UNIVERSITATEA POLITEHNICA din BUCUREŞTI
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei

Metode avansate de control şi automatizare a unui robot de tip Hexapod

Proiect de diplomă

Prezentat ca cerinţă parţială pentru obţinerea titlului de Inginer
În domeniul Electronică şi Telecomunicaţii
Specializarea Tehnologii şi Sisteme de Telecomunicaţii

Conducător ştiinţific

Şl. dr. Ing. Horia Cucu
Prof. dr. Ing. Corneliu Burileanu

Absolvent

Gaiţă Andrei

Bucureşti
2016

Anexa I

Universitatea "Politehnica" din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Departamentul Telecomunicaţii

Aprobat Director de Departament :
Prof. Dr. Ing. Silviu Ciochină

TEMA PROIECTULUI DE DIPLOMĂ
a studentului Gaiţă Andrei, grupa 441 C

1. Titlul temei: *Metode avansate de control şi automatizare a unui robot de tip Hexapod*

2. Contribuţia practică, originală a studentului va consta în:
Proiectarea şi realizarea practică a unui sistem de control la distanţă pentru un robot de tip hexapod din seria phantomX. Robotul va fi echipat cu un microcalculator care va indeplini funcţiile :

** recepţionare si prelucrare in timp real a datelor pe baza cărora robotul va executa diferite funcţii de mişcare (de ex: viteza, direcţia, controlul independent al fiecărui picior, etc)*
** transmiterea unor mesaje de feedback in timp real dispozitivului care va controla robotul (de ex: realizarea cu succes a funcţiei cerute, erori, imagini, etc.)*
Comunicaţia microcalculatorului cu sistemul care il comanda se va realiza wireless prin internet. Va fi dezvoltată o aplicaţie care asigură codarea şi transmiterea datelor de control către robot şi afişarea mesajelor de feedback intr-un mod cât mai inteligibil pentru un utilizator obişnuit.

3. Proiectul se bazează pe cunoştinţe dobândite în principal la urmatoarele 3 discipline: *Programarea Calculatoarelor, Microcontrolere, Circuite Electronice Fundamentale*

4. Proprietatea intelectuală asupra proiectului aparţine: *Laboratorului de cercetare Speech and Dialogue (SpeeD) şi studentului Gaiţă Andrei*

5. Locul de desfăşurare a activităţii: *Laboratorului de cercetare Speech and Dialogue (SpeeD)*

6. Realizarea practică/ proiectul ramân în proprietatea: *Laboratorului de cercetare Speech and Dialogue (SpeeD) şi studentului Gaiţă Andrei*

7. Data eliberării temei: *11.11.2015*

CONDUCATOR LUCRARE:

Prof. Dr. Ing. Corneliu Burileanu

S.L. Dr. Ing. Horia Cucu

STUDENT:

Gaiţă Andrei

DECLARAŢIE DE ONESTITATE ACADEMICĂ

Prin prezenta declar că lucrarea cu titlul "Metode avansate de control şi automatizare a unui robot de tip Hexapod", prezentată în cadrul Facultăţii de Electronică, Telecomunicaţii şi Tehnologia Informaţiei a Universităţii "Politehnica" din Bucureşti ca cerinţă parţială pentru obţinerea titlului de Inginer în domeniul Inginerie Electronică şi Telecomunicaţii, programul de studii Tehnologii şi Sisteme de Telecomunicaţii este scrisă de mine şi nu a mai fost prezentată niciodată la o facultate sau instituţie de învăţământ superior din ţară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referinţe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar şi în traducere proprie din altă limbă, sunt scrise între ghilimele şi fac referinţă la sursă. Reformularea în cuvinte proprii a textelor scrise de către alţi autori face referinţă la sursă. Înţeleg că plagiatul constituie infracţiune şi se sancţionează conform legilor în vigoare.

Declar că toate rezultatele simulărilor, experimentelor şi măsurătorilor pe care le prezint ca fiind făcute de mine, precum şi metodele prin care au fost obţinute, sunt reale şi provin din respectivele simulări, experimente şi măsurători. Înţeleg că falsificarea datelor şi rezultatelor contituie fraudă şi se sancţionează conform regulamentelor în vigoare.

Bucureşti, Data Gaiţă Andrei

CUPRINS

LISTA FIGURILOR

LISTA TABELELOR

LISTA ACRONIMELOR

AAC = Advanced Audio Coding
AES = Advanced Encryption Standard
ALU = Arithmetic Logic Unit
AMP = Apache, MySQL, PHP
ARM = Advanced RISC Machine

BSD = Berkeley *Software* Distribution

CGI = Common Gateway Interface
CISC = Complex Instruction Set Computer
CPU = Central Processing Unit
CSI = Camera Serial Interface

DIRT = Dată Intensive Realtime Applications
DSO = Dynamic Shared Object
DNS = Domain Name System
DVD-ROM = Digital Versatile Disc- Read Only Memory
DV = Digital Video

FTDI = Future Technology Devices Internaţional

GIF = Graphics Interchange Format
GOP = Group of Pictures
GPU = Graphics Processing Unit

HD = High Definition
HDMI = High-Definition Multimedia Interface
HTML = Hyperlink Text
HDV = High Definition Video

IETF = Internet Engineering Task Force
IP = Internet Protocol
ISA = Instruction Set Architecture
ISO/IEC = International Organization for Standardization/Internaţional Electrotechnical Commission

ISP = In-System Programmable
IT = Internet Technology
ITU-T = International Telecommunication Union - Telecommunication Standardization Sector

JPEG = Joint Photographic Experts Group
JSON = JavaScript Object Notation

MicroSDHC = Micro Secure High Capacity
MPEG-4 = Motion Picture Experts Group Layer-4 Video
MP3 = MPEG-1/2 Layer 3
MPM = Multiprocessing Modules
MIPI = Mobile Industry Processor Interface

OSI = Open Source Initiative

PC = Personal Computer
PHP = HyperText Preprocessor (Personal HomePage)
PNG = Portable Network Graphics
POSIX = Portable Operating System Interface
PTZ = Pan-Tilt-Zoom

QCIF = Quarter Common Intermediate Format or Common Interchange Format

Rpi = Raspberry Pi
RAM = Random Access Memory
RFC = Request for Comments
RISC = Reduced Instruction Set Computer

SOC = System on a Chip
SD = Secure Digital
SCL/SDA = Signal Clock/Signal Data
SDHC = Secure Digital High Capacity
SSH = Secure Shell
SSL = Secure *Socket* Layer

UDP = User Datagram Protocol
USB = Universal Serial Bus
URL = Uniform Resource Locator

VoIP = Voice over IP

WMV = Windows Media Video
WMA = Windows Media Audio
WHATWG = *Web* Hypertext Application Technology Working Group

CAPITOL 1

INTRODUCERE

1.1 MOTIVAŢIA LUCRĂRII

Redarea video în timp real a cunoscut o dezvoltare semnificativă în ultimul deceniu, lucru ce a dus la creare unei necesităţi din utilizarea acesteia. Ariile de utilizare a streaming-ului video sunt nenumărate, de la mass-media până la supravegherea propriei locuinţe. Nevoia oamenilor de securitate şi de cunoaştere au dus la dorinţa de a crea o nouă tehnologie stabilă care să poată oferi siguranţă şi rapiditate în viaţa de zi cu zi. Astfel, alegerea acestei teme a fost determinată de o tehnologie relativ nouă, cu o putere mare de a acapara domeniul tehnologic.

Nu în ultimul rând, această tehnologie trebuie dezvoltată şi din punctul de vedere al controlului pe care oamenii îl pot avea asupra ei. De aceea, redarea video trebuie să permită şi unui simplu utilizator să poată interveni în acest proces. De aici provine ideea de **control** introdusă de lucrarea de faţă. În plus, nevoia utilizatorului ca procesul să fie cât mai rapid introduce şi ideea de **automatizare**, care implică anumite configurări iniţiale pentru a permite unui sistem de redare video să poată funcţiona fără alte intervenţii. Astfel, se poate satisface şi ideea de **automatizare**.

1.2 OBIECTIVELE LUCRĂRII

Acest proiect îşi propune realizarea unui sistem de control al unui robot de tip hexapod prin intermediul Internetului cât şi realizarea de redare video prin intermediul unei camere ataşate de robot. Pentru realizarea acestor obiective am folosit următoarele trei sisteme: *server* HTTP pe maşină Ubuntu cu deschidere la *Internet*, *Raspberry Pi* şi hexapodul *PhantomX*.

Aplicaţia va avea la bază un set de *server*e definite în limbajul de *scripting JavaScript*, o aplicaţie realizată în *Python* ce permite comunicaţia serială cu robotul cât şi o aplicaţie realizată în limbajul de programare *Arduino C*, ce permite realizarea diferitelor tipuri de mişcări de către hexapod.

Proiectul de faţă dispune de o interfaţă *web* prin care utilizatorul va avea posibilitatea de a trimite comenzi către robot şi de a vizualiza *streaming-ul* video.

Din punct de vedere funcţional aplicaţia se împarte în două module:

- Modulul de control al robotului
- Modulul de redare video

Cele două module vor fi descrise pe larg în **Capitolul 6 - Dezvoltarea sistemului de control şi automatizare a robotului hexapod.**

1.3 DESCRIERE GENERALĂ A CONŢINUTULUI LUCRĂRII

Lucrarea aceasta este structurată pe 7 capitole, fiecare dintre acestea punând accent pe tehnologiile *software*/*hardware* utilizate, după cum urmează.

Capitolul 2 pune în evidenţă câteva dintre cele mai importante caracteristici *software*/*hardware* ale robotului hexapod *PhantomX* Mark II.

Capitolul 3 prezintă microcalculatorul *Raspberry Pi* utilizat împreună cu modulului camerei pentru realizarea redării video în timp real.

Capitolul 4 descrie tehnologia de redare video, punând accent pe standardul de codare H.264, format în care se obţine fişierul video preluat de *Raspberry Pi* prin intermediul camerei video.

Capitolul 5 pune în evidenţă tehnologiile *software*, protocoalele şi tipurile de conexiuni utilizate în realizarea lucrării.

Capitolul 6 întregeşte toate aceste noţiuni pentru a prezenta paşii urmaţi pentru realizarea fiecărui modul *software* în parte.

CAPITOL 2

ROBOTUL HEXAPOD *PHANTOMX*

2.1 ROBOȚI HEXAPOZI - CONSIDERENTE GENERALE

Un robot hexapod reprezintă un dispozitiv mecanic şi electronic, a cărui mişcare se bazează pe cele şase picioare. Spre deosebire de alte tipuri de roboţi, cu două, trei sau patru picioare, robotul hexapod dispune de o flexibilitate şi de o stabilitate superioară. De aceea, comportamentul unui robot hexapod este mult mai complex, mai ales datorită faptului că nu toate cele şase picioare sunt necesare mişcării; celelalte pot fi folosite pentru a ridica obiecte sau pentru o mai bună direcţionare a robotului spre anumite zone. Se consideră că acest tip de robot a fost creat pentru a testa teorii biologice legate de mişcarea insectelor, controlul de motoare şi neurobilogia. Nu în ultimul rând, astfel de roboţi pot fi folosiţi în misiuni de descoperire sau cercetare, în locuri greu accesibile oamenilor(de exemplu, în zone devastate de cutremure sau alte calamităţi naturale).

Design-ul unui hexapod poate varia din punctul de vedere al aranjării picioarelor. De cele mai multe ori, inspiraţi fiind din anatomia unei insecte, hexapozii vor avea picioarele distribuite simetric. Mai mult decât atât, picioarele dispun de două până la 6 puncte de libertate. Mişcările unui hexapod sunt controlate de tipurile de paşi pe care acesta îi poate face: tripod alternat(utilizarea a trei picioare la un pas), patruped sau târâtor(mişcarea unui singur picior). Pe lângă aceşti paşi standard de control al hexapodului, mişcarea poate fi aleasă în mod diferit în funcţie de mediul exterior. Interacţiunea om-robot are o importanţă sporită şi în cazul roboţilor hexapozi. Controlul exercitat de om asupra hexapodului variază între diferite nivele de autonomie. Omul poate avea control absolut asupra

mişcărilor robotului, pe care le poate comanda prin programarea acestuia; de asemenea, robotul poate fi programat astfel încât comenzile pe care le înregistrează să îl determine să ia decizii mai complexe pentru a îndeplini cerinţa. Există şi roboţi autonomi care pot funcţiona o perioadă îndelungată fără interacţiunea cu omul, reacţionând pe baza unor modele bine definite.

2.2 STRUCTURA *HARDWARE*

PhantomX **[12]** este un robot hexapod de dimensiune medie, creat de specialiştii din cadrul *Interbotix Labs*. Acest robot hexapod are o complexitate medie, dispunând de elemente *software open-source*, create cu intenţia de a încuraja un număr cât mai mare de oameni să experimenteze şi să modifice codul sursă care permite controlul robotului. Miezul robotului *PhantomX* este reprezentat de *Robocontrollerul Arbotix*, care funcţionează pe un sistem de cinematică şi mişcare inversă, comandând reţeaua *Dynamixel AX-12* pentru poziţionarea picioarelor. *Arbotix* acceptă comenzi de navigare prin intermediul **Protocolului de Comandă**, un protocol serial simplu ce permite un control proporţionat al mişcării robotului. *Robocontrollerul* poate comunica *wireless* cu un sistem de comandă manual, prin intermediul acestui protocol şi cu ajutorul unei perechi de module *wireless Xbee*. Acelaşi protocol poate fi utilizat şi pentru comunicaţia dintre un PC şi hexapod, folosind interfaţă USB *Xbee*. Controlul hexapodului *PhantomX* poate fi făcut cu ajutorul oricărui limbaj de programare care este capabil de a transmite date printr-un port serial.

Robotul hexapod *PhantomX* este prezentat în Figura 2.1.

Figura 2.1 - Robotul Hexapod *PhantomX Mark II*
Sursa [1]

2.2.1 Robocontroller ArbotiX-M

Robocontrollerul Arbotix-M **[13]** reprezintă miezul oricărui robot *Interbotix*, prin versatilitatea sa, oferind o soluţie de control avansată pentru actuatoare DYNAMIXEL şi BIOLOID. Acest *microcontroller*, asemenea unuia compatibil cu Arduino, beneficiază de numeroase librării şi exemple *open-source*. Litera "M" din numele acestui *microcontroller* provine de la "mini", considerând că acesta este cu 25% mai mic decât prima versiune *Arbotix*.

Din punct de vedere *hardware*, *Robocontroller*ul are la bază un *microcontroller* AVR, cu o frecvenţă de tact de 16MHz (ATMEGA644p), conţinând şi 2 porturi seriale, unul dedicat *servocontrollerului* Bioloid şi unul pentru programarea *XBEE* de tip radio/FTDI. De asemenea, *Robocontrollerul* mai este compus din 3 pini compatibili DYNAMIXEL TTL, 28 de pini digitali de I/O, ce pot funcţiona şi că pini analogici, 3 pini pentru **GND**, **VCC** şi semnal şi altele. Acestea urmează să fie prezentate pe rând în următoarele rânduri. Mai jos, Figura 2.2 prezintă structura *Robocontrollerului Arbotix-M*.

- **Alimentarea DC**

În cadrul *Robocontroller-ului* întâlnim 2 *jack-uri* pentru alimentarea DC a plăcuţei. Ambii sunt echivalenţi din punct de vedere electric, astfel că alimentarea *controllerului* se poate face folosind oricare dintre aceste două *jack-uri*. Primul accepta 2 fire - unul de masă şi unul pentru voltaj, iar al doilea accepta un *jack* DC de dimensiuni 2.1x5.5 mm. Puterea de alimentare furnizată de la aceste 2 *jack-uri* poate alimenta: orice actuator DYNAMIXEL conectat la *Arbotix-M* şi regulatorul de putere integrat pe plăcuţă (care va furniza la rândul său un voltaj de 5V pentru restul componentelor).

Robocontrollerul Arbotix-M (Figura 2.2) poate funcţiona la tensiuni de alimentare între 7-30V, dar în mod normal se utilizează voltaje de 11-12V, având în vedere că aceasta este tensiunea maximă la care actuatoarele DYNAMIXEL pot funcţiona corect.

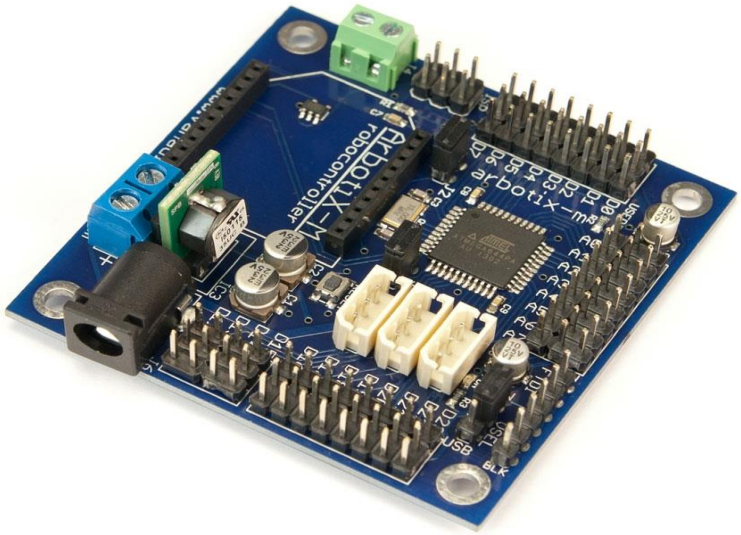


Figura 2.2 - *Controller-ul ArbotiX-M*
Sursa [2]

- **Regulatorul de tensiune**

Regulatorul de tensiune de 5V va prelua tensiunea de intrare de la pinul VIN şi o va aduce la o tensiune stabilă, pentru o funcţionare normală a *controllerului* ATMEGA644p şi a senzorilor şi echipamentelor ataşate. Regulatorul poate funcţiona la tensiuni de intrare de 7-30V. Tensiunile de ieşire pot varia între 1.5 şi 5V. O observaţie importantă legată de acest regulator este faptul că acesta nu furnizează putere pentru pinii digitali D12-D15.

- ***Jumper-ul* pentru selectarea puterii**

Acest *jumper* oferă posibilitatea alegerii sursei de putere pentru alimentarea *Robocontrollerului*. Utilizatorul poate alege între alimentarea la voltajul de 5V furnizat de către regulatorul de tensiune, care primeşte puterea de intrare de la pinul VIN, şi alimentarea de la pinul USB prin intermediul unui cablu FTDI - alimentarea de la conectorul USB al PC-ului.

- **LED-ul indicator al alimentării**

Acest LED se va aprinde de fiecare dată când *Robocontrollerul* este alimentat, fie prin intermediul regulatorului, fie prin intermediul portului USB.

- **Butonul RESET**

Acest buton va reseta *Arbotix* şi va reporni programul încărcat pe *microcontroller*.

- **Portul serial FTDI/Portul de programare**

Acest port este unul dual: atât port de programare, cât şi port serial. Prin conectarea unui dispozitiv FTDI (precum un cablu FTDI sau un UartSBee), utilizatorul poate programa *Robocontrollerul* şi poate comuta către comunicaţia serială.

De asemenea, acest port permite PC-ului conectat să reseteze *Robocontrollerul*. Cei 6 pini ai portului FTDI au următoarele funcţii: resetare(verde), transmisiune(galben), recepţie(portocaliu), alimentare 5V(roşu), legare la împământare(maro) şi ground(negru).

- **Portul de programare ISP**

Acest port reprezintă un mod secundar de a programa *Robocontrollerul Arbotix*-M. Principalul avantaj al utilizării acestui port în dezavantajul celui FTDI este faptul că nu există nevoia unei conexiuni seriale. Acest lucru înseamnă că *Robocontrollerul* poate fi programat prin simpla conectare a unui *Xbee*. Pe de altă parte, dezavantajul este faptul că nu se pot trimite sau primi comunicaţii seriale prin intermediul portului ISP.

Robocontrollerul Arbotix-M poate fi programat prin intermediul portului FTDI doar dacă *microcontrollerul* ATMEGA644p are un *Bootloader* instalat. Toate unităţile *Arbotix* sunt înzestrate cu un *bootloader*, însă programarea prin portul ISP va suprascrie acest *bootloader*.

❖ Pinii digitali de intrare/ieşire

Fiecare dintre grupurile de pini digitali prezintă o configuraţie de tip Semnal-Tensiune-Masă, fiind astfel compatibili cu o gamă largă de senzori ce pot fi utilizaţi în diverse aplicaţii ce implică robotul hexapod *PhantomX*.

❖ Pinii analogici de intrare

Asemenea pinilor digitali, cei analogici prezintă o configuraţie de tip Semnal-Tensiune-Masa. Aceşti pini se comportă ca nişte pini digitali, fiind accesibili prin adresarea pinilor digitali 24-31.

❖ Porturi TTL

Controller-ul Arbotix-M conţine 3 porturi TTL pentru conectarea actuatoarelor DYNAMIXEL. Acestea sunt identice, deoarece toate actuatoarele conectate se vor afla pe acelaşi port serial. Porturile TTL pot fi utilizate doar pentru conectarea unor actuatoare de tip DYNAMIXEL.

❖ *Socket XBee*

Socket-ul Xbee permite adăugarea comunicaţiei *wireless* la *controller-ul Arbotix-M*, prin crearea unei conexiuni seriale fără fir. Această conexiune permite comunicarea lui *Arbotix-M* cu alt *microcontroller* sau cu un PC, prin intermediul unui alt modul *Xbee*. *Socket-ul Xbee* partajează un port serial cu portul FTDI, ceea ce nu permite utilizarea ambelor porturi simultan.

❖ Pinii IIC/I2C/I^{2}C

Controllerul Arbotix-M nu deţine un port dedicat IIC, însă suportă protocul IIC pe pinii 16 şi 17.

2.2.2 Actuatorul DYNAMIXEL AX-12A

Actuatorul AX-12A (Figura 2.3) **[3]** de la *Robotis* este unul dintre cele mai avansate actuatoare de pe piaţă şi a devenit un actuator standard pentru următoarea generaţie de robotice. Acest motor are abilitatea de a-şi monitoriza viteza, temperatura, poziţia, voltajul şi încărcarea. Mai mult decât atât, algoritmul de control utilizat pentru a menţine poziţia corectă a actuatorului poate fi ajustat individual pentru fiecare motor în parte, permiţând controlul vitezei şi puterii pentru răspunsul motorului. Acest management al poziţiei actuatorului este întreprins de către un *microcontroller* integrat. Abordarea aceasta distribuită permite *controllerului* principal să realizeze şi alte funcţii.

Zona de operare a actuatorului AX-12A este determinată de unghiurile climia CW/CCW, care variază între 0°-300°.

