

Universitatea POLITEHNICA din Bucureşti
Facultatea de Electronică, Telecomunicații și Tehnologia Informației

Sistem intelligent bazat pe roboți colaborativi

Lucrare de licență

Prezentată ca cerință parțială pentru obținerea
titlului de *Inginer*
în domeniul *Calculatoare și Tehnologia Informației*
programul de studii *Ingineria Informației*

Conducători științifici

Prof. Dr. Ing. Corneliu Burileanu

As. Univ. Drd. Ing. Ana-Antonia Neacșu

Absolvent

Andrei-Daniel Dedu

Anul 2020

Declarație de onestitate academică

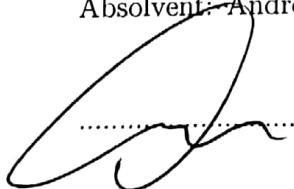
Prin prezenta declar că lucrarea cu titlul *Sistem intelligent bazat pe roboți colaborativi*, prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității "Politehnica" din București ca cerință parțială pentru obținerea titlului de *Inginer* în domeniul Inginerie Electronică și Telecomunicații/ Calculatoare și Tehnologia Informației, programul de studii *Ingineria Informației* este scrisă de mine și nu a mai fost prezentată niciodată la o facultate sau instituție de învățământ superior din țară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referințe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și fac referință la sursă. Reformularea în cuvinte proprii a textelor scrise de către alți autori face referință la sursă. Înțeleg că plagiatul constituie infracțiune și se sancționează conform legilor în vigoare.

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

București, Iunie 2020.

Absolvent: Andrei-Daniel Dedu



TEMA PROIECTULUI DE DIPLOMĂ
 a studentului **DEDU I. Andrei-Daniel , 444A**

1. Titlul temei: Sistem intelligent bazat pe roboți colaborativi

2. Descrierea contribuției originale a studentului (în afara părții de documentare) și specificații de proiectare:

Această lucrare își propune să pună în evidență comportamentul roiului - un comportament de cooperare observat în mod obișnuit în natură și deseori expus de insecte mici, păsări etc. - atrage mișcarea colectivă și unificată a multor indivizi. Inspirate din roiurile din natură, roiurile robot pot prezenta comportamente aparent complexe. Aceștia pot interacționa cu vecinii sau cu mediul local pe baza unui set de reguli, sau pot învăța diferite comportamente folosind algoritmi de inteligență artificială.

Scopul acestui proiect este de a mobiliza un set de 10 roboți (de tip Kilobot) să realizeze în mod cooperativ o singură sarcină globală. Fiecare robot luat individual are un comportament autonom; prin urmare, obiectivul principal este realizarea comunicației eficiente dintre roboți ce se va face prin intermediul unui controller infraroșu cu microcontroler ce va fi proiectat și dezvoltat. Tot în cadrul proiectului se va dezvolta și un circuit de alimentare a roboților în paralel.

Ulterior, sistemul va fi validat într-o aplicație reală ce va pune în evidență comportamentul de roi și capacitatea de cooperare a roboților.

3. Resurse folosite la dezvoltarea proiectului:

Arduino IDE, Atmel Studio, KiloGUI, placă de dezvoltare Arduino, microcontrolerul ATmega 328P, interfata RS232.

4. Proiectul se bazează pe cunoștințe dobândite în principal la următoarele 3-4 discipline:

Arhitectura microprocesoarelor, Microcontrolere, Programarea calculatoarelor, CID.

5. Proprietatea intelectuală asupra proiectului aparține: U.P.B.

6. Data înregistrării temei: 2020-02-20 01:59:54

Conducător(i) lucrare,
 Prof. dr. ing. Corneliu BURILEANU

Student,

Director departament,
 Prof. dr. ing Sever PASCA

Decan,
 Prof. dr. ing. Mihnea UDREA

Cod Validare: **eb90e5f349**

Cuprins

Lista figurilor	ix
Lista tabelelor	x
Lista acronimelor	xi
Introducere	1
Motivație	1
Aplicabilitate	1
Obiective	2
Starea Artei	3
1. Descrierea componentelor hardware	9
1.1. Kiloboți	9
1.2. Overhead Controller	10
1.3. Standard RS-232	11
1.4. KiloGUI	12
1.5. MAX232	14
1.6. Atmel ATmega328p	14
1.7. Comunicare IR	16
1.8. Placă cu LED-uri IR	17
1.9. Încărcător	17
2. Setup experimental	21
2.1. Spațiul de lucru	21
2.2. Calibrare	22
2.3. Iluminare	23
2.4. Rază de acțiune	24
3. Descrierea componentelor software și a aplicațiilor	27
3.1. Atmel Studio	27
3.2. AVRdude	27
3.3. Kilolib	28
3.4. Software Overhead Controller	32
3.5. Bootloader	33

3.6. Aplicații	34
Concluzii	49
Concluzii generale	49
Contribuții personale	49
Dezvoltări ulterioare	50
Bibliografie	51

Lista figurilor

1.	Îmbunătățirea timpilor de așteptare în trafic folosind un sistem colaborativ [1]	3
2.	Formare de inele folosind sisteme de tip reacție-difuzie [2]	4
3.	Segmentare în dungi folosind sisteme de tip reacție-difuzie [2]	4
4.	Exemplu de auto-asamblare. [3]	5
5.	Forme realizate folosind algoritmul de auto-asamblare. [3]	6
6.	Dezbaterea opinilor între roboți prin metoda votului majoritar. [4]	7
7.	Experiment ce are ca scop luarea unei decizii la nivel de colectiv. [4]	7
8.	Obiectul cu inele ce urmează a fi deplasat. [5]	8
9.	Mișcarea de translație a unui obiect. [5]	8
1.1.	Un kilobot și componentele ce îl alcătuiesc. [6]	10
1.2.	Schematic Overhead Controler	12
1.3.	Overhead Controler	13
1.4.	Partea din spate a Overhead Controler-ului	14
1.5.	KiloGUI	15
1.6.	Amplasare Overhead Controler. [6]	16
1.7.	Schematic Placă cu LED-uri IR	17
1.8.	6 Plăci cu LED-uri IR	18
1.9.	Conectori Placă cu LED-uri IR	18
1.10.	Încărător kiloboți	19
2.1.	Organizarea spațiului de lucru	21
2.2.	Suprafața de lucru dreaptă	22
2.3.	Submeniu de calibrare	23
2.4.	Arena kiloboților din laboratorul de robotică al universității din Sheffield. [7] . .	24
2.5.	Măsurători ale distanței de comunicare între kiloboți	25
3.1.	Împrăștierea kiloboților	36
3.2.	Pașii de funcționare al algoritmului de Strângere	37
3.3.	Algoritmul Strângere cu 1 robot de tip <i>beacon</i>	42
3.4.	Algoritmul Strângere cu 2 roboți de tip <i>beacon</i>	42
3.5.	Algoritmul Orbitare cu 3 roboți de tip <i>beacon</i>	43
3.6.	Algoritmul ce aliniaază 5 roboți într-un sir indian	48

Lista tabelelor

1.1.	Componentele ce alcătuiesc un kilobot.	10
1.2.	Parametrii Atmel ATmega328p. [8]	15
2.1.	Valori de calibrare pentru diversi roboti.	23
2.2.	Intensitatea luminoasă în funcție de diversele surse de iluminare [7]	24
2.3.	Evoluția razei de transmisiune în funcție de sursa de lumină folosită.	25

Lista acronimelor

ADC = convertor analogic-digital (din engleză "Analog-to-digital converter")

AVR = din engleză "Advanced RISC"

EEPROM = din engleză "Electrically Erasable Programmable Read-Only Memory"

GND = masă sau punct de potențial 0. (din engleză "Ground")

ID = identificator

IDE = din engleză "Integrated development environment"

IR = infraroșu (infrared în engleză)

ISP = din engleză "In-system programming"

LED = diodă emițătoare de lumină (din engleză "Light-emitting Diode")

Li-Ion = Litiu-Ion

PWM = din engleză "Pulse-width Modulation"

RGB = modelul de culoare roșu, verde, albastru (din engleză "Red, Green, Blue")

RISC = calculator cu set de instrucțiuni redus (din engleză "Reduced Instruction Set Computer")

ROM = din engleză "Read-only memory"

SMT = din engleză: "Surface-mount technology"

SPI = din engleză: "Serial Peripheral Interface"

THT = din engleză: "Through-hole technology"

Introducere

Motivatie

Această lucrare își propune să studieze sistemele automate ce se bazează pe colaborarea unor componente hardware individuale în realizarea unui scop comun bine-definit. Ideea de colaborare între indivizi este inspirată din natură, unde se regăsește cu precădere în comportamentul a diverse grupuri de insecte și mamifere. Colaborarea poate fi observată și în evoluția omenirii, aceasta reprezentând cheia dezvoltării rasei umane.

Întrucât interacțiunile inter-umane reprezintă procese complexe, ce depind de mulți factori externi, majoritatea sistemelor colaborative de pe piață prezintă o analogie cu diferite grupuri de insecte, deoarece este mai ușor de înțeles, de analizat și de reproducus. Cea mai evidentă corespondență este aceea ce asociază sistemele colaborative cu o colonie de furnici. În mod asemănător, în fiecare din cele două cazuri fiecare individ luat separat nu poate duce o sarcină complexă la bun-sfârșit, însă influența și puterea lor de a acționa crește odată cu numărul lor. O colonie de furnici este capabilă să divizeze cantitatea de muncă și cu ajutorul comunicării între indivizi poate rezolva probleme complexe cum ar fi găsirea celui mai scurt drum pentru hrană, construirea unui mușuroi sau apărarea colectivului. Acestea nu au o privire de ansamblu și nu există un coordonator care să decidă mișcarea fiecarui individ. Cu toate acestea, fiecare individ este capabil de a lua decizii bazate pe comportamentul observat în jurul său. Astfel, o colonie poate fi asemănătoare cu un organism singular. Dacă decizile fiecarui individ sunt corecte și nu au un comportament nonconformist, cu timpul, se va instaura un mecanism de gândire colectivă. Studiul acestui comportament colaborativ și încercarea de a îl reproduce este motivată de dorința de a înțelege aceste fenomene ale naturii și de a le putea folosi pentru diverse aplicații ce pot reprezenta un beneficiu pentru societate.

Aplicabilitate

Acste tipuri de sisteme au o aplicabilitate vastă. Printre domeniile în care astfel de sisteme pot fi aplicate se numără domeniul medical, gestionarea traficului, studiul organismelor și al sistemelor întâlnite în natură, dar și simularea diferitelor scenarii în care un grup mare de indivizi de dimensiune redusă ar putea fi folositor, ca de exemplu căutarea de supraviețuitori după diverse dezastre naturale.

Un studiu mai elaborat asupra aplicabilității unor astfel de sisteme în domeniul medical a fost publicat în cadrul Institutului de Chimie Macromoleculară din Iași [9].

Studiul descrie un sistem colaborativ format din nanoroboți ce reprezintă o mulțime de mașini controlabile alcătuite din componente, care prin dimensiunile reduse pot interacționa

sau chiar penetra membrana unei celule. Acești nanoroboți se pot autopropulsa, pot detecta și semnală o problemă și sunt capabili de a procesa informația [9]. O astfel de abordare impune totuși mai multe constrângeri. Unele dintre aceste constrângeri sunt asociate cu sursa de alimentare a nanoroboților, metoda de propulsie, metoda de control a unui astfel de sistem și interferențele electromagnetice. În ciuda acestor dezavantaje, un astfel de sistem poate deveni esențial în transferul ţintit al medicamentelor, tratarea tumorilor la nivel celular și diagnosticarea timpurie a unor boli precum diabet și cancer.

O altă aplicație în care implementarea unui astfel de sistem poate aduce un beneficiu major este gestionarea traficului rutier. În ziua de astăzi tot mai multe persoane preferă să locuiască în orașe mari și dezvoltate datorită oportunităților prezente într-un astfel de mediu. Acest fenomen, împreună cu dorința de a folosi un mijloc de transport privat, cum ar fi automobilul personal, și nu un mijloc de transport în comun, duce la congestiunea circulației rutiere. [10] Această problemă poate fi soluționată, nu în totalitate, dar substanțial prin implementarea unui sistem colaborativ inteligent de gestionare a circulației. Printre primele sisteme de gestionare intelligentă se numără cele bazate pe bucle inductive în carosabil care pot depista dacă pe aceasta se află un corp metalic solid, de exemplu un automobil. Informația este trimisă ulterior unui calculator central care gestionează datele și ia decizii în consecință [11]. Abordarea descrisă este folosită și în prezent, dar datorită sistemului rigid și centralizat nu funcționează cu o eficiență foarte mare.

O soluție ce caută să rezolve neajunsurile sistemului centralizat este implementarea unui sistem colaborativ care are ca actori toți participanții la trafic. Pentru implementarea unui astfel de sistem trebuie ca toți actorii să disponă de senzori și echipamentele necesare. Dacă un singur participant nu dispune de aceste facilități sistemul nu poate funcționa corect, ceea ce reprezintă o constrângere majoră în implementarea sa practică. Majoritatea autovehiculele noi sunt dotate cu un sistem ce permite conducerea semi-autonomă și anume sistemul ACC (pilot automat adaptiv). Pe baza acestui sistem s-a sugerat implementarea unui sistem denumit CACC (pilot automat adaptiv colaborativ) care permite mașinilor să distribuie datele măsurate prin senzorii proprii celorlalte mașini, iar pe baza informației disponibile să se ia cele mai bune decizii pentru siguranță și fluidizarea circulației [12].

Niște exemple concrete prin care sistemul poate să fie util sunt enunțate în continuare: plecarea exact în același timp de la un semafor, indiferent de poziția pe care o ocupă o mașină în coadă, eliminând astfel timpuri morti datorați factorului uman, avertizarea mașinilor din spate cum că porțiunea de drum din față este blocată, pentru ca aceastea să nu parcurgă intersecția pentru a o bloca și schimbarea benzii de deplasare prin acordul dintre două mașini. Un studiu elaborat în cadrul departamentului de Electronică al Universității Harvard a dovedit eficiența unui astfel de sistem prin simulările efectuate. Rezultatele obținute în acest studiu se pot vizualiza în figura 1.

Obiective

Lucrarea are ca obiectiv general implementarea și testarea unui sistem colaborativ bazat pe kiloboți și a tuturor sistemelor auxiliare ce sunt necesare pentru buna funcționare a sistemului colaborativ.

Acest obiectiv general poate fi atins prin îndeplinirea următoarelor obiective specifice:

- **Crearea unui sistem de control al kiloboților:** Acest sistem va recepta pachete de mesaje de la o stație de lucru și le va transpune în semnal IR pentru a comunica cu kilobotii.
- **Crearea unui metode conveniente de încărcare pentru kiloboți:** Sistemul va fi

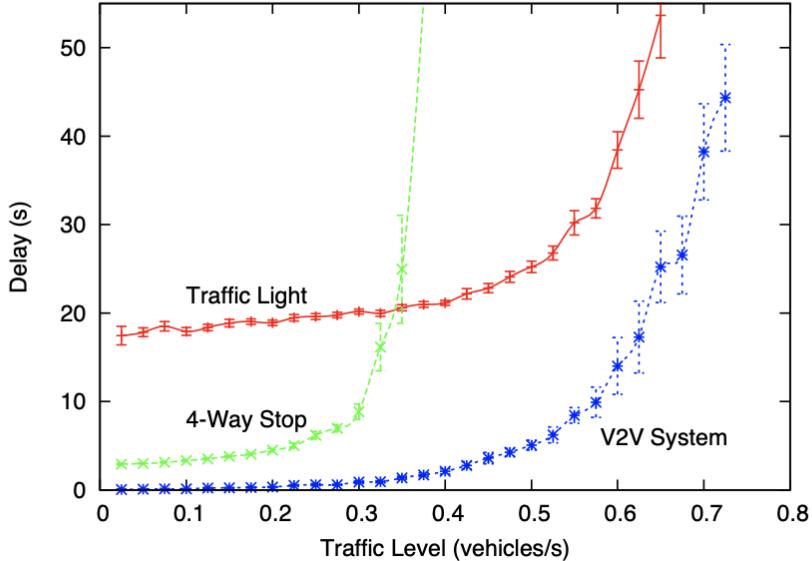


Figura 1: Îmbunătățirea timpilor de așteptare în trafic folosind un sistem colaborativ [1].

capabil să încarce simultan toți cei 10 kiloboți într-un mod convenient ce profită de modul în care aceștia sunt construiți.

- **Organizarea spațiului de lucru:** Se va crea un spațiu de lucru potrivit pentru a putea simula comportamentul colaborativ. Se va ține cont de toate proprietățile kiloboților în proiectarea spațiului de lucru, astfel încât aceștia să funcționeze la capacitate maximă.
- **Realizarea unor aplicații care pun în evidență comportamentul colaborativ:** Aplicațiile dezvoltate vor evidenția interacțiunea dintre kiloboți, acestea neputând fi duse la bun sfârșit de către un singur individ.

Starea artei

Datorită aplicabilității vaste și referințelor continue către natură și lumea ce ne încongoară, acest tip de sisteme au fost cercetate în amănunt, mai ales sistemele ce se folosesc de kiloboți, actorii principali în cadrul sistemelor dezvoltate în această lucrare.

Prin folosirea kiloboților se poate simula un sistem colaborativ și prin rezultatele obținute, să se determine eficiența acestuia.

Aplicațiile ce includ kiloboți sunt limitate datorită imperfecțiunilor regăsite în senzori și motoare [2]. Aceste probleme sunt aproape imposibil de soluționat atunci când un robot încearcă să acționeze individual. Cu toate acestea, soluția acestei limitări se regăsește în interacțiunea dintre roboți. Prin interacțiune acești roboți pot căpăta simțul pozitiei și al direcției, simțuri ce sunt inexistente în lipsa sistemului colaborativ.

