

# UNIVERSITATEA TITU MAIORESCU

FACULTATEA: INFORMATICĂ

DEPARTAMENT: INFORMATICĂ

Programa de studii: INFORMATICĂ

DISCIPLINA: INTELIGENȚĂ ARTIFICIALĂ

## IA - Testul de evaluare nr. 8

Robot Umanoid – BIOLOID

Grupa	Numele și prenumele	Semnătură student	Notă evaluare

Data: \_\_\_\_ / \_\_\_\_ / \_\_\_\_

CS-I dr.ing.

Lucian Ștefăniță GRIGORE

Conf.dr.ing.

Iustin PRIESCU

Ș.L.dr.ing.

Dan-Laurențiu GRECU



## Cuprins

1.	INTRODUCERE .....	3
2.	ROBOTUL UMANOID BIOLOID .....	6
2.1	Hardware și asamblare.....	7
2.2	Sistemul de comandă și control .....	10
3.	METODA N-Human Modeling.....	14
3.1	ROS .....	14
3.2	GAZEBO .....	14
3.3	RViz.....	14
3.4	SAWYER .....	15
3.5	Conceptul de arhitectură MoveIt .....	15
4.	SOFTWARE .....	21
4.1	Software Bioloid.....	21
5.	BIBLIOGRAFIE.....	29

## 1. INTRODUCERE

Dezvoltarea roboților umanoizi bipedali a început cu peste 30 de ani în urmă. În ciuda numeroaselor eforturi de cercetare, controlul unui robot bipedic este încă o sarcină extrem de provocatoare în ceea ce privește adaptabilitatea, robustețea și stabilitatea [1.32]. Cele mai comune abordări sunt Modelul pendulului inversat liniar (LIPM - Linear Inverted Pendulum Model) și metoda Zero Moment Point (ZMP). Problema cu metoda LIPM este că este un sistem de fază non-minimă, care produce un răspuns inferior la un pas de intrare. Pentru a compensa acest efect nedorit, se procedează propunând o nouă metodă de control predictiv de model augmentat (AMPC - Augmented Model Predictive Control) bazată pe observație (Fig. 1-1). Împreună cu sistemul de feedback senzorial, această metodă reduce șocul produs de forțele de impact ale contactului cu solul și, de asemenea, poate îmbunătăți performanța generală a sistemului de urmărire ZMP în cazul unor perturbații externe. Roboții umanoizi sunt proiectați din materiale ușoare și pot fi modelați corect considerându-i ca fiind multi-body, unde fiecare componentă este un sistem „point-mass”.



**Fig. 1-1 Robotul umanoid BIOLOID.**

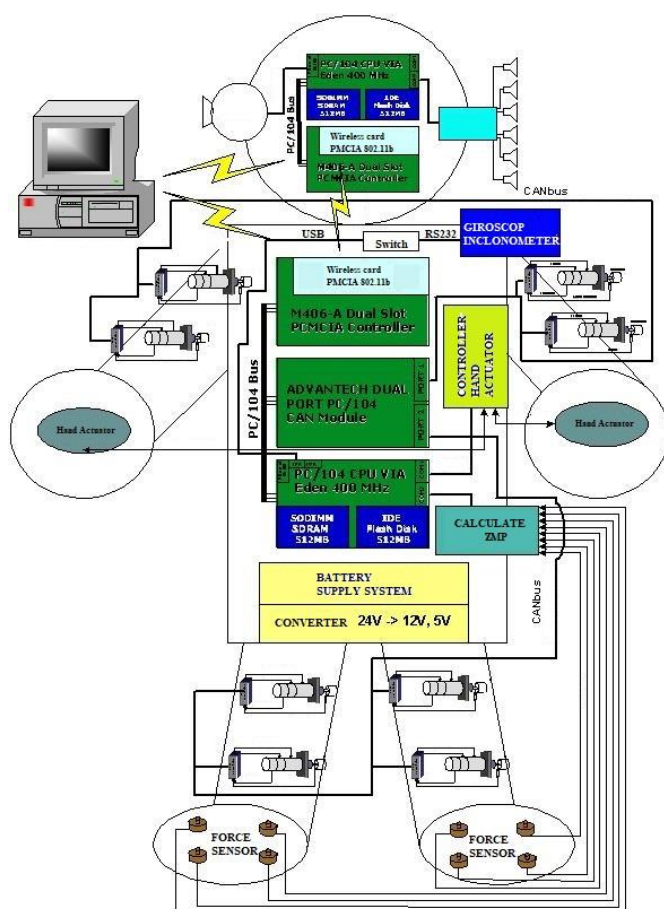
<https://www.generationrobots.com/en/401066-bioloid-premium-kit-robotis.html>

Spre deosebire de roboții industriali, un robot umanoid va interacționa cu o persoană în același spațiu de lucru. Pentru a putea interacționa cu un om și a funcționa într-un mod uman, sunt necesare abilități senzoriale motorii ale robotului. Robotul umanoid trebuie să fie echipat cu dispozitive de acționare și cu un

număr de senzori diferiți pentru a controla mișcările sale și pentru a monitoriza starea sa și pentru a evita coliziunile cu oamenii sau obiectele din mediul înconjurător.

Rezumând cerințele, trebuie îndeplinite următoarele:

- i) arhitectura hardware trebuie să respecte puterea de calcul necesară;
- ii) scalabilitate;
- iii) modularitate;
- iv) interfețe standardizate;
- v) eficiență energetică;
- vi) dimensiuni de gabarit rezonabile;
- vii) greutate cât mai mică;
- viii) asamblare ușoară plug&play.



**Fig. 1-2 Arhitectura hardware a unui robot umanoid.**

<http://roboticslab.uc3m.es/roboticslab/researchtopic/hardware-architecture-humanoids>

Scopul principal al sistemului de control al robotului umanoid este să îi asigure un mers stabil și să evite căderea în jos. Pentru a realiza acest lucru, se generează algoritmi de mișcare pentru fiecare articulație conform teoriei ZMP (Zero Moment Point), altfel spus robotul umanoid nu va cădea în jos atât timp cât ZMP se află în interiorul poligonului suport realizat de picioarele de susținere.

În Fig. 1-2 se regăsește o prezentare generală a unei posibile structuri hardware. Arhitectura prezentată este prevăzută cu un nivel ridicat de scalabilitate și modularitate prin împărțirea sistemului hardware în trei straturi de bază. Fiecare strat este reprezentat ca un controler centrat pe propriul layer de lucru, cum ar fi comunicațiile externe, supravegherea rețelei de mișcare și controlul general.

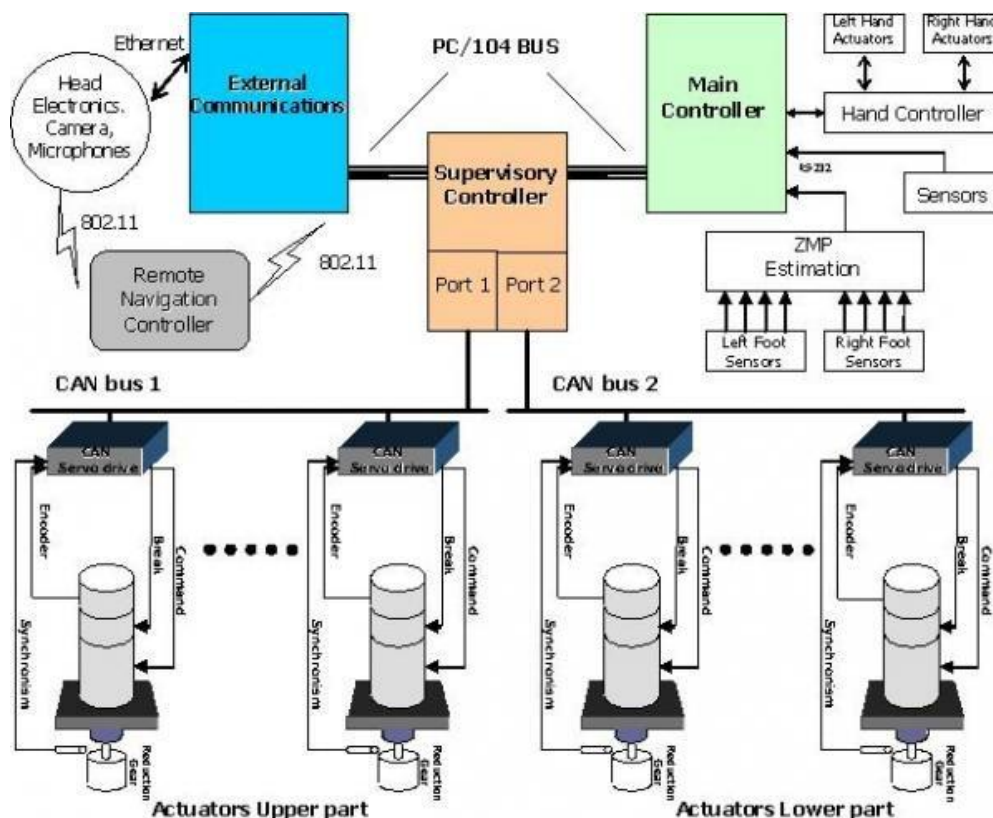


Fig. 1-3 Detaliu hardware robot umanoid – comandă actuatori.

<http://roboticslab.uc3m.es/roboticslab/researchtopic/hardware-architecture-humanoids>

Fiecare robot trebuie calibrat. Calibrare care se referă la modul în care controlerul și efectorii finali răspund la diverse provocări sau sarcini specifice.