Robotul hexapod *PhantomX* dispune de 18 actuatoare Dynamixel AX-12A, dispuse ca în Figură 2.4.

Figura 2.3 - Actuatorul *DYNAMIXEL AX-12A*

Sursa [3]

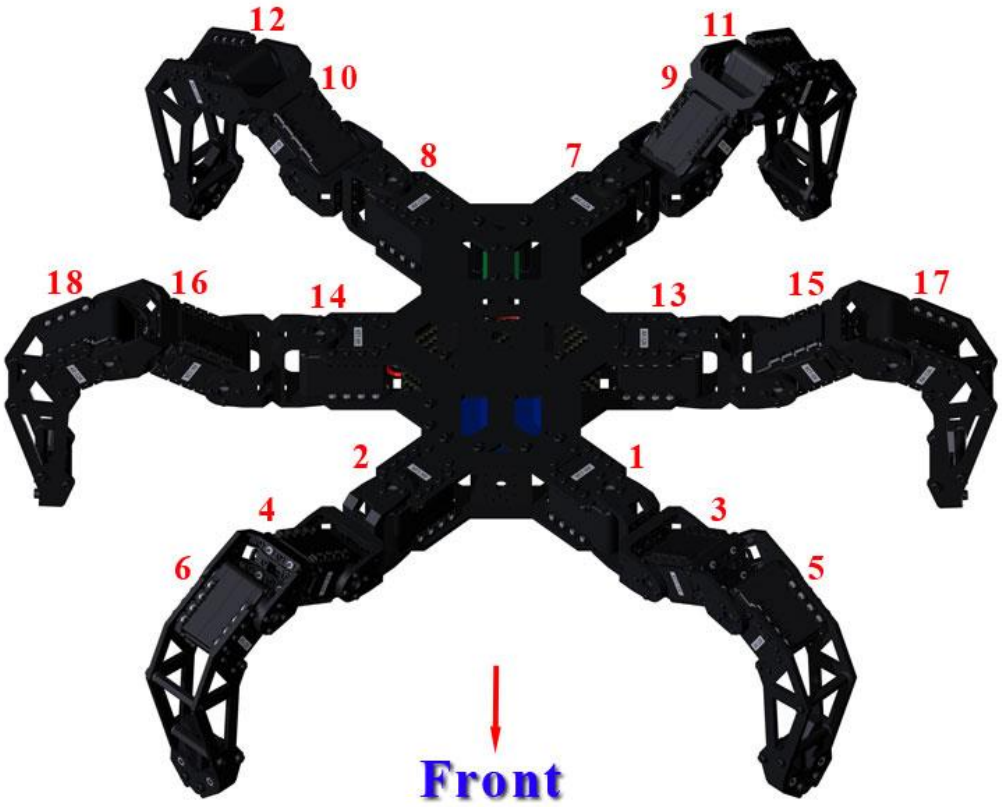


Figura 2.4 - Distribuirea actuatoarelor DYNAMIXEL în robotul hexapod *PhantomX* Mark II

Sursa [4]

Printre specificaţiile *hardware* are actuatorului se numără:

- Greutatea: 53.5g
- Dimensiunea: 32mm*50mm*40mm
- Rezoluţia: 0.29 °
- Temperatura de operare: -5 °C ~ +70 °C
- Tensiunea de alimentare: 9~12V(tensiunea recomandată este de 11.1V)
- Tipul de protocol: half-duplex, ascincron serial
- Legătura fizică: TTL(conector daisy chain)
- Viteza de comunicaţie: 7343bps~1Mpbs
- Material: plastic de inginerie

2.2.3 Acumulatorul Lipo

O baterie Lithium-polymer, sau mai corect spus, Lithium-ion polymer, este o tehnologie recentă apărută în domeniul bateriilor reîncărcabile, construite la dimensiuni mult mai mici şi în forme diverse, fiind de asemenea şi mai puţin rigide decât bateriile precedente.

Acumulatorul cu numele mai general Lithium-Ion este folosit aproape în orice echipament electronic cotidian, de la telefoanele mobile şi laptop-uri până ls vehicule electrice alimentate prin baterie.

Printre avantajele utilizării acestui tip de acumulator se numără: capacitatea de înmagazinare a unei puteri mari, dispune de rate mari de descărcare, ceea ce îl face utilizabil în majoritatea aplicaţiilor electronice şi poate şi realizat în orice formă sau mărime.

CAPITOL 3

SISTEMUL DE PROCESARE *RASPBERRY PI*

3.1 NOȚIUNI GENERALE

Raspberry Pi este un sistem de procesare de dimensiunea unui card, la un preț accesibil pentru publicul general, ce poate fi conectat la monitorul unui calculator sau la un televizor şi poate fi controlat cu ajutorul unei tastaturi sau a unui *mouse*. Acest mic dispozitiv permite oamenilor de toate vârstele să exploreze lumea procesării şi a programării în limbaje de programare că *Scratch* sau *Python*. Acest mic calculator este capabil de performanţe apropiate de cele ale unui calculator de tip *desktop*, aceste performanţe începând cu o simplă căutare pe Internet (*browsing*) sau utilizarea jocurilor video HD, ajungând până la crearea de tabele sau foi de calcul, procesare de cuvinte şi utilizare de jocuri.

În plus, sistemul de procesare *Raspberry Pi* are abilitatea de a interacţiona cu lumea exterioară, a fost şi continuă să fie folosit într-un număr mare de proiecte digitale, de la maşini muzicale până la staţii meteo şi căsuţe pentru păsări ce utilizează camere infraroşu. Unul dintre scopurile principale pentru care acest sistem de procesare a fost creat este orientarea copiilor către lumea programării, pentru a înţelege cum funcţionează calculatoarele şi pentru a învăţa să le utilizeze în mod curent.

Raspberry Pi a fost creat de către Fundaţia *Raspberry Pi*, o fundaţie înregistrată ca una de tip educaţională şi de caritate în Marea Britanie. Ţelul acestei fundaţii este avansarea în educaţie a

copiilor şi a adulţilor, într-un cuvânt a persoanelor de toate vârstele, în special în domeniul IT, al calculatoarelor şi al ştiinţei calculatoarelor, dar şi în domenii adiacente.

Creatorii sistemului de procesare *Raspberry Pi* au făcut publică prima generaţie Rpi în anul 2011, sub numele de *Raspberry Pi Model A* şi o altă variantă mai performantă, *Model B*. Un an mai târziu, în 2012, aceştia au revenit pe piaţă cu alte două modele, A+ şi B+. Trei ani mai târziu, în 2015 au fost lansate şi *Raspberry Pi2 Model B* şi *Raspberry Pi3 Model B* cu performanţe superioare modelelor anterioare.

Deşi diferite din punctul de vedere al puterii de procesare, toate modele de Rpi au câteva caracteristici *software* şi *hardware* comune. Printre acestea, putem aminti faptul că miezul procesării are loc pe un *chip*, de aici şi numele de "sistem pe un *chip*"(SOC) de tip *Broadcom*. Acest *chip* include o unitate de procesare compatibilă ARM, sau mai scurt, procesor (CPU), şi o unitate de procesare video GPU, de tip *VideoCore IV*. Viteza procesorului variază între 700 MHz şi 1.2GHz pentru ultimul model lansat pe piaţă (RPi 3), iar memoria RAM se află în intervalul 256MB şi 1 GB. Deoarece am menţionat faptul că RPi se comportă că un mini-calculator, acesta are nevoie de un sistem de operare care este stocat pe un card SD, fie în dimensiunea SDHC fie în MicroSDHC. Majoritatea modelelor RPi conţin între 1-4 slot-uri USB, HDMI, o ieşire video de tip *composite video* (un canal pentru transmisiuni video analogice, fără semnal audio) şi o altă ieşire pentru semnal audio. Nu în ultimul rând, câteva modele conţin şi port *Ethernet* de tip RJ45, iar ultimul model RPi 3 are inclus şi *WiFi*, standard 802.11n şi *Bluetooth*.

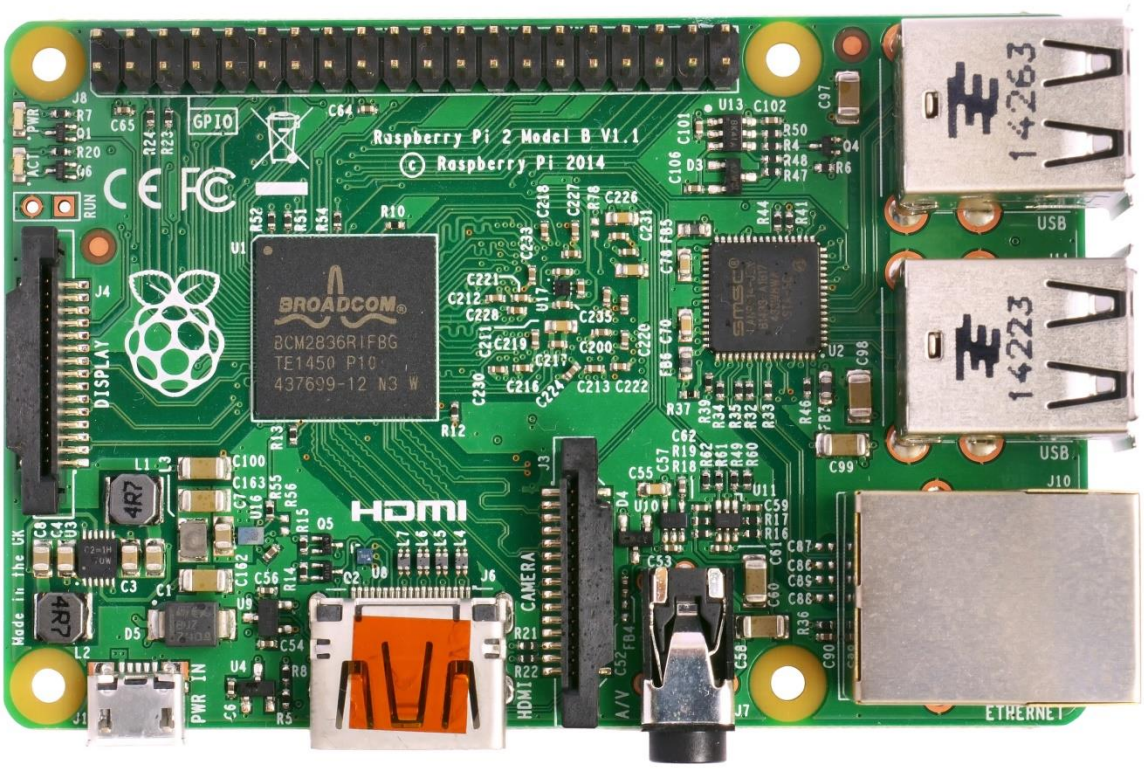


Figura 3.1 Microcalculatorul *Raspberry Pi2 Model B*
Sursa [5]

3.2 STRUCTURA *HARDWARE*

Tabelul 3.1 prezintă comparativ specificaţiile *hardware* pentru diferitele modele de sisteme Raspberry Pi, enumerate şi descrise pe scurt mai sus.

	Model A	**Model B**	**Model A+**	**Model B+**	**Raspberry Pi2**	**Raspberry Pi3**	**Raspberry PiZero**
System-on-a-chip (ŞoC):	Broadcom BCM2835 (CPU + GPU. SDRAM is a separate chip stacked on top)				Broadcom BCM2836	Broadcom BCM2837	Broadcom BCM2835
CPU:	700 MHz ARM11 ARM1176JZF-S core				900MHz quad-core ARMv7 Cortex-A7	1.2GHz 64-bit quad-core ARMv8 Cortex-A53	1000MHz Low Power ARM1176JZ-F
GPU:	Broadcom VideoCore IV, OpenGL ES 2.0,OpenVG 1080p30 H.264 high-profile encode/decode, 250 MHz					Broadcom VideoCore IV, OpenGL ES 2.0,OpenVG 1080p60 H.264 high-profile encode/decode, 400 MHz	Broadcom VideoCore IV
Memory (SDRAM)iB	256 MiB (planned with 128 MiB, upgraded to 256 MiB on 29 Feb 2012)	256 MiB (until 15 Oct 2012); 512 MiB (since 15 Oct 2012)	256 MiB	512 MiB	1024 MiB		512 MiB
SB ports:	1 USB 2.0 (provided by the BCM2835)	2 USB 2.0 (via integrated USB hub în LAN9512)	1 USB 2.0 (provided by the BCM2835)	4 USB 2.0 (via integrated USB hub în LAN9514)			1 Micro USB OTG (On The Go)
Video outputs:	Composite video | Composite RCA, HDMI (not at the same time)		HDMI | Composite video requires 4 Pole Adapter				HDMI, Composite video via unsoldered 2-pin header
Audio outputs:	TRS connector | 3.5 mm jack, HDMI						Multi-Channel HD Audio over HDMI
Audio inputs:	None, but a USB mic or sound-card could be added						
Onboard Storage:	Secure Digital|SD / MMC / SDIO card slot		Micro Secure Digital / MicroSD slot				
Onboard Network:	None	10/100 wired Ethernet RJ45	None	10/100 wired Ethernet RJ45		10/100 wired Ethernet RJ45, integrated 802.11n Wi-Fi & Bluetooth 4.1	None
Low-level peripherals:	26 General Purpose Input/Output (GPIO) pânş, Serial Peripheral Interface Buş (SPI), I²C, I²S, Universal asynchronous receiver/transmitter (UART)		40 General Purpose Input/Output (GPIO) pânş, Serial Peripheral Interface Buş (SPI), I²C, I²S,I2C IDC Pânş, Universal asynchronous receiver/transmitter (UART)				40 General Purpose Input/Output (GPIO) pânş, Serial Peripheral Interface Buş (SPI) (unpopulated)
Real-time clock:	None						
Power ratings:	300 mA, (1.5 W)	700 mA, (3.5 W)	600mA up to 1.2A @ 5V	~650 mA, (3.0 W)		800mA (4.0 W), up to 2.5A	160mA rating

Power source:	5 V (DC) via Micro USB type B or GPIO header						
Size:	85.0 x 56.0 mm x 15mm	85.0 x 56.0 mm x 17mm	65.0 x 56.0 mm x 12mm	85.0 x 56.0 mm x 17mm	85.0 x 56.0 mm x 17mm	85.6 x 56.5 mm x 17mm	65.0 x 30.0 mm x 5mm
Weight:	31g	40g	23g	40g	40g	45g	9g

Tabelul 3.1 - Comparaţie a specificaţiilor *hardware* pentru modelele Rpi

Sursa [6]

În continuare, pentru a putea detalia cele mai importante caracteristici *hardware* ale mini-calculatorului *Raspberry Pi2*, voi clarifica noţiunile implicate de acestea.

3.2.1 Procesorul Cortex A7 - Arhitectura ARMv7

Ahitectura ARM formează bazele pentru orice procesor ARM. De-a lungul timpului, arhitectura ARM a evoluat prin includerea de trăsături arhitecturale care să satisfacă cerinţele în creştere de noi functionalităţi, de integrare a securităţii, de performanţe înalte. Arhitectură ARM suportă implementări într-o gamă largă de puncte de performanţă, stabilindu-se ca fiind arhitectura de vârf în multe segmente de piaţă. Am precizat faptul că acest tip de arhitectură este folosită într-o gamă largă de aplicaţii, atât în implementări simple, cât şi în implementări avansate în care se folosesc tehnici de ultimă oră de micro-arhitecturi. Atributele cheie ale arhitecturii ARM sunt dimensiunile de implementare, performanţa şi consumul mic de putere.

Arhitectura ARM este similară cu o arhitectură de tip RISC, încorporând următoarele caracteristici tipice:

- arhitectura de stocare uniformă a registrelor, în care procesarea de date operează doar pe conţinutul registrelor, nu direct în conţinutul memoriei
- moduri de adresare simple, cu adresele de stocare determinate doar din conţinutul registrelor şi din câmpul instrucţiunilor

Suplimentar, arhitectura ARM aduce următoarele îmbunătaţiri:

- control atât asupra ALU cât şi asupra comutatorului în majoritatea instrucţiunilor de procesare a datelor pentru maximizarea utilizării acestor două componente
- moduri de adresare cu autoincrementare şi autodecrementare pentru a optimiza buclele de program
- încărcarea şi stocarea de instrucţiuni multiple pentru a maximiza cantitatea de date
- executare condiţională a majorităţii instrucţiunilor pentru a maximiza rata execuţiei

Aceste imbunătăţiri aduse unei arhitecturi RISC de bază permite procesoarelor ARM să obţină un bun echilibru de performanţe înalte, de dimensiuni reduse ale codului scris, consum mic de putere şi arii mici de silicon.

Procesorul din centrul sistemului *Raspberry Pi* este un procesor multimedia *Broadcom* BCM2836 de tip *system-on-chip* (SoC) **[14]**. Acest lucru înseamnă că majoritatea componentelor sistemului, inclusiv unitatea de procesare şi cea de procesare grafică, împreună cu unitatea audio şi partea *hardware* de comunicaţii, sunt construite împreună într-o singură componentă ascunsă în spatele unui chip de memorie de 1GB. Nu numai construcţia de tip SoC

este ceea ce diferenţiază acest procesor de cel utilizat la un *laptop* sau la un *desktop,* ci şi setul diferit de instrucţiuni (ISA) al arhitecturii ARM, prezentate anterior.

Arhitectura ARM, dezvoltată de către *Acom Computers* pe la sfârşitul anilor 1980, nu este atât de comună în lumea echipamentelor desktop. Aceasta excelează însa în zona echipamentelor mobile; de exemplu, majoritatea telefoanelor de tip *smartphone* utilizate în zilele moderne conţin un miez de procesare bazat pe arhitectura ARM. Combinaţia dintre un set redus de instrucţiuni şi un consum mic de putere face ca alegerea perfectă să fie cea a unui procesor ARM în locul unuia de tip *desktop* (generaţia x86) cu un consum ridicat de putere şi cu un set mai complex de instrucţiuni(CISC).

Procesorul BCM2836 bazat pe o arhitectură ARM este secretul operării sistemului *Raspberry Pi* la o alimentare de 5V şi 1A, furnizată de către portul micro-USB de pe plăcuţă. De asemenea, tot procesorul este motivul pentru care nu există supraîncalzirea echipamentului: consumul mic de putere duce la emanare de căldură puţină, chiar şi în procese complexe de procesare.

3.2.2 Modulul WIFI

Modulul *WIFI* **[7]** este utilizat în această lucrare pentru controlul *remote* , prin *Internet*, a robotului hexapod, prin intermediul microcalculatorului *Raspberry Pi*.

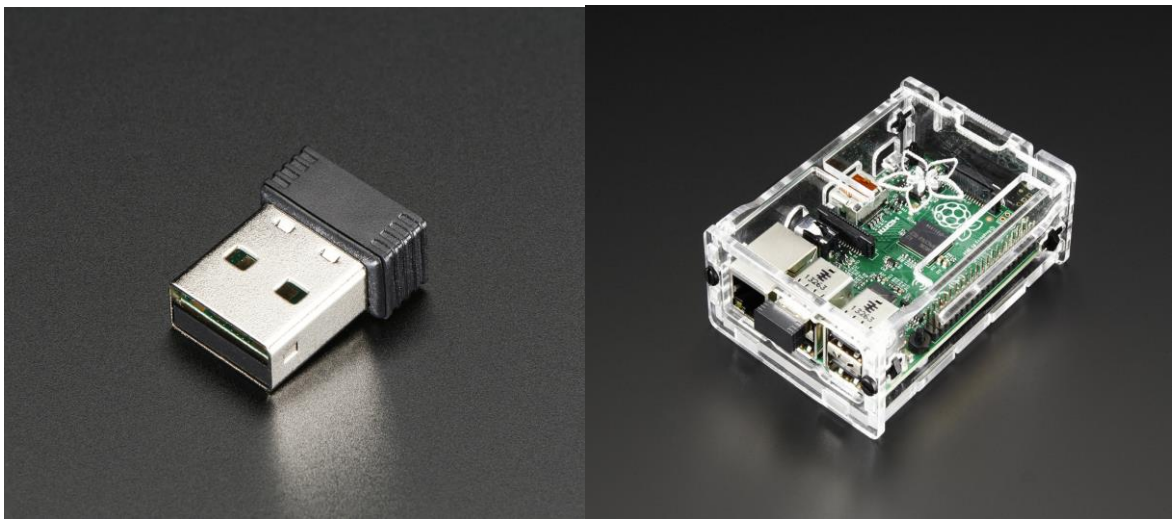


Figura 3.2 - Modulul *WiFi* utilizat de microcalculatorul *Raspberry Pi*

Sursa [7]

Figura 3.2 prezintă modului *WiFi* şi o vedere internă a acestui modul.

Din punct de vedere *hardware*, modului *WiFi* utilizat în această lucrare prezintă următoarele caracteristici:

- *Realtek RTl8192 / 8188CUS Chipset*
- Iese 8mm
- Greutate: 2,17 g
- Standardele *wireless* IEEE 802.11n:, IEEE 802.11g, IEEE 802.11b
- Interfaţa: USB 2.0

- 802.11n: până la 150Mbps (*downlink*) şi până la 150Mbps (*Uplink*), 802.11g: 54/48/36/24/18/12/9/6 Mbps, 802.11b: 11 / 5,5 / 2 / 1 Mbps
- 2.4GHz ISM (*Industrial Scientific Medical*) *Band*
- Frecvenţa RF: 2412 ~ 2462 MHz (America de Nord), 2412 ~ 2472 MHz (Europa), 2412 ~ 2484 MHz (Japonia)
- 1 ~ 14 canale
- Antena Integrată
- Putere de ieşire: 13 ~ 17 dBm (tipic)
- Securitate: WPA, WPA-PSK, WPA2, WPA2-PSK. TKIP / AES
- Ad-Hoc
- Temperatura de operare 0 ° C la 40 ° C
- Temperatura de depozitare -20°C la 75 ° C
- Umiditate de operare 10% ~ 90% (fără condens)
- Umiditatea de depozitare 5% ~ 95% (fără condens)

3.2.3 Socketul pentru cameră

Microcalculatorul *RaspBerry Pi* are un o interfață serială de tipul 2 (CSI-2) pentru modulul camerei (MIPI), care facilitează conectarea acesteia la procesorului principal Broadcom BCM2835. Acesta este un *port* ce furnizează o conexiune electrică la magistrală între două dispozitive. Aceasta este o interfață simplă, ușor de folosit și înțeles.

Scopul acestei interfețe a fost standardizarea procesului de atașare a modului camerei la procesor pentru industria de telefoane mobile. În final, aceasta a ajuns să fie folosită și pentru microcalculatorul *Raspberry Pi*.

