initialize serial com
to_board_serial = serial.Serial('/dev/cu.usbmodem14101', 9600)
```

2. Se extrage un frame, se prelucrează și apoi se extrag fețele din acel frame.

```
# extract current frame
_, frame = camera.read()

# resize frame to fit given dimensions
frame = imutils.resize(frame, width=800, height=400)

# mirror the image for better orientation
frame = cv2.flip(frame, 1)

# convert frame to its grayscale version
gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

# detect face
faces = classifier.detectMultiScale(
    gray_frame,
    scaleFactor=1.1,
    minNeighbors=4,
    minSize=(20, 20),
    flags=cv2.CASCADE_SCALE_IMAGE
)

# draw reference rectangle
cv2.rectangle(frame, ref_btm, ref_top, (0, 0, 255), 2)
```

3. Dacă a fost detectată cel puțin o față, se selectează prima din listă, se calculează centrul dreptunghiului care o încadrează și apoi se calculează diferența dintre centrul acestui dreptunghi și centrul dreptunghiului de referință (cel centrat în mijlocul frame-ului). Diferența dintre cele două dreptunghiuri este apoi folosită pentru a transforma poziția feței într-una dintre următoarele valori: CENTER, N, E, W, S, NW, NE, SW, SE. (această conversie este realizată în funcția **to_position** pe baza unor constante definite la pasul **1** și a semnelor componentelor vectorului diferență) Valoarea rezultată va fi trimisă apoi plăcuței Arduino.

```
# check if any faces were detected
```

```
if len(faces) > 0:
    # draw rectangle around the first face
    (x, y, w, h) = faces[0]
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

    # compute center of face
    center = (x + w / 2, y + h / 2)

    # compute difference between face's center and ref's center
    dif = (center[0] - ref_center[0], center[1] - ref_center[1])

    # convert coord. dif to an eye position
    serial_str = to_position(dif) + '\n'

    # send eye position to board
    to_board_serial.write(serial_str.encode('utf-8'))
```

```
def to_position(coords):
    # extract the x and y values
    x, y = coords

    if x >= x_dif_offset or x <= -x_dif_offset:
        if y_dif_offset >= y >= -y_dif_offset:
            if x > 0:
                return "EAST"
            else:
                return "WEST"
        elif y <= -y_dif_offset:
            if x > 0:
                return "NORTH_EAST"
            else:
                return "NORTH_WEST"
        elif y >= y_dif_offset:
            if x > 0:
                return "SOUTH_EAST"
            else:
                return "SOUTH_WEST"
    else:
        if y >= y_dif_offset:
            return "SOUTH"
        elif y <= -y_dif_offset:
            return "NORTH"

    return "CENTER"
```

4. Se afișează frame-ul și apoi se revine la pasul 1.

```
# Display frame
cv2.imshow('Video', frame)

# wait before proceeding to next frame
```

```
cv2.waitKey(1)
```

## Arduino

Mediu de dezvoltare: Arduino IDE
Librării folosite: MD_MAX72xx (comunicare cu matricele de LED-uri), NewPing (comunicare cu senzorul HC-SR04)

## Descriere generală

Plăcuța Arduino măsoară folosind senzorul HC-SR04 distanța dintre acesta și individ și apoi verifică dacă a primit input pe serială. Dacă distanța respectivă este mai mică decât o constantă (mai exact, individul este “in range”) și a fost primit input pe serială, atunci se updatează starea celor două matrice de LED-uri, se afișează starea respectivă și se resetează valoarea countdown-ului până la intrarea în modul IDLE. Practic, dacă plăcuța nu primește input pe serială sau individul este “out of range” se va decrementa valoarea countdown-ului până va ajunge la 0. Cât timp aceasta este 0 și nu se respectă condiția de input pe serială și cea de range, robotul își va mișca ochii în mod aleator (adică va sta în modul IDLE).