Întrucât roboții umanoizi copiază din caracteristicile morfofuncționale ale unui om, este firesc ca răspunsul senzorilor la acțiunile mediului exterior să fie similare. De aceea una dintre cele mai simple și totodată complexe metode de calibrare constă în dezechilibrarea robotului.

## 2. ROBOTUL UMANOID BIOLOID

Robotul BIOLOID este un kit educațional care se poate configura în 5 configurații majore diferite (Fig. 2-1).



Fig. 2-1 Kit educațional BIOLOID.

<https://www.robotshop.com/products/robotis-bioloid-premium-humanoid-robot-kit>



ROBOTIS PREMIUM<sup>1</sup> este un kit de robot educațional de tip do-it-yourself care folosește servomotoare DC modulare. Este un pachet all-in-one pentru constructorii avansați de roboți care oferă următoarele. Kit-ul permite construirea unei varietăți de roboți, de la un robot cu 1 DOF până la un umanoid cu 18 DOF.

- Manual de asamblare și programe de bază incluse pentru 29 de roboți.
- Instrumente de asamblare și software de programare furnizate.
- Senzor giroscop, senzor de măsurare a distanței, senzor IR inclus.
- Include telecomandă și modul Bluetooth fără fir.
- Software de programare bazat pe GUI (RoboPlus)
- Comunicații digitale prin pachete cu topologie Daisy Chain pentru cablare ușoară.
- Mecanism de expansiune versatil pentru construirea ușoară a robotului.

## 2.1 Hardware și asamblare

Asamblarea robotului Bioloid Premium se face folosind kit-ul de asamblare primit împreună cu celelalte elemente de structură și motoare. Toate piesele mecanice se atașează între ele cu șuruburi și bușe de plastic. Componentele electronice folosesc cabluri cu conectori proprietari Robotis.

În acest capitol ne vom concentra pe construcția de tip umanoid a acestui robot. Există un manual care ajută la îndeplinirea sarcinii cu ușurință. Sunt necesare anumite aptitudini pentru a putea duce la bun sfârșit proiectul, printre care, cele mai importante fiind cunoașterea electronicii și automatizării, abilități mecanice dar și de programare basic.

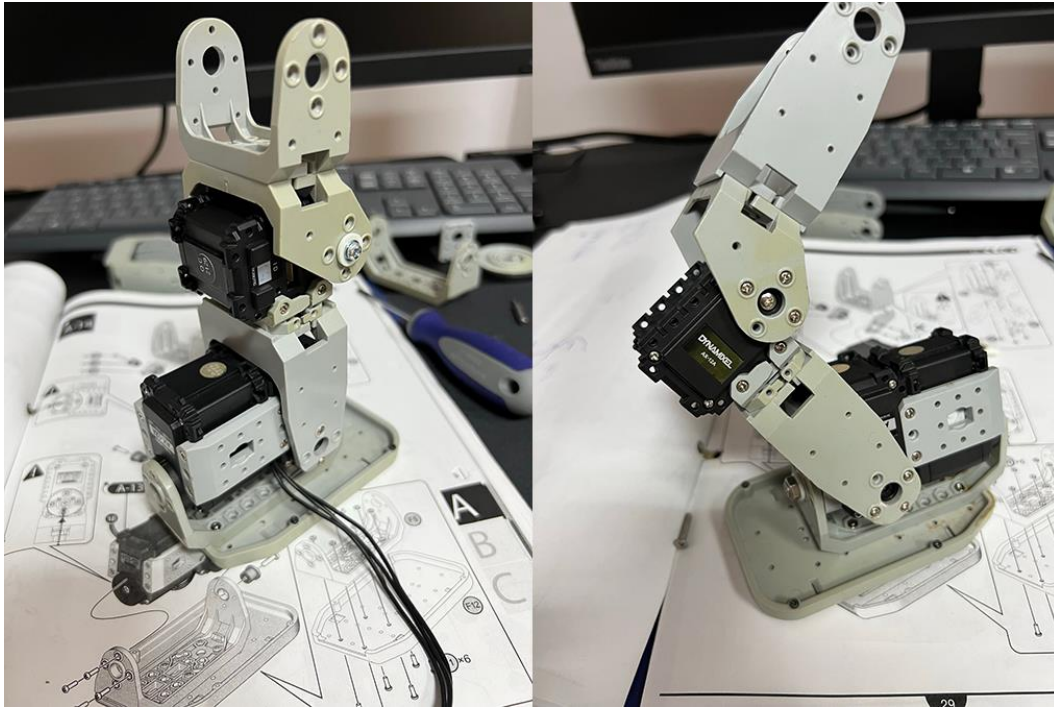
Asamblarea începe cu mâinile (Fig. 2-2), care conțin trei motoare fiecare și câteva elemente de susținere. Motoarele sunt legate în serie, controller-ul fiind punctul inițial de conexiune pentru fiecare dintre mâini.



**Fig. 2-2 Asamblare maini robot umanoid**

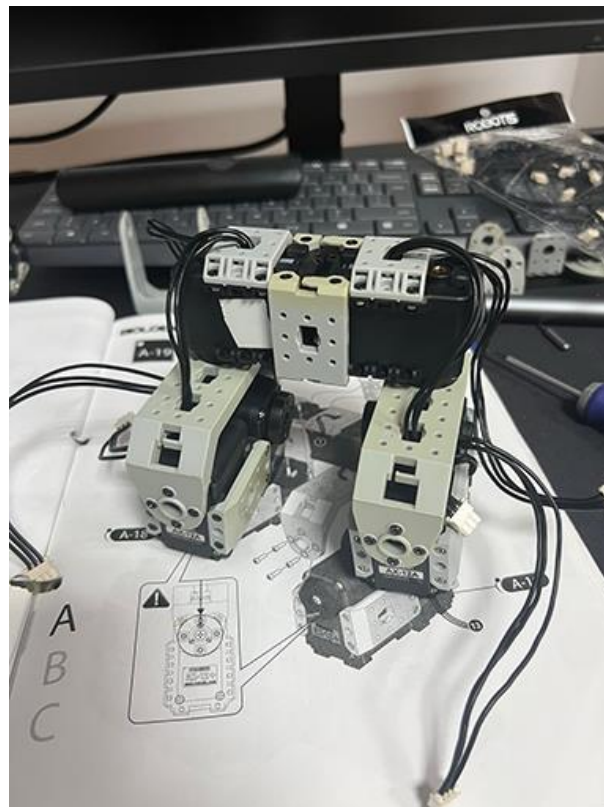
Următorul pas este asamblarea picioarelor (Fig. 2-3), care sunt făcute din mai multe componente. Sunt alcătuite tot din trei motoare și câteva piese de suport, printre care și talpa pe care acesta va sta. Motoarele sunt legate în serie, două dintre ele fiind parte componentă a tălpii și au rol de a stabili poziția robotului.

<sup>1</sup> <https://www.useabot.com/products/robotis-premium>



**Fig. 2-3 Asamblare picioare robot umanoid**

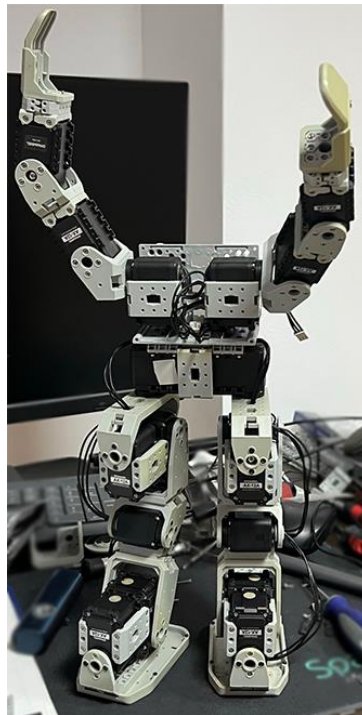
Trecem mai apoi la șolduri (Fig. 2-4), piesa de legatura între picioare și trunchi, din nou o piesă care stabilizează robotul. Este format din șase motoare legate în serie, iar de ele se vor atașa picioarele pe de-o parte, iar trunchiul pe partea cealaltă, de care se va atașa controller-ul și bateria.



**Fig. 2-4 Șolduri roboto umanoid**

Măinile se atașează de structura superioară a trunchiului și se conectează în serie la motoarele care compun șoldurile (Fig. 2-5). Ele vor fi ranforsate cu placa pe care va fi montat controller-ul și bateria.





**Fig. 2-5 Măinile atașate trunchiului robotului umanoid**

Robotul este complet (Fig. 2-6) după ce se atașează capacele decorative pentru piept și cap.



**Fig. 2-6 Robot asamblat**

## 2.2 Sistemul de comandă și control

Sistemul de comandă și control, RoboPlus, este proprietar Robotis, și include mai multe software-uri diferite, GUI și terminal, care permit manipularea completă a robotului prin încărcarea și modificarea programelor care definesc mișcarea și răspunsul robotului.

Comunicația cu controller-ul robotului se face prin cablu, folosind protocolul serial RS232. Pentru aceasta, Robotis au pus la dispoziție un cablu și un adaptor RS232->USB (Fig. 2-7).