MIPI CSI-2 versiunea 1.01 **[19]** suportă până la 4 campuri de date, fiecare dintre câmpuri cu o bandă de frecvență maximă de 1 Gbps. Mai mult decat atât, interfața utilizează numărul minim de conexiuni electrice pentru a reduce complexitatea PCB-ului. Comunicația de date este uni-direcțională, de la cameră către procesor.

Microcalculatorul *Raspberry Pi* are o interfață a camerei ZIF 15, unde un cablu de tip bandă se conectează pentru a stabili conexiunea la magistrală. Conectorul CSI este format din două interfețe mai mici. Prima interfață este folosită pentru transferul de date și semnale de ceas de la cameră către procesor într-o singură direcție. A doua interfață este formată din linii SCL/SDA, o legatură de control bidirecțională.

3.2.4 Conexiunea serială

Conexiunea serială I2C este o conexiune asincronă, ce folosește două fire, fiecare dintre acestea suportând un maximum de 1008 dispozitive de tip *slave*. I2C poate suporta un sistem *multi-master*, permițând mai multor dispozitive *master* să comunice cu toate dispozitivele de pe magistrală (dispozitivele *master* nu pot comunica între ele prin intermediul magistralei și trebuie să folosească liniile de magistrală pe rând).

Fiecare magistrală I2C este formată din două semnale: SCL și SDA, unde SCL reprezintă semnalul de ceas, iar SDA semnalul de date. Semnalul de *clock* este întotdeauna generat de către dispozitivul *master* curent de pe magistrală.

3.3 CARACTERISTICI *SOFTWARE*

O altă diferenţă majoră între sistemul *Raspberry Pi* şi un *laptop* sau un echipament de tip *desktop*, pe lângă dimensiune şi preţ, este sistemul de operare.

Sistemul de operare este cea mai importantă componentă a unui sistem, acesta rulând pe un calculator. Sistemul de operare administrează memoria, procesele, componentele *hardware* şi *software* ale PC-ului. De asemenea, un sistem de operare permite utilizatorului să interacţioneze cu un calculator fără să fie nevoit să folosească limbaj maşină.

Majoritatea calculatoarelor disponibile în zilele noastre pe piaţă funcţionează pe baza a 3 mari sisteme de operare: Microsoft Windows, Apple OS X sau Linux. Primele doua platforme sunt *closed source*, create într-un mediu rezervat, ceea ce permite utilizatorului să vadă doar rezultatul final al unei acţiuni, nu şi procesul prin care rezultatul este obţinut.

Sistemul Raspberry Pi, prin comparaţie, rulează pe sistemul de operare numit GNU/Linux - numit, mai simplu, Linux. Spre deosebire de primele doua sisteme de operare menţionate mai sus, Linux este *open-source*: este posibil să obţinem codul sursă al acestuia pentru a-l modifica după bunul plac, sau pentru a-l vizualiza pur şi simplu. Această caracteristică *open-source* a sistemului de operare Linux a permis portarea uşoară către *Raspberry Pi*, prin simple modificări. Distribuţiile de Linux folosite pentru *Raspberry Pi* sunt Debian, Fedora Remix şi Arch Linux.

Sistemul de operare utilizat pentru microcalculatorul *Raspberry Pi* se numește *Raspbian*, un sistem de operare oferit gratuit, bazat pe sistemul Debian optimizat pentru echipamentul *Raspberry Pi*. Totuși, *Raspbian* furnizează mai mult decât facilitățile unui simplu sistem de operare: acesta încapsulează peste 35000 de pachete, grupuri de software precompilat într-un format ușor de înteles, pentru o instalare rapidă și simplă.

CAPITOL 4

STREAMING VIDEO

4.1 *STREAMING* VIDEO - GENERALITĂŢI

Cu două decade în urmă, firul de telefon din casa fiecărui om reprezenta un mod esenţial de a comunica la distanţă cu familia sau prietenii. Ideea de bază nu s-a schimbat foarte mult din anii 1870, când Alexander Graham Bell (1847-1922) şi alţi câţiva oameni au devenit pionierii tehnologiei telefonice. În secolul 21 însă, oamenii au început să vadă liniile telefonice într-un mod diferit: la momentul actual există conexiuni de bandă largă la Internet, *download* de muzică, video pe *Youtube*, ştiri, noutăţi şi alte tipuri de informaţii, pe lângă apelurile telefonice, la care avem acces din propria casă 24 de ore din 24. *Streaming-ul media*, sau în traducere directă, "curgerea suportului de informaţie" a devenit o parte centrală a revoluţiei informaţiei.

Dacă ajungem să fim de acord cu ideea că o poză valorează cât o mie de cuvinte, atunci putem afirma că o imagine în mişcare valorează cât un milion de cuvinte. Poate de accea tehnologiile de *streaming* ale datelor au avut tranziţii dramatice pentru a ajunge la consumatorii cărora le sunt destinate. Pe măsură ce aceste tehnologii continuă să se dezvolte şi să crească rapid în importanţă, organizaţiile care distribuie *streaming media* trebuie să monitorizeze şi să evalueze constant curentele de pe piaţă pentru a se asigura că aleg tehnologiile care furnizează calitatea cea mai înaltă a datelor.

Tehnologii de *streaming* ca *Adobe Flash*, *Apple QuickTime*, *Microsoft Windows Media* şi *Silverlight* includ câteva componente comune în soluţiile lor. Acestea includ un *player* care să redea informaţia pe calculatorul său pe telefonul mobil al celui care vizualizează, un format definit al fişierului şi deseori o componentă de *server* care oferă trăsături precum drepturi de administrare digitală şi *streaming* în timp real.

Toate tehnologiile de redare video utilizează compresia pentru a micşora dimensiunea fişierelor audio şi video, pentru că acestea să poată fi furnizate şi rulate de către spectatori, de la distanţă, în timp real. Tehnologii de compresie video comune, cunoscute şi sub numele de *codec-uri*, includ: H.264, MPEG-4, VP6 şi VP8, WMV şi MPEG-1/2, în timp ce *codec-uri* audio comune includ: AAC, Vorbis, WMA şi MP3. *Codec-urile* sunt în general independente de tehnologia de *streaming* care le implementează.

Istoria redarii video poate fi privita din 3 puncte de vedere: unul relaţionat cu tehnologiile de *redare video*, următorul legat de viteza de conexiune şi de echipamentul ţintă şi ultimul legat de creşterea furnizorilor de servicii de *streaming*.

4.2 *CODEC*

Codec-urile reprezintă oxigenul pieţei de *media streaming*; fără *codec-uri*, *media streaming* nu ar mai exista. *Codec-urile* sunt implicate în această piaţă de la înregistrare video până la *streaming* ale fişierelor pentru transmitere. Mulţi producători de fişiere video sunt în relaţie cu piaţa DVD-ROM şi Blu-ray, precum şi cea de transmisiuni televizate.

Codec-urile sunt tehnologii de compresie ce au două componente importante: un codor pentru compresia fişierelor şi un decodor pentru decompresia fişierelor. Există *codec-uri* pentru fişiere de tip date(PKZIP), pentru imagini statice(JPEG, GIF, PNG), pentru fişiere audio(MP3, AAC) şi fişiere video(Cinepak, MPEG-2, H.264, VP8).

De asemenea, există două categorii mari de *codec-uri*: *lossless* şi *lossy* (în traducere directă: fără pierderi şi cu pierderi). *Codec-urile* fără pierderi, ca PKZIP sau PNG, reproduc perfect fişierul original după decompresie. În contrast cu acestea, *codec-urile* cu pierderi produc o replică a fişierului original după decompresie, însă nu fişierul original. *Codec-urile* cu pierderi au un mare dezavantaj: cu cât compresia fişierelor este mai amănunţită, cu atât decompresia va produce pierderea mai multor date, deci şi o pierdere a calităţii mai mare.

Tehnologiile de compresie cu pierderi de calitate a fişierelor utilizează două tipuri de compresii: inter-cadru şi intra-cadru. Compresia de tip intra-cadru este la bază compresie de imagine statică aplicată unui fişier video, fiecare cadru fiind comprimat fără o referinţă către oricare alt cadru. De exemplu, *Motion-JPEG* utilizează doar compresie de tip intra-cadru, codând fiecare cadru ca o imagine JPEG separată. *Codec-ul* DV utilizează de asemenea doar compresie intra-cadru, la fel ca DVCPRO-HD, care în esenţă, divizează fiecare cadru HD în 4 blocuri SD DV, toate codate numai prin compresie intra-cadru.

Pe de altă parte, compresia inter-cadru utilizează redundanţe între cadre pentru compresia video. Compresia inter-cadru este mult mai eficientă decât cea intra-cadru, de aceea majoritatea *codec-urilor* sunt optimizate pentru a căuta şi a echilibra informaţiile redundante dintre cadre.

La început, *codec-uri* bazate pe CD-ROM ca *Cinepak* şi *Indeo* utilizau două tipuri de cadre pentru această operaţie: cadre *key* şi cadre *delta*. Cadrele *key* stocau cadrul întreg şi erau comprimate doar cu compresie intra-cadru. În timpul codării, *pixelii* din cadrele *delta* erau

comparaţi cu *pixelii* din cadre anterioare, iar informaţia redundantă era înlăturată. Datele rămase în fiecare cadru "delta" erau de asemenea comprimate utilizând tehnici intra-cadru, necesare pentru a satisface rata de date destinată fişierului.

4.2.1 Formate lungi GOP

Încă din timpul zilelor CD-ROM, tehnicile inter-cadru au avansat şi majoritatea *codec-urilor*, incluzând MPEG-2, H.264 şi VC-1, folosesc acum trei tipuri de cadre pentru compresie: cadre I, cadre B şi cadre P, ca în Figura 4.1. Cadrele I sunt la fel ca cele de bază (cadre *key*) şi sunt comprimate doar cu tehnici intra-cadru, făcându-le mai late şi mai puţin eficiente.

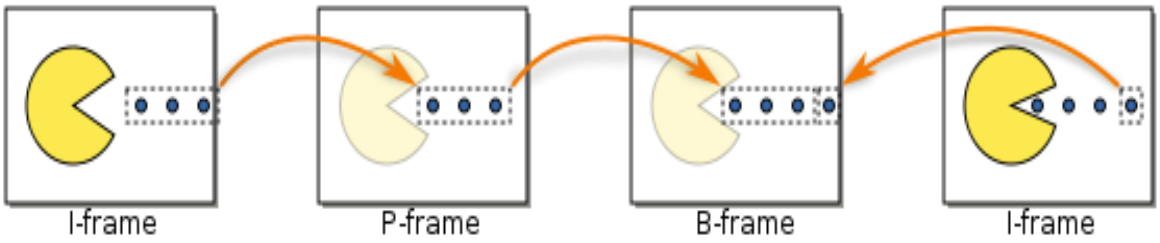


Figura 4.1 - Exemple de cadre I, P, B din cadrul formatului GOP

Sursa [8]

Cadrele B şi P sunt cadre de tip *delta*. Cadrele de tip P sunt cele mai simple şi pot utiliza informaţii redundante în orice cadru anterior I sau P. Cadrele B sunt mai complexe şi pot utiliza informaţii redundante în orice cadru I, B sau P anterior sau consecutiv. Acest lucru face cadrele B cele mai eficiente dintre cele 3 tipuri de cadre.

Aceste tipuri de cadre multiple sunt stocate într-un grup de imagine (GOP), fiecare începând cu un cadru I, incluzând toate cadrele până la următorul cadru I (acesta din urmă nu este luat în considerare). *Codec-urile* ce folosesc toate cele 3 tipuri de cadre se numesc deseori "formate GOP lungi", în special atunci când acestea sunt utilizate în sisteme non-liniare de editare. Acest lucru pune în evidenţă al doilea compromis fundamental al tehnologiei de compresie cu pierderi: calitatea pentru complexitatea decodării. În alte cuvinte, cu cât un *codec* oferă o calitate mai bună, cu atât acea informaţie este mai greu de decodificat, în particular în cazul aplicaţiilor interactive, că editarea de fişiere video.

Primul format lung GOP utilizat în sisteme non-liniare de editare a fost HDV, un format bazat pe MPEG-2, ce introduce o complexitate ridicată. De exemplu, cu DV şi *Motion-JPEG*, fiecare cadru era referenţiat separat, astfel că puteai muta locul iniţial de editare în orice cadru al fişierului video, iar decompresia se făcea în timp real începând cu acel cadru.

Totuşi, în cazul HDV, dacă mutai locul de editare al fişierului video într-un cadru B, editorul non-liniar trebuia să facă decompresia tuturor cadrelor referenţiate de către acel cadru B, iar acele cadre pot fi localizate fie înainte, fie după cadrul B respectiv. Pentru calculatoarele cu performanţele acelor vremuri, majoritatea funcţionând cu sisteme de operare pe 32 de biţi ce puteau adresa maxim 2 GB de memorie, formatele lungi GOP cauzau latenţă semnificativă, ceea ce făcea editarea foarte înceată.

Pe măsură ce camerele video portabile au început să se bazeze pe formate GOP că MPEG-2 şi H.264 pentru a stoca datele video, a apărut un nou tip de *codec*, numit adesea şi *codec* intermediar. Aceste *codec-uri* folosesc doar tehnici de compresie intra-cadru pentru maximizarea răspunsului la editare şi rata de date foarte ridicată pentru conservarea calităţii.

4.3 CODAREA H.264

Standardul actual de compresie video H.264(cunoscut şi sub numele de MPEG-4 Partea 10/AVC pentru codare video avansată) va deveni în scurt timp prima alegere în domeniul *streaming-ului* video.

H.246 **[9]** este un standard licenţiat şi deschis către dezvoltare, ce suportă cele mai eficiente tehnici de compresie video disponibile în zilele contemporane. Un codor H.264 poate reduce dimensiunea unui fişier digital video cu până la 80%, fără a compromite calitatea imaginii, în comparaţie cu formatul Motion JPEG şi cu 50% mai mult faţă de standardul MPEG-4 Part 2. Această caracteristică pune în evidenţă necesitatea unei benzi de reţea mai scăzute şi a unui spaţiu de stocare mai mic. Altfel spus, se poate obţine o calitate mai bună video la o rată de bit dată.

Organizaţiile de standardizare din domeniul telecomunicaţiilor şi din industria IT afirmă că H.264 este considerat a fi adoptat la scară mai largă decât standardele anterioare. H.264 a fost deja introdus în noile dispozitive electronice precum telefoane mobile de tip smartphone şi în playere video digitale, fiind rapid acceptat de către utilizatori. Furnizorii de servicii ca stocare online de fişiere video şi companiile de telecomunicaţii au început de asemenea să adopte acest codor H.264.

În industria de supraveghere video H.264 va fi cel mai probabil utilizat în aplicaţii ce necesită rate mari ale cadrelor şi rezoluţii înalte, precum supravegherea autostrăzilor, a aeroporturilor şi a cazinourilor, unde norma acceptat este de 30/25(NTSC/PAL) cadre pe secundă. Se poate afirma că acesta este domeniul în care se vor observa reduceri drastice ale utilizării benzii reţelei şi a spaţiului de stocare de date.

Nu în ultimul rând, utilizarea codării H.264 va necesita şi adoptarea de camere performante, cu un număr al *megapixelilor* foarte ridicat, din moment ce tehnologia de compresie prezintă o eficienţă ridicată în reducerea dimensiunii fişierelor mari şi a ratei de biţi generată, fără a compromite calitatea imaginii. Pentru ca balanţa să fie echilibrată, există totuşi şi câteva dezavantaje. Odată cu economisirea benzii de reţea şi a costurilor de stocare, va fi nevoie şi de camere şi staţii de monitorizare performanţe.

4.3.1 Dezvoltarea codării H.264

H.264 este rezultatul proiectului cooperativ dintre Grupul de Experţi în Codare Video din cadrul ITU-T şi Grupul de Experţi în Imagini în Mişcare din cadrul ISO/IEC(MPEG). ITU-T este sectorul care coordonează standardele de telecomunicaţii în interesul Uniunii Internaţionale de Telecomunicaţii. ISO/IEC se ocupă de controlul tuturor standardelor pentru tehnologii electrice şi electronice. H.264 este numele folosit de către ITU-T, în timp ce ISO/IEC au numit acest standard MPEG-4 Part 10/AVC, prezentat ca o nouă parte din suita MPEG-4. Suită MPEG-4 include, de exemplu MPEG-4 Part 2, un standard ce a fost utilizat pentru codoare video bazate pe IP şi pe camere de reţea.

Creat pentru a adresa diverse slăbiciuni din standardele de compresie video precedente, H.264 suportă următoarele caracteristici:

- implementări ce furnizează o reducere a ratei de bit cu aproximativ 50%, având o calitate video fixă în comparaţie cu alte standarde video
- robusteţe a erorii, astfel că erorile de transmisie de-a lungul diverselor reţele să fie tolerate
- capabilităţi de latenţă scăzută şi o calitate ridicată pentru latență mai mare
- specificaţii de sintaxă clare, lucru ce simplifică implementările
- decodare perfect asemănătoare, ce defineşte cu exactitate cum trebuie făcute calculele numerice de un codor şi de un decodor, astfel că erorile cumulative să fie evitate

Codarea H.264 are de asemenea flexibilitatea de a suporta o variatate largă de aplicaţii ce necesită rate de biţi foarte diferite. De exemplu, în aplicaţiile video de divertisment - care includ transmisiuni radio, transmisiuni satelitare, transmisiuni prin cablu şi DVD - codarea H.264 va putea livra o performanţă încadrată între 1 şi 10Mbit/s cu o latenţă înaltă, pe când în serviciile de telecomunicaţii prezintă rate ale biţilor sub 1Mbit/s pentru latente scăzute.

4.3.2 Modul de funcţionare a compresiei video

Compresia video se referă la reducerea şi înlăturarea datelor video redunante astfel ca fişierele video digitale să poată fi stocate şi transmise eficient. Procesul implică aplicarea unui algoritm asupra sursei video pentru a crea un fişier comprimat, pregătit pentru stocare sau transmisie. Pentru a derula fişierul comprimat, un algoritm invers celui de compresie este aplicat, pentru a produce un fişier video ce prezintă practic acelaşi conţinut ca sursă video originală. Timpul în care are loc compresia, transmisia, deocompresia şi expunerea fişierului video se numeşte latență. Cu cât algoritmul de compresie este mai avansat, cu atât latență creşte, pentru aceeaşi putere de procesare.

O pereche de algoritmi ce funcţionează împreună pentru a realiza acest proces se numeşte *codec* video (codor/decodor). *Codec-urile* video ce implementează diferite standarde nu sunt de obicei compatibile unul cu celălalt; în alte cuvinte, un conţinut video comprimat cu un anumit standard nu poate fi decomprimat utilizând alt standard. De exemplu, un decodor MPEG-4 Part 2 nu poate funcţiona împreună cu un codor H.264. Acest lucru se datorează faptului că un algoritm nu poate decoda corect rezultatul unui alt algoritm, însă este totuşi posibil să se implementeze diferiți algoritmi în aceeaşi aplicaţie *software* sau *hardware*, pentru a face posibilă compresia mai multor tipuri de formate video.

Aceste standarde diferite de compresie video utilizează diverse metode de reducere a datelor şi astfel, rezultatele diferă atât în calitate, latență, cât şi în rata biţilor.

De asemenea, chiar şi rezultatele de la diferite codoare care folosesc acelaşi standard pot varia, deoarece *designer-ul* codorului poate alege să implementeze seturi de utilitare diferite din cadrul unui standard. Atâta timp cât rezultatul unui codor se mulează pe formatul unui standard şi pe cerinţele unui anumit tip de decodor, este posibil să se creeze diferite implementări. Acest lucru este avantajos deoarece implementări diferite pot avea ţeluri şi bugete diferite. Codoarele *software* care nu funcţionează în timp real, utilizate pentru date optice, au avantajul de a livra fişiere video codate la o calitate mult mai bună decât codoarele *hardware* utilizate în aplicaţii de timp real, precum conferinţele video (posibilitate oferită de dispozitive electronice, ca

laptop-uri, *smartphone-uri*, tablete etc.). De aceea, un anumit standard nu poate garanta o rată de bit sau o calitate fixă. Nu în ultimul rând, performanţa unui standard nu poate fi comparată cu uşurinţă cu cea a altor standarde, sau chiar cu alte implementări ale aceluiaşi standard, fără a fi definit în prealabil modul de implementare.

Spre deosebire de un codor, decodorul trebuie să implementeze toate elementele unui standard pentru a putea decoda şiruri maleabile de biţi. Acest lucru se datorează faptului că standardul specifică exact cum decompresia ar trebui să refacă fiecare bit din fişierului video comprimat.

Graficul din Figura 4.2 prezintă o comparaţie a ratelor de biţi, pentru un anumit nivel al calităţii imaginii pentru următoarele standarde video: Motion JPEG, MPEG-4 Part 2(fără compensare a mişcării), MPEG-4 Part 2(cu compensare a mişcării) şi H.264(profilul de referinţă).

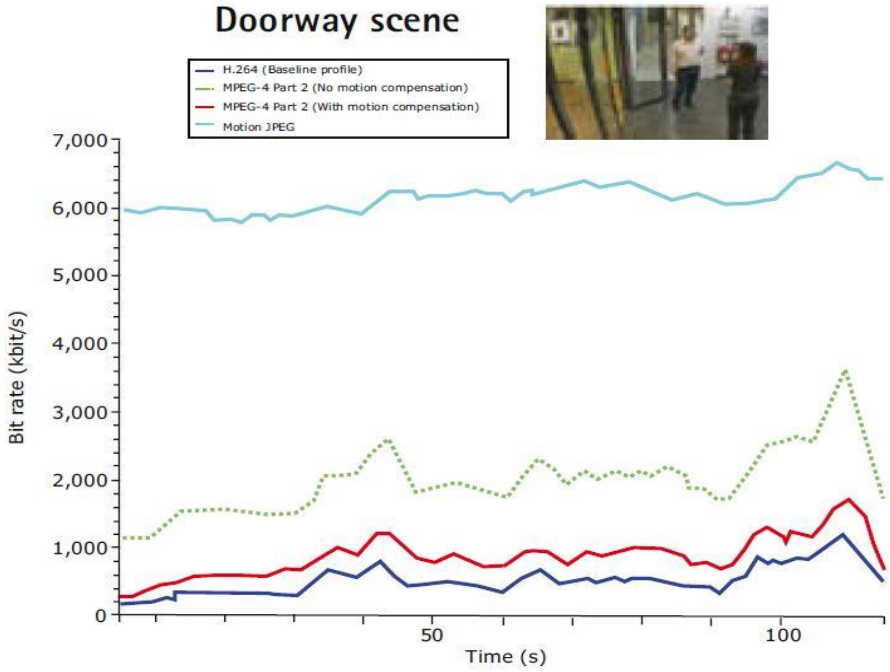


Figura 4.2 - Comparaţie a ratelor de biţi pentru un anumit nivel de calitate al imaginii
Sursa [9]

Figura scoate în evidenţă faptul că un codor H.264 generează cu până la 50% mai puţini biţi pe secundă pentru a secvenţă video decât în cazul unui codor MPEG-4 cu compensarea mişcării. Codorul H.264 este de cel puţin 3 ori mai eficient decât un codor MPEG-4 fără compesarea mişcării şi de cel puţin 6 ori mai eficient decât un codor Motion JPEG.