```
void loop() {
  // get distance from sensor to person
  distance = sensor.ping_cm();

  // check to see if there's input available from serial
  if (Serial.available() > 0 && distance <= MAX_DISTANCE) {
   // read display position string
   display_string = Serial.readStringUntil('\n');

   // convert display string to state (int value)
   int new_display_state = to_display_state(display_string);

   crt_display_state = new_display_state;

   display_eye(crt_display_state);

   idle_countdown = IDLE_COUNTDOWN_VAL;
  } else {
    // decrement idle countdown if no input on serial
    if (idle_countdown > 0)
      idle_countdown--;
  }

  // do idling if the countdown has reached 0
  if (idle_countdown == 0)
    do_idling();
```

```
}
```

# Diverse explicații

**1. Constante**
Pentru a putea citi codul mai ușor am grupat constantele folosite în 3 grupuri:

```
  1) Constante folosite pentru comunicarea cu matricele de LED-uri. Aici am definit pinii folosiți, numărul de matrice de LED-uri și luminozitatea minimă și maximă pe care o pot avea acestea.
  2) Constante folosite pentru comunicarea cu HC-SR04. Aici am definit pinii folosiți, distanța maximă pe care să o detecteze modulul (MAX_DETECTION_DISTANCE) și distanța maximă la care se poate afla individul față de modul (MAX_DISTANCE)
  3) Constante "general purpose", folosite pentru a realiza diverse verificări sau inițializări.
```

**2. Structuri de date**
Am folosit o singură structură de date: un enum care reține toate stările posibile în care se pot afla cele două matrice de LED-uri.

```
enum display_state {
  CENTER,
  NORTH,
  SOUTH,
  WEST,
  EAST,
  SOUTH_EAST,
  SOUTH_WEST,
  NORTH_EAST,
  NORTH_WEST
};
```

Intuitiv, ele corespund următoarelor poziții:
(Liniile colorate reprezintă constantele definite în codul python: x_dif_offset, -x_dif_offset, y_dif_offset, -y_dif_offset)

**3. Variabile globale**
De notat aici este faptul că, pentru fiecare stare din enum-ul **display_state** am definit un vector de 8 valori care reprezintă practic ce LED-uri trebuie să fie aprinse atunci când matricele se află în starea respectivă. De asemenea, am adăugat keyword-ul **volatile** doar în fața variabilelor care sunt modificate în handler-ele de întreruperi.

```
uint8_t closed[] =
{
  0b00111100,
  0b01111110,
  0b01111110,
```

```
  0b01111110,
  0b01111110,
  0b01111110,
  0b01111110,
  0b00111100
};

// current eye display state
int crt_display_state = CENTER;

// display string received from computer
String display_string;

// coundown until entering IDLE state
uint64_t idle_countdown = IDLE_COUNTDOWN_VAL;

// brightness value for the LED matrices
volatile int led_brightness = MIN_BRIGHTNESS;

// distance from sensor to individual
float distance;

// LED matrix control object
MD_MAX72XX mx = MD_MAX72XX(MD_MAX72XX::GENERIC_HW,
                           DATA_PIN, CLK_PIN,
                           CS_PIN, DEVICE_COUNT);

// sensor control object
NewPing sensor(TRIGGER_PIN, ECHO_PIN, MAX_DETECTION_DISTANCE);
```

**4. Întreruperi**

Pentru a adăuga diverse feature-uri la funcționalitatea de bază a proiectului (mai exact, aceea de a urmări cu privirea individul) m-am folosit de întreruperi. Am folosit atât întreruperi interne, cât și întreruperi externe. Cele interne sunt declanșate de către TIMER1 și sunt folosite pentru a face robotul să clipească la fiecare n secunde. Cele externe sunt declanșate de apăsarea unor butoane și sunt folosite pentru a face robotul să facă cu ochiul sau pentru a crește/scădea valoarea intensității cu care luminează LED-urile. În continuare voi atașa secțiunile de cod pentru ISR-ul și setup-ul întreruperilor.

*a) Timer*

Cum spuneam și mai sus, acesta este folosit pentru a face ca robotul să clipească la intervale regulate de timp (aici, 4 secunde).