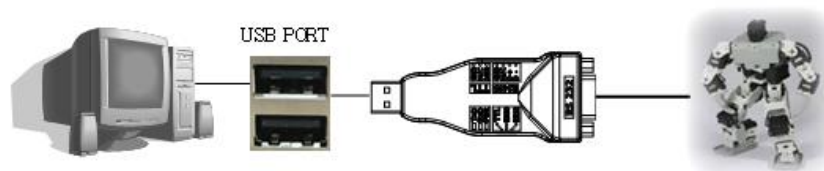


Fig. 2-7 Conectare prin serial la Bioloid

RoboPlus este interfața principală din care putem să accesăm celelalte soft-uri, RoboPlus Task, RoboPlus Manager, RoboPlus Task dar și programele mai avansate, RoboPlus Terminal și Dynamixel Wizard. Acesta din urmă este programul de control și configurare al motoarelor Dynamixel care compun partea de mișcare a robotului. Ele pot fi configurate în totalitate, de la ce nivel de cuplu motor să producă, până la numărul de identificare (ID) și chiar dacă să se comporte ca o roată sau ca o articulație.

RoboManager (Fig. 2-8) este interfața GUI prin care putem accesa celelalte programe care ne ajută să configurăm robotul în cele mai mici detalii.

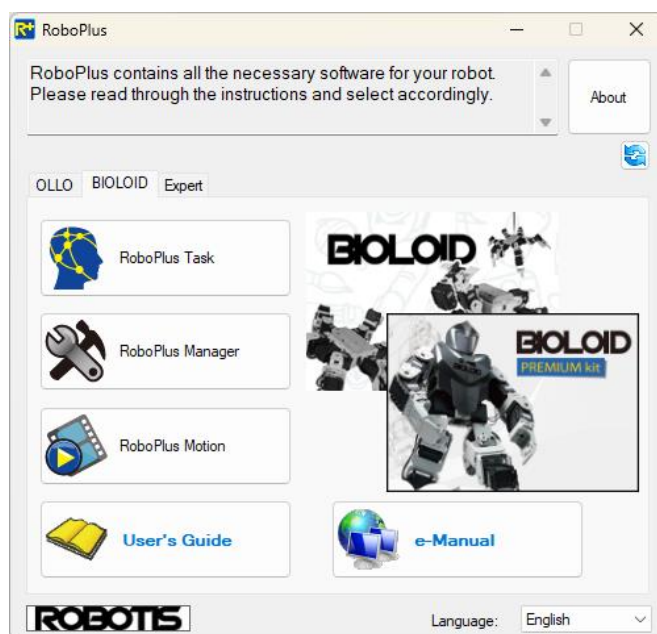
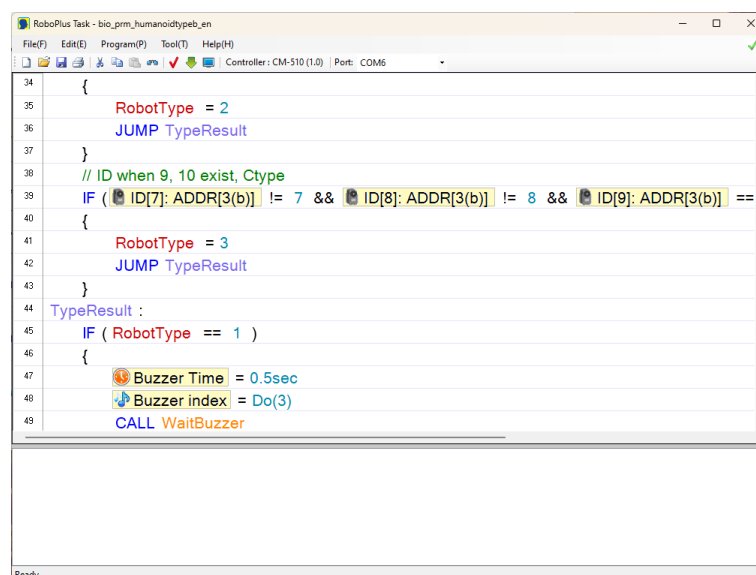


Fig. 2-8 RoboPlus Manager

De aici, putem să accesăm RoboPlus Task, care ne permite să scriem programul care va defini logica pe care o va urma robotul. Limbajul de programare este unul proprietar Robotis dar bazat pe formă pe limbajul

C. Pe lângă programele default pe care Robotis le pune la dispoziția utilizatorilor, folosind RoboPlus Task, putem crea propriul cod prin care robotul poate face orice mișcare ne dorim, prin programarea fiecărui motor individual (Fig. 2-9).



```

34 {
35     RobotType = 2
36     JUMP TypeResult
37 }
38 // ID when 9, 10 exist, Ctype
39 IF ( ID[7]:ADDR[3(b)] != 7 && ID[8]:ADDR[3(b)] != 8 && ID[9]:ADDR[3(b)] ==
40 {
41     RobotType = 3
42     JUMP TypeResult
43 }
44 TypeResult :
45 IF ( RobotType == 1 )
46 {
47     Buzzer Time = 0.5sec
48     Buzzer index = Do(3)
49     CALL WaitBuzzer

```

Fig. 2-9 RoboPlus Task

Împreună cu RoboPlus Task, se folosește software-ul complementar RoboPlus Motion (Fig. 2-10) în care se vor stabili exact mișcările pe care robotul le va face. Fișierul de Task va face ca trecerea între mișcări să fie făcută cât mai liniar prin corelarea acestora cu cele din fișierul de Motion.

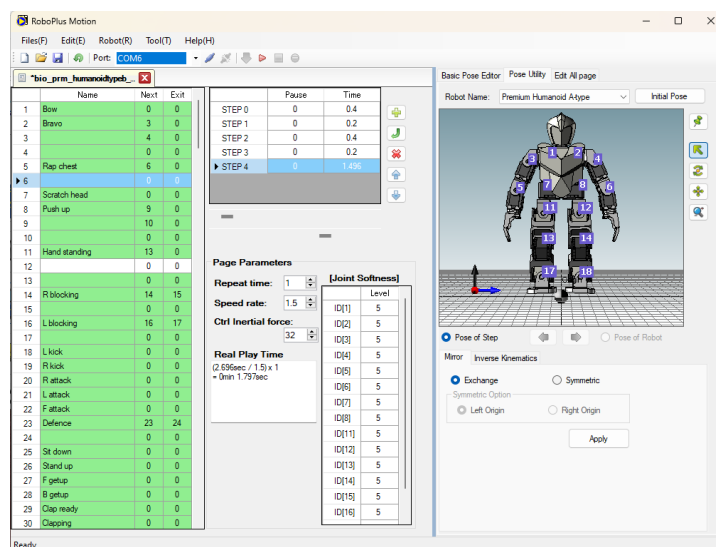
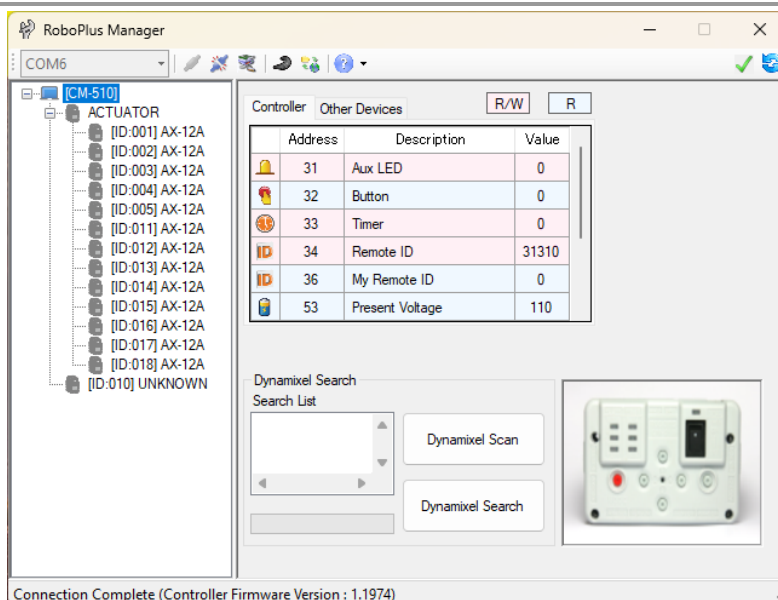


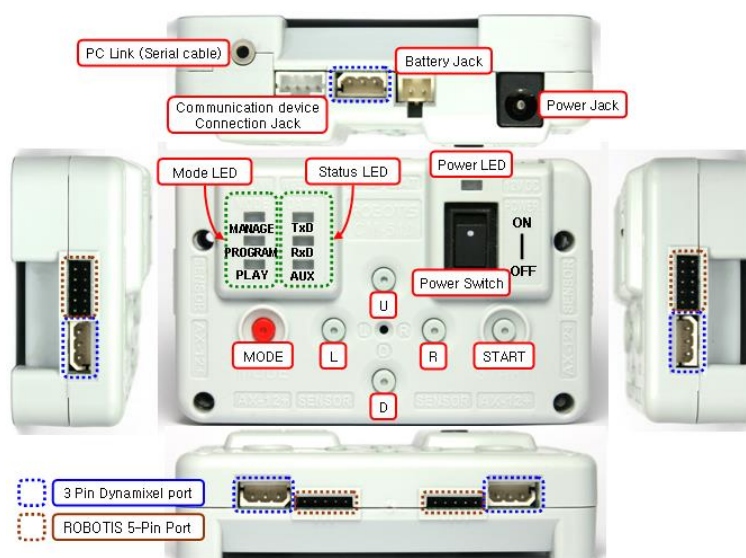
Fig. 2-10 RoboPlus Motion

În RoboPlus Manager, putem configura, face update sau verifica controller-ul robotului dar și testa componentele robotului, cum ar fi motoarele Dynamixel sau operațiunile controller-ului (Fig. 2-11).