4.3.3 Profile şi Nivele H.264

Grupul codor-decodor ce stă la baza definirii codării H.264 s-a concentrat pe crearea unei soluţii simple şi clare, limitând opţiunile şi caracteristicile la minimum. Un aspect important al standardului, la fel ca şi în cazul altor standarde, furnizează capabilităţile în profile (seturi de caracteristici algoritmice) şi în nivele (clase de performanţă) ce suportă în mod optimal produse populare şi formate comune.

Codarea H.264 are şapte profile, fiecare concentrându-se pe o clasă specifică de aplicaţii. Fiecare profil defineşte ce set de caracteristici poate folosi codorul şi limitează complexitatea implementării decodorului.

Camerele de reţea şi codoarele video folosesc de cele mai multe ori profilul de referinţă, care este destinat în principal aplicaţiilor cu resurse computaţionale restrânse. Profilul de referinţă este cel mai potrivit din punctul de vedere al performanţelor codoarelor în timp real, incluse în camere video de reţea. Acest profil permite şi prezenţa unei latenţe scăzute, o cerinţă importantă în supravegherea video şi în particular, pentru controlul PTZ în timp real al camerelor de reţea.

Codarea H.264 are 11 nivele de capabilitate pentru a limita performanţa, banda de frecvenţă şi cerinţele de memorie. Fiecare nivel defineşte rata de bit şi rata de codare în macro-blocuri pe secundă, pentru rezoluţii de la QCIF până la HDTV şi mai departe. Cu cât rezoluţia este mai mare, cu atât nivelul H.264 cerut este mai înalt.

În funcţie de profilul H.264, codorul poate utiliza diferite tipuri de cadre precum cadrul I, cadrul P şi cadrul B.

Cadrul I, sau intra-cadru, este un cadru autonom ce poate fi decodificat indepent, fără nicio referinţă la o altă imagine. Prima imagine dintr-o secvenţă video este întotdeauna un cadru I. **Cadrele I** sunt necesare ca puncte de start pentru noii observatori (puncte de resincronizare), dacă şirul de biţi transmişi este deteriorat. **Cadrele I** pot fi utilizate pentru a implementa funcţii ca rulare a fişierului video, derulare sau alte funcţii de acces aleator. Un codor va insera automat **cadrele I** la intervale regulate sau la cerere, în cazul în care sunt aşteptaţi clienţi noi la vizualizarea unui flux video. Dezavantajul cadrelor I este consumul mare de biţi, însă pe de altă parte, acestea nu generează multe artefacte.

Cadrul P (inter-cadru predictiv) face referire la părţi dintr-un cadru I/cadru P utilizat anterior. **Cadrele P** necesită de cele mai multe ori mai puţini biţi decât cadrele I, dar un dezavantaj este sensibilitatea acestora la erori de transmisie din cauza dependenţei complexe faţă de cadrele anteriore de referinţă P sau I.

Cadrul B (inter-cadru dublu predictiv) este un cadru care face referire atât la un cadru de referinţă anterior, cât şi la un cadru viitor.

Figura 4.3 prezintă o secvenţă tipică, formată din cadre I, B şi P. Un **cadru P** poate face referire doar la un cadru I sau P anterior, pe când un cadru B poate face referire atât la cadre I, cât şi cadre B precedente sau viitoare.

În momentul în care un decodor video reface un fişier video prin decodarea de fluxuri de biţi cadru după cadru, decodarea trebuie să înceapă întotdeauna cu un cadru I. Dacă sunt utilizate, cadrele B şi cadrele P trebuie decodate în relaţie cu cele de referinţă.

În profilul de referinţă H.264, doar cadrele I şi P sunt utilizate. Acest profil este ideal pentru camerele de reţea şi pentru codoarele video, având în vedere că latenţă scăzută este obţinută deoarece cadrele B nu sunt utilizate.

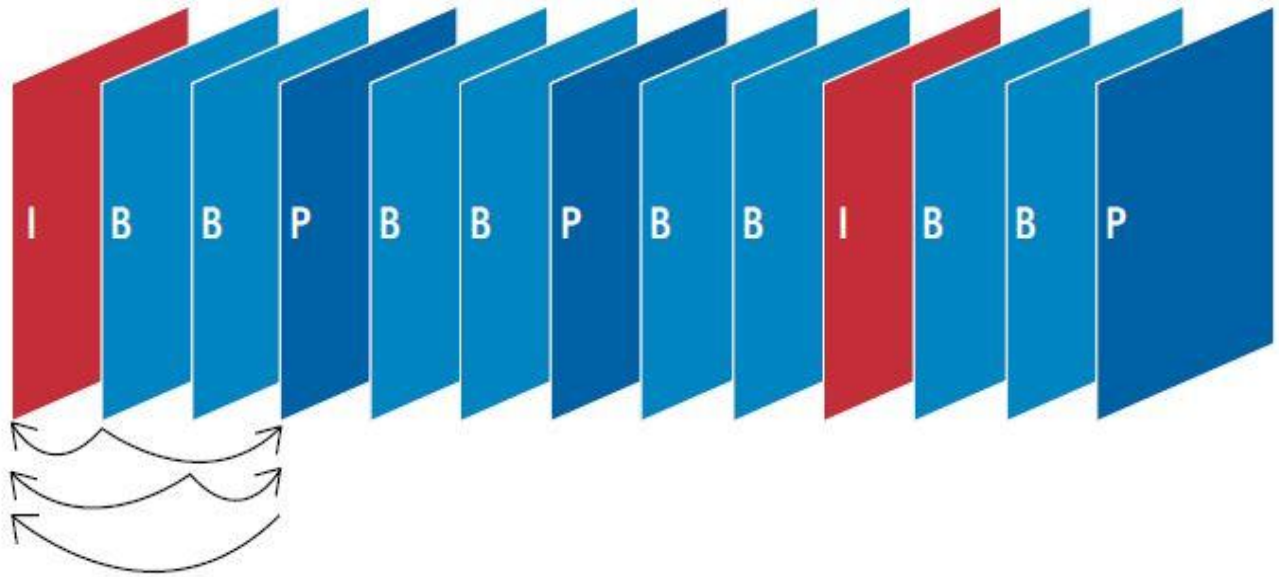


Figura 4.3 - Secvenţă tipică de cadre I, B, P
Sursa [9]

4.3.4 Metode de bază pentru reducerea datelor

Pentru reducerea datelor video se pot folosi o varietate de metode, atât pentru cadrul unei imagini cât şi pentru o serie de cadre.

În cazul cadrului unei imagini, datele pot fi reduse simplu prin eliminarea informaţiilor redundante, lucru ce ar avea un impact asupra rezoluției imaginii.

În cazul unei serii de cadre, datele video pot fi reduse prin metode precum codarea cu diferenţă, care este utilizată de majoritatea standardelor de compresie video inclusiv H.264. În codarea cu diferenţa, un cadru este comparat cu un cadru de referinţă (de exemplu, un cadru I/P anterior) şi doar *pixelii* schimbaţi faţă de cadrul referinţă sunt codaţi. Astfel, numărul de valori ale *pixelilor* ce sunt codaţi şi trimişi este redus.

Pentru formatul *Motion JPEG*, cele 3 imagini din secvenţă prezentată în Figură 4.4 sunt codate şi trimise ca imagini unice separate (cadre I), fără nicio dependenţă una faţă de cealaltă.

La codarea cu diferenţă, doar prima imagine (cadru I) este codată complet. În următoarele două imagini (cadre P), referinţele sunt făcute la prima imagine pentru elementele statice(ex. casă şi părţile care se mişcă, omul care aleargă este codat utilizând vectori de mişcare, astfel reducând cantitatea de informaţie trimisă şi stocată).

Cantitatea de informaţie codată poate redusă mai departe dacă detecţia şi codarea de diferenţe este bazată pe blocuri de *pixeli* (macroblocuri), în locul *pixelilor* individuali; astfel, zone mai mari sunt comparate şi doar blocurile care conţin diferenţe semnificative sunt codate.

Codarea de diferenţă, totuşi, nu ar reduce datele semnificativ dacă ar exista multă mişcare într-un video. În acest caz poate fi folosită compensarea blocurilor de *pixeli* bazate pe mişcare. Această compensare ia în considerare cât de multă informaţie dintr-un nou cadru al unei secvenţe video poate fi găsită într-un cadru anterior, dar posibil într-o altă locaţie. Această tehnică divide un cadru într-o srie de macroblocuri. Bloc cu bloc, un nou cadru - de exemplu, un

cadru P, poate fi compus şi "prevăzut" prin căutarea de blocuri asemănătoare într-un cadru de referinţă. Dacă este găsită o potrivire, codorul codează pur si simplu poziţia unde blocul potrivit va fi găsit în cadrul de referinţă. Prin codarea vectorului de mişcare, după cum se numeşte, se iau în considerare mai puţini biţi faţă de conţinutul actual al unui bloc întreg.

Figura 4.4 - Reducerea numărului de pixeli codaţi pentru 3 cadre succesive

Sursa [9]

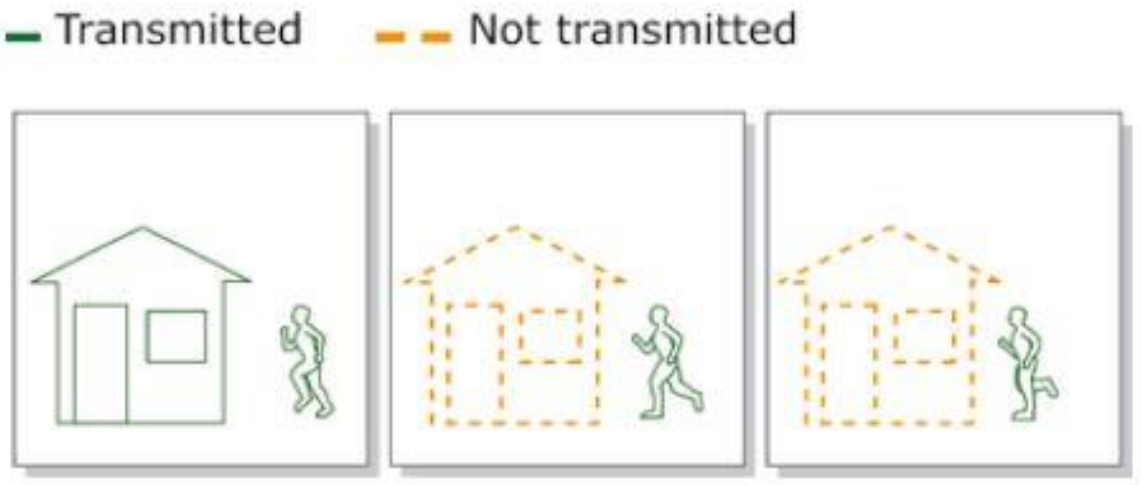


Figura 4.5 - Pixeli transmişi şi netransmişi pentru 3 cadre succesive

Sursa [9]

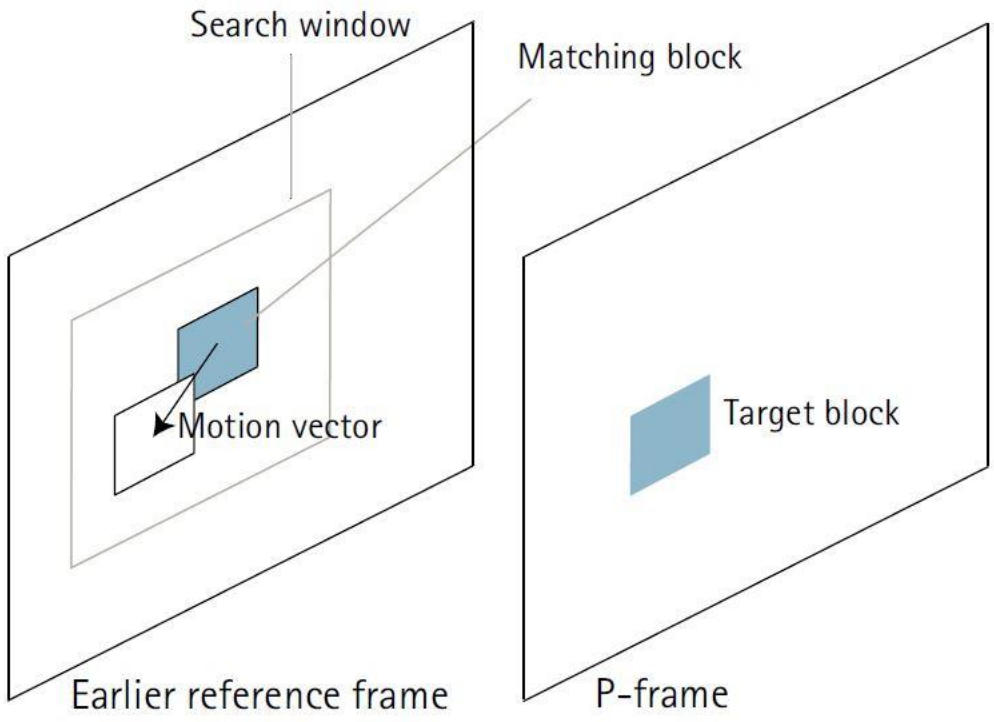


Figura 4.6 – Codarea cu diferenţă

Sursa [9]

4.3.5 Eficienţa codării H.264

H.264 duce tehnologia compresiei video la un nivel înalt. Cu ajutorul acestei codări, o nouă schemă de predicţie este introdusă pentru codarea cadrelor I. Această schemă poate reduce destul de mult numărul biţilor unui cadru I şi poate menţine o calitate înaltă prin activarea predicţiei succesive a blocurilor mici de pixeli din cadrul macroblocului unui cadru. Acest lucru este obţinut prin încercarea de a găsi *pixeli* potriviţi printre *pixelii* codaţi anterior. Prin refolosirea valorilor *pixelilor* ce au fost codaţi în prealabil, dimensiunea de bit poate fi redusă drastic. Nouă intra-predicţie este o parte cheie a tehnologiei H.264 ce s-a dovedit a fi foarte eficientă. Pentru o simplă comparaţie, dacă s-ar folosi doar cadre I într-un flux H.264, fişierul video ar avea o dimensiune mult mai mică decât în cazul unu flux Motion JPEG, care foloseşte doar cadre I.

Figura 4.7 prezintă 3 ilustraţii ale modurilor în care se poate face intra-predictia în codarea unor pixeli de tip 4X4 în cadrul a 16 blocuri conţinute de un macrobloc. Fiecare dintre cele 16 blocuri dintr-un macrobloc pot fi codate folosind moduri diferite.

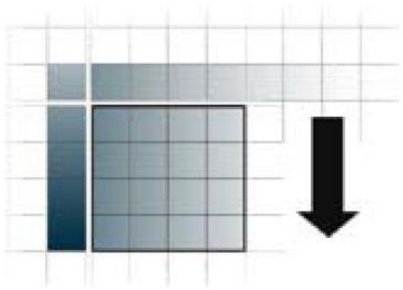
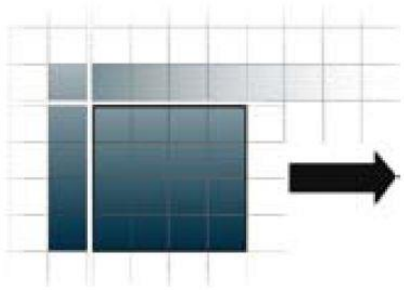
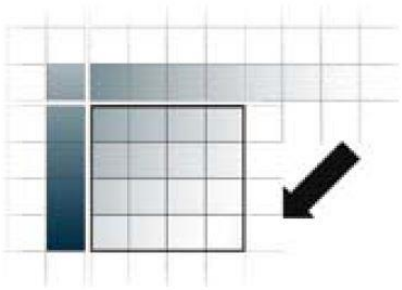
In this mode, four bottom pixels from the block above are copied vertically into part of an intra-coded macro-block.	In this mode, four right-most pixels from the block to the left are copied horizontally into part of an intra-coded macroblock.	In this mode, eight bottom pixels from the blocks above are copied diagonally into part of an intra-coded macro-block.
		

Figura 4.7 - Moduri în care se poate realiza intra-predictia

Sursa [9]

4.4 FORMATUL MP4

MP4 sau MPEG-4 Partea 12 reprezintă un format de *container* digital *multimedia*, utilizat deseori pentru a stoca fişiere video sau audio, dar în acelaşi timp poate stoca şi alte tipuri de date precum subtitrări şi imagini statice. Asemenea altor tipuri de formate de *container*, acesta permite redarea video via Internet. Extensia unui astfel de fişier este de cele mai multe ori **.mp4**, dar poate avea şi alte extensii precum **.m4a** şi **.m4p.** Extensia **.m4a** este folosită în cazul fişierelor exclusiv audio.

Existenţa a două formate diferite, unul exclusiv audio şi unul video şi audio, a creat multe controverse printre utilizatori şi dezvoltatori de *software multimedia*. În cazul unor *manageri* de fişiere, precum *Windows Explorer*, se verifică înaintea deschiderii fişierului dacă acesta este pur audio sau video combinat cu audio, asociindu-i extensia corespunzătoare.

Majoritatea tipurilor de date pot fi ataşate unui fişier MPEG-4 Partea 14 prin intermediul fluxurilor private. Pentru fiecare format în parte sunt definite *codec-urile* corespunzătoare. Astfel că pentru fişiere video în format MPEG-4 Partea 14, *codec-ul* potrivit este cel H.264, prezentat în **subcapitolul 4.3**.

CAPITOL 5

SUPORT *SOFTWARE* DE CONTROL ŞI AUTOMATIZARE A ROBOTULUI

5.1 SHELL SCRIPTING

Sistemul de operare UNIX reprezintă un set de programe care se comportă ca o legătură între un PC şi utilizator. Programele PC-ului care alocă resursele sistemului şi coordonează toate detallile interne ale calculatorului formează sistemul de operare sau *kernel.* Utilizatorii relaţionează cu *kernel-ul* prin intermediul unui program numit *shell. Shell-ul* este un interpretor de comenzi, traducând comenzile introduse de utilizator şi convertindu-le într-un limbaj înţeles de către *kernel.*

Unix a fost iniţial dezvoltat în anul 1969, de către un grup de angajaţi AT&T de la *Bell Labs*. La momentul actual pe piaţă există o întreagă varietate de versiuni Unix, printre care se numără *Solaris Unix*, AIX, HP Unix şi BSD. Linux este de asemenea o variantă de Unix, făcută publică pentru utilizatori în mod gratuit. Un calculator cu sistemul de operare Unix poate fi utilizat de mai mulţi oameni, ceea ce dovedeşte faptul că acest sistem de operare este unul multi-utilizator. De asemenea, un utilizator poate rula multiple programe în acelaşi timp, de accea Unix mai este numit şi multifuncţional.

Arhitectura sistemului UNIX se bazează pe patru concepte de bază care unifică versiunile UNIX:

- ***kernel*** : aceste reprezintă centrul sistemului de operare. Acesta interacţionează cu partea *hardware* şi majoritatea *task-urilor*, precum *managementul* memoriei, planificarea de sarcini şi managementul fişierelor.
- ***shell***: acesta reprezintă o utilitate ce procesează cerinţele utilizatorului. În momentul în care acesta tastează o comandă în terminal, *shell-ul* interpretează comanda şi apelează programul dorit. *Shell-ul* utilizează sintaxe standard pentru toate comenzile. Cele mai cunoscute utilitare *shell* ce sunt disponible cu majoritatea variantelor Unix sunt *C Shell*, *Bourne Shell* şi *Korn Shell*.

- **comenzi şi utilitare**: există diverse comenzi şi utilitare pe care le poţi folosi în activităţile de zi cu zi. Câteva exemple ar fi **cp**, **mv** şi **grep.** Există peste 250 comenzi standard pe lângă altele furnizate de programe *software* asociate Unix. Fiecare dintre aceste comenzi vin împreună cu numeroase opţiuni, care le fac versatile şi utilizabile în multe aplicaţii.
- **fişiere şi directoare** : Toate datele din cadrul UNIX sunt organizate sub forma fişierelor. Toate fişierele sunt organizate în directoare. Aceste directoare sunt organizate în structuri de arbore, numite fişiere de sistem.

Comenzile elementare ale sistemului UNIX sunt prezentate în Tabelul 5.1:

Comandă	Semnificaţie
ls	Listing simplu al fişierelor
ll	Listing al fişierelor, cu anumite detalii
cp	Copiere fişiere
mv	Mutare/redenumire fişiere
rm	Ştergere fişiere
cd	Schimbare director de lucru
pwd	Printare cale curentă a directorului
cât	Listare fişier/fişiere secvenţial
sudo	Rularea unei comenzi cu drept de root
mkdir	Creare director
rmdir	Ştergere director

Tabelul 5.1 - Comenzi elementare în terminalul Linux

Uneori, utilizatorul doreşte să lucreze în linia de comandă cu permisiuni standard, pentru a se asigura că nu distruge fişiere de sistem sau fişiere care aparţin altor utilizatori. Totuşi, există multe situaţii în care se doreşte să se copieze un fişier într-un director de sistem (precum /usr/local/bin) pentru a-l face disponibil altor utilizatori ai sistemului. Doar administratorul de sistem (cunoscut ca utilizatorul *root*) ar trebuie să aibă permisiunea de a modifica orice conţinut al directoarelor de sistem. De accea încercarea de copiere a unui fişier(precum un program descărcat online) în acel director este interzis în mod normal. Deoarece ar fi costisitor ca un utilizator să se logheze întotdeauna că root pentru a realiza cerinţe pe partea administrativă, se poate folosi comanda **sudo** pentru a executa o singură comandă ca utilizator **root.** La executarea unei comenzi precedată de cuvântul cheie **sudo**, utilizatorului i se va cere parolă de **root**.