```
void setup_timer() {
  TCCR1A = 0;
  TCCR1B = 0;
  TCNT1 = 0;

  // set match register in order to obtain a blink every 4s.
  OCR1A = 62500;

  // set timer in CTC mode
  TCCR1B |= (1 << WGM12);
```

```
  // set prescaler to 1024
  TCCR1B |= (1 << CS12) | (1 << CS10);

  // enable TIMER1 interrupt
  TIMSK1 |= (1 << OCIE1A);
}
```

```
ISR(TIMER1_COMPA_vect) {
  do_blink();
}
```

*b) INTn*
Am folosit întreruperile INT0 și INT1 pentru a face să pară ca robotul face cu ochiul. INT0 controlează ochiul stâng, iar INT1 pe cel drept. Pentru a elimina efectul de **debouncing** am introdus o verificare adițională în ISR care se asigură ca butonul a fost apăsat de către utilizator. (Funcția de setup include setup-ul pentru toate întreruperile date de butoane așa că o voi include doar aici)

```
void setup_buttons() {
  // set interrupt on falling edge for INT0
  EICRA |= (1 << ISC01);
  // set interrupt on falling edge for INT1
  EICRA |= (1 << ISC11);

  // enable INT0 interrupt
  EIMSK |= (1 << INT0);
  // enable INT1 interrupt
  EIMSK |= (1 << INT1);
  // enable PCINT20 interrupt
  PCMSK2 |= (1 << PCINT20);
  // enable PCINT21 interrupt
  PCMSK2 |= (1 << PCINT21);

  // set PD2 pin as INPUT
  DDRD &= ~(1 << PD2);
  // set PD3 pin as INPUT
  DDRD &= ~(1 << PD3);
  // set PD4 pin as INPUT
  DDRD &= ~(1 << PD4);
  // set PD5 pin as INPUT
  DDRD &= ~(1 << PD5);

  // enable PULLUP for PD2
  PORTD |= (1 << PD2);
  // enable PULLUP for PD3
  PORTD |= (1 << PD3);
  // enable PULLUP for PD4
  PORTD |= (1 << PD4);
  // enable PULLUP for PD5
  PORTD |= (1 << PD5);
```

```
  // enable PCMSK2 scan
  PCICR |= (1 << PCIE2);
}
```

```
ISR(INT0_vect) {
  static uint64_t last_time = 0;
  uint64_t crt_time = millis();

  if (crt_time - last_time > BOUNCE_TIME) {
    do_wink(LEFT_EYE);
  }

  last_time = crt_time;
}
```

```
ISR(INT1_vect) {
  static uint64_t last_time = 0;
  uint64_t crt_time = millis();

  if (crt_time - last_time > BOUNCE_TIME) {
    do_wink(RIGHT_EYE);
  }

  last_time = crt_time;
}
```

*c) PCINT2*
Deoarece nu mai aveam posibilitatea de a folosi întreruperi generate pe frontul crescător/descrescător al semnalului, am fost nevoit sa folosesc PCINT pentru a incrementa/decrementa valoarea intensității cu care luminează LED-urile. Problema cu această abordare este că luminozitatea se schimbă la fiecare schimbare logică a semnalului, dar, dpdv. al experienței utilizatorului am considerat că este admisibilă abordarea. La fel ca la INTn, am adăugat o verificare adițională pentru a elimina efectul de **debouncing**.