**Fig. 2-11 RoboPlus Manager**

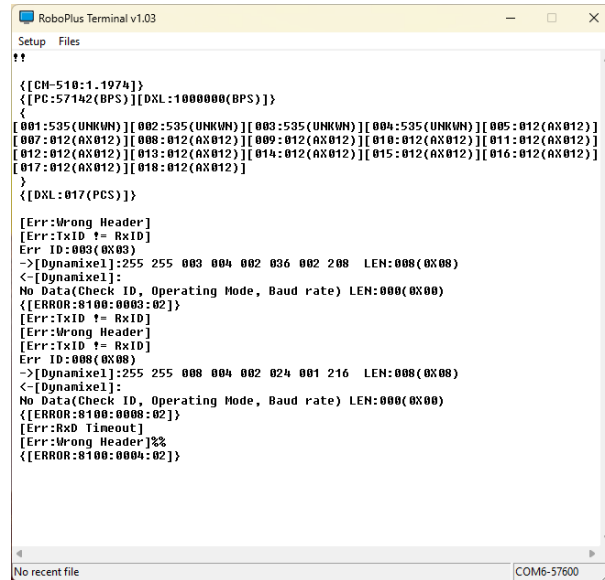
Controller-ul folosit de noi (Fig. 2-12), CM-510, este configurat pentru a fi cât mai versatil, programarea acestuia făcându-se în Embedded C. Acesta are un procesor ATmega2561<sup>2</sup>, un chip low-power pe 8-biți, cu arhitectura RISC. Are 256kb de memorie flash, 8kb de SRAM, 4kb EEPROM și o viteză de 16 MIPS la 16MHz, având nevoie de 4.5-5.5V.



**Fig. 2-12 CM-510, controller Bioloid**

Pentru utilizatorii avansați, care vor să modifice în detaliu structura software și hardware a robotului Bioloid, Robotis pune la dispoziție RoboPlus Terminal (Fig. 2-13), unde prin linii de cod, se poate comunica direct cu bootloader-ul controller-ului și scrie fișiere de firmware customizate pe nevoile utilizatorului.

<sup>2</sup> <https://www.microchip.com/en-us/product/ATmega2561>



```
RoboPlus Terminal v1.03
Setup Files

↑↑

{[CH-510:1.1974]}
{[PC:57142(BPS)][DXL:1000000(BPS)]}
{
  [001:595(UNKNOWN)][002:595(UNKNOWN)][003:595(UNKNOWN)][004:595(UNKNOWN)][005:012(AX012)]
  [007:012(AX012)][008:012(AX012)][009:012(AX012)][010:012(AX012)][011:012(AX012)]
  [012:012(AX012)][013:012(AX012)][014:012(AX012)][015:012(AX012)][016:012(AX012)]
  [017:012(AX012)][018:012(AX012)]
}
{[DXL:017(PCS)]}

[Err:Wrong Header]
[Err:TxID != RxID]
Err ID:003(0X03)
->[Dynamixel]:255 255 003 004 002 036 002 208 LEN:000(0X00)
<-[Dynamixel]:
No Data(Check ID, Operating Mode, Baud rate) LEN:000(0X00)
{[ERROR:8100:0003:02]}
[Err:TxID != RxID]
[Err:Wrong Header]
[Err:TxID != RxID]
Err ID:008(0X08)
->[Dynamixel]:255 255 008 004 002 024 001 216 LEN:000(0X00)
<-[Dynamixel]:
No Data(Check ID, Operating Mode, Baud rate) LEN:000(0X00)
{[ERROR:8100:0008:02]}
[Err:Rx0 Timeout]
[Err:Wrong Header]3%
{[ERROR:8100:0004:02]}
```

Fig. 2-13 RoboPlus Terminal



### 3. METODA N-Human Modeling<sup>3</sup>

N-Human Modeling se referă la modelarea generative a comportamentului multimodal. Utilizează Git LFS pentru a putea lucra cu fișiere foarte mari.

AI Robot Challenge – viitorul aparține fuziunii AI și a roboticii pentru a permite roboților inteligenți, să devină colaborativi și să poată efectua misiuni de asistență socială.

#### 3.1 ROS

ROS (Robot Operating System) este un middleware bazat pe robotică, licențiat sub licență BSD open source. Deși ROS nu este un sistem operativ, acesta oferă biblioteci, hardware, drivere de dispozitiv, transmiterea mesajelor, gestionarea pachetelor și alte instrumente pentru a ajuta în dezvoltarea de software pentru aplicații în domeniul roboților umanoizi mobili. De asemenea, sprijină colaborarea prin Repositories.

ROS este un cadru distribuit de procese (cunoscut sub numele de Noduri), care permite adăugarea individuală a executabilelor, ceea ce face cadrul foarte modular. Aceste procese pot fi grupate în pachete, care pot fi ușor distribuite.

Deși există unele kituri Robotics disponibile pe piață, ROS devine rapid noul standard atât pentru robotica industrială, cât și pentru cercetare, deoarece integrează hardware și software pentru orice tip de aplicații în robotică.

ROS susține sistemele de tip Unix, în special Ubuntu și Mac OS X, însă comunitatea a adăugat suport pentru platformele Fedora, Gentoo, Arch Linux și alte platforme Linux.

#### 3.2 GAZEBO

Gazebo este un instrument de simulare 3D robot care se integrează perfect cu ROS (adică serverul Gazebo poate funcționa într-un mediu ROS). Gazebo permite construirea de scenarii 3D pe calculatorul cu roboți, folosind obstacole și alte obiecte. Acest lucru permite ca roboții să fie testați în diverse scenarii complexe sau periculoase fără a afecta robotul real. De cele mai multe ori este mai rapid și mai eficient să fie rulat un simulator în loc să fie executat întregul scenariu pe un robot real. Utilizează motoarele ODE (Open Dynamics Engine) pentru mișcări realiste.

Gazebo are două componente principale: serverul, care acționează ca un *back-end* pentru a calcula toată logica pentru scenariile de testare și un client, care acționează ca un *front-end* grafic. Acest lucru este foarte util deoarece, uneori, se pot efectua testele fără UI pentru accelerarea procesului de execuție. Gazebo a fost folosit pentru a simula robotul atlas pentru Virtual Robotics Challenge (precursorul DARPA), care a avut drept provocare realizarea unui software pentru misiunile de salvare umanitare.

#### 3.3 RViz

RViz este un vizualizator 3D open-source pentru sistemul de operare ROS. Utilizează datele senzorilor și marcasele personalizate de vizualizare pentru a dezvolta capacitățile robotului într-un mediu 3D.

Caracteristici:

- **planificarea mișcării:** procesul de descompunere a unei sarcini de mișcare dorite în mișcări discrete care satisfac constrângerile de mișcare și optimizează unele aspecte ale mișcării;

---

<sup>3</sup> <https://github.com/StanfordASL/NHumanModeling>

- **detectarea obiectelor:** vizualizarea și recunoașterea obiectelor care utilizează aparatul foto, de exemplu recunoașterea cuburilor de culori diferite;
- **calibrare:** pentru calibrarea camerei geometrice în vederea estimării parametrilor interni ai camerei care afectează procesarea imaginilor;
- **Debugging;**
- RViz widget – vizualizare;
- **redare 3D stereo:** când se utilizează 2 camere conectate pentru a înregistra imagini 3D, acesta afișează o vedere diferită față de fiecare ochi, astfel încât scena pare să aibă profunzime.

RViz oferă un instrument CLI care vă permite să executați scripturi Python sau C++ cu controale.

### 3.4 SAWYER

Sawyer este o soluție integrată de robot colaborativ (aka cobot) concepută cu viziune încorporată, clești inteligenți swappable și control de forță de înaltă rezoluție. Scopul robotului este acela de a automatiza sarcinile industriale repetitive specifice. Acesta vine cu un braț care are un dispozitiv de prindere care poate fi înlocuit cu ușurință de una dintre opțiunile disponibile din kitul de prindere ClickSmart.

Caracteristici:

- dispune de senzori de cuplu foarte sensibili încorporați în fiecare articulație, lucru care permite să fie controlată forța în cazul în care introducerea delicată a pieselor este critică sau eventual să fie folosită forța dacă este necesar. Poate fi manevrat în spații strânse și are o lungime activă de până la 1260 mm;
- dispune de un sistem integrat de vizualizare folosit pentru poziționarea robotului și permite atașarea de sisteme externe de vizionare precum camerele de luat vederi;
- este foarte rapid pentru a implementa cât mai multe piese, deoarece este construit pe sistemul plug & play cu senzori integrați.

### 3.5 Conceptul de arhitectură MoveIt

Conceptul de arhitectură MoveIt!<sup>4</sup> (Fig. 3-1) se poate observa că este un model de arhitectură MoveIt! de nivel înalt pentru un nod primar, denumit generic move\_group. Acest nod servește ca integrator, astfel:

- extrage toate componentele individuale și le adună împreună pentru a oferi un set de acțiuni și servicii ROS (Robot Operating Systems);
- permite beneficiarilor să utilizeze acțiunile și serviciile ROS.

Interfața cu utilizatorul permite accesarea acțiunilor și serviciilor oferite de move\_group, astfel:

- pachetul move\_group\_interface oferă o interfață pentru C++, ușor de instalat pentru move\_group;
- pachetul moveit\_commander oferă o interfață pentru **Python**;
- **interfața grafică (GUI)** – folosește pluginul Planning Motion pentru RViz (ROS Visualizer - vizualizatorul ROS).