O aplicație interesantă prin care se poate studia simpla luare de decizii a roboților este realizată cu ajutorul unui sistem de reacție-difuzie. Un sistem de reacție-difuzie reprezintă o serie de modele matematice care descriu anumite fenomene fizice, de exemplu schimbarea în timp și spațiu a concentrației, sau a temperaturii într-o substanță chimică [13]. Ecuațiile ce guvernează funcționarea unui astfel de sistem nu conțin gradienți ce țin de coordonatele spațiale, proprietate ce face aplicarea acestui posibilă, deoarece în sisteme colaborative nu putem determina cu exactitate coordonatele fiecărui individ. Prin aplicarea acestor sisteme se pot studia aceste tipuri de interacțiuni și se pot obține modele ca cele din Figura 2.

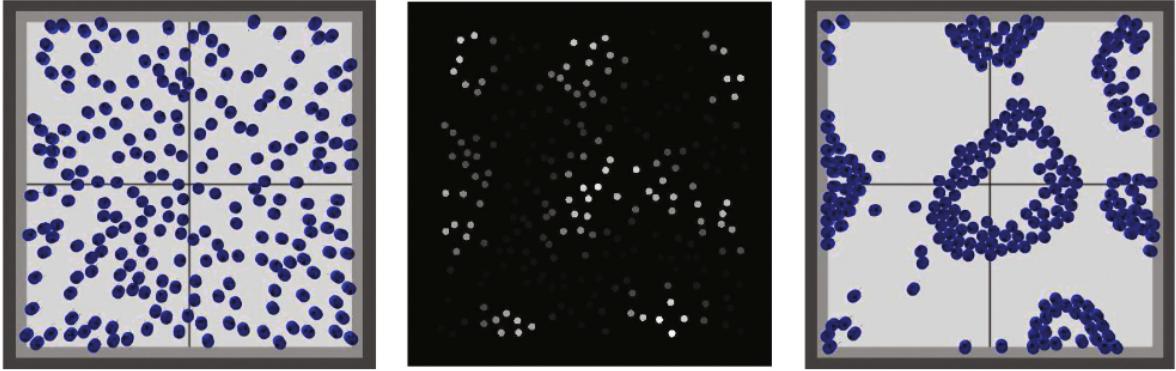


Figura 2: Formare de inele folosind sisteme de tip reacție-difuzie [2].

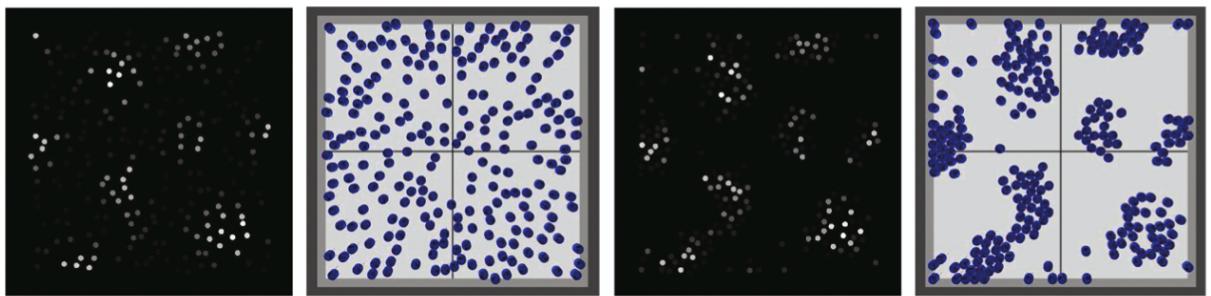


Figura 3: Segmentare în dungi folosind sisteme de tip reacție-difuzie [2].

Figura 2 prezintă cele trei stări în care sistemul funcționează. Sistemul de kiloboți folosit în acest experiment nu este unul fizic, ci unul simulație în mediul ARGoS. Prima imagine reprezintă starea inițială în care kiloboții sunt distribuiți aleator pe o suprafață de lucru, urmând ca aceștia să rămână în această poziție pentru 10^4 unități de timp. După această perioadă, fiecare robot își alocă un rol în funcție de gradul de activare al fiecărui. În a doua stare gradul de activare mai ridicat este reprezentat prin culoarea albă, iar gradul de activare mai scăzut prin negru, oricare alt grad de activare este reprezentat prin culoarea gri, mai închis, sau mai deschis, în funcție de intensitate. Cele două roluri importante au fost notate cu r_a , respectiv r_m . Dacă nivelul de activare este mai mare decât un prag ales, kilobot-ul va primi rolul r_a , iar, în caz contrar va primi rolul r_m . Roboții ce au rolul r_a sunt răspunzatori pentru a forma conturul inelului, iar roboții marcați r_m au rolul de a se alătura acestora și de a alcătui forma dorită, după cum se poate observa din starea trei din figura 2. După ce distanța unui robot r_m față de un robot r_a este mai mică ca valoarea unui prag ales acesta va căpăta și el rolul r_a , devenind astfel un reprezentant al conturului pentru ceilalți roboți cu proprietatea r_m [2].

Asemănător cu modelul inelului descris mai sus se pot forma multe alte figuri. Un alt exemplu este cel prezentat în figura 3, în care se poate observa cum kiloboții acționează similar unui algoritm de clustering, în care aceștia sunt capabili să formeze grupuri în jurul centroizilor, centroizii fiind reprezentanți de roboți cu rolul r_a [2].

O altă aplicație a kiloboților ce poate fi considerată benefică este auto-asamblarea și auto-repararea a unor forme arbitrară.

Tehnicile de auto-asamblare devin din ce în ce mai atractive în aplicații la scară microscopică, unde nu există alte alternative de fabricare. Astfel față de abordarea clasică de fabricarea

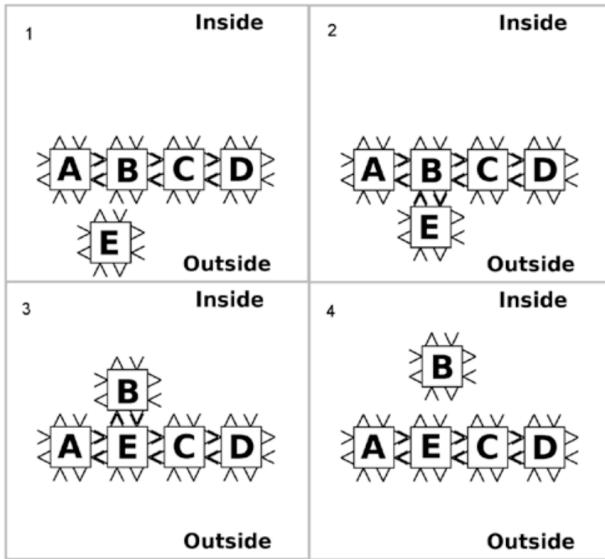


Figura 4: Exemplu de auto-asamblare. [3]

folosirea nanoroboților oferă avantajul auto-asamblării și chiar al auto-reparării diferitelor dispozitive sau sisteme în care s-a folosit această tehnologie ca metodă de producție. [3] Procesul de auto-asamblare este inspirat, ca și sistemele colaborative ce prezintă scopul lucrării, din natură. Există numeroase exemple de astfel de structuri: mai mulți atomi care se organizează într-o moleculă, celule care se organizează în țesuturi sau chiar în organisme. [3] Această tehnică vine cu multiple avantaje, dar și dezavantaje. Avantajul major este dat de unicitatea metodei de control a indivizilor; în cazul nostru există un singur cod sursă care este încărcat pe roboți, iar ulterior aceștia îndeplinesc scopul fără a mai fi necesară o a doua interacțiune. În cazul în care unul dintre indivizi, din anumite motive obiective, nu mai funcționează corect, se poate introduce un alt individ configurat identic ca cel anterior care să îl înlocuiască pe cel defect. Se poate folosi și un surplus de astfel de indivizi în fabricarea sistemului, pentru a oferi o componentă redundantă ce conferă siguranță. Metoda enunțată oferă proprietatea de auto-reparare. Un alt avantaj este reutilizarea indivizilor în alte aplicații în cazul în care sistemul dezvoltat anterior nu mai este necesar. Această reutilizare presupune extragerea indivizilor din vechiul sistem, reconfigurarea lor conform noilor cerințe și introducerea lor în noul sistem. Prin această metodă se distinge proprietatea de reutilizare. Printre dezavantaje se numără costul ridicat de fabricarea și de manipulare a unor astfel de indivizi la dimensiuni atât de mici. În studiul elaborat de [3], s-a folosit următoarea abordare pentru a putea recrea orice poligon: s-au ales o serie de indivizi care alcătuiesc conturul, de menționat este faptul că toți roboții ce se găsesc în contur sunt conectați între ei, iar restul roboților care ajung la frontieră conturului împing cel mai apropiat individ în interiorul conturului și îi revendică locul ca fiind un robot ce face parte din contur. [3]

Figura 4 arată un exemplu clasic de interacțiune dintre un individ ce alcătuiește conturul și un individ care se află în imediata vecinătate a primului menționat. În starea 1 indivizii A,B,C și D alcătuiesc o muchie a conturului. În starea 2 individul E găsește această muchie și intră într-un schimb de informație cu individul B. În starea 3 individul E îi ia locul lui B. Iar în ultima stare B este liber să se deplaceze în interiorul formei. Prin această metodă se poate forma origine poligon care are toate muchiile conectate între ele. [3]

În dezvoltarea acestor forme partea mai dificilă este crearea acestui contur. Pentru a crea acest contur s-a presupus că indivizii ce iau parte la acțiune au noțiunea de direcție. Pentru a alcătui orice formă este necesară transmiterea unei liste ca parametru care să conțină direcția și lungimea fiecărei muchii. Cunoșcând direcția și lungimea fiecărei muchii putem determina

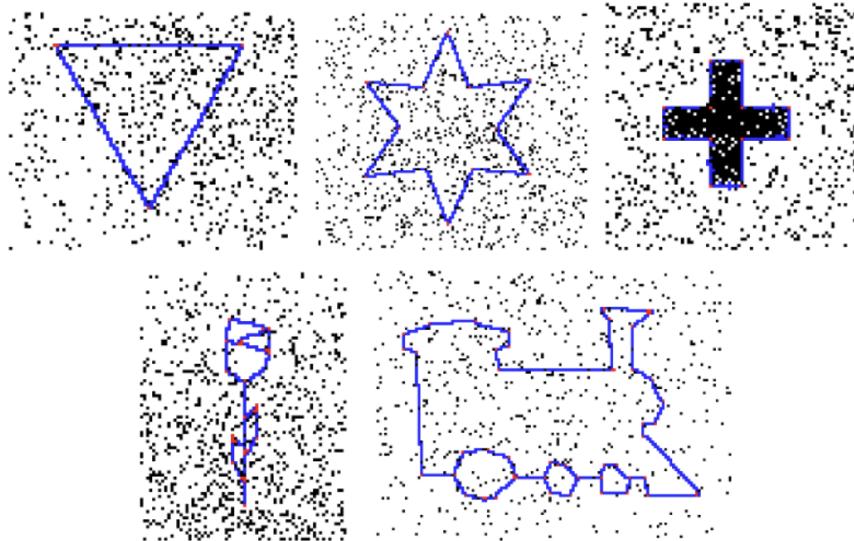


Figura 5: Forme realizate folosind algoritmul de auto-asamblare. [3]

punctele în care există noduri, adică punctele în care o muchie se termină, iar alta începe. Plecând de la un singur individ poziționat într-un nod putem forma fiecare muchie pe rând, prin propagarea mesajelor din ce în ce mai departe, până în momentul în care conturul este complet. Ulterior se poate trece la etapa următoare care a fost prezentată mai sus și care presupune umplerea formei obținute.

Figura 5 prezintă câteva forme obținute folosind algoritmul descris mai sus. De menționat că și aceste rezultate au fost obținute în urma unei simulări virtuale.

O altă aplicație ce prezintă interes este luarea unor decizii corecte la nivel de grup de roboți în punerea unei probleme ce are răspuns binar. Luarea decizilor la nivel de colectiv reprezintă unul dintre cele mai vitale acțiuni pe care un individ le poate face într-un sistem colaborativ [4]. Prin rezolvarea acestei probleme se ajunge la o decizie colectivă și după o perioadă de timp în care aceasta se propagă, chiar unanimă. Ca mai toate aplicațiile ce țin de sisteme colaborative, inspirația se află în natură, mai exact în grupurile de furnici și în stupii de albine, unde toți membrii acestor structuri iau decizii colective pentru bunăstarea colectivului. Pentru implementarea acestei tehnici se va folosi o strategie denumită modulararea directă a deciziei majoritare (DMMD) [14]. Experimetal s-a realizat prin crearea unui cuib în care au fost poziționați 100 de kilobroți. La stânga și la dreapta acestui cuib se află două zone de interes ce au un anumit factor de calitate. Aceste zone de interes sunt din sticlă, iar pe partea cealaltă a suprafeței există câte 5 roboți care transmit informația care le spune roboților care se află în această zonă de interes identificatorul zonei și factorul de calitate asociat acesteia. Inițial 50 de roboți consideră zona din stânga ca fiind opțiunea cea mai bună, pe când ceilalți 50 consideră zona din dreapta ca fiind obținerea potrivită. La început toți roboții se află într-o stare de explorare, aceștia sunt încurajați să părăsească cuibul și să caute aceste zone de interes. După găsirea zonelor de interes și recepționarea datelor privind factorul de calitate, roboții se vor deplasa înapoi în cuib unde va interacționa cu restul roboților și vor dezbatе soluția optimă. [4]

În figura 6 este reprezentată interacțiunea dintre un robot i și doi vecini ai acestuia, respectiv j și h . Dacă în opinia robotului i soluția a este cea mai bună, se poate observa cum această opinie se poate schimba în funcție de toate combinațiile posibile ale opiniilor vecinilor. Bineînțeles se poate lua în considerare și factorul de calitate al soluției, lucru ce ar face soluția

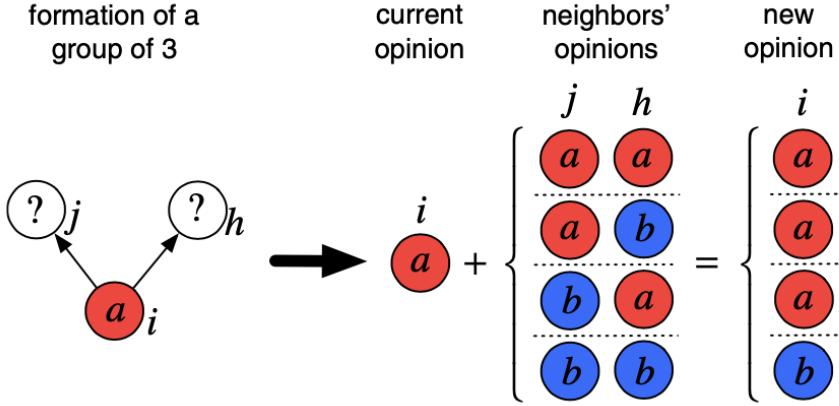


Figura 6: Dezbaterea opinilor între roboți prin metoda votului majoritar. [4]

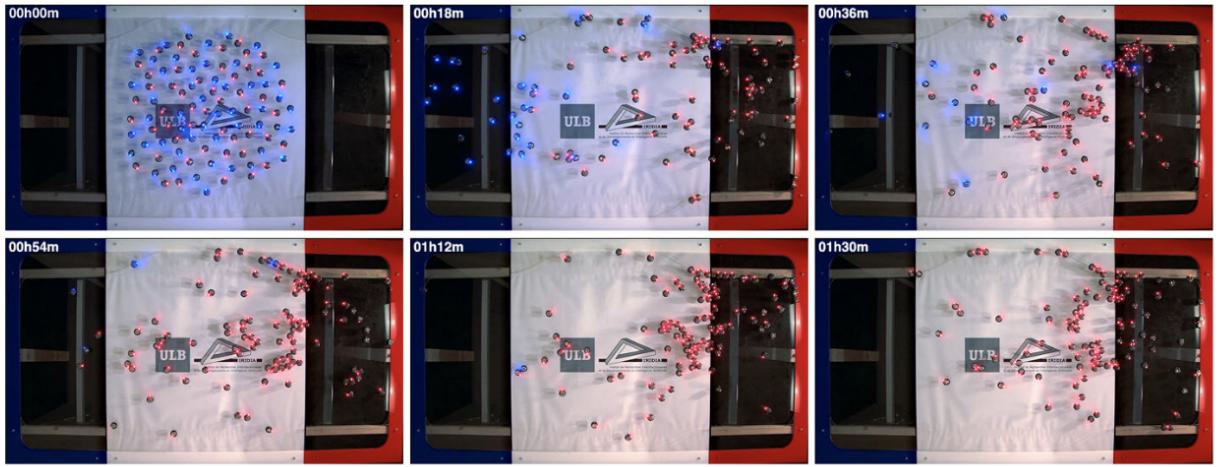


Figura 7: Experiment ce are ca scop luarea unei decizii la nivel de colectiv. [4]

calitativă mai influentă. Prin folosirea acestei metode soluția corectă se va propaga în întreg sistemul, până când se va ajunge la o decizie unanimă.

În Figura 7 se poate observa metodologia algoritmului enunțat mai sus asupra grupului de 100 de kiloboți. Zona din dreapta, reprezentată de culoarea roșie detaliază soluția mai calitativă. Se poate observa cum după o perioadă de timp această soluție începe să se propage, iar în final se ajunge la o decizie unanimă. Există totuși un compromis între timpul de execuție și performanța algoritmului. Cu cât timpul de execuție este mai mare, cu atât performanța sistemului este mai ridicată și sunt sănse mai mari ca toți kiloboți să ajungă la un consens. Dacă timpul de execuție trebuie să fie mai redus, din anumite motive obiective, performanța sistemului va avea de suferit. Un alt factor major ce influențează performanța algoritmului este numărul de indivizi ce iau parte la acțiune. Dacă numărul este suficient de mare atunci propagarea se poate efectua mai rapid, deoarece roboții nu mai sunt nevoiți să se deplaseze pe distanțe lungi pentru a interacționa între ei.