În cadrul acestei lucrări, noţiunea de *shell scripting* a fost utilizată pentru controlul modulului de cameră din cadrul microcalculatorului *Raspberry Pi*. În acest scop am utilizat utilitarul **raspivid**, disponibil pe Rpi odată cu instalarea modulului de cameră şi a sistemului de operare. Câteva dintre comenzile din linia de comandă utilizate pentru a pregăti procesul de capturare video şi pentru a realiza practic aceasta sunt următoarele:

- verificarea spaţiuliui de stocare pentru fişierul/fişierele video, **df -h,** care produce un rezultat asemănător cu cel de mai jos:

```
pi@raspberrypi ~ $ df -h
Filesystem      Size  Used Avail Use% Mounted on
rootfs          7.3G  3.4G  3.6G  49% /
/dev/root       7.3G  3.4G  3.6G  49% /
devtmpfs        180M     0  180M   0% /dev
tmpfs            38M  228K   38M   1% /run
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs            75M     0   75M   0% /run/shm
/dev/mmcblk0p1   56M   19M   38M  33% /boot
```

Rezultatul comenzii ne arată faptul că avem 3GB de memorie liberi, pentru a captura un fişier video de 30 min în format HD, cu aproximativ 115MB pe minut.

- Capturarea unui fişier video de 5 secunde poate fi realizată cu următoarea comandă:

  raspivid -o myvid.h264

Dacă se doreşte capturarea unui fişier de o durată mai lungă, se poate preciza acest lucru prin adăugarea opţiunii **-t,** însoţită de durata acestuia în milisecunde (în exemplul de mai jos se doreşte o durată de 60 de secunde)

raspivid -o myvid.h264 -t 60000

- Precizarea unei anumite rezoluţii video se poate face astfel:

  raspivid -o myvid.h264 -w 1280 -h 720

Ca rezultat, vom obţine un fişier video cu rezoluţia 1280x720. Standardul în rezoluţie al capturilor video utilizând acest modul Rpi este de 1920x1080, astfel că în lipsa celor 2 opţiuni de la finalul comenzii, rezoluţia va fi cea standard.

- Alegerea unei rate de bit se face astfel:

  raspivid -0 myvid.h264 -w 1280 -h 720 -b 8000000

Rata de bit specificată este de 8000Kbs sau 8Mb. Rata de bit standard este de 17000000. Nu există o rată de bit considerată corectă sau incorectă, ea fiind aleasă încercând diverse variante şi observând care este cea potrivită aplicaţiei în cauză.

- Convertirea unui fişier video din format brut H.264 în format MP4 este de cele mai multe ori necesară deoarece multe programe de rulare a fişierelor video nu suportă decât formate împachetate într-un format de *container*. Pentru a realiza această împachetare se

poate folosi librăria MP4Box, foarte uşor de utilizat. Pentru instalare se folosesc următoarele comenzi:

sudo apt-get update
sudo apt-get install -y gpac

5.2 CONEXIUNE *SSH*

Accesarea unui *shell* (sau a unei linii de comenzi) se poate face prin diferite metode pe majoritatea sistemelor Linux/Unix. Una dintre cele mai consacrate modalităţi este programul *telnet*, care este disponibil pe majoritatea sistemelor de operare ce suportă conexiunea la reţea. Accesarea unui cont *shell* prin intermediul unei metode *telnet* supune totuşi utilizatorul la un risc în ceea ce priveşte trimiterea sau primirea informaţiilor în sesiunea de *telnet*, deoarece acestea sunt vizibile (în *plain-text*) în reţeaua locală şi în reţeaua locală a maşinii la care acesta se conectează. Astfel, oricine poate interveni pentru a vizualiza conexiunea dintre cele două maşini, putând afla numele de utilizator, parola, *e-mail-ul* şi comenzile rulate. Din cauza acestor motive a apărut nevoia de creare a unui program mai sofisticat decât *telnet* pentru că un utilizator să se poată conecta la un *host* din depărtare.

SSH, acronim pentru *Secure Shell*, a fost creat şi implementat pentru a furniza o securitate ridicată la accesarea unui alt PC din depărtare. Pe lângă faptul că acest utilitar criptează sesiunea, acesta furnizează şi facilităţi mai bune de autentificare, precum şi transfer de fişiere securizat, *port forwading, forwarding* de sesiuni şi altele, pentru a se creşte securitatea acestor protocoale. Se pot folosi diverse forme de criptare, începând de la criptarea pe 512 biţi până la 32768 biţi; de asemenea, se pot folosi chei de criptare că AES, DES triplu, *Blowfish*, CAST128 sau *Arcfour*. Desigur, cu cât numărul biţilor de criptare creşte, cu atât timpul de generare şi utilizare a cheilor va creşte, pe lângă timpul de transmisie a datelor prin această conexiune.

La început, pentru conectarea cu un PC de la depărtare, va trebui rulată comandă **ssh hostname** pe maşină locală. *Hostname-ul* furnizat ca argument al acestei comenzi este cel al maşinii din depărtare. Mai departe, va trebui furnizată parola sistemului la care utilizatorul doreşte să se conecteze. Dacă numele de utilizator dorit există, iar parola tastata este corectă, atunci sistemul îi va permite utilizatorului să aibă acces la PC-ul din depărtare.

O metodă pentru a evita utilizarea unei parole pentru autentificarea pe o maşină din depărtare este utilizarea autentificării prin cheie publică. Aceasta este considerată a fi mai sigură împotriva atacurilor de tip **brute-force**. De asemnea, autentificarea prin cheie publică permite automatizarea rutinei de logarea între maşini. Poate să simplifice şi procesul de logare, fără a compromite securitatea parolei.

Autentificarea prin cheie publică utilizează o pereche de chei generate - una publică şi una privată - pentru autentificarea între un client şi un *host*. Cheia publică este derivată din cheia privată. La autentificare, maşina gazdă compara cheia publică cu cea privată pentru a verifica veracitatea cheii publice. Dacă acestea se potrivesc, accesul este garantat. Securitatea sistemului depinde de securitatea cheii private.

Paşii care trebuie urmaţi pentru a realiza această procedură sunt: generarea cheilor privată şi publică necesară, pe maşină gazdă, transferarea/adăugarea cheii publice în fişierul **authorized_keys** de pe client şi logarea prin intermediul autentificării cu cheie publică.

5.3 CONEXIUNE *TCP*

TCP reprezintă un standard ce definește stabilirea și menținerea conversațiilor în rețea prin intermediul programelor de aplicație care interschimbă date. TCP funcționează prin intermediul Protocolului Internet(IP), ce definește modul în care PC-urile transmit pachete de date de la unul la celălalt. Împreună, TCP și IP formează regulile de bază pentru definirea Internetului. TCP este definit de către IETF în standardele RFC.

TCP este cunoscut ca un protocl orientat pe conexiune, ceea ce înseamnă că o conexiune este stabilită și întreținută până când programul aplicație de la fiecare capăt a terminat interschimbul de mesaje. Protocolul determină cum se împart datele din aplicație în pachete pe care rețeaua le poate livra, cum se trimit acestea și cum sunt acceptate în nivelul rețea, controlează debitul pachetelor și se ocupă cu retransmiterea pachetelor refuzate sau eronate, precum și recunoașterea tuturor pachetelor primite (datorită faptului că este un protocol ce furnizează transmisia datelor fără erori). În modelul de comunicație OSI, TCP ocupă părți ale nivelului 4, cel de transport, și părți ale nivelului 5, cel sesiune.

De exemplu, atunci când un *server Web* trimite un fișier HTML către un client, acesta folosește protocolul HTTP. Nivelul de program HTTP interoghează nivelul TCP pentru a seta conexiunea și pentru a trimite fișierul. Stiva TCP împarte fișierul în pachete pe care le numerotează și pe care le trimite mai departe individual către nivelul IP. Deși pe parcursul transmisiei fiecare pachet va avea aceeași sursă și aceleași adrese IP destinație, pachetele pot fi trimise pe multiple rute. Nivelul program TCP în PC-ul clientului așteaptă până când toate pachetele au ajuns, apoi le recunoaște pe cele primite și interoghează pentru retransmisia pachetelor lipsă (în funcție de numărul acestora), după care le ansamblează într-un fișier pe care îl livrează aplicației destinație.

Retransmiterea și nevoia de a reordona pachetele după ce acestea ajung la destinație poate introduce latență în fluxul TCP. Aplicațiile foarte sensibile la timpul de transmisie, precum VoIP și *streaming-ul video*, se bazează în general pe transport (UDP) care reduce latență și *jitter-ul* (variații ale latenței), însă acestea nu prezintă probleme de reordonare a pachetelor sau primire de date incomplete.

Un utilitar Unix foarte des utilizat pentru a realiza conexiuni TCP cu un *host* din depărtare este **netcat.** Acesta este folosit pentru a citi și scrie date de-a lungul unei conexiuni de rețea, folosind protocoalele TCP și UDP. *Netcat* a fost creat ca un *tool* de *back-end,* putând fi direct utilizat de alte programe sau *script-uri*. În același timp, vine în ajutor în depanarea rețelelor și în aplicații de explorare, având în vedere că poate crea aproape orice tip de conexiune de care un utilizator ar putea avea nevoie și că are câteva capabilități interne foarte puternice. Pentru o utilizare simplă a acestui *tool*, comandă de *shell* **nc [options] host port** creează o conexiune TCP la portul precizat al mașinii țintă. Intrarea standard este apoi trimisă către acea mașină, iar orice date care vin înapoi prin intermediul acelei conexiuni sunt trimise înapoi ieșirii standard. Acest proces continuă la nesfârșit, până când partea de rețea a aplicației este oprită.

Dacă se dorește să se trimită un pachet UDP în locul inițierii unei conexiuni TCP, se va folosi opțiunea **-u**:

netcat -u host port

De asemenea, se poate specifica un interval de *port-uri*, prin adăugarea unei cratime:

netcat host startport-endport

Utilitarul *netcat* poate fi folosit și pentru scanarea de *port-uri*. Deși acesta nu este una dintre cele mai bune soluții pentru a realiza această acțiune, el poate fi folosit în aplicații simple, că identificarea *port-urilor* deschise. Acest lucru se realizează prin utilizarea opțiunii **-z**, în locul inițierii unei conexiuni TCP(se va folos și opțiunea **-v**, pentru informații mai detaliate):

netcat -z -v domain.com 1-1000

```
nc: connect to domain.com port 1 (tcp) failed: Connection refused
nc: connect to domain.com port 2 (tcp) failed: Connection refused
nc: connect to domain.com port 3 (tcp) failed: Connection refused
nc: connect to domain.com port 4 (tcp) failed: Connection refused
nc: connect to domain.com port 5 (tcp) failed: Connection refused
nc: connect to domain.com port 6 (tcp) failed: Connection refused
nc: connect to domain.com port 7 (tcp) failed: Connection refused
. . .
Connection to domain.com 22 port [tcp/ssh] succeeded!
. . .
```

Rezultatul comenzii ne arată dacă scanarea *port-urilor* a fost realizată cu succes, sau nu. Totuşi, scanarea *port-urilor* va fi mult mai rapidă dacă se cunoaşte adresa IP. În acest caz se poate folosi *flag-ul* **-n** pentru a specifica faptul că nu este nevoie de determinarea adresei IP, cu ajutorul DNS:

netcat -z -n -v 111.111.111.111 1-1000

Netcat nu este restricţionat la trimiterea de pachete TCP şi UDP. Acesta poate să asculte un *port* şi pentru conexiuni sau pachete. Acest lucru aduce oportunitatea conectării a două instanţe de *netcat* într-o relaţie de tip *client-server*. Alegerea calculatorului client şi a celui *server* este doar o distincţie relevanţă în timpul configurării iniţiale. După ce conexiunea este stabilită, comunicaţia este exact la fel în ambele direcţii. Pe una dintre maşini, *netcat* poate fi comandat să asculte pe un anumit *port* pentru conexiuni. Putem face acest lucru prin furnizarea parametrului **-l** şi alegerea unui port:

netcat -l 4444

Această comandă îi va indica lui *netcat* să asculte *portul* 4444 pentru conexiuni TCP. Ca utilizator normal (de tip *non-root*), nu vei putea fi capabil să deschizi *porturi* sub *portul* 1000, ca o măsură de securitate. Pe un al doilea *server*, ne putem conecta la prima maşină pe un *port* cu numărul ales de noi. Putem face acest lucru stabilind conexiunea anterioară:

netcat domain.com 4444

Printr-o conexiune TCP standard, putem transmite aproape orice tip de informaţie. Nu ne limităm la mesaje dintr-un *chat* ce sunt tastate de un anumit utilizator. Putem folosi această opţiune pentru a transforma *netcat* într-un program de transfer de fişiere.

Încă o dată, trebuie să alegem unul dintre capetele conexiunii pentru a asculta. Totuşi, în locul afişării informaţiilor pe ecran, că în exemplul anterior, vom distribui informaţiile direct într-un fişier:

netcat -l 4444 > received_file

Pe cel de-al doilea PC, vom crea un simplu fişier text cu următoarea comandă:

echo "Hello, this is a file" > original_file

În continuare, putem folosi acest fişier că intrare în conexiunea *netcat* pe care o vom stabili cu PC-ul care ascultă. Acest fişier va fi transmis că şi cum l-am fi scris în mod interactiv:

netcat domain.com 4444 < original_file

Acum putem observa pe calculatorul care aşteaptă conexiunea că avem un noi fişier numit **received_file**, ce conţine acelaşi lucru că fişierul de pe celălalt calculator.

5.4 SERVER APACHE

"HTTP este un protocol client-*server* de nivel aplicaţie bazat pe cereri ale clientului şi răspunsuri corespondenţe ale *server*ului. Tradiţional, HTTP foloseşte pentru transportul de date protocolul TCP (portul 80), însă poate funcţiona peste orice alt protocol care oferă o conexiune punct-la-punct garantată. Informaţiile transferate prin HTTP pot fi în format text(text obişnuit, HTML, XML) sau binar(ex:muzică, imagini etc).

*Server*ul *web* este un soft ce "vorbeşte" HTTP şi care publică resurse - fie fişiere aflate în sistemul său de fişiere, fie conţinut generat pe loc ca urmare a unei cereri de client. În primul caz, fişierele publicate vor fi cele aflate dedesubtul unui director desemnat ca rădăcina a paginilor *web*. Pentru un *server web* configurat cirect, clienţii nu vor avea acces decât la fişierele publicate, nu la întregul sistem de fişiere al *server*ului.

Clienţii specifică resursă dorită sub forma unei adrese, denumită în general URL, de formă http://www.example.com:80/ex/index.html. URL-ul este compus din 3 părţi:

- **http://** - specifică protocolul folosit. Această deoarece soft-urile client au deseori capabilitatea de a folosi mai multe protocoale(HTTP, FTP etc) şi trebuie să ştie pe care dintre acestea să-l folosească pentur resursă cerută
- **www.example.com:80** - desemnează adresa *server*ului - fie sub forma ei numerică, fie sub forma unui nume DNS căruia trebuie să corespundă în fişierul zonă o înregistrare de tip A sau CNAME - şi portul pe care acesta asculta
- **/ex/index.html** - reprezintă aşa-numitul URL-path : calea către resursa dorită, aşa cum o vede clientul. Ceea ce clientul consideră a fi / (rădăcina paginilor *web*) este de fapt conţinutul directorului desemnat ca atare în configurarea *server*ului.

Pentru fiecare cerere, *server*ul trebuie să identifice pe baza URL-path-ului calea reală către resursa solicitată. Implicit, operaţia se realizează prefizand URL-path-ului cu calea către directorul rădăcină a paginilor *web*, însă acest mod de lucru poate fi schimbat din fişierul de configurare al *server*ului. *Server*ul poate prezenta clietului orice altă structură a resurselor detinue (*web*space) care să nu corespundă 100% celei reale din sistemul de fişiere.

Apache funcţionează că un *server Web*. Rolul său principal este de a analiza orice fişier cerut de către un browser şi să afişeze rezultatele corecte în conformitate cu codul sursă din acel fişier. Apache este destul de puternic şi poate realiza aproape orice sarcină pe care proprietarul *server*ului(*Web*master) o doreşte.

Cele mai importante caracteristici ale *server*ului *Web* Apache sunt enumerate în continuare:

- Controlul accesului la resurse la nivel de fişier/director/URL-path(atât pentru filesystem cât şi pentru *web*space)
    - În funcţie de adresa clientului său/ *server*ului
    - În funcţie de numele domeniului din care face parte clientul
    - Prin crearea de fişiere/directoare parolate
- Virtual hosting - găzduirea mai multor site-uri pe aceeaşi maşină, în două moduri:
    - IP-based - host0ul în cauză dispune de mai multe adrese IP, fiecare cu propriul nume DNS
    - Name-based - maşina are un singur IP, iar departajarea *server*elor virtuale se face după numele *server*ului aşa cum apare el în URL cerut de client
- Site-uri personale ale utilizatorilor definiţi în sistem. Facilitate utilă în cazul ISP-urilor. Utilizatorii pot fi împuterniciţi să seteze unele opţiuni ale site-ului personal printr-un fişier de configurare individual(eliminând necesitatea intervenţiei permanente a administratorului)
- Posibilitatea de generare de pagini dinamice, prin folosirea altor programe:
    - Prin intermediul CGI, *server*ul poate interacţiona cu aplicaţii scrise în orice limbaj de programare
    - Există interpretoare sub formă de modul DSO pentru majoritatea limbajelor de *scripting* importante(Perl, PHP etc), cu avantajul vitezei sporite faţă de variant CGI
- Posibilitatea de manevrare a *web*space-ului(ierarhia de resurse văzută de către client)
- Filtrare I/O - trecerea datelor de I/O printr-un lanţ de module(prelucrare succesivă)
- Securizarea comunicaţiei cu *server*ul prin intermediul modulului SSL ce permite autentificarea şi criptarea comunicaţiei între client s *server* pe bază de chei publice
- MPM - module DSO care dau posibilitatea *server*ului de a se adapta la caracteristicile de multi-procesare ale sistemului de operare pe care rulează (până la versiunea 2, apache utiliza thread-urile POSIX, ceea ce îl făcea să ruleze ineficient pe windows sau Netware)" [18]

În conformitate cu *site-ul Web Netcraft* (**www.netcraft.com**), în acest moment *Apache* funcţionează pe mai mult de 27 de milioane de *servere* în Internet, mai mult decât *Microsoft*, *Sun ONE* şi *Zeus* la un loc. *Apache* a devenit o alegere atât de populară datorită flexibilităţii, puterii şi desigur, datorită preţului. Poate fi utilizat pentru a găzdui un site *Web* pentru publicul general, sau o companie răspândită în intranet, sau pur şi simplu pentru testarea paginilor create înainte că acestea să fie încărcate într-un servat securizat sau pe o altă maşină.

Pentru instalarea *serverului Web Apache*, se foloseşte următoarea comandă pe o distribuţie de Ubuntu/Debian Linux:

sudo apt-get install apache2

Pentru vizualizarea fişierelor de configurare ale *serverului*, prin tastarea comenzilor **cd /etc/apache2** şi **ls -F,** vom obţine următorul rezultat:

```
apache2.conf  envvars     magic            mods-enabled/  sites-available/
conf.d/       httpd.conf  mods-available/  ports.conf     sites-enabled/
```

În continuare voi descrie pe scurt cele mai importante dintre aceste fișiere:

apache2.conf : acesta este fișierul elementar de configurare al *serverului*. Aproape toate configurările pot fi făcute din acest fișier, deși se recomandă utilizarea de fișiere separate, cu un anumit scop precis, pentru simplitate. Acest fișier va configura setările standard și va fi punctul central de acces al *server*ului la citirea detaliilor de configurare.

Ports.conf : acest fișier este folosit pentru a specifica porturile pe care trebuie să asculte *host-urile* virtuale

Conf.d/ : directorul este utilizat pentru controlul unor aspecte specifice ale configurației *Apache*

Sites-available : acest director conține toate fișierele *host-urilor* virtuale ce definesc diferite *site-uri web*. Acestea vor stabili ce conținut este servit pentru fiecare request.

Sites-enabled/ : acest director stabilește care definiții de host virtual sunt de fapt utilizate

5.5 PROGRAMARE DE *SOCKET*

Un *socket* reprezintă capătul unei legături de comunicație bidirecțională dintre două programe care rulează într-o rețea. Un *socket* este legat de un număr de *port*, astfel că nivelul TCP să poată identifica aplicația spre care datele trebuie trimise. Capătul unei legături (*endpoint*) este o combinație între o adresă IP și un număr de *port*. Orice conexiune TCP poate fi unic identificată de capetele sale. În acest fel, pot există conexiuni multiple între un *host* și un *server*.

Un *socket* este de cele mai multe ori folosit în interacțiune dintre client și *server*. Configurații tipice ale unui sistem plasează *server-ul* pe o mașină și clienții pe alte mașini. Clienții se conectează la *server*, interschimba informații și apoi se deconectează.

Socket-ul are un curs tipic al evenimentelor. Într-un model client-*server* orientat pe conexiune, *socket-ul* de pe procesul *serverului* așteaptă cererile de la client. Pentru această, *serverul* stabilește la început o adresă pe care clientul o poate folosi pentru a-l găsi. După stabilirea adresei, *serverul* așteaptă cererile de servicii de clienți. Schimbul de date de la client la *server* are loc atunci când un client se conectează la *server* prin intermediul unui *socket*. *Serverul* rezolvă cererea clientului și îi trimite înapoi răspunsul.

Figura 5.1 arată în amănunt acest proces al evenimentelor pentru o sesiune *socket* orientată pe conexiune:

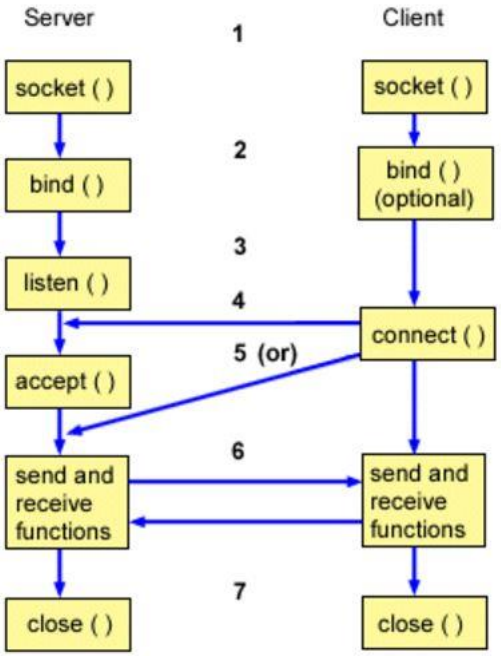


Figura 5.1 - Procesul evenimentelor pentru o sesiune *socket* orientată pe conexiune
Sursa [10]

Există patru tipuri de *socket-uri* **[18]** disponibile pentru utilizatori, cele mai utilizate fiind cele de *streaming* şi cele datagrama. În cazul *socket-urilor* de *streaming*, transmisia într-un mediu de tip reţea este garantat. Dacă sunt trimise 3 elemente prin fluxul de *socket* "A, B, C", la recepţie ele vor fi primite în aceeaşi ordine. Aceste tipuri de *socket* folosesc TCP pentru transmisia de date. Dacă transmisia este imposibilă, transmiţătorul va primi un indicator de eroare.