<sup>4</sup> <http://moveit.ros.org/documentation/concepts/>

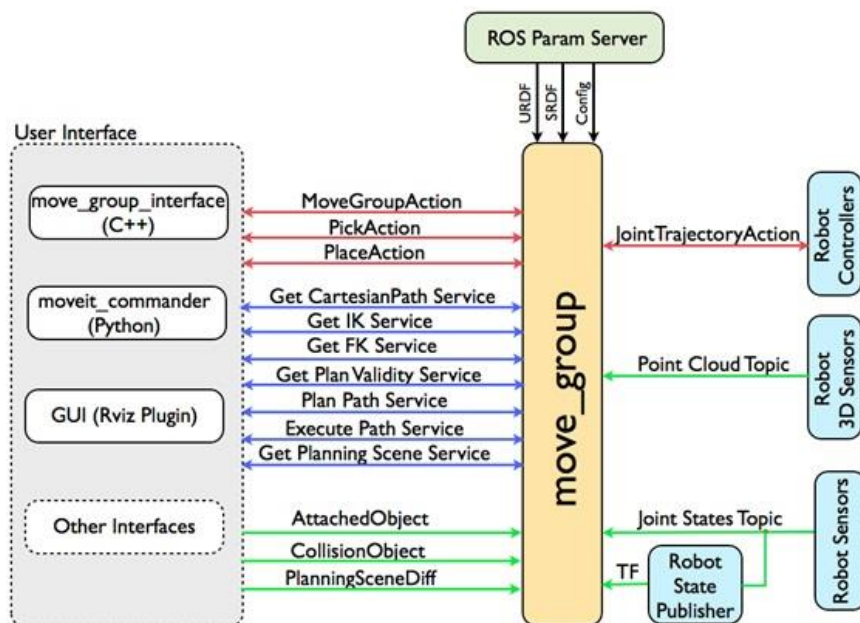


Fig. 3-1 Model de arhitectură MoveIt.

<https://moveit.ros.org/documentation/concepts/fig 1>

Interfața robotului: *move\_group* reprezintă, în această situație, calea de dialog cu robotul prin subiecte și acțiuni ROS<sup>5</sup>. Comunicarea are rolul de a obține informațiile de stare actuale (poziții ale îmbinărilor, hărți de puncte și alte date prelevate de la senzorii robotului) care să poată fi transmise ulterior operatorilor.

Informații standard de stare: *move\_group* comunică cu topicul *joint\_states* (stări comune) pentru a determina starea de fapt (actuală) – adică poziția exactă a elementelor robotului. Modelul *move\_group* comunică simultan cu toți efectorii cum ar fi, pentru: mobilitate, apucare, prelucrare, imagine etc. Modelul *move\_group* nu are rol de configurare al propriului editor, aceasta fiind făcută de către utilizator.

Transformarea informațiilor: *move\_group* monitorizează transformarea informațiilor utilizând biblioteca ROS TF, unde biblioteca TF (Transform Frame) permite utilizatorului să urmărească multiple cadre (frame-uri) de coordonate în același timp. Acest lucru permite nodului să obțină informații globale despre poziția robotului (printre altele). De exemplu, stiva de navigație ROS va publica transformarea dintre cadrul hărții și cadrul de bază al robotului la TF. Modelul *move\_group* poate:

- folosi TF pentru a descoperi această transformare pentru uz intern. Modelul *move\_group* comunică doar cu TF;
- publica informații TF de la robot, doar când rulează subrutina *robot\_state\_publisher*.

Interfața de comandă: *move\_group* comunică cu operatorii robotului folosind interfața *FollowJointTrajectoryAction* (Fig. 3-2). Aceasta este o interfață de acțiune ROS. Un server pe robot

<sup>5</sup> <http://www.ros.org/>

trebuie să *deservească* această acțiune – acest server nu este furnizat de *move\_group* în sine, el satisface un singur client din acest punct de vedere, adică al controlului asupra robotului.

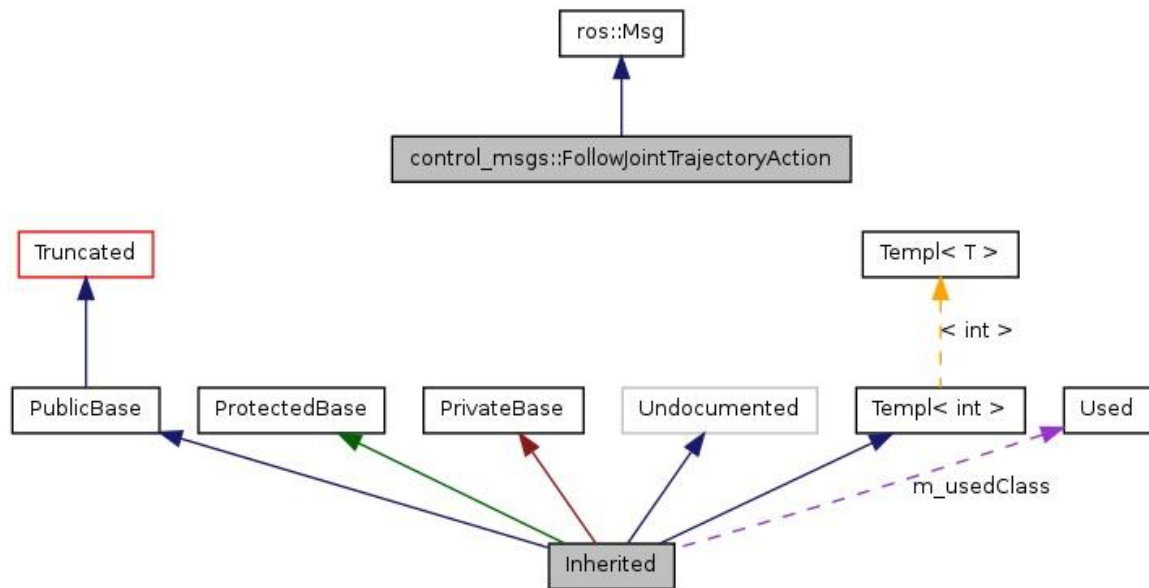


Fig. 3-2 FollowJointTrajectoryAction.

[http://mirror-ap.docs.ros.org/groovy/api/robot\\_mechanism\\_controllers/html/graph\\_legend.html](http://mirror-ap.docs.ros.org/groovy/api/robot_mechanism_controllers/html/graph_legend.html)

/\*! Invisible class because of truncation \*/

```
class Invisible { };
```

/\*! Truncated class, inheritance relation is hidden \*/

```
class Truncated : public Invisible { };
```

/\* Class not documented with doxygen comments \*/

```
class Undocumented { };
```

/\*! Class that is inherited using public inheritance \*/

```
class PublicBase : public Truncated { };
```

/\*! A template class \*/

```
template<class T> class Templ { };
```

/\*! Class that is inherited using protected inheritance \*/

```
class ProtectedBase { };
```

/\*! Class that is inherited using private inheritance \*/

```

class PrivateBase { };

/*! Class that is used by the Inherited class */
class Used { };

/*! Super class that inherits a number of other classes */
class Inherited : public PublicBase,
                  protected ProtectedBase,
                  private PrivateBase,
                  public Undocumented,
                  public Templ<int>
{
private:
    Used *m_usedClass;
};

```

Planificarea scenariului: *move\_group* folosește monitorizarea planificării scenelor pentru a menține o planificare pe monitorul de lucru (Fig. 3-3), afișând atât reprezentarea elementelor din exterior dar și cele interne ale robotului. Starea robotului poate include orice obiecte transportate de robot care sunt considerate a fi atașate rigid la acesta.

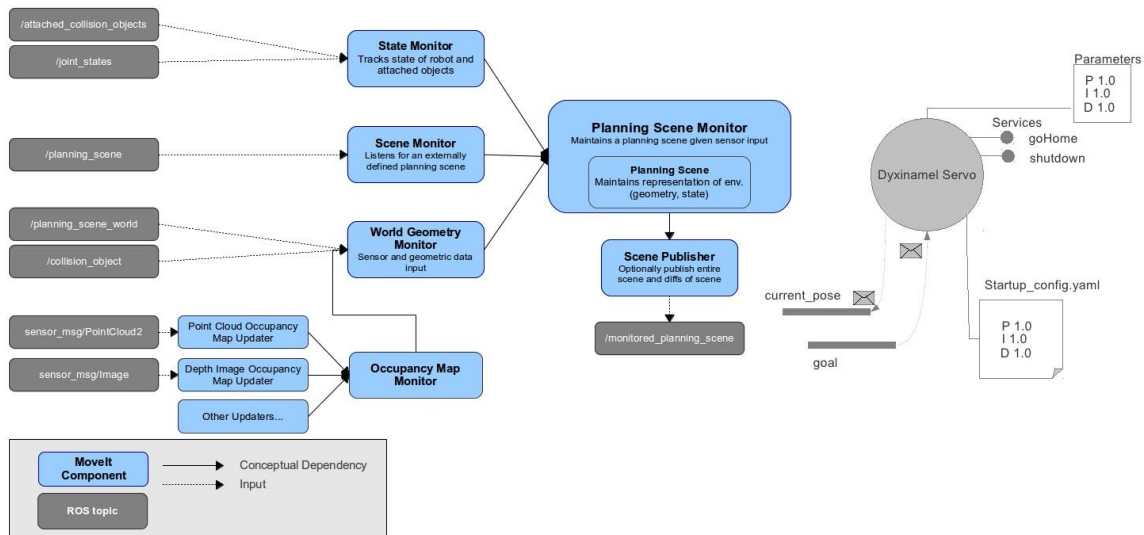


Fig. 3-3 Planning Scene Monitor Diagram - MoveIt! și parametri distribuiți ROS yaml.