Unul dintre cele mai importante lucruri pe care un colectiv de indivizi poate să îl realizeze împreună și nu individual este deplasarea de obiecte [5]. Putem observa acest comportament mereu în mușuroaie de furnici în momentul în care acestea doresc să deplaseze hrana sau alte lucruri necesare. În experimentul realizat în articolul [5] s-au considerat mai multe forme arbitrară care prezintă inele în anumite zone. În interiorul inelelor vor fi poziționați kiloboții care fac posibilă deplasarea obiectului. Această construcție se poate observa în figura 8. Dacă



Figura 8: Obiectul cu inele ce urmează a fi deplasat. [5]

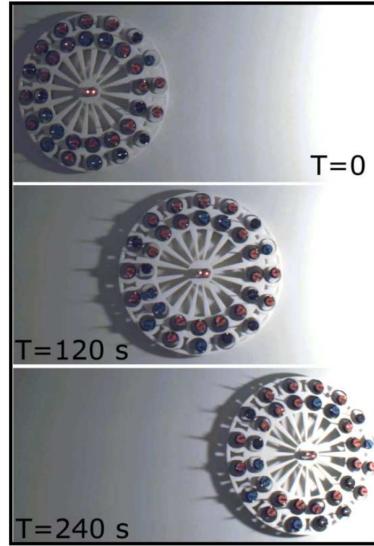


Figura 9: Mișcarea de translație a unui obiect. [5]

sunt suficienți roboți ce iau parte la acțiune se va învinge forța de frecare statică, iar obiectul se va deplasa. Pentru a alege direcția în care să se deplaseze obiectul s-a folosit și senzorul de lumină ambientală împreună cu o sursă de lumină pe suprafața de lucru care marchează destinația obiectului.

Deoarece indivizii nu au noțiunea de direcție sunt necesare câteva mișcari de rotație, în primă fază, pentru a căuta câteva informații privind direcția corectă în care se găsește sursa de lumină, adică destinația finală a obiectului. După această stare toți roboții încearcă să se plaseze în cea mai bună poziție pentru a putea efectua deplasarea, se dorește ca direcția de deplasare a roboților să fie paralelă cu o dreaptă trasa din mijlocul obiectului până la punctul unde se dorește să se afle centrul după mișcarea de translație. Acest lucru nu se întâmplă mereu aşa că nu toată puterea roboților se transformă în putere utilă pentru îndeplinirea scopului. O altă consecință a acestui fenomen este apariția unor mici mișcări de rotație care pot devia traectoria cu câteva grade față de traectoria ideală. O realizare a experimentului poate fi observată în figura 9.

Este evident că dacă creștem numărul de agenți viteza va crește. Acest lucru se întâmplă până la un anumit punct în care viteza obiectului tinde asymptotic la viteza de deplasare a unui agent în mișcare liberă.

În concluzie, putem spune că există o multitudine de aplicații ce folosesc implementări ale sistemelor colaborative. Este de necontestat utilitatea și aplicabilitatea acestor algoritmi și putem spune că sistemele collaborative chiar reprezintă o alternativă viabilă a sistemelor centralizate.

Capitolul 1

Descrierea componentelor hardware

1.1 Kiloboti

Sistemul prezentat în această lucrare are la baza robotii denumiți kiloboti. Aceștia reprezintă un grup de 10 roboti, fiecare fiind dotat cu un microcontroler Atmel ATMega 328p conectat la diverse dispozitive periferice pentru a putea simula cu succes un sistem colaborativ. În construirea unui kilobot s-a pus accentul pe două aspecte importante: costul și funcționalitatea. Robotul are nevoie de funcționalitate pentru a fi capabil să îndeplinească sarcinile colective la care este supus, dar în același timp, trebuie să fie suficient de simplu pentru a menține un cost scăzut de producție [15]. Unitatea centrală de procesare, Atmel ATMega 328p este un microcontroler pe 8 biți ce dispune de 32 KB de memorie *Flash*, utilizată atât pentru memoria de program, cât și pentru bootloader; 1 KB de memorie *EEPROM* ce poate fi folosită pentru stocarea informației non-volatile. Frecvența de lucru este 8 MHz. [8]

Fiecare kilobot este echipat cu o baterie Li-Ion reîncărcabilă de 3,7V ce permite o durată de funcționare de până la 3 săptămâni în modul sleep. De asemenea, fiecare robot dispune de un circuit ce permite încărcarea acestuia atunci când pe unul dintre picioarele acestuia este aplicată o tensiune de +6V, iar pe cărligul de încărcare GND (0V) [6]. Kilobotii pot comunica pe o raza de aproximativ 7 cm cu ajutorul unui transmițător și receptor cu IR. Atât transmițătorul cât și receptorul se află pe fată inferioară; astfel, transmisiunea se realizează prin reflexia semnalului de pe suprafață pe care se află robotii, ceea ce impune o alegere atentă a suprafetei de lucru. O caracteristică vitală a acestui sistem o reprezintă faptul că robotul poate estimă distanța până la transmițător pe baza instensității semnalului recepționat. O altă parte importantă o reprezintă senzorul de lumină cu ajutorul căruia se poate determina intensitatea luminii ambientale, pe baza căreia se pot lua decizii în consecință.

Kilobotii se deplasează cu ajutorul a 2 motoare cu vibrații ce pot fi controlate independent, permitând robotului să se miște și să efectueze viraje. Fiecare motor are 255 de valori diferite de putere și necesită calibrare în funcție de suprafață de lucru aleasă. Sistemul dispune și de un LED RGB ce poate fi folosit pentru a semnala diferențe stări de funcționare ale kilobot-ului, de exemplu, putem afișa nivelul de încărcare al bateriei cu ajutorul acestuia. Bootloader-ul ce se regăsește pe roboti permite controlul lor prin interfața **kiloGUI** cu ajutorul căreia putem controla și induce robotii în diferențe stări de funcționare (**sleep**, **voltage**, **run**, etc.), dar să încărcăm și codul C compilat pe aceștia.

În tabela 1.1 se pot recunoaște elementele marcate în figura 1.1.

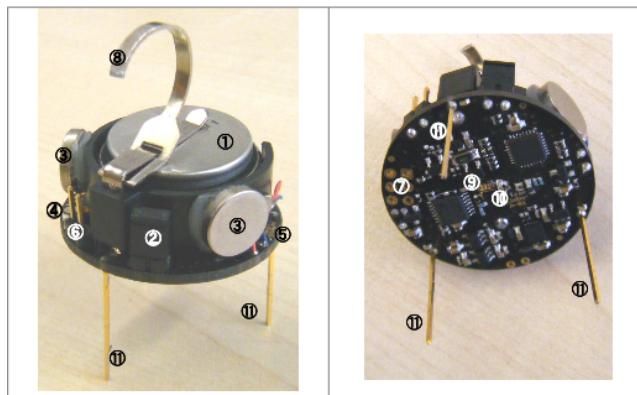


Figura 1.1: Un kilobot și componente ce îl alcătuiesc. [6]

Număr de identificare	Denumire componentă
1	Baterie Li-Ion de 3,7V
2	Jumper de pornire
3	Motoare cu vibrații
4	LED RGB
5	Senzor de lumină ambientală
6	Ieșire serială pentru debug
7	Socket pentru interfață ISP
8	Cârlig pentru încărcare
9	Transmițător de semnal infraroșu
10	Receptor de semnal infraroșu
11	Picioarele robotului (folosite și pentru încărcare)

Tabela 1.1: Componentele ce alcătuiesc un kilobot.

1.2 Overhead Controller

Pentru a putea trimite diferite instrucțiuni roboților este nevoie de o platformă ce poate transpune mesajele primite de la calculator în semnal IR, ce poate fi ulterior înțeles de către kiloboți. Această platformă o vom denumi în continuare **Overhead Controller**. Overhead Controller-ul are la bază un microcontroler **Atmel ATmega328P** (același microcontroler ce se regăsește și pe kiloboți), care primește informația de la un calculator folosind o magistrală serială, mai exact standardul **RS-232**.

Componentele ce alcătuiesc Overhead Controller-ul:

- **Circuit integrat MAX232:** Standardul RS-232 folosește alte nivele logice pentru a transmite informația serială. Aplicarea tensiunii corespunzătoare nivelelor logice direct pe pinii microcontrolerului poate duce la defectarea acestuia. Pentru a evita acest scenariu, s-a folosit un circuit integrat, **MAX232**, ce este capabil de a transforma nivelele logice folosite de standardul RS232 în nivelele logice TTL folosite de microcontrolerul **ATMega328p**. Transformarea se realizează atât pentru transmiterea, cât și pentru recepționarea de date prin interfață serială.
- **Interfață ICSP:** Overhead Controller-ul dispune și de un pin header pentru interfață ICSP, ce este folosită în cazul în care dorim adaptarea codului încărcat pe microcontroler, sau în cazul în care acesta se defectează și este necesară înlocuirea acestuia.

- **Conecatori pentru placile cu LED-uri:** Pentru a comunica cu placile cu LED-uri IR s-au montat 6 seturi a cate 2 conectori. Polaritatea acestor conectori este marcată cu "+" și "-". Conectorul "+" este conectat la sursa de 12V, iar conectorul "-" la drena unui tranzistor. Fiecare ieșire către placile cu LED-uri este controlată de către un tranzistor care are poarta conectată la un pin al microcontroler-ului, drena la ieșirea circuitului de pe placa cu LED-uri (adică conectorul "-") și sursa la punctul de potential 0V. Prin acționarea tranzistorului, circuitul se închide prin placa cu LED-uri, aplicând astfel 12V pe circuitul de alimentare al LED-urilor. Astfel, prin controlarea nivelului de tensiune de pe pin-ul la care este conectat tranzistorul putem controla dacă LED-urile IR sunt aprinse sau nu.
- **Regulator de tensiune 12V la 5V:** Overhead Controller-ul este alimentat de la o singură sursă de 12V. Acest potențial este folosit pentru alimentarea placilor cu LED-uri IR. Cu toate acestea, avem nevoie și de un potențial de 5V pentru a alimenta microcontrolerul, dispozitivele ce ajută la funcționarea corectă a acestuia și circuitul integrat MAX232. Pentru a obține acest potențial s-a folosit un regulator de tensiune L78S05CV.
- **Buton de reset:** Butonul este legat la pinul de reset al microcontroler-ului și este folosit, după cum sugerează și numele, pentru resetarea Overhead Controller-ului în cazul în care acesta nu mai răspunde la comenzi sau dacă apare o problemă în exploatarea acestuia.
- **LED:** Pentru a putea verifica starea de funcționare a sistemului se folosește un LED care semnalează primirea de pachete de la stația de lucru. Se poate adăuga și altă utilitate prin modificarea codului încarcat pe microcontroler.
- **Header pentru transmisie serială:** Această header se poate folosi pentru a putea depărtă transmisiunea serială. Cei 2 conectori sunt inscripționați cu etichetele "TX" și "RX" și pot fi conectați la un dispozitiv ce dispune de un periferic de transmisie și recepționare serială cu nivele TTL de 0 - 5V, cum ar fi o placă de dezvoltare Arduino.

În figura 1.2 se poate observa schema electrică a Overhead Controller-ului și modul de apăsare a tuturor componentelor enunțate mai sus.

Figura 1.3 arată implementarea fizică a Overhead Controller-ului. Se poate observa marcate toate elementele ce se folosesc în interfațarea cu utilizatorul pentru a evita confuzie.

Figura 1.4 arată modul în care a fost construit Overhead Controller-ul, și anume cu trasee din fludor pe un perfboard, iar unde au fost necesare salturi peste aceste trasee s-au folosit fire de cupru izolate. Componentele folosite sunt fabricate în tehnologia THT, mai puțin tranzistoarele care sunt fabricate în tehnologia SMT.

1.3 Standard RS-232

Standardul RS-232 (Recommended Standard 232) este folosit pentru transmisiunea serială de date, introdus prima oară în anul 1960. Acesta definește toate caracteristicile electrice, sincronizarea de semnale, utilitatea lor, dimensiunea și pinout-ul conectorilor. Se folosește, de cele mai multe ori, pentru transmisiunea de date de la calculator către un circuit extern care poate gestiona aceste date, fiind astfel foarte util în această aplicație.

Standardul a apărut mult înainte de folosirea nivelor TTL, utilizate la scară largă în ziuă de azi. Nivelele logice folosite de transmițător se află între +5 - +15V pentru "0" logic și -5 - -15V pentru "1" logic. Pentru receptor s-a introdus o marjă de eroare de 2V pentru a putea face sistemul mai puțin influențabil de zgomot, astfel că nivalele logice pentru receptor se află între +3 - +15V pentru interpretarea mesajului ca fiind "0" logic și -3 - -15V pentru "1"

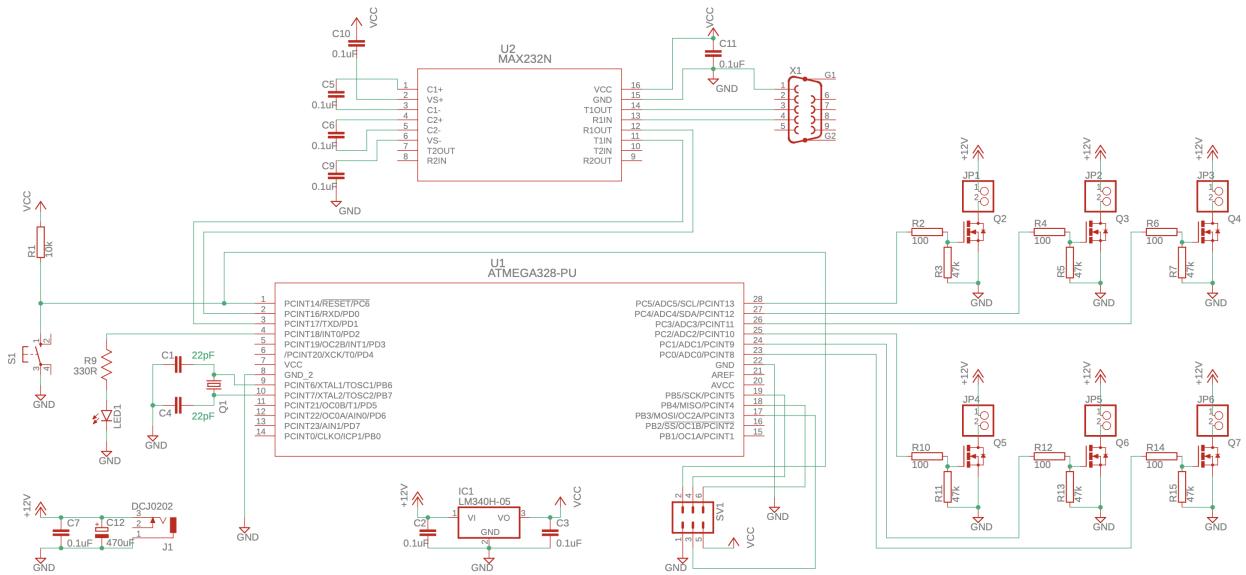


Figura 1.2: Schematic Overhead Controller.

logic [16]. Se poate observa că interpretarea nivelelor de tensiune se face invers față de nivelele TTL.

1.4 KiloGUI

KiloGUI este componenta software ce reprezintă interfața cu sistemul robotic. Acesta pune la dispoziția utilizatorului o varietate de comenzi pe care le poate transmite către Overhead Controller. Una dintre cele mai importante comenzi este comanda “upload” ce încarcă fișierul .hex ce a fost creat în urma compilării codului C, urmând ca acesta să fie încarcat pe kiloboți.

Pe lângă această funcție de upload a codului, KiloGUI permite selecția stării de funcționare a kiloboților. Modurile de funcționare sunt următoarele:

- **Run:** Kiloboții execută programul încărcat pe aceștia.
- **Pause:** Se oprește execuția programului și se intră într-un mod de așteptare în care kiloboții nu se mișcă; acest mod de funcționare este prezentat prin aprinderea și stingerea LED-ului cu culoarea verde la un interval scurt de timp. După ce se părăsește modul Pause execuția se va relua din punctul unde s-a indus starea.
- **Sleep:** Roboții intră într-o stare de hibernare ce este semnalată prin aprinderea și stingerea LED-ului alb la un interval mai mare de timp. În acest mod de funcționare kiloboții au o autonomie a bateriei de până la 3 săptămâni, deci aceștia pot fi plasați în acest mod atunci când nu sunt folosiți.
- **Voltage:** Fiecare kilobot va verifica nivelul de încărcare al bateriei proprii și va afișa o culoare reprezentativă pentru acest nivel folosind LED-ul ce se regăsește pe aceștia.
- **Bootload:** Prin această comandă se pregătește încărcarea fișierului .hex pe roboti. Pentru ca această comandă să aibă efect trebuie ca roboții să se afle în modul Pause. Dacă comanda a fost recepționată cu succes de către kiloboți, aceștia vor ține LED-ul aprins constant cu culoarea albastră.