***Socket*.IO** este o librărie *JavaScript* (prezentat mai detaliat în **subcapitolul 5.7**) pentru aplicaţii *web* în timp real. Această librărie permite comunicaţia bidirecţională, în timp real, dintre clienţii *web* şi *servere*. Librărie conţine 2 componente: cea de client ce rulează pe *browser* şi cea de *server*, cunoscută ca librăria *Node.js*. Ambele componente conţin API-uri aproape identice.Asemenea lui *Node.js*, ***Socket*.IO** este orientat pe evenimente.

***Socket*.IO** utilizează în principal protocolul *WebSocket*. Putând fi folosit ca un *wrapper* pentru *WebSocket*, această librărie furnizează multe alte beneficii, incluzând transmitere de date *broadcast* către multiple *socket-uri*, stocare de date asociate clienţilor şi intrări/ieşiri asincrone. Mai mult decât atât, ***Socket*.IO** are abilitatea de a implementa aplicaţii analitice, *streaming* binar, mesagerie instantă şi colaborare de documente. Librăria manipulează conexiunea în mod transparent şi se poate actualiza la *WebSocket* dacă există posibilitatea.

Pe partea de *server*, ***Socket*.IO** funcţionează prin adăugarea de "ascultători"(*listeneri*) de evenimente la o instanţă a **http.*Server***. Pentru a întocmi acest lucru, se poate scrie un cod asemenător cu cel de mai jos:

```
var server = require("net").createServer();
var io = require("socket.io")(server);

var handleClient = function (socket) {
    // we've got a client connection
    socket.emit("tweet", {user: "nodesource", text: "Hello, world!"});
};

io.on("connection", handleClient);

server.listen(8080);
```

Pe partea de client, *serverul* HTTP poate să deservească librăria clientului la /*socket*.io/*socket*.io.js. Pentru conectarea la *server*ul ***Socket*.IO** creat anterior, următorul *tag* trebuie adăugate următoarele linii în *tag-ul body*:

```
<script src="/socket.io/socket.io.js"></script>
<script>
    var socket = io.connect("http://localhost");
</script>
```

Variabila globală *socket* este un obiect asemănător unui emițător de eveniment. Putem atașa un *listener* care să fie declanșat atunci când apare conexiunea la *server* astfel:

```
socket.on("connect", function () {
    console.log("Connected!");
});
```

Între client și *server* se pot transmite diverse evenimente și obiecte JSON serializabile, dar și fluxuri de date. Astfel că se pot atașa fluxuri *Readable* de date *browser-ului* din *server*. Dacă dorim să trimitem conținutul unui *script* din *server*, am putea scrie următoarele linii:

```
var fs = require("fs");
var ss = require("socket.io-stream");

io.on("connection", function (socket) {
    ss(socket).emit("script", fs.createReadStream(__filename));
});
```

Pentru a vizualiza fiecare bucată din fluxul de date în *browser*, vom asculta evenimentul "dată" din obiectul rezultat:

```
socket.on("script", function (stream) {
    var buffer = "";
    stream.on("data", function (data) {
        buffer += data.toString();
    });
    stream.on("end", function () {
        console.log(buffer);
    });
});
```

5.6 LIMBAJE DE *SCRIPTING SERVER-SIDE*

5.6.1 Server-side.Client-Side

Internetul înfăţişează două tipuri majore de *scripting*: *server-side* şi *client-side*. *Scripting-ul* de tip *client-side* este compus în mare parte din *Javascript*, care este responsabil de multe dintre caracteristicile *web* pe care le vedem deseori, cum ar fi ferestre *pop-up*, animaţii sau meniuri de tip *drop-down*. Motivul pentru care această parte este pe partea clientului este reprezentat de executarea codului pe maşina utilizatorului, după ce pagină este încărcată.

Utilizând *script-uri* pe partea de client se permite aducerea de schimbări asupra unei pagini fără că această să fie reîncărcată; de asemenea, se facilitează validarea iniţială de formă şi simplificarea imbunătăţirilor aduse interfeţei de utilizator.

Totuşi, utilizarea de *script-uri client-side* necesită că utilizatorii să aibă *Javascript* activat sau ca *browserul* să suporte acele tipuri de *script-uri*. Acest lucru înseamnă că aceste *script-uri client-side* nu trebuie utilizate pentru autentificarea de utilizatori sau pentru manipularea de informaţii sensibile, din cauza abilităţii utilizatorului de a modifica sau de a dezactiva *script-uri* de pe partea de client.

Scripting-ul server-side pe de altă parte este efectuat pe *serverul* găzduitor al *site-ului* înainte că pagina să fie distribuită către utilizator. Acest lucru înseamnă că orice schimbări ce trebuie aduse *script-ului* au nevoie de o reîncărcare a paginii. Utilizarea de *scripturi server-side* este benefică pentru autentificarea utilizatorilor, salvarea informaţiilor într-o bază de date, extragerea datelor pentru afişare şi multe alte sarcini.

Dezavantajul *scripting-ului server-side* constă în principal în reîncărcarea paginilor. Deoarece *script-ul* este procesat înainte că acesta să fie distribuit la *browser*, utilizatorul nu are acces la codul intern. Acest lucru face ca *script-urile server-side* să fie cea mai bună alegere în cazul informaţiilor sensibile.

1. PHP

PHP este un limbaj de *scripting* de tip *server-side* (pe partea de *server*) ce permite unui *site Web* să fie cu adevărat dinamic. PHP este acronimul pentru *Hypertext Preprocessor*. Flexibilitatea sa şi dificultatea scăzută în a învăţa acest limbaj de *scripting*(mai ales pentru programatorii cu experienţă în limbajele C, *Java* sau *Perl*) îl fac unul dintre cele mai populare limbaje de *scripting*. Popularitatea limbajului PHP continuă să crească pe măsură ce afacerile şi persoanele fizice de peste tot îl iau în considerare că o alternativă în locul limbajului *ASP Microsoft* şi înteleg că beneficiile PHP depăşesc cu siguranţă costurile. *Zend Technologies, Ltd*., sursa centrală de îmbunătăţiri aduse asupra PHP şi dezvoltatorii *Interpretorului Zend* care suportă aplicaţiile PHP, au afirmat că acum codul PHP poate fi găsit în aproximativ 9 milioane de *site-uri Web.*

PHP a fost creat iniţial de către Rasmus Lerdorf în 1995, pentru a satisface nevoia de a procesa date mai uşor atunci când se creează pagini pentru *World Wide Web*. De fapt, dorinţa originală a acestuia a fost să creeze un *script* care să păstreze o evidenţă a tuturor vizitelor pe care CV-ul său online le-a primit. Dintr-un mic moft, popularitatea scriptului creat de Rasmus a crescut treptat, aşa că el a continuat să dezvolte acest limbaj.

La început, acronimul PHP a însemnat altceva (*Personal Home Page*) şi a fost făcut public că un proiect open source gratis.

Spre deosebire de HTML, care este parsat de către browser atunci când o pagină este încărcată, PHP este preprocesat de către maşina care deserveşte documentul (această maşină fiind referită că *server*). Tot codul PHP conţinut de către acest document este procesat de către *server* înainte că documentul să fie trimis către browserul vizitatorului.

Cele trei zone majore în care *scripting-ul* PHP este folosit sunt:

- *scripting* pe partea de *server*: pentru care este nevoie de 3 elemente esenţiale (*parser-ul* PHP, un *server Web* şi un browser *Web*),
- *scripting* în linia decomandă : în acest caz, *script-ul* PHP poate rula fără un *server* în partea de browser, acesta având nevoie doar de un *parser*,
- dezvoltare de aplicaţii desktop: PHP nu este cea mai bună alegere în acest caz, pentru programatorii care nu cunosc foarte bine acest limbaj de *scripting*; se pot folosi capacităţi puse la dispoziţie de către extensia PHP-GTK.

În cadrul acestui limbaj de *scripting* pe partea de *server*, un element foarte important folosit în cadrul lucrării este acela de protejare a parolei utilizatorilor care se loghează în aplicaţia *Web*, sau *password hashing*. Aceasta reprezintă una dintre mai primare modalităţi de securitate a parolelor, considerată necesară atunci când se creează aplicaţii care acceptă parole de la utilizatori. Fără acest mecanism, orice parolă stocată în baza de date a aplicaţiei poate fi furată dacă baza de date este compromisă, lucru ce nu compromite doar aplicaţia în sine, ci şi conturile utilizatorilor şi în alte servicii, dacă aceştia nu folosesc parole unice pentru conturi diferite.

Funcţiile comune pentru acest mecanism de securizare a parolelor, precum **md5()** şi **sha1()** nu sunt suficiente pentru a cripta o parolă. Două soluţii alternative de îmbunătăţire a efectului celor două funcţii de securizare ar fi următoarele: **salt** şi funcţia **crypt()**. Desigur, prima este cea mai eficientă. Mecanismul salt se referă la adăugarea unor date suplimentare la parola criptată, pentru o dificultate şi mai crescută a decriptării de către răufăcători. Pentru aceasta, se folosesc funcţia **password_hash().**

În urma acestor paşi, o parolă criptată va arăta asemănător cu cea din Figura 5.3:

Figura 5.3 - Exemplu de parolă criptată în PHP
Sursa [11]

O altă necesitate a unui site *web* în ceea ce priveşte securitatea este noţiunea de variabile de sesiune. Acestea sunt variabile PHP ce conţin informaţii despre un singur utilizator al aplicaţiei, valabile pe toate paginile aplicaţiei. Astfel, se evita amestecul de informaţii şi mai mult decât atât, pierderea securităţii acestora, de la mai mulţi utilizatori. Prin intermediul acestor variabile de sesiune, se păstrează o evidenţă a tuturor utilizatorilor care au deschis aplicaţia, care se află încă într-o sesiune, care au închis o sesiune.

2. Node.JS

Node.JS este o platformă de tip *server-side* construită cu ajutorul mecanismului *Google Chrome Javascript*(versiunea 8). *Node.js* a fost dezvoltat de către Ryan Dahl în anul 2009, iar în cadrul documentaţiei celei mai noi versiuni v0.10.36, definiţia acestei platforme este următoarea:

"*Node.js* este o platformă construită pe baza mediului de rulare *Chrome JavaScript* pentru o structurare mai rapidă şi scalabilitatea aplicaţiilor de reţea. Node.js utilizează un model orientat pe evenimente, non-blocant pe intrare/ieşire ce îl face uşor de utilizat şi eficient, perfect pentru aplicaţii în timp real şi pentru manipulare intensivă de date, putând fi rulat pe mai multe echipamente distribuite. "

Node.js este un mediu de rulare *open-source*, cross-platform pentru dezvoltarea de aplicaţii de reţea sau de tip *server-side*. Aplicaţiile Node.js sunt scrise în *JavaScript* şi pot fi construite în cadrul mediului *Node.js* pe OS X, *Microsoft Windows* şi *Linux*. *Node.js* mai furnizează şi o librărie bogată în module *JavaScript* ce simplifică dezvoltarea aplicaţiilor *web*.

Cele mai importante caracteristici ale platformei *Node.js* sunt următoarele:

- **Asincron şi orientat pe evenimente**: toate API-urile librăriei *Node.js* care sunt asincrone sunt şi non-blocante. Acest lucru înseamnă, în esenţă, că un *server Node.js* nu aşteaptă niciodată că un API să returneze date, ci se mută la următorul API după chemarea acestuia, iar un mecanism de notificări a evenimentelor *Node.js* ajută *serverul* să primească răspunsul de la un *call* API precedent.

- **Foarte rapid**: fiind construit pe baza mecanismului *Google Chrome V8 JavaScript*, librăria *Node.js* este foarte rapidă în executarea codului

- **Rulare pe un singur *thread* cu o scalabilitate ridicată**: *Node.js* utilizează modelul de *thread* unic în buclă de evenimente. Mecanismul evenimentelor ajută *serverul* să răspundă într-un mod non-blocant, ceea ce face ca *serverul* să fie scalabil spre deosebire de *serverele* tradiționale care creează *thread-uri* limitate pentru a manipulare cererile.

- **Lipsa conceptului de buffering**: Aplicațiile *Node.js* nu stochează niciodată date într-un *buffer*. Acest aplicații oferă datele pe bucăți.

Domeniile de utilizare ale *Node.js* sunt, în mare parte, următoarele:

- Aplicații de I/O
- Aplicații pentur streaming de date
- DIRT
- Aplicații bazate pe API-uri JSON
- Aplicații *Single Page*

5.7 Limbaje de *scripting client-side*

1. JavaScript

JavaScript, de cele mai multe prescurcat că JS, este un limbaj de *scripting* orientat pe obiecte, cu o construcție ușoară, fiind cunoscut ca unul dintre cele mai utilizate limbaje utilizate în cadrul paginilor *Web*. Acest limbaj de *scripting* descrie acțiunile care au loc în cadrul site-ului *web* și face o conexiune ușoară între partea de *server* și partea de client a aplicației (JS fiind limbaj de *scripting* pe partea de client). *JavaScript* este un limbaj puternic utilizat la scară largă pentru a descrie comportamentul paginilor *Web*. *JavaScript* a fost creat pentru a îngloba atât concepte din limbajul de programare Java, cât și din C++ pentru a reduce numărul de noțiuni noi pe care dezvoltatorii de aplicații trebuie să le învețe.

2. HTML

HTML reprezintă un limbaj de marcare folosit în crearea paginilor *Web*, care împreună cu CSS și *JavaScript* formează o tehnologie de vârf în domeniul aplicațiilor *Web*. *Browser-ele Web* pot citi și execută fișiere HTML în pagini *web* vizile sau audibile. HTML descrie din punct de vedere semantic structura unui *web*site, iar înainte de aparația limbajului CSS pentru îmbunătățirea aspectului paginilor *web*, HTML se ocupă și de acest aspect; de accea, în era curentă el este considerat mai mult un limbaj de marcare decât un limbaj de programare.

În cadrul HTML, sunt definite *tag-uri* între care se creează blocuri de text, instrucțiuni, apeluri de funcții.

3. CSS

CSS este un limbaj de descriere a aspectului unei pagini *Web* scrise într-un limbaj de marcare că HTML, XHTML și chiar XML. Acest limbaj de bazează în special pe separarea conținutului unui document de documentul de prezentare, incluzând aspecte de proiectare, culori și *font-uri*. Această separare poate îmbunătăți accesibilitatea la conținutul documentului, poate

permite paginilor HTML să partajeze un tip de formatare prin specificarea fişierului CSS relevant cu extensia .css şi poate reduce complexitatea şi repetitivitatea din conţinutul structural.

CSS utilizează o sintaxă simplă cu un număr foarte mare de cuvinte cheie în limba engleză, pentru a specifica numele diferitelor proprietăţi şi stiluri.

4. jQuery

jQuery este o librărie *JavaScript* de tip *cross-platform*, creată pentru a simplifica *scripting-ul* HTML pe partea de client. *JQuery* este una dintre cele mai utilizate librării în zilele noastre. Sintaxa acestei librării are rolul de face navigarea documentelor mai uşoară, de a selecta elemente DOM, de a crea animaţii, de la manipula evenimente şi de a dezvolta aplicaţii *Ajax*. La baza, *jQuery* este o librărie de manipulare DOM (o reprezentare structurală de tip arbore a tuturor elementelor dintr-o pagină *Web*), care uşurează găsirea, selectarea şi jonglarea cu aceste elemente DOM.

Avantajele librăriei *jQuery* sunt:

- **Încurajarea separii JavaScript de HTML**: în locul adăugării atributelor de evenimente HTML care apelează funcţii *JavaScript*, se utilizează sintaxe smple pentru adăugare de manipulatoare de evenimente la DOM
- **Concizie şi claritate**
- **Eliminarea incompatibiliatilor *cross-browser***: mecanimesmele *JavaScript* de pe diferite browsere pot să difere atât de mult între ele, cât să nu poată coexistă; *jQuery* elimina acest neajuns, prinţ crearea unei interfeţe consistente ce funcţionează pe mai multe tipuri de browsere

5. BootStrap

BootStrap este un *framework web open-source*, folosit pe partea de *front-end* pentru construirea *site-urilor web* şi a aplicaţiilor *web*. Aceasta conţine elemente de CSS şi HTML, formând diverse *template-uri* de *design* pentru tipografie, forme, butoane, navigare şi alte elemente de interfaţă, precum şi extensii opţionale de *JavaScript*.

BootStrap este modular, conţinând în esenţe o serie de *Less stylesheets* ce implementează diversele componente. Dezvoltatorii de aplicaţii pot adapta un fişier *BootStrap* după bunul plac, alegând doar elemente care le sunt necesare şi eliminând pe cele redundante.

BootStrap aduce cu sine s câteva componente *JavaScript* sub formă de *plugin-uri jQuery*. Acestea furnizează elemente de interfaţa cu utilizatorul adiţionale, precum elemente de tip *dialog box, carousels*. Acestea extind de asemenea şi funcţionalitatea diverselor elemente de interfaţă deja existente, incluzând funcţia de auto-completare pentru câmpuri ce primesc ca intrare un text.

6. MySQL

Un alt favorit al limbajelor open source, MySQL este constructorul de baze de date care permite limbajului PHP şi *server*ului Apache să lucreze împreună pentru a accesa şi afişa date într-un format ce poate fi citit de către un *browser*. Acesta este un *server* cu un limbaj de interogare structurat construit pentru date numeroase şi pentru procesarea de interogări complexe. Ca un sistem de bază de date relaţional, MySQL permite ca diverse tabele să fie unite pentru o eficienţă şi o viteză maximă.

Cele mai populare caracteristici ale limbajului MySQL sunt prezentate în continuare:

- utilizarea de multe CPU prin intermediul *thread-urilor de kernel*
- operaţii *multi-platform*
- numeroase tipuri de coloane ce acoperă aproape orice tip de date
- funcţii de grup pentru calcule şi sortări matematice
- comenzi ce permit informaţiilor despre baza de date să fie afişată uşor şi succint de către administrator
- nume de funcţii ce nu afectează tabelele sau numele de coloane
- sistem de verificare de parolă şi de utilizator pentru o securitate suplimentară
- până la 32 de indecşi per tabelă; această caracteristică a fost implementată cu succes la nivelul de 60.000 tabele şi 5.000.000.000 rânduri
- raportare de erori la nivel internaţional, utilizabilă în ţări diferite

MySQL este alegerea potrivită pentru furnizarea de date via *Internet* datorită abilităţii sale de a manipula cantităţi mari de informaţie şi datorită măsurilor sale de securitate avansate.

MySQL este o componentă integrată a platformelor LAMP şi WAMP (*Linux/Windows-Apache-MySQL-PHP/Perl/Python*). Este uşor de folosit şi integrat în aplicaţii, iar pentru administrarea bazelor de date de acest tip se pot folosi diverse aplicaţii grafice sau linia de comandă. Cele mai utilizate operaţii ale bazelor de date sunt următoarele:

Comandă	**Semnificaţie**
CREATE	Creare bază de date sau tabelă
DROP	Ştergere bază de date sau tabelă
INSERT	Adăugare înregistrare într-o tabelă
DELETE	Ştergere înregistrări dintr-o tabelă
UPDATE	Actualizare înregistrări din tabelă
SELECT	Selectare tabelă
ALTER	Alterare tabelă

Tabelul 5.2 - Comenzi elementare pentru baze de date

CAPITOL 6

DEZVOLTAREA SISTEMULUI DE CONTROL ŞI AUTOMATIZARE A ROBOTULUI HEXAPOD

Din punct de vedere funcţional, lucrarea este compusă din două subsisteme:

- subsistemul de control
- subsistemul de redare video

Pe lângă aceste două subsisteme, lucrarea dispune şi de o interfaţă *Web*, prin intermediul căreia utilizatorul are posibiltatea să trimită comenzi robotului hexapod în vederea unor mişcări complexe, cât şi posibilitatea de redare video cu ajutorul camerei ataşate microcalculatorului *Raspberry Pi*.

6.1 SUBSISTEMUL DE CONTROL

Subsistemul de control a fost realizat cu ajutorul a două limbaje de *scripting*, *JavaScript* şi *Python* şi un limbaj de programare Arduino, bazat pe C. Pentru atingerea obiectivului lucrării, acela de a transmite comenzi unui robot hexapod, am folosit următoarele unităţi funcţionale:

- *browser Web*
- *server* HTTP pe maşină *Ubuntu* cu deschidere la *Internet*
- *raspberry Pi* cu modul de *WiFi* şi conexiunea la *Internet*
- hexapodul conectat la *Raspberry* prin intermediul protocolului I2C

Interconectarea dintre aceste unităţi funcţionale este descrisă în următoarea schemă bloc, din Figura 6.1:

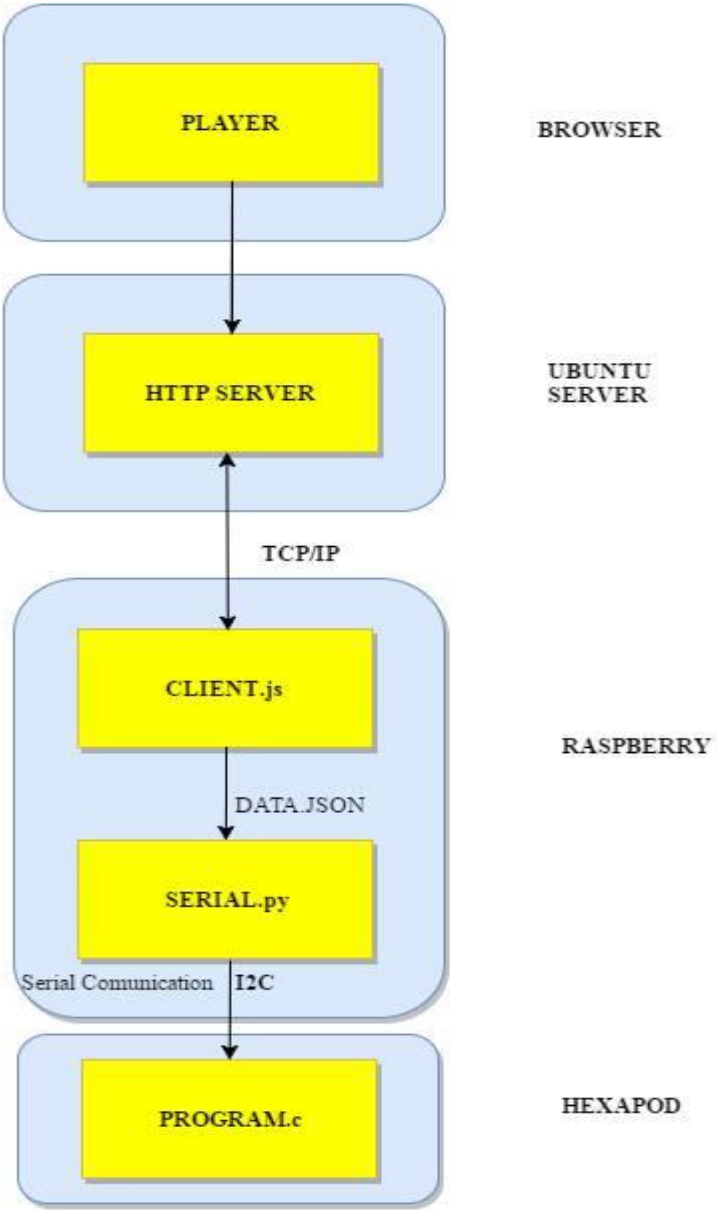


Figura 6.1 - Schemă funcţională a modulului de control

În continuare, voi trata fiecare unitate funcţională în parte şi voi descrie modul în care aceste sub-blocuri comunică între ele.