Extinderea Capabilităților: *move\_group* este structurat pentru a fi ușor extins din punct de vedere al capabilităților individuale cum ar fi: alegerea locului de acționare, cinematica, planificarea mișcării.



Toate acestea sunt de fapt implementate ca plugin-uri separate cu o clasă de bază comună. Plugin-urile pot fi configurate folosind ROS printr-un set de parametri „[ROS yaml](#)”, care reprezintă o programare distribuită și care utilizează biblioteca [ROS pluginlib](#). Utilizatorii nu trebuie să configureze pluginurile [move\\_group](#), deoarece acestea sunt configurate automat în fișierele de lansare generate de MoveIt!

### **Planificarea mișcării (Motion Planning)**

[Pluginul pentru planificarea mișcării](#) ([The Motion Planning Plugin](#)) (Fig. 3-4): Modelul [MoveIt!](#) lucrează cu planificatorii de mișcare printr-o interfață de *plugin*. Acest lucru permite modelului [MoveIt!](#) să comunice cu diferite planificatoare de mișcare, care pot accesa mai multe biblioteci, ceea ce va conduce la o facilitate de extindere a modelului [MoveIt!](#). Interfața cu planificatorul de mișcări se realizează printr-o acțiune ROS sau un serviciu (oferit de nodul [move\\_group](#)). Planificatorii de mișcare implicați pentru [move\\_group](#) sunt configurați folosind OMPL (Open Motion Planning Library) și [MoveIt!](#) este utilizat ca interfață la OMPL.

[Cererea planului de mișcare](#) ([The Motion Plan Request](#)): Prin cererea planului de mișcare specifică clar ce se dorește de la planificatorul mișcării. În mod obișnuit i se cere planificatorului de mișcări să miște un braț într-o altă locație (într-un spațiu al articulației) sau ca efectorul final să ocupe o nouă poziție. Tot aici sunt verificate posibilele coliziuni (inclusiv autocoliziuni). Se poate atașa un obiect la efectorul final (sau la orice parte a robotului). Aceasta permite planificatorului de mișcare să țină seama de mișcarea obiectului în timpul planificării căilor. De asemenea, se pot specifica constrângeri pentru ca planificatorul de mișcare să verifice – constrângerile încorporate, fiind vorba despre *constrângerile din punct de vedere cinematic*, astfel:

- [Position Constraints](#) – constrângeri de poziție, care restricționează poziția unei legături într-o regiune a spațiului;
- [Orientation Constraints](#) – constrângeri de orientare, care restricționează orientarea unei legături, astfel încât, să se încadreze în limitele circulare, înălțime sau înclinare specificate;
- [Visibility Constraints](#) – constrângeri de vizibilitate, care restricționează un punct de pe o legătură, astfel încât, să se afle în conul de vizibilitate pentru un anumit senzor;
- [Joint Constraints](#) – constrângeri standard prin restrângerea unei îmbinări între două valori prestabilite;
- [User-specified Constraints](#) – constrângeri specificate de utilizator, care se pot specifica ca fiind constrângeri proprii printr-un apel invers definit de utilizator.

[Rezultatul planului de mișcare](#) ([The Motion Plan Result](#)): Nodul [move\\_group](#) va genera o traiectorie dorită ca răspuns la solicitarea planului de mișcare. Această traiectorie va muta brațul (sau orice grup de articulații) în locația dorită. Rezultatul în urma acțiunii din [move\\_group](#) este o traiectorie și nu doar o cale. Nodul [move\\_group](#) va utiliza vitezele și accelerațiile maxime dorite (dacă este specificat acest lucru) pentru a genera o traiectorie care respectă constrângerile de viteză și de accelerare la nivelul comun.

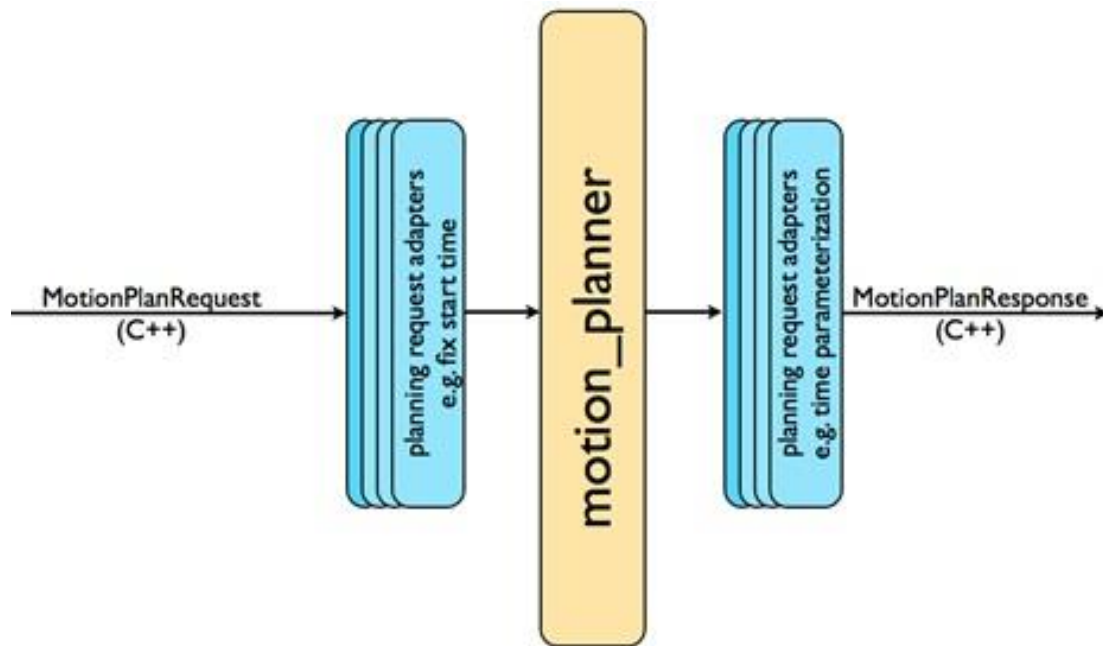


Fig. 3-4 Motion Planning.

<https://moveit.ros.org/documentation/concepts/> fig. 2

## 4. SOFTWARE

### 4.1 Software Bioloid<sup>6</sup>

```

"""This module provides the Bus class which knows how to talk to Bioloid
devices, and the BusError exception which is raised when an error is
encountered.

"""

import pyb

from bioloid import packet
from bioloid.dump_mem import dump_mem
from bioloid.log import log

class BusError(Exception):
    """Exception which is raised when a non-successful status packet is received."""

    def __init__(self, error_code, *args, **kwargs):
        super(BusError, self).__init__(self, *args, **kwargs)
        self.error_code = error_code

    def get_error_code(self):
        """Retrieves the error code associated with the exception."""
        return self.error_code

    def __str__(self):
        return "Rcvd Status: " + str(packet.ErrorCode(self.error_code))

class Bus:
    """The Bus class knows the commands used to talk to bioloid devices."""

    SHOW_NONE = 0

```

```
SHOW_COMMANDS = (1 << 0)
```

```
SHOW_PACKETS = (1 << 1)
```

<sup>6</sup> <https://github.com/dhylands/bioloid3/blob/master/bioloid/bus.py>

```

def __init__(self, serial_port, show=SHOW_NONE):
    self.serial_port = serial_port
    self.show = show

def action(self):
    """Broadcasts an action packet to all of the devices on the bus.
    This causes all of the devices to perform their deferred writes
    at the same time.

    """
    if self.show & Bus.SHOW_COMMANDS:
        log('Broadcasting ACTION')
        self.fill_and_write_packet(packet.Id.BROADCAST, packet.Command.ACTION)

def fill_and_write_packet(self, dev_id, cmd, data=None):
    """Allocates and fills a packet. data should be a bytearray of data
    to include in the packet, or None if no data should be included.
    """
    packet_len = 6
    if data is not None:
        packet_len += len(data)
    pkt_bytes = bytearray(packet_len)
    pkt_bytes[0] = 0xff
    pkt_bytes[1] = 0xff
    pkt_bytes[2] = dev_id
    pkt_bytes[3] = 2    # for len and cmd
    pkt_bytes[4] = cmd
    if data is not None:
        pkt_bytes[3] += len(data)
        pkt_bytes[5:packet_len - 1] = data

```

```

    pkt_bytes[-1] = ~sum(pkt_bytes[2:-1]) & 0xff
    if self.show & Bus.SHOW_PACKETS:
        dump_mem(pkt_bytes, prefix=' W', show_ascii=True, log=log)
    self.serial_port.write_packet(pkt_bytes)

def ping(self, dev_id):
    """Sends a PING request to a device.

```

Returns true if the device responds successfully, false if a timeout occurs, and raises a bus.Error for any other failures.

raises a BusError for any other failures.

"""

```
self.send_ping(dev_id)
```

```
try:
```

```
    self.read_status_packet()
```

```
except BusError as ex:
```

```
    if ex.get_error_code() == packet.ErrorCode.TIMEOUT:
```

```
        return False
```

```
    raise ex
```

```
return True
```

```
def read(self, dev_id, offset, num_bytes):
```

```
    """Sends a READ request and returns data read.
```

Raises a bus.Error if any errors occur.