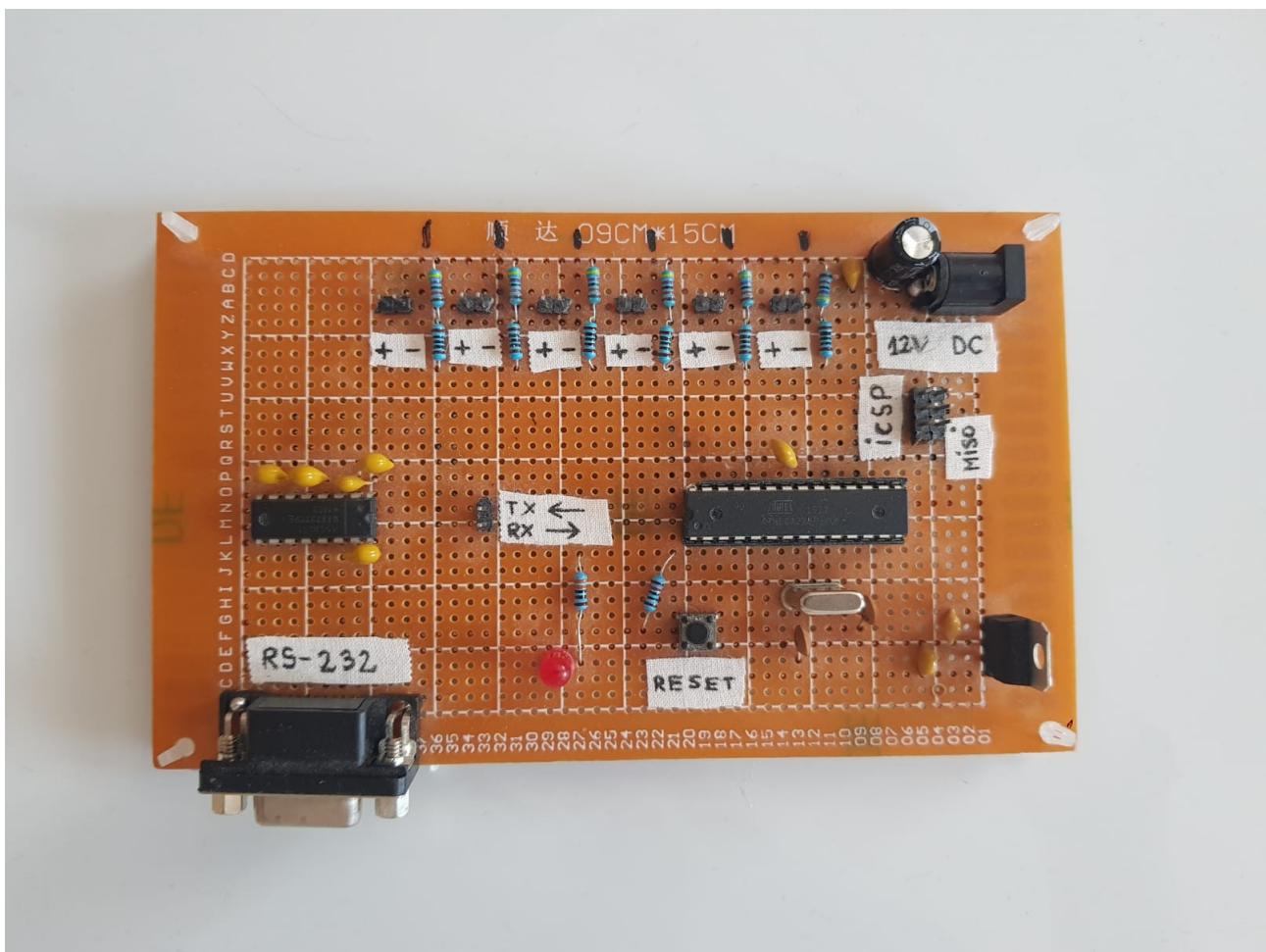


Figura 1.3: Overhead Controler.

- **Reset:** Acest mod de funcționare este destul de similar cu modul **Pause**. Diferența dintre acestea se poate observa în momentul în care se execută din nou comanda **Run**. Dacă robotul se află în modul de funcționare **Pause** atunci reluarea execuției programului se va face din punctul în care s-a primit comanda. Pe când, dacă robotul se află în modul **Reset** acesta nu își va mai relua execuția din punctul curent, ci va relua execuția programului de la început.
- **Serial Input:** Este un submeniu prin care se pot vizualiza datele primite de la kilobotii dacă aceștia funcționează în modul **Debug** și dacă au conectorul interfeței seriale cu 2 fire conectat. Modul **Debug** poate fi indus prin implementare software. Acest submeniu este foarte util dacă se dorește depanarea sistemului.
- **Calibration:** Este tot un submeniu care ajută la calibrarea valorilor de acționare a motoarelor, dar și pentru a aloca fiecărui kilobot un ID unic ce oferă o metodă de identificare a unui individ în cadrul sistemului. Calibrarea este necesară deoarece procesul de fabricare nu este identic pentru toți roboții, iar din această cauză apar inconistențe între aceștia. Prin procesul de calibrare se dorește eliminarea acestor inconstențe, procesul ajută în minimizarea acestora, dar este departe de a fi ideal. Submeniul de calibrare poate fi observat în Figura 2.3.

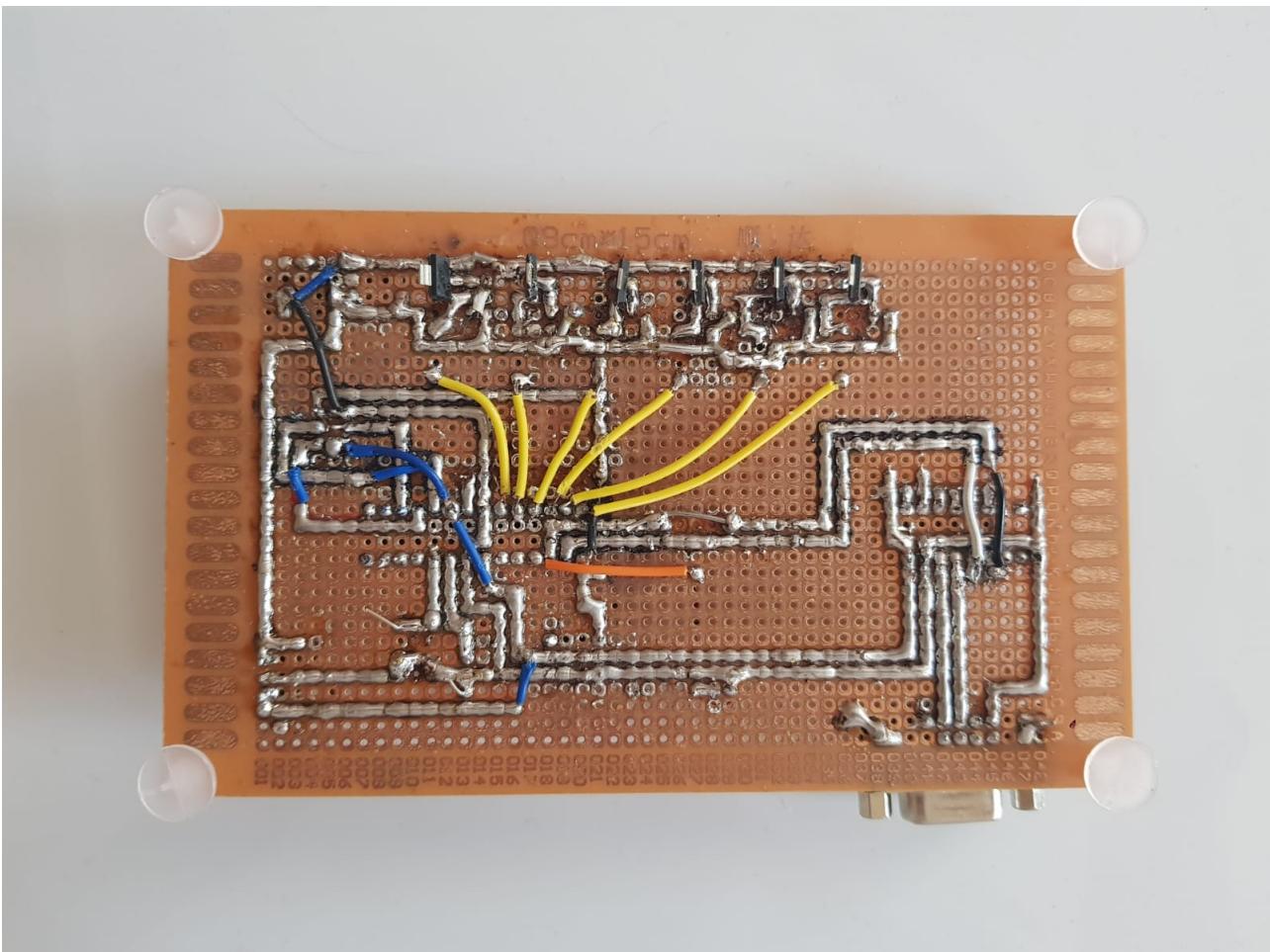


Figura 1.4: Partea din spate a Overhead Controler-ului.

1.5 MAX232

Pentru a putea face conversia dintre datele primite de la calculator, prin transmisiunea serială folosind standardul RS-232, și nivele TTL pe care le poate înțelege microcontroler-ul ATmega328p s-a folosit circuitul integrat MAX232CPE.

Max232 este un dispozitiv ce poate fi caracterizat ca fiind un driver/receptor cu 2 canale. Acesta include un generator de tensiune capacitive, capabil să genereze nivele de tensiune specifice standardei TIA/EIA-232-F folosind doar o singură sursă de alimentare de 5V. Fiecare receptor face conversia nivelor TIA/EIA-232-F de la intrare în nivele TTL/CMOS (5V), iar cele 2 drivere fac conversia din nivelurile TTL/CMOS aflate la intrare în nivele TIA/EIA-232-F. Valoarea de prag tipică a receptorului este de 1,3V. [17]

În această aplicație este necesară folosirea unei singure perechi receptor/driver. Driver-ul este conectat la pinul **TX** al microcontroler-ului ATmega328p, pe când receptorul este conectat la pinul **RX**. Atât driver-ul, cât și receptorul sunt legați la pinii aferenți conectorului de transmisiune serială.

1.6 Atmel ATmega328p

Atmel ATmega328p este un microcontroler oferit de Atmel și face parte din familia megaAVR.

ATmega328 este un circuit integrat ce are la bază un microcontroler RISC AVR pe 8 biți. Parametrii microcontroler-ului pot fi vizualizați în tabela 1.2.

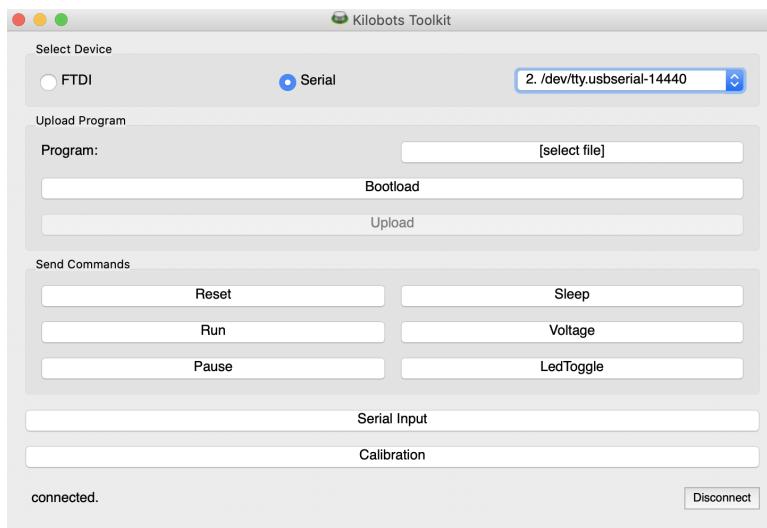


Figura 1.5: KiloGUI

Parametrii	Valori
Flash	32 Kbytes
RAM	2 Kbytes
EEPROM	1 Kbytes
Număr de Pini	28
Frecvență maximă de funcționare	20 MHz
CPU	8 bit AVR
Pini maximi de intrare/ieșire	23
Întreruperi externe	2

Tabela 1.2: Parametrii Atmel ATmega328p. [8]

Microcontroler-ul dispune de mai multe dispozitive periferice ce pot fi folosite în diverse aplicații. Cele mai importante sunt:

- **2 Timere pe 8 biți cu mod de comparare.**
- **1 Timer pe 16 biți cu mod de comparare și de captură.**
- **6 Canale PWM.**
- **Convertor analog-digital pe 8 canale și acuratețe de 10 biți.**
- **Interfață serială USART.**
- **Interfață serială SPI.**
- **Interfață serială I^2C .**
- **Întreruperi pe schimbarea valorii unui pin.**

Pentru utilizarea mai eficientă a energiei, microcontroler-ul combină cinci moduri software diferite de economisire a energiei. Dispozitivul poate funcționa în gama 1,8 - 5,5V.

În această aplicație microcontroler-ul se regăsește atât pe kiloboți, cât și pe Overhead Controller. În ambele cazuri acesta are rolul de unitate centrală de procesare.

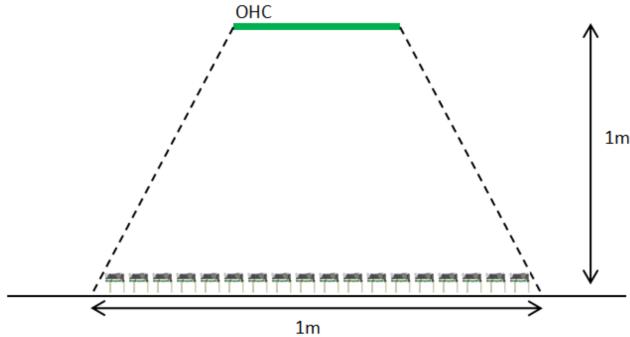


Figura 1.6: Amplasare Overhead Controller. [6]

În structura **Overhead controller**-ului, microcontroler-ul gestionează pachetele primite prin portul de transmisie serială. După aceasta el controlează cele 6 tranzistoare ce sunt conectate la pinii microcontroler-ului. Aceste tranzistoare aprind sau sting o serie de LED-uri IR ce sunt folosite pentru a transmite comenzi către kiloboți.

În cadrul kiloboților microcontroler-ul demodulează și gestionează pachetele primite de la **Overhead Controller**. Pachetele primite pot fi încărcare direct în memoria de program a microcontroler-ului, ceea ce permite încărcarea de programe direct prin IR, fără a fi nevoie de conectarea unui fir fizic. Acest lucru aduce și un dezavantaj, deoarece comunicarea fără fir poate fi mai puțin fiabilă decât cea prin fir. Dacă un singur bit nu este recepționat, sau este recepționat greșit se poate ajunge la incapacitatea robotului de a funcționa corect, fiind necesară reluarea procesului de încărcare al programului.

1.7 Comunicare IR

Kiloboții sunt proiectați astfel încât aceștia să funcționeze fără o conexiune fizică cu **Overhead Controller**-ul. Totuși, de multe ori, este necesar ca aceste două dispozitive să comunice între ele. Pentru a soluționa această problemă, atât kiloboții, cât și **Overhead Controller**-ul sunt dotati cu dispozitive ce facilitează comunicarea prin IR între acestia. Kiloboții dispun atât de transmițător, cât și de receptor cu IR, deoarece aceștia trebuie să comunice și între ei, nu doar cu **Overhead Controller**-ul, pe când **Overhead Controller**-ul dispune doar de transmițător.

Comunicația se bazează pe reflexia fasciculu lui luminos de pe suprafața pe care se află kiloboții, deci alegerea acestei suprafețe este foarte importantă. Kiloboții ar trebui operați pe o suprafață dreaptă și netedă pentru a asigura mobilitatea robotilor. Pentru a facilita comunicația, suprafața ar trebui să fie lucioasă. Un exemplu bun de suprafață ar fi o tablă albă lucioasă [6].

Overhead Controller-ul ar trebui să se afle la o distanță de aproximativ 1 metru deasupra suprafeței de lucru, astfel robotii ce se află pe un diametru de 1 metru ar trebui să aibă o conexiune stabilă la **Overhead Controller** [6]. Amplasamentul descris se poate observa și în Figura 1.6.

Pentru a avea posibilitatea de a detecta semnalul de la **Overhead Controller**, puterea semnalului trebuie să fie mai mare decât valoarea de prag a fotodiodei găsită pe roboti. Cu cât distanța față de sursă este mai mare, cu atât puterea semnalului scade. Din acest motiv se folosește o sursă de iluminare externă. Sursa externă ar trebui să acopere o mare parte din valoarea de prag a diodei, fiind necesar doar un mic impuls din partea **Overhead Controller**-ului pentru ca un kilobot să recunoască mesajul codat [7].

1.8 Placă cu LED-uri IR

Rolul Overhead Contoler-ului este de a primi comenzi de la o stație de lucru și de a le transpune în secvențe binare ce urmează să fie transmise ulterior robotilor. După cum se poate observa și din schema Overhead Contoler-ului, Figura 1.2, există 6 conectori ce sunt folosiți pentru a realiza conexiunea dintre controler și placile cu LED-uri IR.

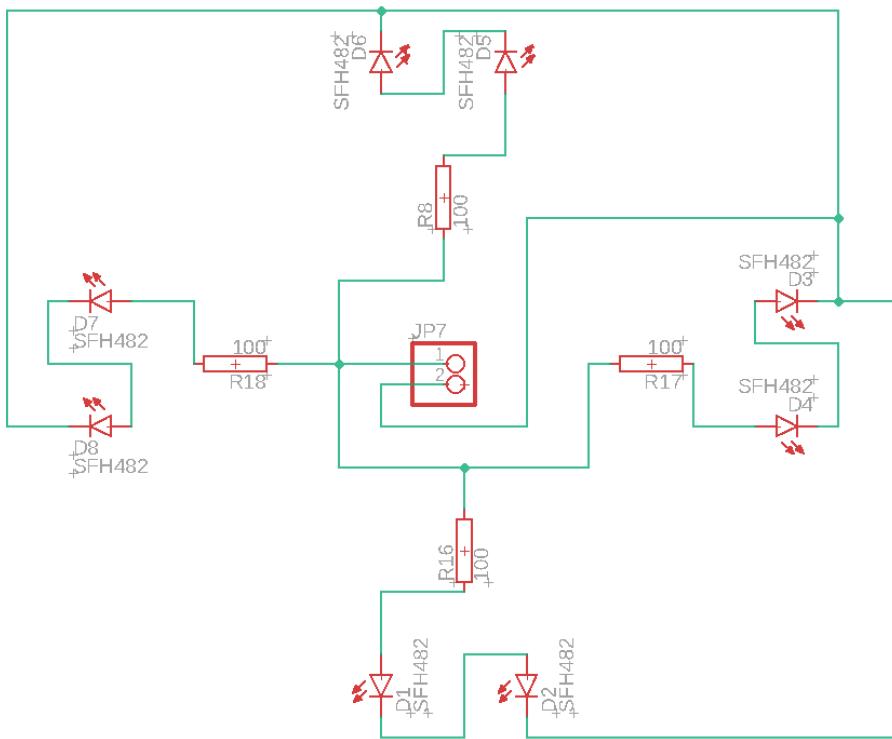


Figura 1.7: Schematic Placă cu LED-uri IR.

Pentru funcționarea corectă este necesară doar o singură placă cu LED-uri IR, dar pentru a acoperi o suprafață cât mai mare se pot folosi până la 6 astfel de plăci. În cazul în care se utilizează o sursă de iluminare artificială pentru suprafața de lucru, trebuie ca așezarea plăcilor să fie minuțioasă, astfel încât acestea să nu împiedice sursa de lumină să ajungă pe suprafața de lucru.