6.1.1 Browser-ul Web

După procesul de autentificare ce va fi descris în subcapitolul 6.3, *browser-ul* primește de la *server* pagina **player.html,** listat în Anexa 1. În cadrul acestui *script*, am definit o modalitate de comunicare cu *server-ul Ubuntu* prin intermediul metodei POST din cadrul protocolului HTTP. Comunicarea se face prin intermediul *click-urilor* utilizatorului primite la intrare de către *browser*, care inițializează câte o metodă POST care trimite pe portul 15001 un obiect JSON cu informații legate de comanda dorită de utilizator (deplasare stânga, deplasare dreapta, deplasare înainte/înapoi, stop, întoarcere stânga, întoarecere dreapta).

Rolul *browser-ului* este acela de a interpreta *script-ul* care realizează interfațarea cu utilizatorul. *Browser-ele* utilizate în cadrul lucrării pentru testare au fost *Google Chrome, Mozilla FireFox, Opera și Microsoft Edge*.

6.2.2 Server-ul HTTP pe mașină Ubuntu

Server-ul HTTP este definiți în limbajul de *scripting JavaScript* de pe platforma *Node.js*, având rolul de a asculta pe portul 15001 metodele POST trimise de *browser* care au ca și conținut obiecte JSON. De asemenea, în cadrul acestui *server*, la conexiunea cu un client definit pe *Raspberry Pi*, se creează un *socket* între client și *server*. *Server-ul* HTTP primește obiectele de tip JSON, le verifică și le trimite clientului prin conexiune TCP/IP stabilită în momentul deschiderii *socket-ului*.

În plus, *server-ul* HTTP dispune de o **automatizare** a conexiunii *client-server*, astfel: pentru stabilirea conexiunii dintre client și *server*, nu este nevoie de cunoașterea adresei IP a mcirocalculatorului Raspberry Pi. De aceea, avantajul major al acestei automarizarii este plasarea *RaspBerry-ului* atașat de robot în orice tip de rețea, precum rețeaua de tip NAT.

Acest *server* este deschis în permanență, el aflându-se într-o stare de așteptare a conexiunilor cu noi clienți. Odată încheiată o conexiune, se tratează evenimentul de tip *end* și se distruge conexiunea TCP creată anterior.

Crearea *server-ului* este pusă în evidență de *script-ul* ***server*.js,** aflat în Anexa 1. Pentru crearea *server-ului* am folosit modulul *http* definit în platforma Node.js și am apelat funcția *http.createServer(handle_incoming_request),* unde *handle_incoming_request* este funcția *callback* care descrie funcționalitatea *server-ului*. Această funcție verifică dacă pe portul 15001 se primește metoda POST și dacă obiectul primit este un JSON valid. Nu în ultimul rând, funcția mai este folosită și pentru trimiterea datelor către *Raspberry P*i pe *socket-ul* deschis. Pentru crearea *socket-ului* am folosit modulul *net*, definit de asemenea în cadrul platformei *Node.js* și funcția *function(conection)* de tip *callback. Socket-ul* se deschide în momentul în care pe portul 15002 se conectează un client.

6.2.3 Raspberry Pi

Pentru a configura conexiunea între *server-ul* HTTP și *Raspberry Pi*, cât și conexiunea dintre *Raspberry Pi* și hexapod, trebuie rulate o serie de aplicații.

În primă instanță, se inițializează o conexiune TCP între *server-ul* HTTP și Raspberry Pi, prin rularea *script-ului* **client.js** (Anexă 1) în fundal. Acest *script* are ca rol menținerea conexiunii TCP, primirea datelor venite de la *server-ul* HTTP și transformarea comenzilor recepționate ca obiect JSON într-un fișier text, cu numele **data.txt,** salvat în același director.

În continuare, se rulează scriptul **serial.py** (Anexă 1) cu comanda **python serial.py** în terminalul Linux. Acest *script* are ca rol citirea datelor în mod continuu din fişierul **data.txt** şi transmiterea lor prin conexiunea serială I2C către *controllerul Arbotix* al hexapodului.

6.2.4 Robotul Hexapod

Controllerul robotului hexapod preia, prin conexiune serială I2C, comenzile utilizatorului. Aceste comenzi sunt transpuse în limbajul *controllerului*, ceea ce înseamnă că acesta va primi una dintre cifrele din mulţimea {0, 1…,6}, fiecare corespunzătoare mişcării pe care robotul trebuie să o întreprindă.

Pe *controller-ul* robotului hexapod sunt verificate datele primite cu ajutorul unui program scris în limbajul Arduino C. În funcţie de cifra identificată, programul va comanda *controllerele* actuatoarelor DYNAMIXEL pentru mişcarea corespunzătoare.

Pentru fiecare cifră în parte se vor efectua următoarele mişcări:

0 - stop (oprire mişcare)

1 - mişcare înapoi

2 - mişcare dreapta

3 - mişcare stânga

4 - mişcare înainte

5 - întoarcere dreapta

6 - întoarcere stânga

Programul Arduino C **hexapod.c** corespunzător se găseşte în Anexa 1.

6.2 SUBSISTEMUL DE REDARE VIDEO

Acest modul funcţional are în componenţa sa *server-ul* de redare video care interacţionează cu *browser-ul Web*. Primul pas în iniţializarea redării video este pornirea *server-ului* video din terminalul Liux, cu ajutorul comenzii **sudo node *server*video.js.**

Odată pornit, *server-ul* va asculta pe portul 80 şi va aştepta primirea unei comenzi de start din *browser* pentru începerea redării video. De asemenea, interfaţa *Web* va dispune şi de un buton de *stop*, care va opri imediat redarea video. Pentru a fi posibilă această redare video, este nevoie să realizăm acţiunea de *port forwarding* în reţeaua locală, aceasta fiind o configurare standard în momentul în care porturile sunt blocate de către *firewall-ul router-ului*.

La primirea comenzii de *start*, se apelează utilitarul **raspivid** care permite interfaţarea cu camera video şi pornirea acesteia. De asemenea, în momentul primirii comenzii *stop* camera se opreşte, iar la primirea comenzii *disconnect* se opreşte atât camera, cât şi *server-ul* de redare video.

Schema funcţională a acestui subsistem este prezentată în Figură 6.2.

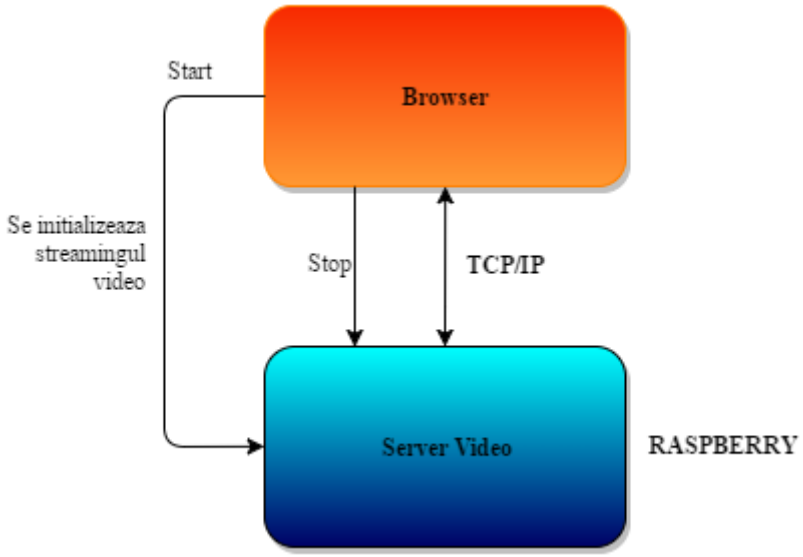


Figura 6.2 - Schemă funcţională a subsistemului de redare video

Camera ataşată microcalculatorului *Raspberry Pi* redă un semnal video **RAW H.264.** Din acest motiv, am fost nevoit să folosesc un modul *software* de încapsulare în timp real a unui flux video H.264 în formatul MP4 ce poate fi redat într-un *browser*. Modulul de încapsulare este bazat pe librăria **broadway** definită în platforma *Node.js*.

Pentru redarea video în cadrul paginii **player.html** am folosit *tag-ul* <**iframe>,** ce permite apelarea *server-ului* video şi stabilirea unei conexiuni TCP între pagina *Web* şi *server-ul* video.

6.3 INTERFAŢA *WEB*

Interfaţa *WEB* este compusă din două pagini *WEB*: **login.html** respectiv **player.html.** Aceasta este găzduită pe un *server Apache,* localizat pe maşina virtuală pe care se găseşte şi *server-ul* HTTP. Interfaţa *WEB* poate fi accesată pe adresa ***gaiţă.studenţi.speed.pub.ro,*** iar la accesarea acesteia utilizatorul este trimis la pagina de logare, unde îi sunt cerute datele contului, respectiv adresa de *e-mail* şi parola.

Schemele funcţionale ale logării sunt prezentate mai jos, în Figura 6.3 şi Figura 6.4.

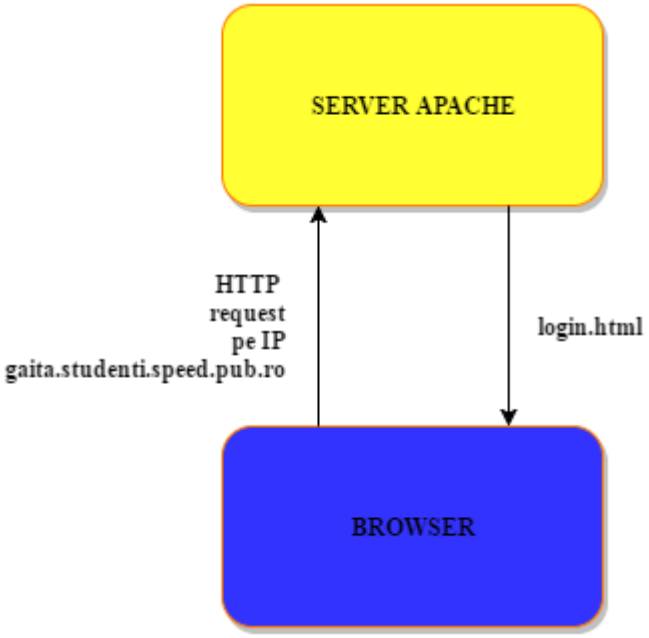


Figura 6.3 - Schemă funcţională a accesării interfeţei *Web*

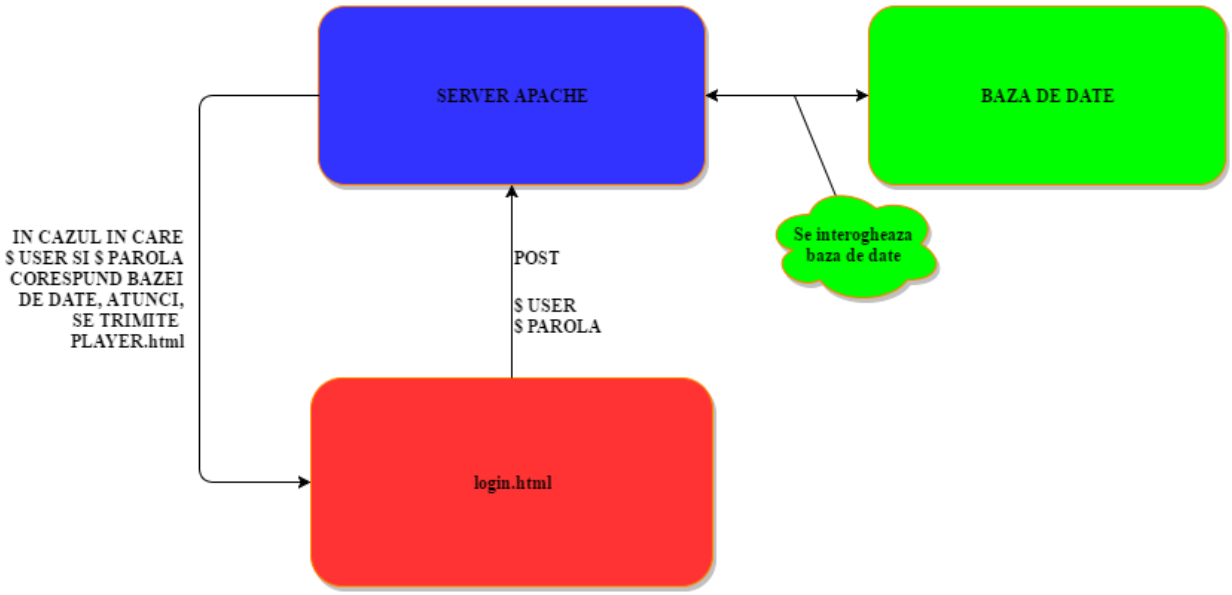


Figura 6.4 - Schemă funcţională a procesului de logare

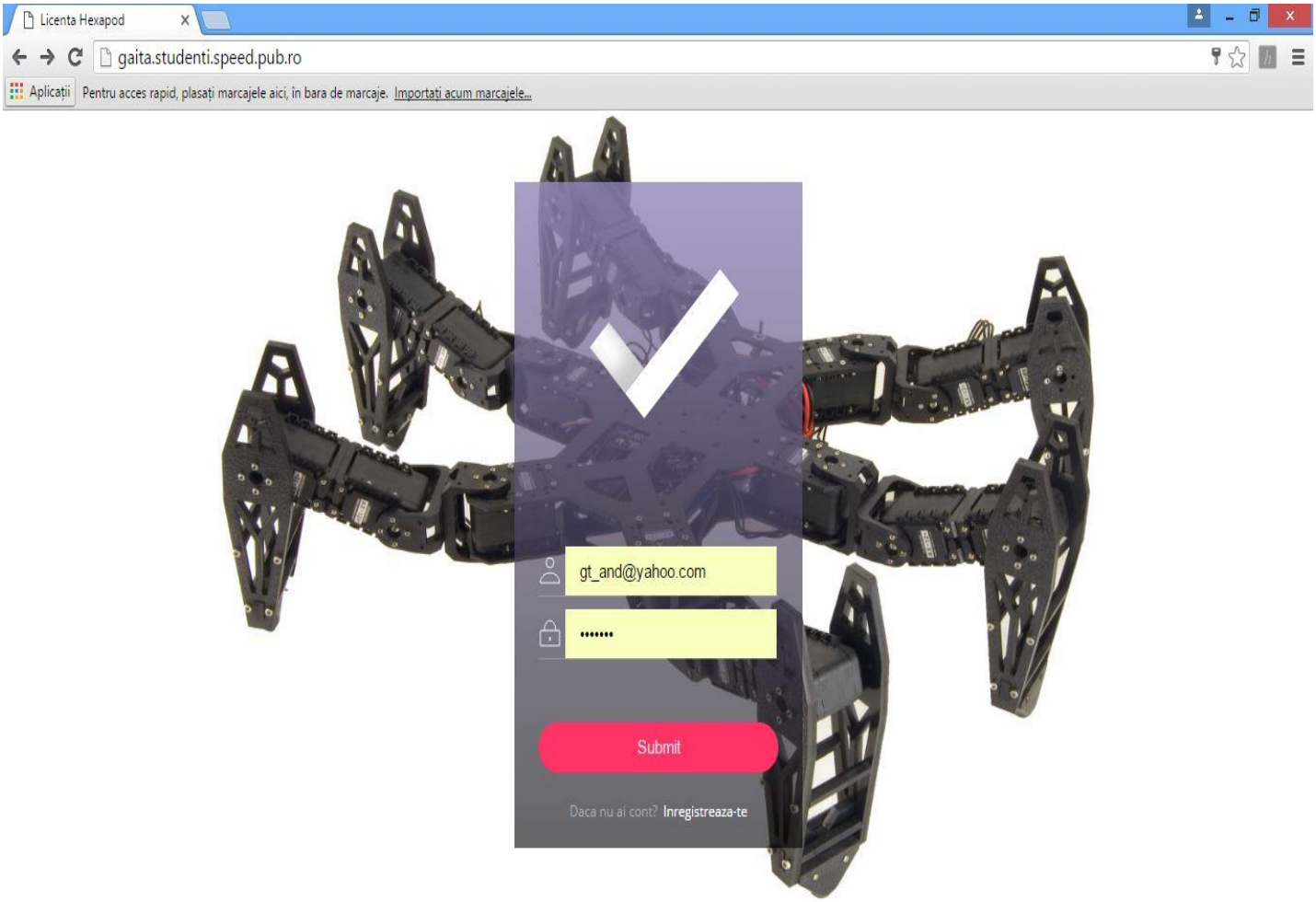


Figura 6.5 - Pagina de logare

Pentru a ajunge la pagina de control şi redare video(Figura 6.5), utilizatorul trebuie să introducă datele în câmpurile de intrare corespunzătoare, date care vor fi verificate pe partea de *server* cu baza de date.

În momentul apăsării butonului de logare, se interoghează baza de date şi se verifică dacă datele introduse corespund cu înregistrările din baza de date. Această verificare este descrisă în *script-ul* **login.html** care pe partea de *server* este definită în limbajul de *scripting* PHP. Dacă verificarea a fost cu succes, utilizatorul este trimis către pagina de redare video.

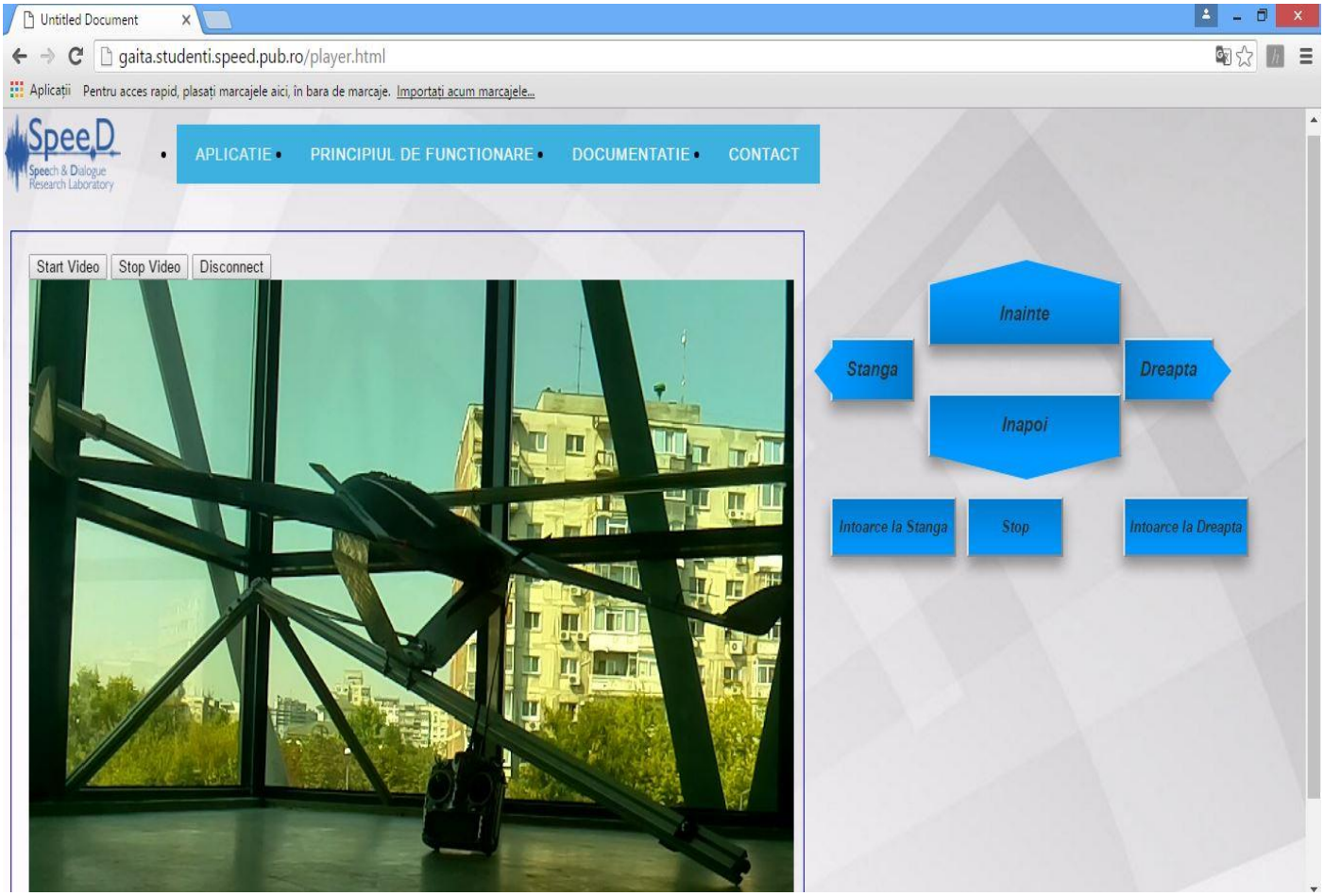


Figura 6.6 - Pagina de redare video

În caz contrar (Figura 6.7, Figura 6.8), acestuia i se va afișa un mesaj de eroare care are atașat un *link* către pagina de logare.