```
ISR(PCINT2_vect) {
  static uint64_t last_time = 0;
  uint64_t crt_time = millis();

  if ((PIND & (1 << PD4)) == 0) {
    if (crt_time - last_time > BOUNCE_TIME) {
      if (led_brightness < MAX_BRIGHTNESS)
        led_brightness++;
    }
  } else if ((PIND & (1 << PD5)) == 0) {
    if (crt_time - last_time > BOUNCE_TIME) {
      if (led_brightness > MIN_BRIGHTNESS)
        led_brightness--;
    }
  }
  last_time = crt_time;
```

```
}
```

**5. Idling**
Atunci când robotul intră în starea de idling, el va selecta în mod aleator o poziție a ochiului și va actualiza starea curenta a ochiului cu cea aleasă în mod aleator. Deși nu este “good practice”, am fost nevoit sa adaug un delay după fiecare apel al funcției **do_idling** pentru ca tranziția dintre două poziții să nu se facă mult prea repede.

```
void do_idling() {
  // randomly choose a new display state
  int new_display = rand() % 9;

  if (new_display != crt_display_state) {
    crt_display_state = new_display;
    display_eye(new_display);
  }

  delay(1000);
}
```

**6. Funcții afișare ochi**
În principiu, pentru a afișa un ochi, am parcurs cele 8 coloane ale matricei de LED-uri și am setat pe rând valoarea coloanei în funcție de starea curentă a ochiului (mai exact, dacă ochiul este în starea **CENTER**, atunci mă voi folosi de vectorul **center** pentru a face afișarea).În cazul blink-ului, am afișat mai întai elementele vectorului **closed** pentru efectul de închidere al ochilor și apoi am afișat elementele vectorului corespunzător stării curente a ochilor. (între cele două am fost nevoit să adaug din nou un delay pentru a se putea observa faptul că se închid ochii) Wink-ul este făcut exact ca blink-ul doar că elmentele vectorului **closed** se afișează doar în cazul unui ochi, celălalt rămânând neschimbat.

```
void display_eye(int eye_state) {
  mx.control(MD_MAX72XX::INTENSITY, led_brightness);

  // display each column of current eye state
  for (int i = 0; i < 8; i++)
  {
      if (eye_state == CENTER) {
        mx.setColumn(i, center[i]);
        mx.setColumn(i + 8, center[i]);
      } else if (eye_state == NORTH) {
        mx.setColumn(i, north[i]);
        mx.setColumn(i + 8, north[i]);
      } else if (eye_state == SOUTH) {
        mx.setColumn(i, south[i]);
        mx.setColumn(i + 8, south[i]);
      } else if (eye_state == EAST) {
        mx.setColumn(i, east[i]);
        mx.setColumn(i + 8, east[i]);
      } else if (eye_state == WEST) {
        mx.setColumn(i, west[i]);
        mx.setColumn(i + 8, west[i]);
```