Fiecare placă are 8 LED-uri IR și 2 conectori marcați cu "+" și "-" ce trebuie legați la Overhead Contoler respectând polaritatea marcată.

Pentru ca mesajul să fie transmis acesta trebuie transpus într-o secvență binară ce va fi transmisă serial către placa cu LED-uri prin acționarea unui tranzistor. Astfel un "1" din secvența binară este asociat LED-ului aprins, pe când simbolul "0" este asociat LED-ului stins. Frecvența de transmisie trebuie să fie aceeași atât pentru Overhead Contoler, cât și pentru roboți. Această frecvență poate fi modificată din firmware-ul Overhead Contoler-ului și din bootloader-ul robotilor.

Distanța la care trebuie amplasate aceste plăci cu LED-uri depinde în principiu de suprafața de lucru și de alte surse de iluminare auxiliare folosite.

1.9 Încărcător

Fiecare robot este alimentat de o baterie de 3,7V. Este evident utilă găsirea unei metode convenabile prin care se pot încărca mai mulți roboți simultan. Pentru a încărca un kilobot



Figura 1.8: 6 Plăci cu LED-uri IR.

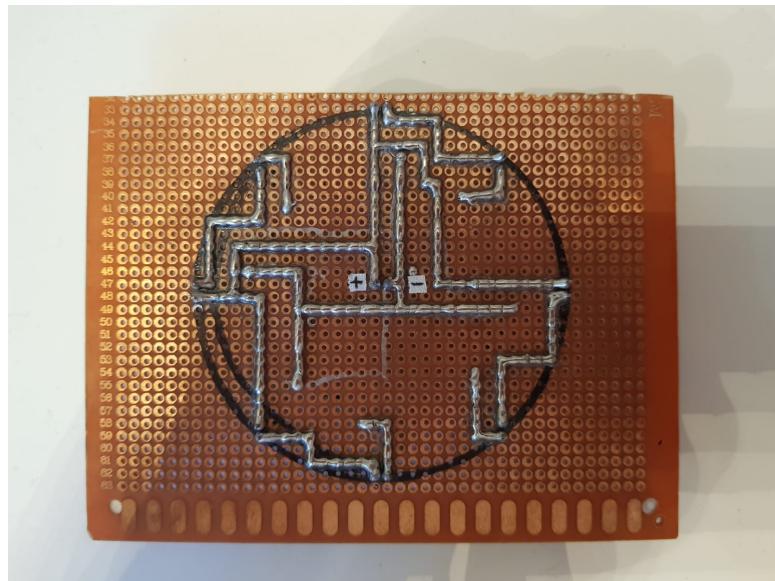


Figura 1.9: Conectori Placă cu LED-uri IR.

este necesară aplicarea unei diferențe de potențial între unul dintre picioarele acestuia și cârligul de încărcare. Componentele unui kilobot pot fi reconsultate în figura 1.1.

Pentru a ușura metoda de încărcare s-a proiectat un sistem alcătuit din două tije metalice, una dintre ele fiind legată la potențialul de $+6V$, iar cealaltă la masă ($0V$). Acest sistem profită la maxim de cârligul de încărcare pe care îl au roboții, deoarece permit ca aceștia să stea agătați de bara de potențial 0 și sprijiniți de bara de potențial 6.

Sistemul este alimentat de la priză printr-un transformator ce convertește cei $220V$ curent alternativ în $12V$ curent continuu, urmând ca acesta să treacă print-o sursă coborâtoare ce menține la ieșire un potențial stabil de $6V$. Transformatorul poate debita până la $2,5A$, adică o putere totală de $30W$, iar un robot folosește în jur de $100mA$ în momentul în care se încarcă. Sursa coborâtoare mai poate consuma maxim $100mA$, deci circuitul poate consuma maxim $1,1A$ dacă se află sub solicitare maximă. Orice valoarea de consum mai ridicată de atât pune sub semnul întrebării starea de bună funcționare a încărcatorului.

Încărcătorul poate alimenta până la 10 roboți, această limitare provenind din dimensiunea

acestuia și nu din pricina puterii debitate de sursa de alimentare. Pentru ca un robot să fie încărcat este necesar ca acesta să fie pornit și aflat în modul de funcționare "Sleep" sau "Pause" (mai multe informații despre modurile de funcționare se pot regăsi în secțiunea kiloGUI 1.4) pentru ca circuitul de încărcare să fie activat. Se poate verifica foarte ușor dacă robotoul se încarcă prin consultarea valorii curentului debitat de sursă de pe display-ul ce se regăsește pe încărcător. Prin modul de proiectare al încărcătorului eliminăm riscul de inversare al polarității și riscul ca acesta să nu facă contact suficient cu bornele robotului.

Durata tipică de încărcare este în jur de 1 oră pentru fiecare robot. Este destul de dificil de estimat o durată exactă de funcționare a bateriei, deoarece depinde foarte mult de modul utilizat, sarcinile pe care trebuie să le îndeplinească robotul și nivelul de uzură al bateriei. De exemplu, un robot care folosește motoarele cu vibrații pentru o perioadă mai îndelungată decât alt robot se va descărca mult mai rapid. Din mai multe experimente realizate s-a observat că în momentul în care bateria este aproape descărcată robotul nu mai funcționează corect și începe să ia decizii eronate, lucru ce impune verificarea constantă a nivelului de încărcare al bateriei. Dacă nivelul de uzură al bateriei este atât de ridicat încât un kilobot nu mai poate fi operat corect putem recurge la înlocuirea bateriei cu o celulă asemănătoare LI-ION 3.6V LIR2477.

Baza încărcătorului este din lemn peste care s-a adăugat autocolant. Cele 2 forme "K", ce susțin cele două bări între care există diferența de potențial, au fost realizate prin tehnologia tăierii cu laser. Bările ce susțin robotii sunt fabricate dintr-un aliaj de cupru.

Sursa coborâtoare poate suporta o tensiune de intrare între 6 și 32V curent continuu și poate pune la ieșire o tensiune între 1,5 și 32V. Sursa prezintă protecție la scurtcircuit, supra-temperatură și alimentare inversă și are o eficiență în jur de 96%.

În figura 1.10 se poate observa încărcătorul în timp ce este folosit pentru reîncărcarea celor 10 kilobotii.



Figura 1.10: Încărcător kiloboti.

Capitolul 2

Setup experimental

2.1 Spațiul de lucru

Realizarea experimentelor necesită pregătirea unui spațiu de lucru ce trebuie să îndeplinească mai multe condiții. Pentru funcționarea corectă trebuie ca suprafața să fie perfect netedă, ca în Figura 2.2, deoarece orice imperfecțiune de pe suprafață poate împiedica mișcarea roboților. Totodată suprafața trebuie să fie dreaptă pentru ca mișcarea roboților să fie previzibilă și consistentă în fiecare direcție. Spațiul de lucru folosit în cadrul acestui proiect este ilustrat în Figura 2.1.



Figura 2.1: Organizarea spațiului de lucru.



Figura 2.2: Suprafața de lucru dreaptă.

2.2 Calibrare

Deoarece în procesul de fabricație există inconsistențe, roboții trebuie calibrati corespunzător înainte de a fi folosiți. Calibrarea include alocarea unui ID unic fiecarui participant la experimente, pentru ca acestia să se poate identifica între ei, dar și calibrarea motoarelor. Ambele acțiuni pot fi îndeplinite cu ajutorul KiloGUI 1.5. Prin procesul de calibrare se modifică informația găsită la anumite adrese din memoria EEPROM, valori ce pot fi citite și în programul scris de utilizator.

Prin calibrare se dorește eliminarea în proporții cât mai mari a tuturor diferențelor ce există între roboți, diferențe ce pot apărea din cauza procesului de fabricare, sau din alte cauze obiective. Cele mai mari diferențe ce pot influența experimentele pe care le desfășuram sunt regăsite în motoarele cu vibrații. Acestea necesită o calibrare fină pentru a putea efectua mișcari la stânga, la dreapta, dar mai ales în față, deoarece mișcarea în față presupune acționarea ambelor motoare simultan. Dacă un motor îl depășește pe celălalt în intensitate, robotul nu va avea o traекторie dreaptă, ci mai degrabă, o traectorie ce se asemănă cu un arc de cerc.

De asemenea, în procesul de calibrare trebuie luată în calcul și suprafața de lucru. Unele suprafete necesită o acțiune mai puternică a motoarelor, pe când pe alte suprafete această acțiune poate duce la o mișcare haotică, ce nu este absolut deloc previzibilă și nu poate fi controlată.

În procesul de calibrare se poate asocia și un ID unic ce permite fiecarui robot să fie unic și ușor identificabil. Spațiul de memorie la care se regăsește ID-ul poate fi citit oricând în faza dezvoltării software. După citire, acesta poate fi folosit pentru individualizarea anumitor secvențe de cod pentru anumiți roboți, sau pentru identificarea individului în comunicarea cu ceilalți roboți.

În figura 2.3 se poate vizualiza submeniul de calibrare din KiloGUI. Valorile sunt pentru calibrarea robotului cu ID-ul 2.

Tabelul 2.1 prezintă valorile de calibrare pentru mai mulți roboți. Se poate observa că nu există nicio corelație între valorile de calibrare a oricare doi roboți, lucru ce impune calibrarea individuală a fiecarui robot în mod empiric, neexistând o metodă prin care să se efectueze calibrarea fără a testa în mod repetat valorile alese.

Faptul că fiecare robot are o intensitate diferită de acționare a motoarelor impune încă o constrângere ce trebuie tratată în implementarea software. Constrângerea apare atunci când se

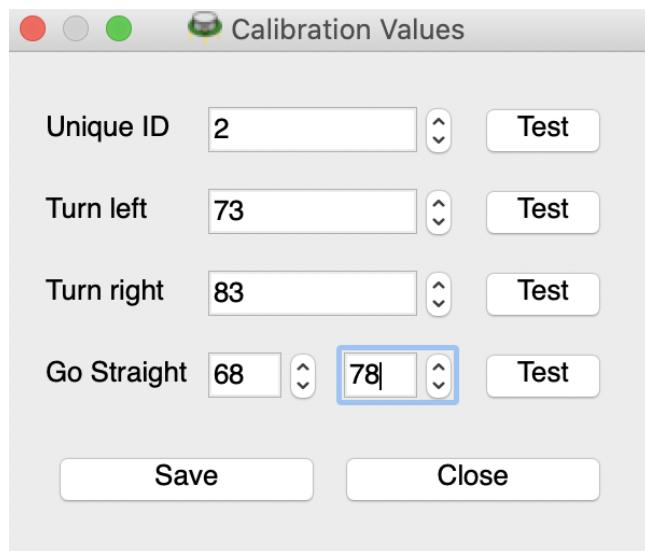


Figura 2.3: Submeniu de calibrare.

ID unic	Viraj stânga	Viraj dreapta	Înainte
2	73	83	68 — 78
3	75	77	68 — 72
5	66	75	59 — 65

Tabela 2.1: Valori de calibrare pentru diversi roboti.

încercă mișcari de prezicte cum ar fi rotirea cu 90°. Datorită calibrării anterioare, fiecare robot are un timp unic în care poate efectua o astfel de mișcare.

Un factor ce poate induce fals sentimentul de calibrare greșită a robotului este nivelul scăzut al bateriei. Este firesc ca motoarele să nu funcționeze în modul dorit atunci când nu mai există resurse de energie pentru a menține funcționarea acestora. Utilizatorului îi revine răspunderea de a verifica nivelul de încărcare înainte de calibrare și în mod regulat după aceasta.

2.3 Iluminare

Kiloboții dispun atât de un transmițător de semnal IR, cât și de un receptor, după cum se poate observa în figura 1.1. Comunicarea între roboti și **Overhead Controler** se realizează fără fir, prin intermediul undelor IR. Datorită caracteristicilor intrinseci a acestui tip de comunicare, nivelul și calitatea iluminării capătă o însemnatate majoră, influențând puternic calitatea transmisiei de date.

Departamentul de informatică al universității din Sheffield a elaborat un studiu minuțios ce dezbată influența iluminării asupra comunicării dintre kiloboți și **Overhead Controler**. Aceștia au ajuns la concluzia că distanța de comunicare este direct influențată de sursa de iluminare auxiliară folosită [7].

În [7], suprafața de lucru folosită a fost împărțită în patru regiuni. La rândul lor, aceste patru regiuni au fost împărțite în alte patru subregiuni: U (up, sus), D (down, jos), L (left, stânga) și R (right, dreapta), ca în figura 2.4. S-au testat diferitele surse de iluminare asupra regiunilor și s-a ajuns la datele obținute în tabela 2.2. Valorile din tabelă reprezintă intensitatea luminoasă exprimată în Lumeni ($1lm = 1cd \times sr$). Este de menționat că projectorul LED a fost amplasat între suprafața A3 și A4 la o distanță de 1,8m de masă și la o înălțime de 1,8m



Figura 2.4: Arena kilobotilor din laboratorul de robotică al universității din Sheffield. [7]

față de podea, pe când banda cu LED-uri a fost montată pe un suport la o distanță de 25cm față de masă și la 1,5m față de podea. Suprafața de lucru a kilobotilor se află la o elevație de 1,02m față de podea.

	Fără sursă de lumină		Proiector LED		Bandă cu LED-uri	
	Regiunea 3	Regiunea 4	Regiunea 3	Regiunea 4	Regiunea 3	Regiunea 4
R	15.7	15.1	164	146.5	224	233
L	15.5	15.6	120	171	235	241
U	18	18.4	122	119	132	149
D	12.6	12.9	218	232.5	451	446

Tabela 2.2: Intensitatea luminoasă în funcție de diversele surse de iluminare [7]

2.4 Rază de acțiune

Studiul iluminării suprafeței este necesar și foarte important, deoarece acest parametru influențează direct distanța de comunicare dintre **Overhead Controler** și kiloboti, dar și transmisiunea de date dintre kiloboti.

Articolul universității din Sheffield [7] a intrat în amănunt despre comunicarea dintre **Overhead Controler** și kiloboti. Aceștia au conchis că cea mai potrivită suprafață de lucru pentru folosirea kilobotilor este o tablă albă lucioasă, fapt ce a influențat și alegerea suprafeței de lucru pentru experimentele ce vor fi realizate în această lucrare.

Pentru studiul razei de acțiune în interacțiunea dintre kiloboti am efectuat mai multe măsurători. Toate aceste măsurători pot fi consultate în tabela 2.3. Pentru semnificația punctelor și poziția lor exactă se poate consulta figura 2.5. Semnul "X" din mijloc reprezintă punctul în care a fost poziționat un robot care avea ca scop unic emiterea unui mesaj. În experiment s-a folosit și un alt doilea robot care a fost poziționat din ce în ce mai aproape până când semnalul transmis de robotul din centru era primit și înțeles clar de fiecare dată când acesta a fost transmis. Punctele și distanțele marcate cu culoarea neagră sunt punctele în care comunicația este stabilă în lipsa unei surse de iluminare, pe când cele cu roșu sunt în prezența unei surse suplimentare de iluminat. Se poate observa că prin adăugarea unei surse de iluminat putem

îmbunătății semnificativ comunicarea dintre kiloboți. Această îmbunătățire este observabilă cel mai mult în momentul execuției experimentelor.

Fără sursă de lumină		Cu sursă de lumină (lampă montată deasupra suprafeței de lucru)	
Puncte	Distanță (cm)	Puncte	Distanță (cm)
A1	9.5	B1	11
A2	10	B2	11.5
A3	11	B3	14
A4	10	B4	16
A5	10.5	B5	14
A6	10.5	B6	14
A7	10	B7	12.5
A8	9	B8	13.5
A9	9	B9	12
A10	10	B10	13
A11	11	B11	13.5
A12	11	B12	13
A13	13	B13	14.5
A14	12.5	B14	17
A15	12.5	B15	16
A16	10.5	B16	13
Medie	10.63	Medie	13.7

Tabela 2.3: Evoluția razei de transmisiune în funcție de sursa de lumină folosită.

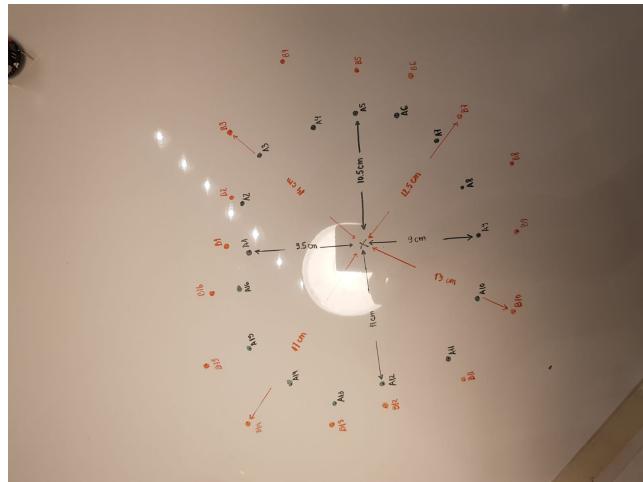


Figura 2.5: Măsurători ale distanței de comunicare între kiloboți.

Capitolul 3

Descrierea componentelor software și a aplicațiilor

Componenta software este alcătuită în cea mai mare parte de codul C embedded încărcat pe kiloboti și pe **Overhead Controller**. Codul a fost scris în **Atmel Studio** deoarece oferă cel mai complet pachet pentru programarea de microcontrolere din familia AVR produse de **Atmel**. În lucrarea de față este vorba despre microcontroler-ul **Atmel ATmega 328p** (descriis mai în detaliu în secțiunea 1.6).

3.1 Atmel Studio

Atmel Studio este un IDE specializat în programarea de microcontrolere fabricate de **Atmel**. De asemenea **Atmel Studio** este considerat și un IDP (platformă de dezvoltare pentru proiecte integrate) ce oferă toate uneltele necesare dezvoltării unui proiect integrat. Unul dintre punctele forte ale IDE-ului este reprezentat de mediul ușor de folosit care permite scrierea, construcția și depanarea aplicațiilor scrise în limbajul C, C++ sau chiar și în limbaj de asamblare. Pe lângă acestea, prin folosirea IDE-ului se pot elimina mai mulți pași intermediari între compilarea codului și încărcarea efectivă pe cip, lucru ce poate influența exponențial timpul de dezvoltare al unei aplicații.