"""

```
self.send_read(dev_id, offset, num_bytes)
```

```
pkt = self.read_status_packet()
```

```
return pkt.params()
```

```
def read_status_packet(self):
```

```
    """Reads a status packet and returns it.
```

Rasises a bioloid.bus.BusError if an error occurs.

"""

```
pkt = packet.Packet(status_packet=True)
```

```
while True:
```

```
    # start = pyb.micros()
```

```
    byte = self.serial_port.read_byte()
```

```
    if byte is None:
```

```
        if self.show & Bus.SHOW_COMMANDS:
```

```
            log('TIMEOUT')
```

```
        if self.show & Bus.SHOW_PACKETS:
```



```

        dump_mem(pkt.pkt_bytes, prefix=' R', show_ascii=True, log=log)
        raise BusError(packet.ErrorCode.TIMEOUT)
    err = pkt.process_byte(byte)
    if err != packet.ErrorCode.NOT_DONE:
        break
    if err != packet.ErrorCode.NONE:
        err_ex = BusError(err)
        if self.show & Bus.SHOW_COMMANDS:
            log(err_ex)
        if self.show & Bus.SHOW_PACKETS:
            dump_mem(pkt.pkt_bytes, prefix=' R', show_ascii=True, log=log)
        raise err_ex
    err = pkt.error_code()
    if self.show & Bus.SHOW_COMMANDS:
        log('Rcvd Status: { } from ID: { }'.format(packet.ErrorCode(err), pkt.dev_id))
    if self.show & Bus.SHOW_PACKETS:
        dump_mem(pkt.pkt_bytes, prefix=' R', show_ascii=True, log=log)
    if err != packet.ErrorCode.NONE:
        raise BusError(err)
    return pkt

```

```

def reset(self, dev_id):
    """Sends a RESET request.

    Raises a bus.Error if any errors occur.
    """
    self.send_reset(dev_id)
    if dev_id == packet.Id.BROADCAST:
        return packet.ErrorCode.NONE
    pkt = self.read_status_packet()
    return pkt.error_code()

def scan(self, start_id=0, num_ids=32, dev_found=None, dev_missing=None):
    """Scans the bus, calling devFound(self, dev) for each device
    which responds, and dev_missing(self, dev) for each device
    which doesn't.

```

```

Returns true if any devices were found.
"""

end_id = start_id + num_ids - 1
if end_id >= packet.Id.BROADCAST:
    end_id = packet.Id.BROADCAST - 1
some_dev_found = False
for dev_id in range(start_id, end_id + 1):
    if self.ping(dev_id):
        some_dev_found = True
        if dev_found:
            dev_found(self, dev_id)
    else:
        if dev_missing:
            dev_missing(self, dev_id)
return some_dev_found

def send_ping(self, dev_id):
    """Sends a ping to a device."""

```

```

if self.show & Bus.SHOW_COMMANDS:
    log('Sending PING to ID {}'.format(dev_id))
self.fill_and_write_packet(dev_id, packet.Command.PING)

def send_read(self, dev_id, offset, num_bytes):
    """Sends a READ request to read data from the device's control
    table.
    """
    if self.show & Bus.SHOW_COMMANDS:
        log('Sending READ to ID {} offset 0x{:02x} len {}'.format(
            dev_id, offset, num_bytes))
    self.fill_and_write_packet(dev_id, packet.Command.READ, bytearray((offset, num_bytes)))

def send_reset(self, dev_id):
    """Sends a RESET command to the device, which causes it to reset the
    control table to factory defaults.
    """
    if self.show & Bus.SHOW_COMMANDS:
        if dev_id == packet.Id.BROADCAST:

```

```

        log('Broadcasting RESET')
    else:
        log('Sending RESET to ID {}'.format(dev_id))
    self.fill_and_write_packet(dev_id, packet.Command.RESET)

```

```
def send_write(self, dev_id, offset, data, deferred=False):
```

```

    """Sends a WRITE request if deferred is False, or REG_WRITE
    request if deferred is True to write data into the device's
    control table.

```

```

    data should be an array of ints, or a bytearray.

```

```

    Deferred writes will occur when and ACTION command is broadcast.

```

```

    """

```

```
if self.show & Bus.SHOW_COMMANDS:
```

```
    cmd_str = 'REG_WRITE' if deferred else 'WRITE'
```

```
    if dev_id == packet.Id.BROADCAST:
```

```
        log('Broadcasting {} offset 0x{:02x} len {}'.format(cmd_str, offset, len(data)))
```

```
    else:
```

```
        log('Sending {} to ID {} offset 0x{:02x} len {}'.format(cmd_str, dev_id, offset, len(data)))
```

```
    cmd = packet.Command.REG_WRITE if deferred else packet.Command.WRITE
```

```
    pkt_data = bytearray(len(data))
```

```
    pkt_data[0] = offset
```

```
    pkt_data[1:] = data
```

```
    self.fill_and_write_packet(dev_id, cmd, pkt_data)
```

```
def sync_write(self, dev_ids, offset, values):
```

```

    """Sets up a synchroous write command.

```

```

    dev_ids should be an array of device ids.

```

```

    offset should be the offset that the data will be written to.

```

```

    values should be an array of bytearrays. There should be one bytearray
    for each dev_id, and each bytearray should be of the same length.

```

```

    raises ValueError if the dimensionality of values is incorrect.

```

```
"""
```

```
if self.show & Bus.SHOW_COMMANDS:
```

```
    ids = ', '.join(['{}'.format(id) for id in dev_ids])
```

```
    log('Sending SYNC_WRITE to IDs {} offset 0x{:02x} len {}'.format(ids, offset, len(values[0])))
```

```
num_ids = len(dev_ids)
```

```
if num_ids != len(values):
```

```
    raise ValueError('len(dev_ids) = {} must match len(values) = {}'.format(num_ids, len(values)))
```

```
bytes_per_id = len(values[0])
```

```
param_len = num_ids * (bytes_per_id + 1) + 2
```

```
data = bytearray(param_len)
```

```
data[0] = offset
```

```
data[1] = bytes_per_id
```

```
data_idx = 2
```

```
for id_idx in range(num_ids):
```

```
    if len(values[id_idx]) != bytes_per_id:
```

```
        raise ValueError('len(values[{}]) not equal {}'.format(id_idx, bytes_per_id))
```

```
    data[data_idx] = dev_ids[id_idx]
```

```
    data_idx += 1
```

```
    data[data_idx:data_idx + bytes_per_id] = values[id_idx]
```

```
    data_idx += bytes_per_id
```

```
self.fill_and_write_packet(packet.Id.BROADCAST, packet.Command.SYNC_WRITE, data)
```

```
def write(self, dev_id, offset, data, deferred=False):
```

```
    """Sends a WRITE request if deferred is False, or a REG_WRITE
```

```
    request if deferred is True. Deferred writes will occur when
```

```
    and ACTION command is broadcast.
```

```
    data should be an array of ints, or a bytearray.
```

```
    Raises a bus.Error if any errors occur.
```

```
    """
```

```
self.send_write(dev_id, offset, data, deferred)
```

```
if dev_id == packet.Id.BROADCAST:
```

```
    return packet.ErrorCode.NONE
```

```
pkt = self.read_status_packet()
```

```
return pkt.error_code()
```