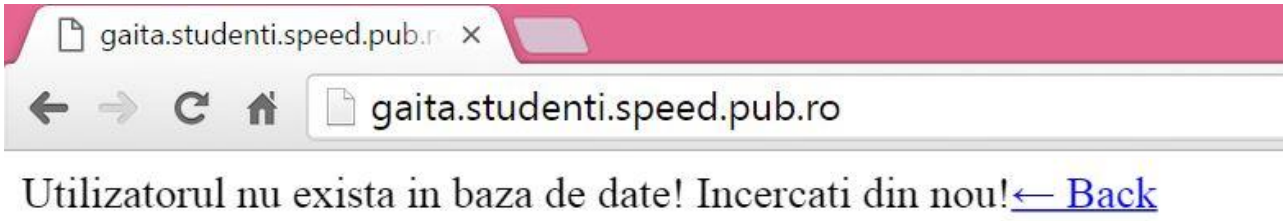


Figura 6.7 - Mesaj de eroare - utilizator inexistent în baza de date

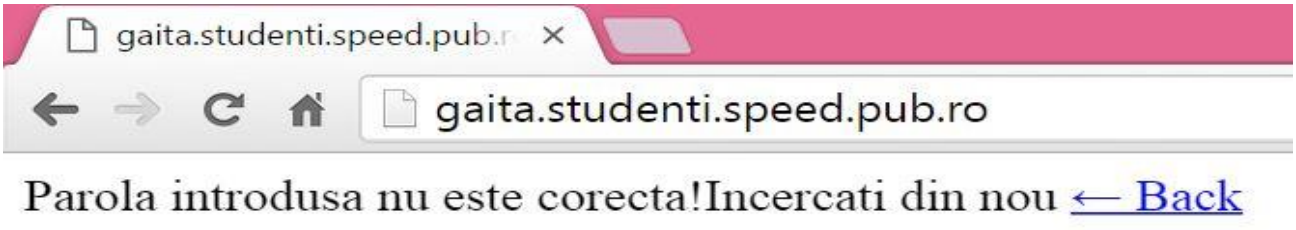


Figura 6.8 - Mesaj de eroare - parola incorectă

În cadrul paginii de redare video, există definirea *iframe-ului*, în care se realizează redarea video și 6 butoane ce au atașate câte un identificator unic prin intermediul căruia se apelează nişte funcţii de tip *Ajax* definite în limbajul *JavaScript JQuery*. La fiecare apăsare pe buton, se trimite câte un POST către *server*, care conţine un obiect JSON cu direcţia de deplasare a robotului.

CAPITOL 7

CONCLUZII

7.1 CONCLUZII GENERALE

Lucrarea de diplomă prezentat a avut ca obiectiv realizarea redării video în cadrul unei interfețe *Web*, cu un flux video preluat de la cameră video ataşată microcalculatorului *Raspberry Pi*. Tehnologiile *hardware* utilizate au fost sistemul de procesare menţionat anterior, robotul hexapod *PhantomX Mark II*, iar tehnologiile *software* au fost limbajele de *scripting server-side*, PHP, platforma *Node.js*, *Python* alături de limbajele de *scripting client-side* HTML, CSS, *Ajax*, *JavaScript*, *JQuery*. În timpul deplasării hexapodului în funcţie de comenzile utilizatorului, cameră video ataşată microcalculatorului *Raspberry Pi* transmite în browser imaginile dinamice capturate în timp real.

Realizarea practică a lucrării este funcţională, indeplinindu-si obiectul iniţial. Robotul hexapod poate întreprinde 6 mişcări diferite (deplasare înainte/înapoi, deplasare dreapta/stânga, rotire stânga/dreapta), iar imaginile dinamice preluate de camera video sunt transmise în *browser-ul Web*, pentru ca utilizatorul să le poată vizualiza în timp real.

7.2 CONTRIBUȚII PERSONALE

Contribuţiile personale, descrise pe larg în **Capitolul 6**, sunt următoarele:

- realizarea interfeţei *Web*
- realizarea *serverelor* ce permit controlul robotului
- realizarea comunicaţiei dintre componentele *hardware* utilizate
- realizarea *serverului* de redare video
- implementarea *softului* de mişcare pentru robot

7.3 DEZVOLTĂRI ULTERIOARE

Ansamblul de sisteme create în cadrul acestei lucrări ar putea fi utilizate într-o serie de aplicaţii ulterioare, la care se pot adăuga următoarele îmbunătăţiri asupra:

- calităţii video
- mişcărilor robotului hexapod, cum ar fi urcarea scărilor
- automatizării conexiunii de *streaming video*

REFERINȚE

[1] *Trossen Robotics*

http://www.trossenrobotics.com/resize/shared/images/PImages/RK-*PhantomX*-Hexapod-AX-12a.jpg?bw=1000&bh=1000

Accesat la data: 28.05.2016

[2] *Trossen Robotics*

http://learn.trossenrobotics.com/images/tutorials/*arbotix*M/*arbotix*m_main_photo.jpg

Accesat la data: 03.06.2016

[3] ROBOTIS e-Manual v1.25.00

http://support.robotis.com/en/product/dynamixel/ax_series/dxl_ax_actuator.htm

Accesat la data: 23.06.2016

[4] *Trossen Robotics*

http://learn.trossenrobotics.com/images/tutorials/*PhantomX*Hexapod/*PhantomX*HexapodId.jpg

Accesat la data: 15.05.2016

[5] *Cool Components*

https://d1dr2mxwsd2nqe.cloudfront.net/media/catalog/product/cache/1/image/9df78eab33525d08d6e5fb8d27136e95/2/5/252522540.jpg

Accesat la data: 15.05.2016

[6] *Embedded Linux Wiki*

http://elinux.org/RPi_*Hardware*

Accesat la data: 16.05.2016

[7] *RoboFun*

https://www.robofun.ro/*wireless*/*wireless*-wifi/miniature-wifi-802-11b-g-n-module-for-raspberry-pi-and-more

Accesat la data: 16.05.2016

[8] *Wikimedia Commons*

https://upload.wikimedia.org/wikipedia/commons/thumb/6/64/I_P_and_B_frames.svg/2000px-I_P_and_B_frames.svg.png

Accesat la data: 18.08.2016

[9] *Axis Communications, H.264 video compression standard*

www.axis.com

[10] *The TCP/IP Guide*

http://www.tcpipguide.com/free/t_TCPConnectionTermination-2.htm

Accesat la data: 14.08.2016

[11] *PHP.net*

https://secure.php.net/

Accesat la data: 14.08.2016

[12] *Trossen Robotis*

http://www.trossenrobotics.com/*PhantomX*-ax-hexapod.aspx

Accesat la data: 20.08.2016

[13] *Trossen Robotics*

http://learn.trossenrobotics.com/*arbotix*/*arbotix*-getting-started/38-*arbotix*-m-*hardware*-overview#&panel1-16

Accesat la data: 04.09.2016

[14] *TechWeek Europe*

http://www.techweekeurope.co.uk/workspace/arm-unveils-big-little-low-power-cortex-a7-architecture-42947

Accesat la data: 07.09.2016

[15] *streamingMedia.com*

http://www.streamingmedia.com/Articles/Editorial/What-Is-.../What-is-a-*Codec*-74487.aspx

Accesat la data: 04.04.2016

[16] *streamingMedia.com*

http://www.streamingmedia.com/Articles/Editorial/What-Is-.../What-is-H.264-74735.aspx

Accesat la data: 04.04.2016

[17] *TutorialsPoint*

http://www.tutorialspoint.com/unix_*sockets*/what_is_*socket*.htm

Accesat la data: 01.09.2016

[18] *InfoAcademy, Curs 11 Linux – Server-ul Web*, pag. 2-3

[19] *Peter Vis*

http://www.petervis.com/Raspberry_PI/Raspberry_Pi_CSI/Raspberry_Pi_CSI_Camera_Interface.html

Accesat la data: 12.08.2016

Anexa 1

- **login.html**

```
<?php
        $link=mysql_connect("localhost","root","cuante");
        if(!$link){
                die('Eșec la  conectare: ' .mysql_error());
        }
        #echo 'Conectare cu succes la serverul mysql';
        $db_select=mysql_select_db('utilizatori',$link);
        if(!$db_select){
                die('Nu  s-a  putut  realiza  conectarea  la  baza  de  date:  '
.mysql_error());
        }
        #echo "<br> Conectarea la baza de date a fost realizată </br>";
        if($_SERVER["REQUEST_METHOD"] == "POST") {
                if($_POST['user'] && $_POST['Password']){


                        $email=mysql_real_escape_strâng($_POST['user']);
                        $pass=mysql_real_escape_strâng(hash("sha512"        ,
$_POST['Password']));



                        $test="SELECT * FROM users WHERE email = '$email'";
                        $testquery=mysql_query($test, $link);

                        $usermysql=mysql_fetch_array($testquery);

                        $passmysql=hash("sha512", $usermysql['Parolă']);
                        $nameuser=$usermysql['Nume'];
                        if($usermysql=='0'){
                                die("Utilizatorul nu există în baza de date!
Încercaţi din nou!<a href='index.php'>&larr; Back</a>");
                        }
                        if($passmysql != $pass){
                                die("Parola      introdusă      nu      este
corectă!Încercaţi din nou <a href='index.php'>&larr; Back</a>");
                        }
```

```
                        header('Location:player.html');


                }
        }
?>



<html>
 <head>
    <meta charset="UTF-8">
    <title>Licenţă Hexapod</title>
    <meta name="viewport" content="width=device-width, initial-scale=1, user-
scalable=yes">
    <link                        rel='stylesheet                        prefetch'
href='http://fonts.googleapis.com/css?family=Open+Sans'>
        <link rel="stylesheet" href="css/style.css">
  </head>
   <body>
  <div class="cont">
  <div class="demo">
    <div class="login">
      <div class="login__check"></div>
         <form action="" method="post">
      <div class="login__form">
        <div class="login__row">
          <svg class="login__icon name svg-icon" viewBox="0 0 20 20">
            <path d="M0,20 a10,8 0 0,1 20,0z M10,0 a4,4 0 0,1 0,8 a4,4 0 0,1
0,-8" />
          </svg>
          <input     type="text"     name='user'class="login__input     name"
placeholder="Email"/>
        </div>
        <div class="login__row">
          <svg class="login__icon pass svg-icon" viewBox="0 0 20 20">
            <path  d="M0,20  20,20  20,8  0,8z  M10,13  10,16z  M4,8  a6,8  0  0,1
12,0" />
          </svg>
          <input  type="password"  name='Password'  class="login__input  pass"
placeholder="Parolă"/>
        </div>
```

```
          <input       type="submit"       name="Logare"       value="Submit"
class="login__submit"  />

        <p  class="login__signup">Dacă  nu  ai  cont?   <a>Inregistreaza-
te</a></p>
      </div>
          </form>
    </div>

  </div>
</div>
    <script src=#></script>

        <script src=#></script>
  </body>
</html>
```

- **player.html**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"> <html
xmlns="http://www.w3.org/1999/xhtml"> <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
        <meta charset='utf-8'>
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width,
initial-scale=1">

        <script src="http://code.jquery.com/jquery-latest.min.js"
type="text/javascript"></script>

        <script src="menubar.js"></script>
        <!--Ajax Post aş Json-->
        <script
src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js"></scri
pt>
        <script type="text/javascript"
src="http://code.jquery.com/jquery-1.12.4.js"></script>
        <script type="text/javascript">
        <?php
```

```
                $projectName="";
                $nameBranch="";
                $randPort=rând(3000,4000);
        ?>
      $(document).ready(function(){
                $("#înainte").click(function(){
                        $.ajax({
                                url:
"http://gaiţă.studenţi.speed.pub.ro:15001",
                                type: "POST",
                                crossDomain: true,
                                data: JSON.stringify({
                                direction:"4",
                                }),
                                dataType: "application/json;charset=utf-8",
                                success: function(response) {
                                        var resp = JSON.parse(response)
                                        //alert(resp.status);
                                },
                                error: function(xhr,status){
                                        //alert("error");
                                }
                        });
        });
      });
      $(document).ready(function(){
                $("#înapoi").click(function(){
                        $.ajax({
                                url:
"http://gaiţă.studenţi.speed.pub.ro:15001",
                                type: "POST",
                                crossDomain: true,
                                data: JSON.stringify({
                                direction:"1",
                                }),
                                dataType: "application/json;charset=utf-8",
                                success: function(response) {
                                        var resp = JSON.parse(response)
                                        //alert(resp.status);
                                },
                                error: function(xhr,status){
```

```
                                        //alert("error");
                                }
                        });
        });
      });
      $(document).ready(function(){
                $("#stânga").click(function(){
                        $.ajax({
                                url:
"http://gaiță.studenți.speed.pub.ro:15001",
                                type: "POST",
                                crossDomain: true,
                                data: JSON.stringify({
                                direction:"3",
                                }),
                                dataType: "application/json; charset=utf-8",
                                success: function(response) {
                                        var resp = JSON.parse(response)
                                        //alert(resp.status);
                                },
                                error: function(xhr,status){
                                        //alert("error");
                                }
                        });
        });
      });
      $(document).ready(function(){
                $("#dreapta").click(function(){
                        $.ajax({
                                url:
"http://gaiță.studenți.speed.pub.ro:15001",
                                type: "POST",
                                crossDomain: true,
                                data: JSON.stringify({
                                direction:"2",
                                }),
                                dataType: "application/json;charset=utf-8",
                                success: function(response) {
                                        var resp = JSON.parse(response)
                                        //alert(resp.status);
                                },
```

```
                                error: function(xhr,status){
                                        //alert("error");
                                }
                        });
        });
      });
      $(document).ready(function(){
                $("#stop").click(function(){
                        $.ajax({
                                url:
"http://gaiţă.studenţi.speed.pub.ro:15001",
                                type: "POST",
                                crossDomain: true,
                                data: JSON.stringify({
                                direction:"0",
                                }),
                                dataType: "application/json;charset=utf-8",
                                success: function(response) {
                                        var resp = JSON.parse(response)
                                        //alert(resp.status);
                                },
                                error: function(xhr,status){
                                        //alert("error");
                                }
                        });
        });
      });
      $(document).ready(function(){
                $("#turnLeft").click(function(){
                        $.ajax({
                                url:
"http://gaiţă.studenţi.speed.pub.ro:15001",
                                type: "POST",
                                crossDomain: true,
                                data: JSON.stringify({
                                direction:"6",
                                }),
                                dataType: "application/json;charset=utf-8",
                                success: function(response) {
                                        var resp = JSON.parse(response)
                                        //alert(resp.status);
```

```
                                },
                                error: function(xhr,status){
                                        //alert("error");
                                }
                        });
        });
      });
      $(document).ready(function(){
                $("#turnRight").click(function(){
                        $.ajax({
                                url:
"http://gaiță.studenți.speed.pub.ro:15001",
                                type: "POST",
                                crossDomain: true,
                                data: JSON.stringify({
                                direction:"5",
                                }),
                                dataType: "application/json;charset=utf-8",
                                success: function(response) {
                                        var resp = JSON.parse(response)
                                        //alert(resp.status);
                                },
                                error: function(xhr,status){
                                        //alert("error");
                                }
                        });
        });
      });
    </script>

        <link rel="stylesheet" type="text/css" href="mystyle.css">
        <title>Untitled Document</title>
</head>

<body class="body">
                <div >

                                <div  id='cssmenu'  STYLE="position:absolute;
left:140px; top:20px;">
                                        <ul>
                                                <li          class='active'><a
href='#'>Aplicație</a></li>
```

```
                                        <li><a href='#'>Principiul de
funcţionare</a></li>
                                        <li><a
href='#'>Documentaţie</a></li>
                                        <li><a
href='#'>Contact</a></li>
                                    </ul>

                                </div>

                        <div > <span>
                                <img    STYLE="position:absolute;    TOP:3px;
LEFT:2px; WIDTH:120px;
                                HEIGHT:80px;                                    "
src="http://gaiţă.studenţi.speed.pub.ro/speed.png" />
                              </span>
                        </div>

                        <div class="streaming_window">
                                <iframe  src="http://10.90.2.71"  width="800"
height="600" frameborder="0">
                                </iframe>
                        </div>



                        <button   type="button"   id="dreapta"   class="right-
arrow">Dreapta</button>
                        <button    type="button"    id="înapoi"    class="down-
arrow">Înapoi</button>
                        <button    type="button"    id="stânga"    class="left-
arrow">Stânga</button>
                        <button   type="button"  id="înainte"  class="up  up-
arrow">Înainte</button>
                        <button type="button" id="turnLeft" class="turnLeft"
>Întoarce la Stânga</button>
                        <button          type="button"          id="turnRight"
class="turnRight" >Întoarce la Dreapta</button>
                        <button    type="button"    id="stop"    class="stop"
>Stop</button>



</body>
</html>
```

- ***server*.js**

```
var http=require('http');
```

```
văr bodyParser = require ("body-parser");
văr enableDestroy=require('server-destroy');
văr jsonfile=require('jsonfile');
văr datetime = require('node-datetime');
var net=require('net');
var Promise=require('promised-io/promise');

văr myTcpConnection;
var command='';
văr pastTime=datetime.create();

        văr deferred=Promise.defer();
        myTcpConnection = deferred.promise;


var server = net.createServer(function(connection) {
        console.log('client connected');
        connection.on('end', function() {
                                console.log('client disconnected');
                                server.close();
        });

        //văr deferred=Promise.defer();
        //myTcpConnection = deferred.promise;
        deferred.resolve(connection);
});
server.listen(15002, function() {
        console.log('server is listening on 15002 port');
});
server.on('error',function(err){
        console.log("Eroare la server");
});


function handle_incoming_request(req, res){
        văr json_dată="";
        res.setHeader('Access-Control-Allow-Origin', '*');
        res.setHeader('Access-Control-Request-Method', '*');
        req.on(
                "readable",
                function(){
```

```
                        var d = req.read();
                        if(typeof(d)=='strâng'){
                                json_dată +=d;

                        }
                        else if(typeof d == 'object' && d instanceof Buffer)
                                json_dată +=d.toString("utf8");

                }
        );
        req.on(
                "end",
                function(){
                        var out ='';
                        if(!json_dată)
                                out= "I got no JSON";

                        else{
                                văr json;
                                try{
                                        json=JSON.parse(json_dată);
                                }catch(e){}
                                if(!json){
                                        out = "Invalid Json";
                                        console.log("Invalid Json");
                                        }
                                else {
                                        out = "VALID JSON"+json_dată;
                                        console.log(json.CEVA);
                                        console.log(json.direction);


myTcpConnection.then(function(myConnection){

myConnection.write(json.direction);

myConnection.pipe(myConnection);
                                        });

                                }
                            }
                        res.end(out);
```

```
                }
        );

}

var s = http.createServer(handle_incoming_request);
s.listen(15001);

s.on('connection', function(socket){
        console.log("New connection");
        socket.setTimeout(30*1000);
});
```

❖ **client.js**

```
var net = require('net');
var fs = require('fs');
văr jsonfile = require('jsonfile');
var file='/home/pi/nodetest/dată.json';
var x=0;

var client = new net.Socket();
client.connect(15002, 'gaiţă.studenţi.speed.pub.ro', function() {
      console.log('Connected');
      //client.write('Raspberry s-a conectat la server');

});

client.on('dată', function(dată) {
      x=x+1;
      obj={index:x,conn:dată}
      jsonfile.writeFile(file,dată, function (err) {
      console.error(err)
      })
      fs.writeFileSync('dată.txt',dată,'utf-8');
      if(Strâng(dată)=="right"){console.log("megereweetgsd ");}
      console.log("Received: "+ dată);
      // client.destroy(); // kill client after server's response
});


client.on('close', function() {
```

```
      console.log('Connection closed');
});
```

❖ **serial.py**

```
import smbus
import time

buş=smbus.SMBus(1)
adress=0x18
val=0
def writeNumber(value):
      buş.write_byte(adress,value)
      return -1
while(1):
      f=open("dată.txt","r")
      try:
            val=int(f.readline())
      excepţ:
            print "Not a number"
      print(val)
      time.sleep(0.2)
      writeNumber(val)
```

❖ **hexapod.c**

```
#include <ax12.h>
#include <Bioloid*Controller*.h>
#include <Commander.h>
#include "nuke.h"
#include <SharpIR.h>
#include <Wire.h>

#define SLAVE_ADDRESS 0x18
#define AX12_HEXAPOD
Commander command = Commander();
int multiplier;

#define RIPPLE_SPEED    1
#define AMBLE_SPEED     3
#define TRIPOD_SPEED    5

#ifdef AX12_HEXAPOD
```

```
#define TOP_SPEED      10
#endif
int walkv;
int rotate;
int serial;
int dist;
int walkh=0;
int lookh=0;
int i=1;
int count;
boolean Seen;

void setup(){
  pinMode(0,OUTPUT);
  // setup IK
  setupIK();
  gaitSelect(AMBLE_SMOOTH);
  // setup serial
  Serial.begin(9600);

  // wait, then check the voltage (LiPO safety)
  delay (1000);
  float voltage = (ax12GetRegister (1, AX_PRESENT_VOLTAGE, 1)) / 10.0;
  Serial.printţ ("System Voltage: ");
  Serial.printţ (voltage);
  Serial.println (" volts.");
  if (voltage < 10.0)
    while(1);

  // stand up slowly
  bioloid.poseSize = 18;
  bioloid.readPose();
  doIK();
  bioloid.interpolateSetup(1000);
  while(bioloid.interpolating > 0){
    bioloid.interpolateStep();
    delay(3);
  }
  multiplier = AMBLE_SPEED;

  Wire.begin(SLAVE_ADDRESS);
```

```
  Serial.println("READY");
}
void receiveData(int byteCount){
 while(Wire.available()){
  dist=Wire.read();
  Serial.printț("dată received:");
  Serial.println(dist);
}}


void loop(){
  Wire.onReceive(receiveData);
    serial=dist;
    if(serial==1){
      walkv=50;
      walkh=0;
      rotate = 0;
    }

    if(serial == 2){
      walkh=-50;
      walkv=0;
      rotate = 0;
    }
    if(serial == 3){
      walkh=50;
      walkv=0;
      rotate = 0;
    }
    if(serial==4){
      walkv=-50;
      walkh=0;
      rotate=0;
    }
    if(serial == 5){
      walkh=0;
      walkv=0;
      rotate = -50;
    }
    if(serial== 6){
```

```
      walkv=0;
      walkh=0;
      rotate= 50;
    }



    if(walkv > 5 || walkv < -5 ){
      Xspeed = (multiplier*walkv)/2;
    }
    else
    {
      Xspeed = 0;
    }
    if(walkh > 5 || walkh < -5 ){
    Yspeed = (multiplier*walkh)/2;
    }
    else
    {
     Yspeed = 0;
    }
    if( rotate > 5 || rotate < -5 ){
    Rspeed = -(rotate)/100.0;
    }
    else
    {
      Rspeed = 0;
    }
     if(bioloid.interpolating == 0){
    doIK();
    bioloid.interpolateSetup(tranTime);
  }
  // update joints
  bioloid.interpolateStep();
}
```

- ***servervideo.js***

```
var http    = require('http');
var express = require('express');
```

```
var Stream = require('./lib/raspivid');

var app  = express();

  //public website
app.use(express.static(__dirname + '/public'));


var server  = http.createServer(app);
var sys = new Stream(server);

server.listen(80);
```