În cadrul acestei lucrări, fiecare proiect realizat în IDE-ul **Atmel Studio** este scris în limbajul C și este configurat pentru a funcționa pe microcontroler-ul ATmega 328p.

Pentru fiecare proiect realizat în **Atmel Studio** se pot defini anumite simboluri care vor fi luate în considerare la momentul execuției. Cel mai vital simbol definit în experimentele realizare este F_CPU căruia îi este asociată valoarea 8000000 ce reprezintă 8 MHz, adică frecvența la care rulează microntolerul. Pe lângă aceste simboluri definite pe proiect se pot defini și simbolurile clasice din cod, prin directiva **#define**.

3.2 AVRdude

Orice sistem ce are la baza un microcontroler din familia AVR se folosește de **AVRdude**. **AVRdude** este o unealtă ce permite descărcarea, încărcarea și manipularea memoriei ROM și EEPROM prin tehnica ISP.

AVRdude poate fi utilizat foarte ușor din linia de comandă, poate identifica microcontrolerul la care este conectat, poate citi și modifica biți de tip *fuse* care sunt folosiți în setarea celor mai importanți parametrii de funcționare ai microcontroler-ului, cum ar fi multiplicatorul pentru frecvența de lucru. Datorită construcției ce nu ține cont de sistemul de operare al stației de

lucru, comenziile puse la dispoziție pot fi implementate foarte ușor într-un script, sau într-un fișier de tipul *makefile*, prin care putem printr-un singur click să compilăm și codul, dar să-l și încărcăm pe microcontroler.

O extensie ce ușurează și mai mult utilizarea programului **AVRdude** este **AVRDudeSS** care adaugă o interfață grafică peste toate comenziile oferite de **AVRdude**, ușurând astfel interacțiunea dintre utilizator și program. **AVRDudeSS** se va folosi în cadrul acestei lucrări pentru manipularea memoriei microcontroler-ului de pe **Overhead Controler**, dar și pentru a reîncărca bootloader-ul pe kiloboți în cazul în care acesta se corupe. Componența software nu va fi folosită totuși și pentru încărcarea de programe pe kiloboți. De acest lucru se ocupă **Overhead Controler**-ul care comunică cu bootloader-ul de pe kiloboți care manipulează biții din memoria de program.

3.3 Kilolib

Kilolib este principala librărie pe care o vom folosi în dezvoltarea programelor. Aceasta este o librărie open-source dezvoltată de K-Team, aceeași companie care fabrică kiloboții. Deoarece aceștia sunt responsabili atât pentru componenta hardware, cât și pentru cea software, putem spune că este de așteptat ca fiabilitatea sistemului să fie una destul de mare.

Prin librăria **kilolib** se adaugă un nivel de interfațare între resursele microcontroler-ului și programator. Prin adăugarea acestui nivel se micșorează cantitatea de muncă pe care trebuie să o depună programatorul, deoarece acesta nu mai este nevoie să facă programarea la nivel de registre și adrese de memorie, lucru ce face programul mult mai ușor de înțeles.

Kilolib prezintă mai multe funcții vitale în exploatarea kiloboților. Acestea sunt:

- `uint8_t estimate_distance(const distance_measurement_t *d);`

Această funcție primește ca parametru un pointer la un obiect de tip `distance_measurement` ce conține intensitatea semnalului IR recepționat. Pe baza acestei intensități se poate estimă o distanță față de robotul care a transmis mesajul. Funcția returnează un număr întreg fară semn ce reprezintă distanța în milimetri față de transmițător. Această funcție se bazează pe un tabel de calibrare intern ce este stocat în memoria EEPROM a kiloboților. Deoarece kiloboții sunt achiziționați de la K-team aceștia vin cu acest tabel deja populat și nu este necesară o recalibrare.

- `void delay(uint16_t ms);`

Prin această funcție se poate introduce o întârziere în execuția programului. Întârzierea are un caracter blocant, adică procesorul nu poate efectua alte operații în timpul în care se dorește să se execute întârzierea. Unicul parametru `ms` este un număr întreg, fară semn, pe 16 biți ce reprezintă durata în milisecunde a întârzierii. Nu este cea mai eficientă metodă de a crea întârzieri datorită caracterului blocant.

- `uint8_t rand_hard();`

Funcția se folosește pentru a genera un număr aleator. Orice generator de numere aleatoare are nevoie de un *seed*. Aceasta este generată prin măsurarea tensiunii bateriei robotului și introducerea valorii analogice în ADC-ul microcontroler-ului și generează un număr aleator pe baza valorii celui mai puțin semnificativ bit din numărul obținut. Ultimul bit se schimbă relativ frecvent, iar acest lucru contribuie la caracterul aleator al generatorului. Valoarea returnată este un număr întreg, fară semn, pe 8 biți și reprezintă numărul aleator generat.

- `uint8_t rand_soft();`

Această funcție, ca și cea de mai sus, încorporează un generator de numere aleatoare. În acest caz *seed*-ul generatorului este specificat prin funcția `rand_seed`. Această variantă de generare a numerelor aleatoare este mai rapidă decât varianta hardware descrisă mai sus.

- `void rand_seed(uint8_t seed);`

Funcția schimbă *seed*-ul generatorului de numere aleatoare folosit în funcția `rand_soft`. Parametrul ce trebuie transmis funcției este un număr întreg, fără semn, pe 8 biți ce reprezintă valoarea *seed*-ului.

- `int16_t get_ambientlight();`

Kiloboții au incorporat un senzor de lumină ambientală ce permite măsurarea cantității de lumină detectată de fotodiodă. Valoarea returnată este o valoare pe 16 biți, dar doar 10 biți dintre aceștia sunt folosiți pentru stocarea valorii, adică valoarea măsurată este între 0 și 1023. Măsurarea trebuie făcută cu atenție deoarece tensiunea măsurată este trecută prin același ADC care este folosit și de funcția de estimare a distanței, adică valoarea măsurată poate să fie eronată din cauza unor suprapuneri de instrucțiuni în cadrul ADC-ului.

- `int16_t get_voltage();`

Funcția returnează o valoare pe 10 biți (între 0 și 1023) ce reprezintă nivelul de tensiune rămas în baterie. Această valoare poate fi folosită ulterior pentru a lua decizii, sau chiar pentru a opri execuția programului în cazul în care nivelul de încărcare devine critic.

- `int16_t get_temperature();`

Această funcție returnează o valoare pe 10 biți (între 0 și 1023) ce reprezintă temperatura măsurată pe kilobot. Senzorul nu este tocmai precis și sesizează doar schimbări mari în temperatură. Ordinul de mărime după care se sesizează o schimbare este în jur de 2° Celsius sau mai mult.

- `void set_motors(uint8_t left, uint8_t right);`

Această funcție permite acționarea motoarelor cu vibrații de pe kiloboți pentru a putea efectua mișcări controlate. Motoarele sunt controlate folosind semnale PWM generate hardware. Cei 2 parametrii sunt numere întregi, fără semn, pe 8 biți și reprezintă factorul de umplere al semnalelor ce vor fi aplicate pe motoare. Cu cât acest număr este mai mare cu atât motorul va fi acționat mai puternic. O acționare mai puternică nu înseamnă totuși un rezultat mai bun, o valoare foarte mare al acestui parametru poate face un kilobot să efectueze o mișcare ce nu este previzibilă și nu poate fi controlată. Valoarea 0 reprezintă un factor de umplere de 0% adică motorul nu este acționat deloc, iar valoarea 255 este asociată unui factor de umplere de 100%, adică motorul este acționat la întreaga sa capacitate. Valorile pentru parametrii ar trebui să fie mai mereu valorile stabilite pentru calibrarea motoarelor ce pot fi accesate prin variabilele oferite de `kilolib`. Pentru mai multe informații legate de calibrarea motoarelor se poate consulta secțiunea 2.2.

Dacă se dorește tranziția factorului de umplere de la 0% la un factor de umplere mai mare de 10%, trebuie ca motoarele să fie pornite la intensitate maximă timp de 10ms pentru a înginge frecarea cu suprafața de lucru.

O recomandare a producătorului este ca într-un interval de 2 secunde motoarele să nu stea la intensitate maximă mai mult de 50ms, deoarece poate provoca daune ireversibile motorului. Valorile tipice ce sunt folosite ca parametrii pentru această funcție sunt între 50 și 90.

- `void spinup_motors();`

Această funcție efectuează acțiunea necesară, descrisă mai sus, pentru ca motoarele să depășească frecarea cu suprafață de lucru, și anume să acționeze motoarele la intensitate maximă pentru $15ms$. Folosirea funcției oferă un rezultat identic cu execuția codului următor:

```
set_motors(255, 255);
delay(15);
```

- `void set_color(uint8_t color);`

Funcția primește ca parametru un număr întreg, fără semn, pe 8 biți care este folosit pentru a determina culoarea pe care o va avea LED-ul RGB de pe kilobot. Fiecare culoare are o rezoluție pe 2 biți, deci există 4 valori ce reprezintă intensitatea culorii respective (0 reprezintă că LED-ul este stins, iar 3 că LED-ul este la intensitate maximă).

Pentru a ușura interacțiunea cu această funcție s-a creat macro-ul `RGB` care poate fi folosit pentru setarea mai usoară a bitilor pentru fiecare culoare. Spre exemplu `RGB(0,0,0)` reprezintă că LED-ul nu are niciun canal de culoare pornit, prin urmare acesta este închis complet, pe când `RGB(0,3,0)` pornește canalul de verde la intensitate maximă, prim urmăre LED-ul va afișa culoarea verde intens.

- `void kilo_init();`

Prin apelarea acestei funcții se inițializează toate componentele hardware ale kilobot-ului pentru a funcționa în modul dorit pentru a beneficia de toate avantajele librăriei `Kilolib`. Funcția se ocupă de calibrarea oscilatorului hardware, setarea timerelor hardware, configurarea porturilor, configurarea ADC-ului, înregistrarea intreruperilor de sistem și inițializarea subsistemului de mesaje. Este recomandat ca funcția să fie apelată cât mai repede, de preferat apelul funcției să fie chiar prima instrucțiune din funcția `main`.

- `void kilo_start(void (*setup)(void), void (*loop)(void));`

Această funcție primește ca parametrii alte două funcții. Prima funcție, `setup`, se execută o singură dată, iar cea de a doua, `loop` se execută în continuu; se poate întrerupe printr-o comandă de la `Overhead Controller` sau prin tăierea alimentării kilobot-ului.

Funcția `setup` ar trebui să conțină orice inițializare suplimentară care ar trebui efectuată, iar funcția `loop` conține programul efectiv care va rula în buclă.

Apelul funcției este echivalent cu execuția codului:

```
int main() {
    setup();
    while(1) {
        loop();
    }

    return 0;
}
```

Pe lângă toate funcțiile descrise mai sus, Kilolib oferă și o mulțime de variabile de sunț legate de spații de memorie în care sunt stocate informații ce sunt foarte utile în programarea kilobotilor.

Cam toate variabilele folosite sunt declarate ca fiind **extern** și **volatile**. **Extern** specifică faptul că variabila este definită înfărata oricărei funcții, că aceasta există undeva dar nu este necesară alocarea de memorie. Cuvântul cheie **volatile** îi spune compilatorului că această variabilă își poate schimba valoarea oricând, chiar dacă programul nu interacționează cu aceasta. Dacă variabila este declarată de tip **volatile** valoarea acesteia se va citi mereu din memorie și nu va fi copiată într-un registru pentru acces mai rapid.

Aceste variabile sunt:

- **extern volatile uint32_t kilo_ticks;**

Această variabilă stochează un număr întreg, fără semn, pe 32 de biți care este inițializat cu valoarea 0 de fiecare dată când un kilobot este pornit sau resetat și se incrementează de aproximativ 32 de ori pe secundă, sau o dată la 30 de milisecunde. Putem folosi această variabilă pentru crea evenimente ce au loc la momente bine stabilite, fără caracterul blocant ce este introdus de funcția **delay()**.

- **extern volatile uint16_t kilo_tx_period;**

Această variabilă reprezintă perioada de timp la care se încearcă trimitera unui mesaj IR. Dacă un kilobot este configurat să transmită mesaje, acesta va transmite în continu mesajul stabilit fără a putea fi oprit. Se poate totuși schimba conținutul mesajului, iar prin această variabilă putem modifica intervalul de timp dintre mesaje.

- **extern uint16_t kilo_uid;**

În această variabilă se stochează identificatorul unic asociat fiecărui robot. Acest identificator poate fi folosit pentru individualizarea unor secvențe de cod în funcție de ID-ul fiecărui robot, sau chiar poate fi transmis ca mesaj pentru ca roboții din aria robotului transmîtător să știe de la cine este mesajul.

- **extern uint8_t kilo_turn_left;**

Variabila aceasta stochează valoarea pe 8 biți a factorului de umplere pe care trebuie să îl aibă semnalul PWM pentru ca robotul să efectueze un viraj corect la stânga. Valoarea este folosită în funcția **set_motors**.

- **extern uint8_t kilo_turn_right;**

Variabila aceasta stochează valoarea pe 8 biți a factorului de umplere pe care trebuie să îl aibă semnalul PWM pentru ca robotul să efectueze un viraj corect la dreapta. Valoarea este folosită în funcția **set_motors**.

- **extern uint8_t kilo_straight_left;**

Variabila stochează valoarea pe 8 biți a factorului de umplere ce trebuie să îl aibă semnalul PWM aplicat motorului stâng pentru ca robotul să meargă în față. Este necesară acționarea ambelor motoare astfel încât în apelul funcției **set_motors** se va transmite ca parametru atât această variabilă, cât și cea de mai jos.

- **extern uint8_t kilo_straight_right;**

Variabila stochează valoarea pe 8 biți a factorului de umplere ce trebuie să îl aibă semnalul PWM aplicat motorului drept pentru ca robotul să meargă în față. Este necesară acționarea ambelor motoare astfel încât în apelul funcției **set_motors** se va transmite ca parametru atât această variabilă, cât și cea de mai sus.

În cadrul **Kilolib** există și declararea unor funcții de tip *callback* ce oferă programatorului libertatea de a implementa diverse comportamente atunci când se îndeplinesc anumite condiții, sau la declansarea unui eveniment. Funcțiile *callback* puse la dispoziție de **Kilolib** sunt:

- **extern message_rx_t kilo_message_rx;**

Această funcție *callback* este apelată de fiecare dată când se primește un mesaj cu succes. Funcția de *callback* implementată trebuie să primească 2 parametrii, un pointer la mesajul decodat și un pointer la distanța măsurată.

- **extern message_tx_t kilo_message_tx;**

Funcția *callback* este apelată ori de câte ori un mesaj este programat pentru a fi trimis. Aceasta returnează un pointer la mesajul care ar fi trebuit trimis, dacă pointerul este **null** atunci mesajul nu a fost transmis. Prin folosirea acestei funcții se pot lua măsuri speciale în cazul în care nu s-a reușit transmiterea mesajului.

- **extern message_tx_success_t kilo_message_tx_success;**

Această funcție *callback* este apelată de fiecare dată când un mesaj este transmis cu succes. Totuși este necesară precauție, deoarece kilobot-ul ce primește mesajul nu răspunde cu un mesaj de confirmare –i.e. transmiterea cu succes a unui mesaj este asociată cu transmiterea unui mesaj când nu a fost detectat zgomot pe canalul de transmisiune.

După toate cele enunțate mai sus, este usor de înțeles de ce **Kilolib** este indispensabil în implementarea software a sistemelor ce se bazează pe kiloboți.

3.4 Software Overhead Controller

K-team oferă spre vânzare varianta lor de **Overhead Controller**. Costul unei asemenea unități este unul ridicat, deși componentele folosite și sistemul pe care îl formează nu sunt chiar atât de complexe. Având aceste lucruri în vedere s-a decis construirea unei variante proprii a **Overhead Controller**-ului. Pentru implementarea hardware se poate consulta secțiunea 1.2.

În pachetul în care se află librăria **kilolib** se regăsește și versiunea codului C embedded folosit de K-team pentru varianta lor de **Overhead Controller**. Acest cod a reprezentat un punct bun de plecare în dezvoltarea codului final care se regăsește în momentul de față pe prototipul dezvoltat în această lucrare.

Diferențele majore au constat în redefinirea porturilor și a pinilor la care sunt conencate diversele circuite ce ajută la funcționarea corectă a sistemului. De exemplu, în varianta inițială placă cu LED-uri IR era conectată la pinul 4 al portului D, iar LED-ul RGB la portul 0 al portului C.

```
#define ir_port PORTD
#define ir_ddr DDRD
#define ir_mask (1<<4)
#define led_port PORTC
#define led_ddr DDRC
#define led_mask (1<<0)
```

În varianta implementată se pot conecta mai multe plăci cu LED-uri IR, mai exact 6. Circuitele ce realizează comunicația cu plăcile cu LED-uri IR sunt concetate la pinii de la 0

la 5 ai portului C, iar LED-ul RGB este conectat la pinul 2 al portului D. Conexiunile pot fi observate și în schematic-ul **Overhead Controler**-ului din secțiunea 1.2.

```
#define ir_port PORTC
#define ir_ddr DDRC
#define ir_mask 0b00111111
#define led_port PORTD
#define led_ddr DDRD
#define led_mask (1<<2)
```

Pe lângă acestea au fost necesare câteva modificări și în cadrul secvenței de transmitere a mesajelor, scrisă în limbaj de asamblare. Modificarea a fost necesară deoarece **Kilolib** este folosit atât de **Overhead Controler**, cât și de kilobăti, dar sunt secvențe ce trebuie particularizate în funcție de dispozitivul pe care este executată funcția. Pentru **Overhead Controler** trebuie eliminată instrucțiunea **cli** din funcția de transmitere a mesajelor. Dacă instrucțiunea rămâne scrisă atunci întreruperile sistemului sunt dezactivate și se vor omite mai multe pachete primite de la **KiloGUI**, deoarce cât timp se execută funcția de trimisere de pachete, restul de pachete primite nu sunt luate în considerare.