## 5. BIBLIOGRAFIE

- 1.1. Argentieri, S., P. Danès, și P. Soueres. 15 January 2007. „Modal analysis based beamforming for nearfield or farfield speaker localization in robotics.” *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Beijing, China: IEEE. 866-871. Accesat 03 03, 2019. doi: 10.1109/IROS.2006.281739.
- 1.2. Argentieri, Sylvain, P. Danès, și P. Souères. 2015. „A Survey on Sound Source Localization in Robotics: from Binaural to Array Processing Methods.” *Elsevier Ltd. - Computer Speech and Language* 34: 87-112. Accesat 03 02, 2019. doi:10.1016/j.csl.2015.03.003.
- 1.3. Arsenio, Artur M. 2004. „Teaching Humanoid Robots like Children: Explorations into the World of Toys and Learning Activities.” *International Journal of Humanoid Robotics* (WSPC - World Scientific Publishing Company) 21. Accesat 02 23, 2019. <https://www.researchgate.net/publication/228969815>.
- 1.4. Artem Rudskyy, M. Sc. 11 April 2012. *Design und Steuerungsaspekte humanoider Roboter*. Thesis, Fakultät für Elektrotechnik und Informationstechnik, Otto-von-Guericke-Universität Magdeburg, Magdeburg, Germany: Universitätsbibl. Otto von Guericke University Library, Magdeburg, Germany., 140. Accesat 02 18, 2019. doi:<http://dx.doi.org/10.25673/5448>.
- 1.5. Benichoux, V., A. Brown, K. L. Anbuhl, și D. J. Tollin. 2017. „Representation of Multidimensional Stimuli: Quantifying the Most Informative Stimulus Dimension from Neural Responses.” *The Journal of Neuroscience : The Official Journal of the Society for Neuroscience* 37 (31): 7332–7346. Accesat 03 04, 2019. doi:10.1523/JNEUROSCI.0318-17.2017.
- 1.6. Blow, M.P., K. Dautenhahn, A. Appleby, C. L. Nehaniv, și D. Lee. September 6-8 2006. „Perception of robot smiles and dimensions for human-robotinteraction design.” *Proceedings of the 15th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN06)*. Hatfield, UK: IEEE. 469-474. Accesat 02 28, 2019. doi:10.1109/ROMAN.2006.314372.
- 1.7. Brooks, R. A., C. Breazeal (Ferrell), R. Irie, C.C. Kemp, M. Marjanović, B. Scassellati, și M. M. Williamson. July 26-30, 1998. „Alternative Essences of Intelligence.” *Proceedings of 15th National Conference on Artificial Intelligence (AAAI-98)*. Monona Terrace, Madison, Wisconsin, USA. 961 - 968. Accesat 02 24, 2019. <http://www.aaai.org/Papers/AAAI/1998/AAAI98-136.pdf>.
- 1.8. Dallali, H., P. Kormushev, N. G. Tsagarakis, și D. G. Caldwell. 2014. „Can active impedance protect robots from landing impact?” *2014 IEEE-RAS International Conference on Humanoid Robots*. Madrid, Spain: IEEE. 1022-1027. Accesat 02 12, 2019. doi:10.1109/HUMANOIDS.2014.7041490.
- 1.9. Grimes, J. A., și J. W. Hurst. 2012, 23-26 july. „The design of atrias 1.0 a unique monopod, hopping robot.” *Fifteenth International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*. Baltimore, MD, USA: HFSP grant number RGY0062/2010 and DARPA grant number W91CRB-11-1-0002. 548-554. Accesat 02 12, 2019. doi:10.1142/9789814415958\_0071.
- 1.10. Huang, J., T. Supaongprapa, I. Terakura, F. Wang, N. Ohnishi, și N. Sugie. 1999. „A model based sound localization system and its application to robot navigation.” *Robotics and Autonomous Systems* (Elsevier B.V.) 27 (4): 199-209. Accesat 03 03, 2019. doi:10.1016/S0921-8890(99)00002-0.
- 1.11. Hudson, Ralph E., Kung Yao, și Joe C. Chen. 2003. „Acoustic Source Localization and Beamforming: Theory and Practice.” *EURASIP journal on advances in signal processing* (Springer International Publishing) 2003 (4): 359-370. Accesat 03 03, 2019. doi:10.1155/S1110865703212038.
- 1.12. Hutter, M., C. D. Remy, M. A. Hoepflinger, și R. Siegwart. 2011. „ScarLETH: Design and control of a planar running robot.” *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. San Francisco, CA, USA: IEEE. Accesat 02 12, 2019. doi:10.1109/IROS.2011.6094504.
- 1.13. Isaiah, A., T. Vongpaisal, A. J. King, și D. E. H. Hartley. 2014. „Multisensory Training Improves Auditory Spatial Processing following Bilateral Cochlear Implantation.” 34 (33): 11119 – 11130. Accesat 03 04, 2019. doi:10.1523/JNEUROSCI.4767-13.2014.
- 1.14. Itoh, Kazuko, Hiroyasu Miwa, Hideaki Takanobu, și Atsuo Takanishi. July–August 2005. „Application of neural network to humanoid robots—development of co-associative memory model.”

- Editor D. Prokhorov, D. Levine, F. Ham și W. Howell. Neural Networks. Elsevier. 666-673. Accesat 11 06, 2018. doi:10.1016/j.neunet.2005.06.021.
- 1.15. Liljegren, G. E., și E. L. Foster. Dec. 18, 1990. „Back Projected Image Using Fiber Optics.” Technical report. Accesat 11 06, 2018. <http://www.freepatentsonline.com/4978216.html>.
- 1.16. Lincoln, P., G. Welch, A. Nashel, A. Ilie, și H. Fuchs. October 19 - 22, 2009. „Animatronic Shader Lamps.” *8th IEEE International Symposium on Mixed and Augmented Reality - ISMAR*. Orlando: IEEE Computer Society Washington, DC, USA ©2009 . 27-33. Accesat 11 06, 2018.
- 1.17. Macpherson, E. A., și A. T. Sabin. 2007. „Binaural weighting of monaural spectral cues for sound localization.” *The Journal of the Acoustical Society of America* 121 (6): 3677-3688. Accesat 03 04, 2019. doi:10.1121/1.2722048.
- 1.18. Nakadai, K., H. G. Okuno, și T. Mizumoto. 2017. „Development, Deployment, and Applications of Robot Audition Open Source Software HARK.” *Journal of Robotics and Mechatronics* 29 (1): 16-25. Accesat 03 02, 2019. doi:10.20965/jrm.2017.p0016.
- 1.19. Nakadai, K., K. Hidai, H. G. Okuno, și H. Kitano. 11-15 May 2002. „Real-time speaker localization and speech separation by audio-visual integration.” *IEEE International Conference on Robotics and Automation*. Washington, DC, USA, USA: IEEE. Accesat 03 04, 2019. doi:10.1109/ROBOT.2002.1013493.
- 1.20. Pang, Wee Ching, Burhan, și Gerald Seet. October 3-5, 2012. „Design Considerations of a Robotic Head for Telepresence Applications.” *International Conference on Intelligent Robotics and Applications*. Montreal, QC, Canada: Intelligent Robotics and Applications. 131-140. Accesat 11 06, 2018. doi:10.1007/978-3-642-33503-7\_14.
- 1.21. Prattand, G. A., și M. M. Williamson. 1995. „Series Elastic Actuators.” *Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots'*. Pittsburgh, PA, USA: IEEE/RSJ International Conference. 399-406. Accesat 02 12, 2019. doi:10.1109/IROS.1995.525827.
- 1.22. Raschka, Sebastian. September 23rd 2015. *Python Machine Learning*. 1st edition. Packt Publishing. Accesat 07 05, 2018. [python-machine-learning-book/docs/equations/pymle-equations.pdf](http://python-machine-learning-book/docs/equations/pymle-equations.pdf).
- 1.23. Rascon, C., I. Meza, G. F. Pineda, L. Salinas, și L. Pineda. 2015. „Integration of the Multi-DOA Estimation Functionality to Human-Robot Interaction.” *International Journal of Advanced Robotic Systems* 12 12 (2): 14. Accesat 03 04, 2019. doi:10.5772/59993.
- 1.24. Rascon, C., și L. Pineda. 12 September 2013. „Multiple Direction-of-Arrival Estimation for a Mobile Robotic Platform with Small Hardware Setup.” Cap. 16 în *IAENG Transactions on Engineering Technologies*, de H. K. Kim și et al. , editor H. K. Kim, S-L Ao, M. A. Amouzegar și B. B. Rieger, 718. DaeGu, Korea: Springer Science & Business Media Dordrecht. Accesat 03 04, 2019. doi:10.1007/978-94-007-6818-5\_16.
- 1.25. Rascon, Caleb, și Ivan Meza. 2017. „Localization of sound sources in robotics: A review .” *Robotics and Autonomous Systems* (Elsevier) 96: 184-210. Accesat 03 02, 2019. doi:10.1016/j.robot.2017.07.011.
- 1.26. Rodemann, T., G. Ince, F. Joubin, și C. Goerick. 14 October 2008. „Using binaural and spectral cues for azimuth and elevation localization.” *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*. Nice, France: IEEE. 2185– 2190. Accesat 03 04, 2019. doi:10.1109/IROS.2008.4650667.
- 1.27. Sato, M., A. Sugiyama, O. Hoshuyama, N. Yamashita, și Y. Fujita. 2005. „Near-Field Sound-Source Localization Based on a Signed Binary Code.” *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences* 88-A (8): 2078-2086. Accesat 03 03, 2019. doi:10.1093/ietfec/e88-a.8.2078.
- 1.28. Sinapayen, L., K. Nakamura, K. Nakadai, H. Takahashi, și T. Kinoshita. September 2015. „Consensus-based sound source localization using a swarm of micro-quadrocopters.” Editor Honda Research Institute. *The 33rd Annual Conference of the Robotics Society of Japan (RSJ2015)*. Robotics Society of Japan. 4. Accesat 03 03, 2019. [http://jglobal.jst.go.jp/detail?JGLOBAL\\_ID=201502205724307397](http://jglobal.jst.go.jp/detail?JGLOBAL_ID=201502205724307397).
- 1.29. Smith, M. G., K. B. Kim, și D. J. Thompson. 17-18 October 2007. „Noise source identification using microphone arrays.” *Autumn Conference of the Institute of Acoustics 2007: Advances in Noise and Vibration Engineering*. London, UK: Institute of Acoustics ( IOA ). 120-127. Accesat 03 04, 2019.

- 1.30. Valin, J., F. Michaud, J. Rouat, și D. Letourneau. 27-31 Oct. 2003. „Robust Sound Source Localization Using a Microphone Array on a Mobile Robot.” *IEEE/RSJ International Conference on Intelligent Robots and Systems - IROS 2003*. Las Vegas, NV, USA, USA: IEEE. 6. Accesat 03 03, 2019. doi:10.1109/IROS.2003.1248813.
- 1.31. Wang, L., și A. Cavallaro. 2017. „Microphone-Array Ego-Noise Reduction Algorithms for Auditory Micro Aerial Vehicles.” *Sensor Journal (IEEE)* 17 (8): 2447-2455. Accesat 03 03, 2019. doi:10.1109/JSEN.2017.2669262.
- 1.32. Wee, Teck-Chew, Astolfi Alessandro, și Ming Xie. 2013. „The Design and Control of a Bipedal Robot with Sensory Feedback.” *International Journal of Advanced Robotic Systems*. doi:10.5772/56572 .
- 1.33. Youssef, K., S. Argentieri, și J. L. Zarader. 3-7 Nov. 2013. „A learning-based approach to robust binaural sound localization.” *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*. Tokyo, Japan: IEEE. 2927–2932. Accesat 03 04, 2019. doi:10.1109/IROS.2013.6696771.