```
      } else if (eye_state == NORTH_EAST) {
        mx.setColumn(i, north_east[i]);
        mx.setColumn(i + 8, north_east[i]);
      } else if (eye_state == NORTH_WEST) {
        mx.setColumn(i, north_west[i]);
        mx.setColumn(i + 8, north_west[i]);
      } else if (eye_state == SOUTH_EAST) {
        mx.setColumn(i, south_east[i]);
        mx.setColumn(i + 8, south_east[i]);
      } else if (eye_state == SOUTH_WEST) {
        mx.setColumn(i, south_west[i]);
        mx.setColumn(i + 8, south_west[i]);
      }
  }
}
```

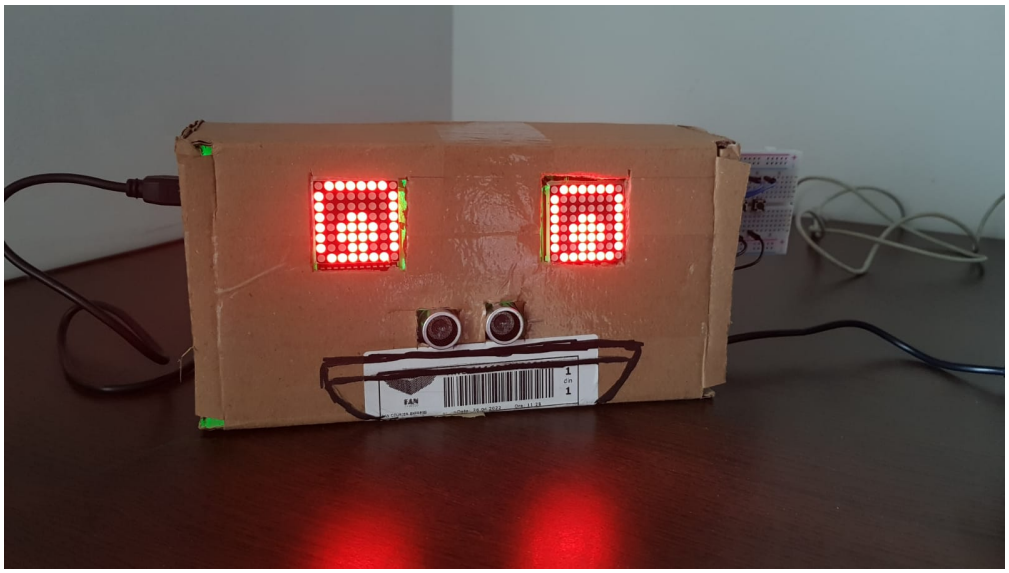
```
void do_blink() {
  mx.control(MD_MAX72XX::INTENSITY, led_brightness);

  // clear display
  mx.clear();

  // close eye
  for (int i = 0; i < 8; i++)
  {
      mx.setColumn(i, closed[i]);
      mx.setColumn(i + 8, closed[i]);
  }

  // add delay before printing eye
  delay(100000);

  // display current state
  display_eye(crt_display_state);
}
```

```
void do_wink(int eye) {
  mx.control(MD_MAX72XX::INTENSITY, led_brightness);

  if (eye == LEFT_EYE) {
    // left eye wink
    for (int i = 0; i < 8; i++)
    {
      mx.setColumn(i, closed[i]);
    }

  } else {
    // right eye wink
    for (int i = 0; i < 8; i++)
    {
      mx.setColumn(i + 8, closed[i]);
```

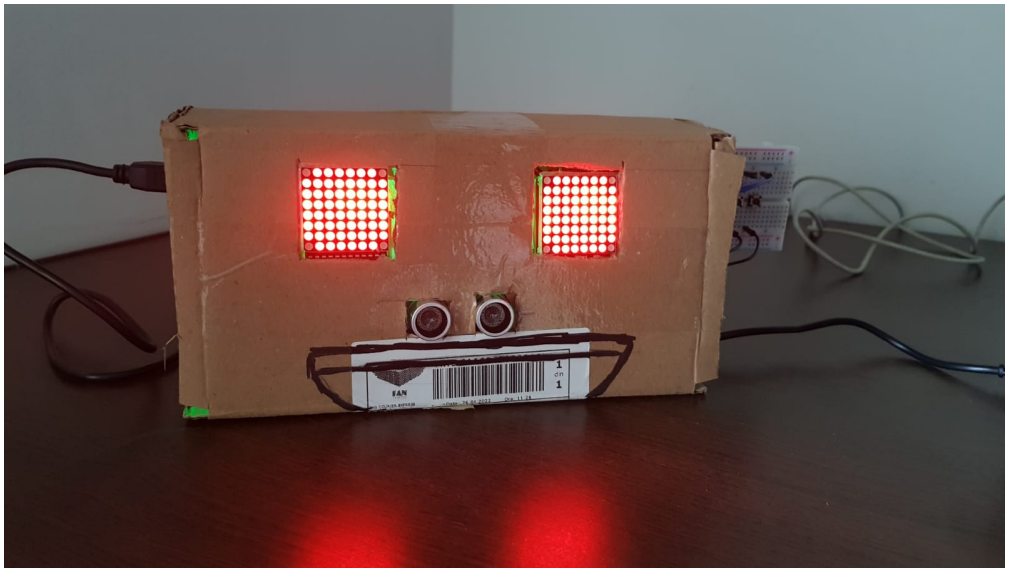
```
    }
  }