După aceste modificări programul poate fi compilat, iar fișierul **.hex** obținut se poate încărca pe **Overhead Controler** folosind unealta **AVRdude**, ce folosește un dispozitiv **USBASP** ce programează microcontroler-ul prin interfața serială **SPI**. Bineînteles că se pot efectua și alte modificări la codul sursă al **Overhead Controler**-ului dacă se dorește acest lucru. Dacă totul a decurs bine, **Overhead Controler**-ul va semnala buna funcționare prin aprinderea și stingerea intermitentă a LED-ului roșu de 5 ori atunci când controlerul este alimentat.

3.5 Bootloader

Bootloader-ul unui kilobot reprezintă codul (firmware-ul) ce este încărcat pe microcontroler-ul **ATmega 328p** de pe kilobăti. Acest bootloader este responsabil pentru gestionarea pachetelor primite de la **Overhead Controler**. Aceste pachete pot conține instrucțiuni precum cele ce sunt trimise de **KiloGUI 1.4** sau chiar conținutul fișierului **.hex** ce a fost rezultat în urma compilării codului. Dacă pachetele conțin codul compilat, bootloader-ul va transferă biți primiți în memoria de program a microcontroler-ului. Acest lucru înseamnă că bootloader-ul este primul strat ce se află peste resursele hardware ale unui kilobot, adică programul scris de utilizator trece prin bootloader și acesta se ocupă de execuția lui. Pe lângă execuția programului scris, bootloader-ul trebuie să se ocupe și de celelalte rutine de sistem ce asigură buna funcționare a kilobot-ului, în acest mod ne mai fiind necesară implementarea lor în programul scris de utilizator. Aceste rutine pot fi de exemplu întreruperi la anumite momente de timp pentru a verifica dacă s-au recepționat alte comenzi de la **Overhead Controler** sau pentru a ține cont în ce mod de funcționare se află robotul.

O definiție mai simplă a bootloader-ului ar fi că acesta reprezintă o interfață între programul scris de utilizator și resursele hardware ale kilobot-ului. Acest nivel de interfațare ușurează și mai mult interacțiunea cu kilobătii și scutește utilizatorul de a mai efectua diverse operații în codul său.

Deoarece acest bootloader se ocupă de scrierea codului primit prin IR în memoria de program a microcontroler-ului fără a fi implementată o metodă de verificare a datelor primite, există posibilitatea ca scrierea să nu se realizeze corect de fiecare dată. Această eroare poate fi datează zgromotului regăsit pe canalul de transmisie, distanței mari dintre **Overhead Controler**

și robot (fapt ce poate duce la o intersitate scăzută a semnalului IR care nu depășește valoarea de prag a fotodieodei pe kiloboți), sau din alte cauze obiective. Dacă o asemenea eroare are loc, conținutul memoriei de program al microcontroler-ului de pe robot poate fi considerat compromis. Este foarte probabil ca și bootloader-ul să devină corupt și să nu mai răspundă la comenzi trimise. În această situație este necesară reîncărcarea bootloader-ului cu aceeași metodă prin care s-a încarcat programul pe **Overhead Controler**; se folosește aplicația **AVRDudeSS** pentru a încărca fișierul **.hex** ce conține bootloader-ul prin dispozitivul **USBASP**, ce comunică cu microcontroler-ul prin interfața serială **SPI**. În realizarea acestui proces este nevoie de o atenție sporită, deoarece conectorul de pe kiloboți nu este unul clasic **ISP** cum este regăsit pe cele mai multe dispozitive. Pinii de alimentare și masă sunt inversați între ei față de un pinout standard **ISP**, iar prin conexiunea greșită a programatorului robotul se poate defecta. Pentru a evita situația în care această metodă este necesar ca utilizatorul să verifice cu atenție conexiunea dintre **Overhead Controler** și kiloboți.

3.6 Aplicații

Aplicațiile reprezintă implementările software care s-au realizat pe parcursul acestui proiect pentru a demonstra modul de funcționare al sistemelor colaborative și cum putem simula acest tip de comportament folosind kilobotii.

Toate aplicațiile au fost scrise în limbajul **C**, folosind **Atmel Studio**, cu toate proiectele configurate pentru microcontroler-ul **Atmel ATMega 328p**, cu simbolul **F_CPU** definit ca fiind **8000000** și cu librăria **kilolib** inclusă.

Multe dintre aceste programe au în comun diverse secvențe sau funcții care pot fi folosite indiferent de aplicație. Un exemplu de astfel de funcție este **set_motion** care permite controlul asupra robotilor într-un mod mai convenabil.

În secvența de cod de mai jos se poate observa modul de implementare a funcției. Pentru folosirea ei s-au definit 4 macro-uri ce reprezintă fiecare tip de mișcare pe care poate să o efectueze un kilobot. Pe lângă aceste macro-uri s-a folosit și o variabilă care memorează mișcarea curentă. Parametrul primit reprezintă noua direcție de deplasare.

```
#define STOP 0
#define FORWARD 1
#define LEFT 2
#define RIGHT 3

int current_motion = STOP;

void set_motion(int new_motion) {
    if (current_motion != new_motion) {
        current_motion = new_motion;

        if (current_motion == STOP) {
            set_motors(0, 0);
        }
        else if (current_motion == FORWARD) {
            spinup_motors();
            set_motors(kilo_straight_left, kilo_straight_right);
        }
        else if (current_motion == LEFT) {
            spinup_motors();
        }
    }
}
```

```

        set_motors(kilo_turn_left, 0);
    }
    else if (current_motion == RIGHT) {
        spinup_motors();
        set_motors(0, kilo_turn_right);
    }
}
}

```

• Împrăștiere

În această aplicație kiloboții pleacă dintr-un grup compact și încearcă să se întindă pe o suprafață cât mai mare, dar totuși să rămână oarecum în aceeași zonă de acțiune. Pentru a realiza acest lucru trebuie ca kiloboții să se depărteze unul de celălalt până când distanța estimată pe baza intensității semnalului luminos este mai mare decât un prag ales. Când acest lucru se întamplă kiloboții vor semnala prin aprinderea LED-ului de culoare verde. Chiar dacă robotul a ajuns în această stare el verifică în continuare dacă distanța față de ceilalți roboți este corespunzătoare. În caz contrar, acesta începe iar să se deplaseze.

Algoritmul folosit se bazează pe funcția `rand_hard` pentru a genera un număr aleator. Pe baza acestui număr se decide ce fel de mișcare o să efectueze robotul. Prin folosirea generatorului de numere aleatoare se asociază o probabilitate de 0.5 pentru ca robotul să meargă în fată, o probabilitate de 0.25 pentru a efectua un viraj la stânga și tot 0.25 pentru un viraj la dreapta. În fiecare secundă se verifică dacă s-a recepționat un mesaj nou și dacă răspunsul este pozitiv atunci robotul generează un nou număr și își schimbă direcția de deplasare. Dacă nu s-a recepționat niciun mesaj atunci înseamnă că robotul nu mai are niciun alt vecin în zona de distanțare. Mai jos este codul folosit în implementarea acestui comportament.

```

if(kilo_ticks > (last_changed + 32)) {
    last_changed = kilo_ticks;
    if(message_rcved == 1) {
        message_rcved = 0;

        random_number = rand_hard();
        dice = random_number % 4;

        if (dice == 0 || dice == 1) {
            set_motion(FORWARD);
            set_color(RGB(1,0,0));
        } else if (dice == 2) {
            set_motion(LEFT);
            set_color(RGB(0,0,1));
        } else if (dice == 3) {
            set_motion(RIGHT);
            set_color(RGB(0,1,1));
        } else {
            set_motion(STOP);
            set_color(RGB(0,0,0));
        }
    }
}

```

```

        }
        else {
            set_motion(STOP);
            set_color(RGB(0,1,0));
        }
    }
}

```

Pentru a seta raza în care nu se dorește să se afle alt robot s-a folosit macro-ul `#define Distance_Threshold 75`.

Funcția de *callback* pentru recepționare de mesaje filtrează mesajele în funcție de distanță măsurată. Dacă distanța este mai mică decât valoarea de prag precizată atunci mesajul se ia în considerare, în caz contrar acesta se omite.

```

void message_rx(message_t *msg, distance_measurement_t *dist_measure) {
    rx = *msg;
    rcvd_data = msg->data[0];
    dist = estimate_distance(dist_measure);
    if(dist < Distance_Threshold) {
        message_rcved = 1;
    }
}

```

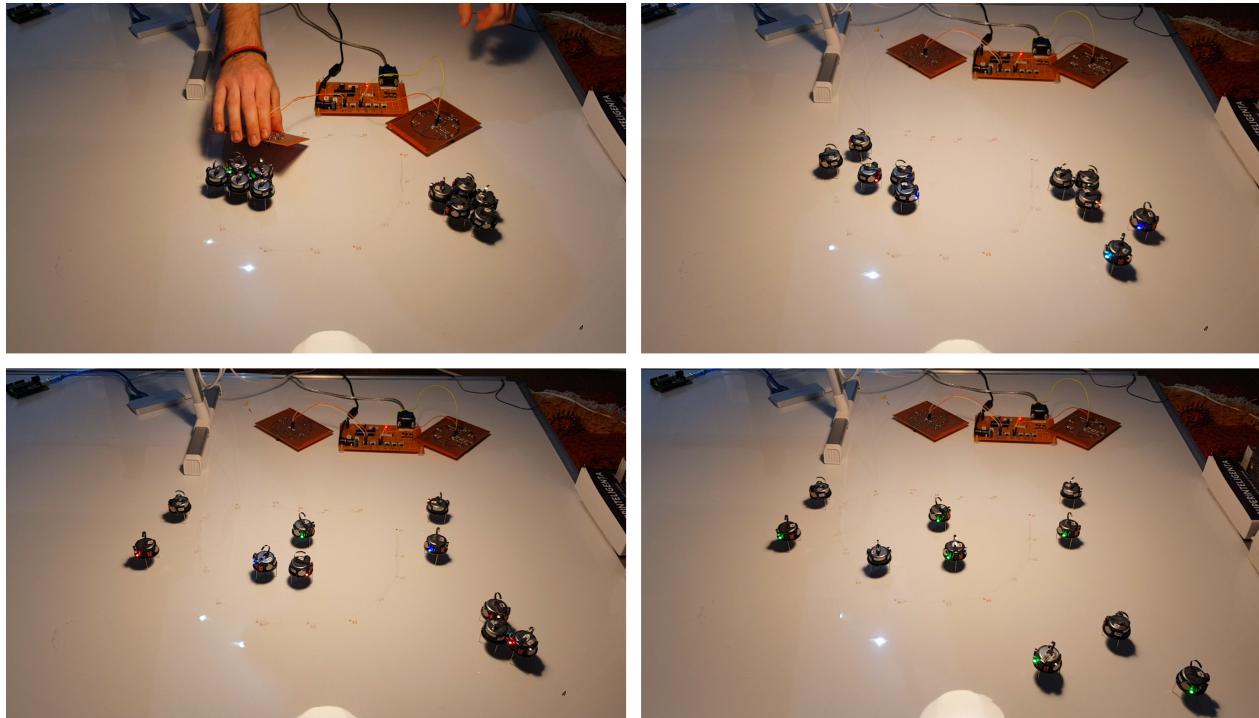


Figura 3.1: Împrăștirea kiloboților.

În figura 3.1 se poate observa desfășurarea algoritmului descris mai sus. Inițial roboții au fost poziționați în 2 grupuri, urmând ca aceștia să se împrăștie pe suprafața de lucru.

• Strângere

În această aplicație se dorește ca roboții să se grupeze într-o structură mai mare, asemănător unui magnet care atrage mai multe obiecte metalice din jur. Pentru a realiza acestă aplicație s-au folosit câțiva roboți care au ca scop marcarea zonei unde o să se adune restul roboților. Acești roboți au fost denumiți roboți *beacon*. Roboții de tip *beacon* emit un mesaj ce poate fi folosit de ceilalți roboți pentru a estima distanța față de ei.

Ceilalți roboți au ca scop găsirea roboților de tip *beacon* și de a se apropiă de aceștia până distanța este foarte mică. Dacă robotul ajunge suficient de aproape de unul de tip *beacon* acesta mostenește comportamentul acestuia și începe și el la rândul său să emită mesajul specific *beacon*. Dacă robotul nu primește niciun mesaj de la un robot de tip *beacon*, acesta se va mișca aleator pe suprafața de lucru până va recepționa un mesaj.

Implementarea acestei aplicații este complicată destul de mult de faptul că roboții nu au un sens al direcției, distanța fiind singurul lucru pe care aceștia pot să îl estimateze.

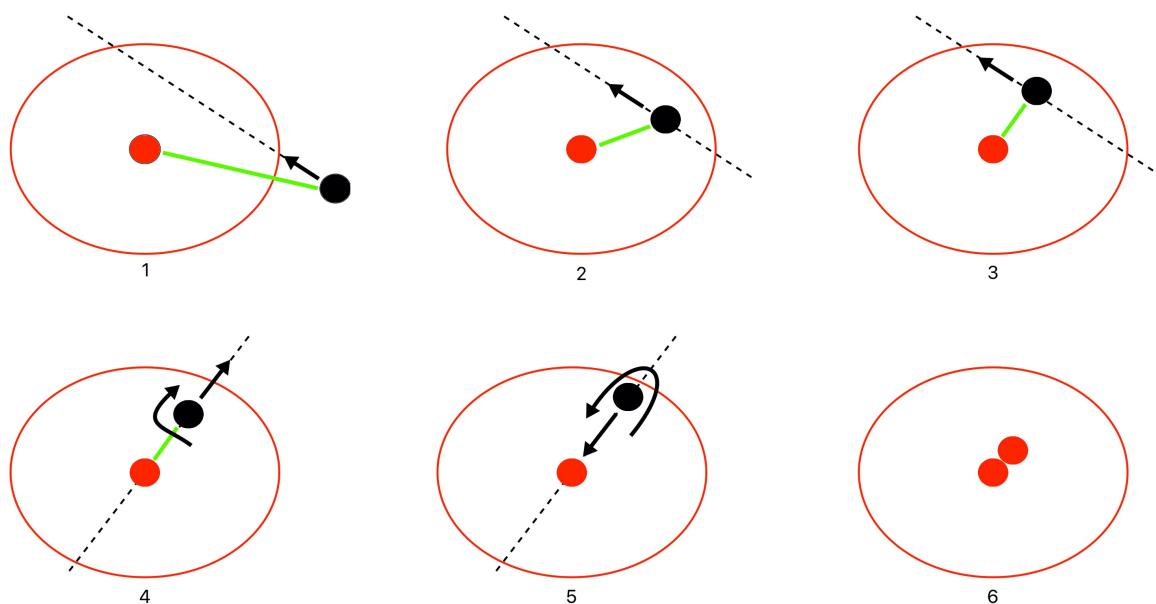


Figura 3.2: Pașii de funcționare al algoritmului de Strângere.

În figura 3.2 este reprezentat modul de funcționare al algoritmului. În figură cercul umplut de culoarea roșie reprezintă robotul de tip *beacon*, cercul umplut de culoarea neagră este robotul care încearcă să ajungă la robotul *beacon*, iar cercul roșu exterior reprezintă aria în care se recepționează mesajul transmis de robotul *beacon*. Linia punctată arată traectoria robotului, linia verde ilustrează distanța dintre robotul care se mișcă și robotul *beacon*. Săgețile sugerează orientarea robotului, dar și mișările de rotație pe care le efectuează.

În pasul 1 un robot se apropie de aria de acțiune a robotului de tip *beacon*. O dată intrat în acesta se va mișca doar în față și va măsura distanța de ori de câte ori poate. Indiferent de direcția din care a venit robotul, distanța față de *beacon* se va micșora mereu, atât timp cât acesta merge în față. Acesta se va deplasa în față până când unghiul format dintre traectoria robotului și dreapta ce are lungimea distanței dintre cei 2 roboți este de 90° . Punctul în care se întâmplă acest lucru se poate vizualiza în pasul 3 din figură. După ce se ajunge în acest punct, robotul se mai deplasează puțin în față pentru a sesiza că distanța curentă s-a mărit față de cea anterioară, moment în care robotul se oprește.

Robotul se reorientizează, încearcând să găsească direcția în care se află robotul de tip *beacon*. Acesta neavând un sens al direcției efectuează o rotire la dreapta cu 90° și se mișcă puțin în față pentru a vedea dacă distanța după efectuarea mișcării scade sau nu față de *beacon*. Acest comportament poate fi vizualizat în pasul 4. Dacă distanța față de *beacon* scade înseamnă ca direcția este corectă și că robotul va ajunge lângă *beacon* dacă continuă să se miște în față. În cazul prezentat în figură distanța crește după ce se efectuează rotirea la dreapta. Robotul sesizează acest lucru și efectuează o rotire cu 180° de grade la stânga (pasul 5) și continuă deplasarea în față până când ajunge foarte aproape de robotul *beacon*. După ce a ajuns în acest punct acest robot devine și el unul de tip *beacon* (pasul 6).

Cazul prezentat mai sus este un caz ideal. În realitate este foarte dificil ca un kilobot să efectueze mișcări atât de precise, deoarece sunt mulți factori ce influențează mișcarea roboților, factori ce țin de calibrarea motoarelor, suprafața de lucru și estimarea greșită a distanței. Erorile sunt reprezentate de traiectorii care ar trebui să fie drepte, dar nu sunt și de rotiri care nu sunt de exact 90° ceea ce îngreunează aplicarea abordării prezentate mai sus. Pentru a combate cât mai mult aceste fenomene, programul verifică în continuu distanța curentă și o compară cu cea anterioară și încearcă pe baza mișcărilor de rotație anterioare să determine care este cea mai bună direcție de deplasare. Mișcările pe care robotul le va efectua se regăsesc în figura 3.2, doar că acestea se pot repeta până când robotul ajunge în punctul dorit. Există și cazuri apropiate de cel ideal în care parcurgerea celor 6 pași din figură sunt suficienți pentru a poziționa robotul.