  // add delay before printing eye
  delay(100000);

  // display current state
  display_eye(crt_display_state);
}
```

## Rezultate Obţinute

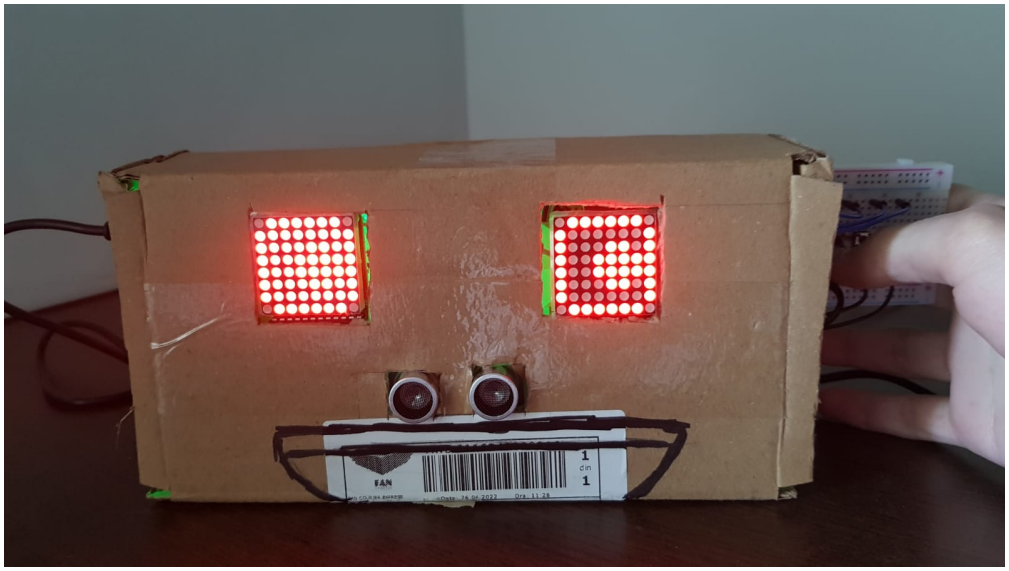
## Vedere generală



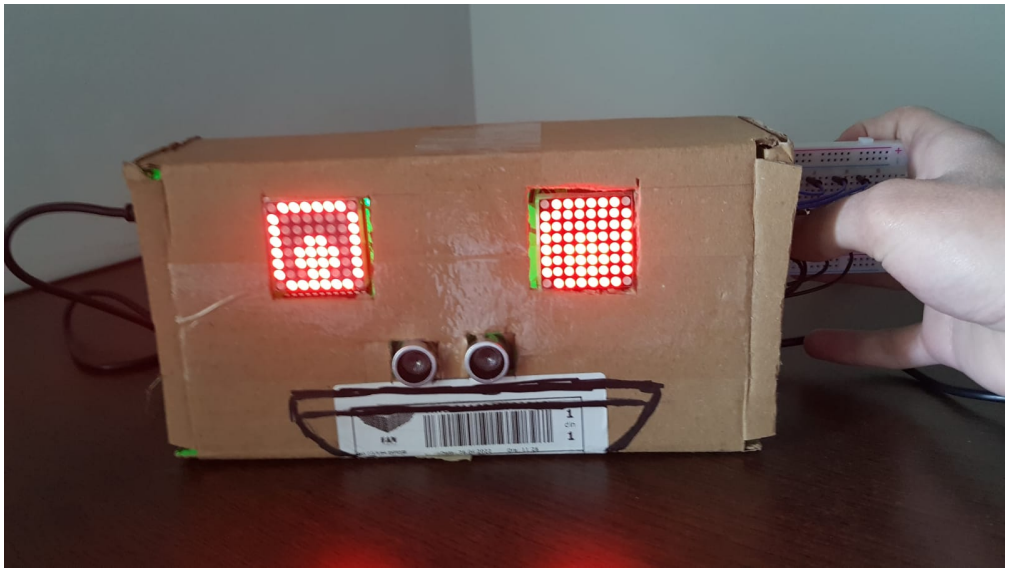
## Blink



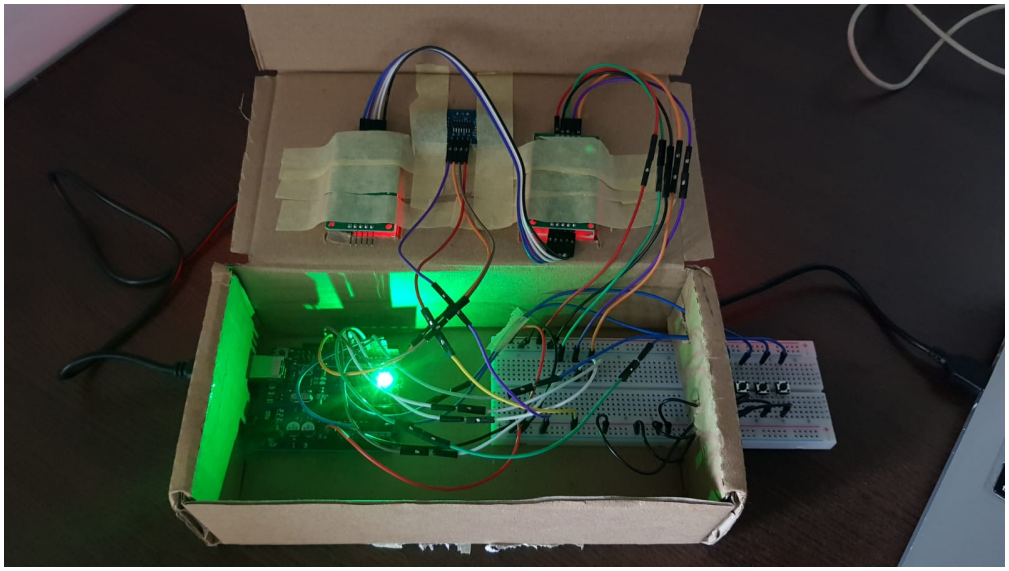
## Wink ochi stâng



## Wink ochi drept



## Vedere sub capotă



## Concluzii

În principiu proiectul și-a atins scopul. Am înțeles mai bine cum se utilizează întreruperile, timer-ele și m-am obișnuit cu mediul de dezvoltare embedded. Pe lângă asta, am explorat și o modalitate prin care se poate îmbina partea de ML cu partea de hardware. Totu-i bine când se termină cu bine...Acum la o bere...

## Download

PDF File: PriviBot
Cod Arduino:privibot_arduino.zip
Cod Python: privibot_python.zip

## Bibliografie/Resurse


- Ideea de matrice de LED-uri pe post de ochi + utilizare MD_MAX72xx + utilizare NewPing: https://create.arduino.cc/projecthub/unexpectedmaker/ultrasoniceyes-b9fd38?ref=platform&ref_id=424_trending___&offset=157
- Prelucrarea imaginilor + ideea de a folosi OpenCV pentru face tracking: https://create.arduino.cc/projecthub/shubhamsantosh99/face-tracker-using-opencv-and-arduino-55412e
- Utilizare întreruperi: https://ocw.cs.pub.ro/courses/pm/lab/lab2-2022
- Idee tratare debouncing: https://forum.arduino.cc/t/debouncing-an-interrupt-trigger/45110/2