Programul are 2 ramuri, una pentru roboții de tip *beacon* și una pentru roboții care se deplasează. Pentru roboții de tip *beacon* programul este destul de simplu:

```
void message_tx_success() {
    message_sent = 1;
}

void setup1() {
    transmit_message.type = NORMAL;
    transmit_message.data[0] = 1;
    transmit_message.crc = message_crc(&transmit_message);
}

void loop1() {
    if(message_sent == 1) {
        message_sent = 0;
        set_color(RGB(1, 0, 0));
        delay(100);
        set_color(RGB(0, 0, 0));
    }
}
```

Robotul stă pe loc și transmite un mesaj și semnalează acest lucru prin aprinderea și stingerea intermitentă a LED-ului RGB cu culoarea roșie.

Robotul care trebuie să se apropie se folosește de funcția `move_to_beacon` pentru a avea comportamentul descris mai sus.

Funcția se comportă ca o mașină cu număr finit de stări, în cazul acesta sunt 2 stări. Prima stare de funcționare face ca robotul să se depleteze în față până când distanța

curentă față de *beacon* este mai mare decât cea anterioară. Când acest lucru se întâmplă se trece în starea 2 în care robotul încearcă să găsească orientarea *beacon*-ului. Când consideră că a găsit direcția bună trece înapoi în starea 1.

```

void move_to_beacon() {
    if(state == 1) {
        if(firstMove) {
            firstMove = 0;
            set_motion(FORWARD);
            set_color(RGB(1,0,0));
        }

        else if (dist > last_dist) {
            set_motion(STOP);
            set_color(RGB(0,0,1));
            state = 2;
            firstMove = 1;
        }
    }
    else if(state == 2) {
        if(firstMove) {
            last_dist2 = dist;
            set_motion(correct_motion);
            set_color(RGB(1,0,1));
            mayGetOutOfRange = 1;
            delay(turnMultiplier * 1500);
            turnMultiplier = 1;
            set_motion(FORWARD);
            delay(2000);
            firstMove = 0;
        }
        else if (firstMove == 0 && dist >= last_dist2) {
            set_motion(STOP);
            set_color(RGB(1,1,0));
            if(correct_motion == RIGHT)
                correct_motion = LEFT;
            else if(correct_motion == LEFT)
                correct_motion = RIGHT;
            turnMultiplier = 3;
            firstMove = 1;
            mayGetOutOfRange = 0;
        }
        else if (firstMove == 0 && dist < last_dist2) {
            mayGetOutOfRange = 0;
            state = 1;
            firstMove = 1;
        }
    }
    last_dist = dist;
}

```

Funcția de mai sus este apelată în funcția `loop` de fiecare dată când se primește un mesaj de la `beacon` iar robotul nu este în poziția corectă.

```

void loop2() {

    if (startingMove) {
        move_random();
        startingMove = 0;
    }
    if(new_message1 == 1) {

        if(dist <= inPositionDistance && (data == 1 || data == 2)) {
            set_motion(STOP);
            set_color(RGB(1,1,1));
            if(firstTimeInPosition) {

                transmit_message.type = NORMAL;
                transmit_message.data[0] = 2;
                transmit_message.crc = message_crc(&transmit_message);

                firstTimeInPosition = 0;
                isInPosition = 1;
            }
        }
    }

    else {
        if(data == 1 && !isInPosition)
            move_to_beacon();
    }
    new_message1 = 0;
}

if (kilo_ticks > last_update2 + 3 * SECOND)
{
    last_update2 = kilo_ticks;

    if(new_message2) {
        new_message2 = 0;
    }
    else {
        if (mayGetOutOfRange == 1) {
            if (correct_motion == RIGHT) {
                correct_motion = LEFT;
            }
            else if (correct_motion == LEFT){
                correct_motion = RIGHT;
            }
            set_motion(correct_motion);
            delay(3*1500);
            set_motion(FORWARD);
            delay(2000);
        }
        else {
    }
}

```

```

        transmit_message.type = NORMAL;
        transmit_message.data[0] = 0;
        transmit_message.crc = message_crc(&transmit_message);

        correct_motion = RIGHT;
        isInPosition = 0;
        state = 1;
        firstMove = 1;
        last_dist = 1000;
        move_random();
        set_color(RGB(0,1,1));
    }
}
}
}

```

Prima dată când se apelează funcția `loop` variabila semafor `startingMove` are valoarea `true` și impune robotului să își înceapă mișcarea aleatoare până la găsirea semnalului de la *beacon*. În funcție se verifică, pe baza distanței esitmate, dacă robotul se află în poziția corectă. Dacă da, atunci robotul își oprește mișcarea și devine și el unul de tip *beacon* și începe să emită și el la rândul lui mesaje. În cazul în care distanța față de robotul *beacon* este mai mare decât pragul ales, dar totuși se recepționează mesaje de la acesta se va apela funcția `move_to_beacon`. Se verifică de asemenea dacă după o perioadă de 3 secunde s-a recepționat cel puțin un mesaj de la *beacon*. Dacă nu s-a recepționat înseamnă că robotul a ieșit din zona de acțiune a *beacon*-ului și va reseta variabilele folosite și se va mișca aleator. Tot în această ramură a programului se verifică valoarea variabilei semafor `mayGetOutOfRange` care este setată în starea 2 din funcția `move_to_beacon`, unde există riscul ca robotul să iasă din zona de acțiune a *beacon*-ului. Dacă variabila are valoarea `true`, înseamnă că robotul a ieșit din zona acțiune în timp ce încerca să găsească direcția corectă către *beacon*, iar programul va încerca să facă robotul să se întoarcă înapoi spre *beacon*.

În programul scris distanța de prag este definită astfel `int inPositionDistance = 33;`, adică pentru ca un robot să poată să considere că se află într-o poziție corectă trebuie să se afle la o distanță mai mică sau egală de 33mm față de un robot *beacon*.

În figurile 3.3 și 3.4 se pot observa rezultatele obținute după executarea codului descris mai sus. În prima figură s-a început experimentul cu un singur robot de tip *beacon*, iar în a doua figură experimentul s-a realizat cu doi roboți de tip *beacon*. Se observă că în ambele cazuri toți kiloboții reușesc să se alăture grupului format.

• Orbitare

Această aplicație își propune să reproducă mișcarea de orbitare, similar cu modul în care planetele orbitează în jurul soarelui. Mișcarea presupune existența unui kilobot care să reprezinte centrul (soarele), similar cu roboții *beacon* de la aplicația anterioară, care să fixeze punctul în jurul căruia vor orbita ceilalți kiloboți.

Roboții care orbitează trebuie să aibă definită direcția în care se dorește să se facă orbitarea și inversul ei. În programul implementat aceste direcții sunt definite astfel:

```

uint8_t direction = LEFT;
uint8_t countrDirection = RIGHT;

```



Figura 3.3: Algoritmul Strângere cu 1 robot de tip *beacon*.



Figura 3.4: Algoritmul Strângere cu 2 roboți de tip *beacon*.

Pe lângă aceste direcții trebuie definită și o distanță de prag pe baza căreia robotul trebuie să decidă dacă să se apropie de centru, sau dacă trebuie să se îndepărteze. În varianta implementată această distanță este de 40mm, `uint8_t orbitDistance = 40;`

Codul roboților centroizi este similar cu cel de la aplicația de Strângere, deoarece aceștia îndeplinesc un rol asemănător, fiind doar un reper pentru toți ceilalți roboți.

Pentru roboții care orbitează s-a ales o abordare simplă, bazată pe mișcări la stânga și la dreapta. Codul din funcția `loop` a roboților ce orbitează este următorul:

```
void loop2() {
    if(new_message) {
        new_message = 0;
```

```

        if(dist < orbitDistance) {
            set_motion(counterDirection);
            set_color(RGB(2,0,0));
        }
        else if(dist >= orbitDistance) {
            set_motion(direction);
            set_color(RGB(0,0,2));
        }
    }
}

```

Se poate observa că fiecare robot așteaptă până când primește un mesaj de la *beacon*, iar pe baza distanței estimate decide dacă să efectueze o mișcare la stânga sau la dreapta.

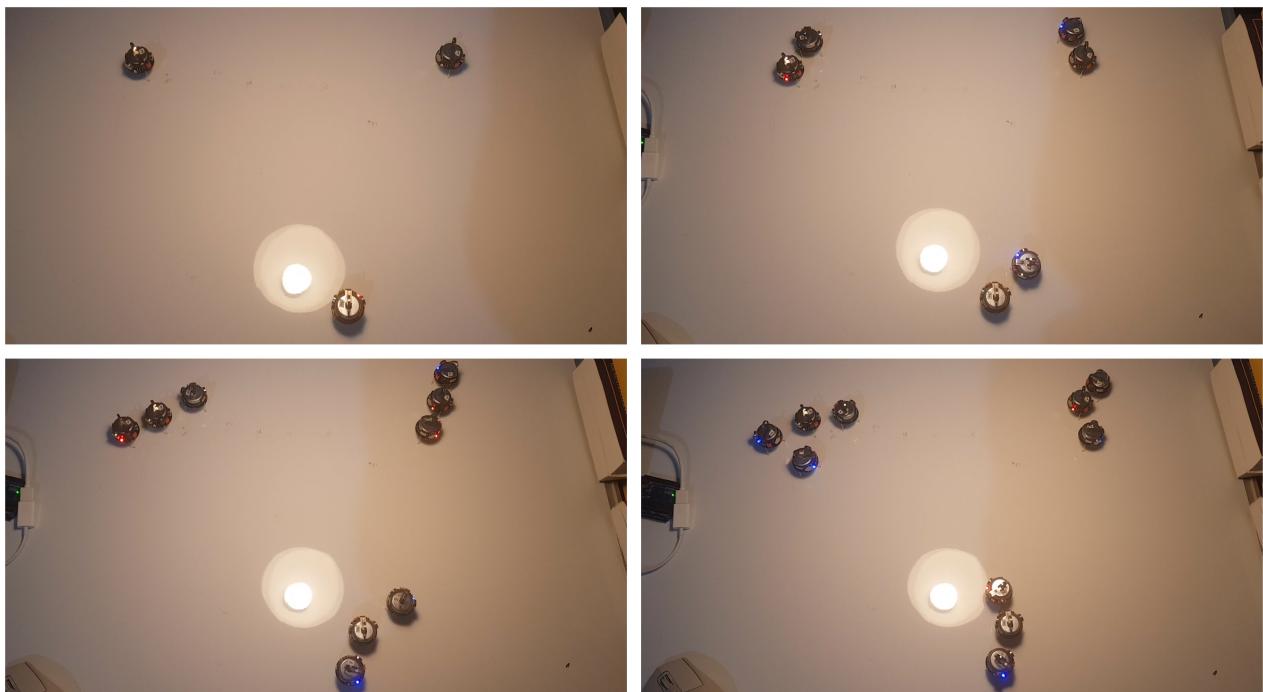


Figura 3.5: Algoritmul Orbitare cu 3 roboți de tip *beacon*.

În figura 3.5 se poate observa modul în care mai mulți kiloboți orbitează în jurul a trei roboți de tip *beacon*.

- **Şir Indian**

Această aplicație are ca scop alinierea robotilor pe o dreaptă. Ca și în cazurile anterioare primul punct din linie este reprezentat de un robot *beacon*. Al doilea robot va încerca să ajungă la primul robot print-o metodă asemănătoare aplicației de Strâgere, iar poziția în care va ajunge acesta va determina direcția în care se va organiza șirul indian, toți restul robotilor urmând să se poționeze în spatele acestuia. Al doilea robot consideră că se află în poziția corectă atunci când distanța față de robotul *beacon* este mai mică decât o valoare de prag aleasă, în cazul acesta 35mm, #define IN_POSITION_DISTANCE 35. Restul robotilor ce urmează să se așeze în linie trebuie să îndeplinească două condiții de distanță. Aceștia trebuie să aibă distanța față de ultimul robot din șir mai mică de 35mm și distanța față de penultimul robot din șir mai mare decât 60mm ,#define IN_POSITION_DISTANCE2 60.

După ce un robot se lipește de șirul format până în acel moment, va încerca să se deplaseze pe conturul liniei până își va găsi locul. Mișcarea de urmărire a conturului se realizează asemănător cu mișcarea de orbitare, adică kilobot-ul va încerca să stea la o distanță fixă față de linie și să orbiteze pe lângă fiecare kilobot ce formează linia. În momentul în care distanța față de un robot este mai mică decât distanța față de kilobot-ul curent, acesta se va schimba în cel față de care distanța este mai mică.

Funcția *callback* de recepționare cu succes a unui mesaj este puțin mai diferită în această aplicație, deoarece dorim să semnalăm momentul în care una dintre cele două condiții de distanță este îndeplinită. Atunci când prima condiție este îndeplinită, LED-ul RGB se va aprinde intermitent în culoarea albastră, iar când a doua condiție este îndeplinită LED-ul RGB se va aprinde intermitent în culoarea verde.

```
void message_rx(message_t *msg, distance_measurement_t *dist_measure) {
    rcvd_message = *msg;
    data1 = msg->data[0];
    data2 = msg->data[1];

    dist = estimate_distance(dist_measure);

    if(data1 == ID - 1) {
        distanceTo1 = dist;
        if(dist < IN_POSITION_DISTANCE) {
            set_color(RGB(0,0,1));
            delay(100);
            set_color(RGB(0,0,0));
            delay(100);
        }
    }
    else if(data1 == ID - 2) {
        distanceTo2 = dist;
        if(dist > IN_POSITION_DISTANCE2) {
            set_color(RGB(0,1,0));
            delay(100);
            set_color(RGB(0,0,0));
            delay(100);
        }
    }
    new_message = 1;
}
```

Kiloboții se vor organiza în linie în ordinea ID-urilor. Dacă un kilobot detectează că agentul ce are valoarea ID-ului mai mică cu unu decât valoarea proprie nu a ajuns în poziția corectă, acesta se va opri și își va aștepta rândul.

Tot algoritmul de apropiere s-a inglobat într-o singură funcție cu denumirea *goTo* ce primește 2 parametrii, primul este ID-ul robotului față de care se dorește să se facă apropierea, iar al doilea parametru este distanța curentă față de acel kilobot.

```
void goTo(uint8_t id, uint8_t distance) {
```

```

    if(new_message && data1 == id) {
        new_message = 0;
        current_dist = dist;

        if(current_dist <= distance) {
            firstMove = 1;
            inPosition = 1;
        }
        else {
            if(goToState == 1) {

                if(current_dist > last_dist) {
                    goToState = 2;
                }
                set_motion(FORWARD);
                delay(1000);
                set_color(RGB(1,0,0));

            }

            else if(goToState == 2) {
                set_motion(STOP);
                set_color(RGB(0,0,1));
                set_motion(correct_motion);
                delay(turnMultiplier * 1500);
                turnMultiplier = 1;
                set_motion(FORWARD);
                delay(1500);
                goToState = 3;
            }

            else if(goToState == 3) {
                set_color(RGB(1,1,0));
                if(current_dist > last_dist) {
                    if(correct_motion == RIGHT)
                        correct_motion = LEFT;
                    else if(correct_motion == LEFT)
                        correct_motion = RIGHT;
                    turnMultiplier = 3;
                    goToState = 2;
                }
                else {
                    goToState = 1;
                }
            }
        }

        last_dist = current_dist;
        set_motion(STOP);
    }
}

```

După cum se poate observa și din cod, implementarea a fost făcută sub forma unei mașini

cu număr finit de stări. În prima stare robotul merge în față atâtă timp cât distanța față de robotul *beacon* scade. Când s-a detectat prima creștere se trece în starea 2, în care se încearcă deplasarea într-o direcție și apoi se trece în starea 3. În starea 3 se verifică dacă decizia legată de direcție luată în starea 2 este corectă. Dacă este corectă se va trece în starea 1, dacă nu, mașina se va întoarce în starea 2. Robotul se va mișca în direcția opusă față de cea anterioară o perioadă mai îndelungată de timp pentru compensa pentru mișcarea greșită de la pasul anterior.

Această metodă este apelată în cadrul funcției `loop` care este bazată și ea pe o mașină cu număr finit de stări.

```

void loop2() {
    if(state == 1) {
        move_random();
        state = 2;
    }

    else if(state == 2) {
        set_color(RGB(1,0,1));
        if(new_message && data1 == ID - 1 && data2 == 0) {
            set_motion(STOP);
            new_message = 0;
        }
        else if(new_message && data1 == ID - 1 && data2 == 1) {
            state = 3;
            new_message = 0;
        }
    }
    else if(state == 3) {
        set_color(RGB(0,1,1));
        if(!inPosition) {
            if (new_message && data1 != ID - 1 && data2 == 1 && dist <
GO_TO_DISTANCE) {
                new_message = 0;
                state = 4;
                nextToId = data1;
                nextToIdDistance = dist;
            }
            else {
                if(ID == 2)
                    goTo(ID - 1, IN_POSITION_DISTANCE);
                else
                    goTo(ID - 1, GO_TO_DISTANCE);
            }
        }
        else {

            if(ID == 2)
                state = 5;
            else if (new_message && data1 == goToId) {
                nextToId = goToId;
                nextToIdDistance = dist;
                state = 4;
            }
        }
    }
}
```

```

        firstMove = 1;
    }

}

else if(state == 4) {
    set_color(RGB(0,1,1));
    set_motion(STOP);
    if (new_message && data1 != nextToId && dist < nextToIdDistance){
        nextToId = data1;
        new_message = 0;
    }

    else if(new_message && data1 == nextToId) {

        new_message = 0;
        current_dist = dist;
        nextToIdDistance = current_dist;

        if(current_dist < ORBIT_DISTANCE) {
            set_motion(orbit_motion);
            delay(ORBIT_DELAY);
        }
        else if(current_dist >= ORBIT_DISTANCE ) {
            set_motion(orbit_counterMotion);
            delay(ORBIT_DELAY);
        }
    }

    if(distanceTo1 < IN_POSITION_DISTANCE && distanceTo2 >=
IN_POSITION_DISTANCE2) {
        set_motion(STOP);
        state = 5;
    }
    new_message = 0;
}

else if(state == 5) {
    set_motion(STOP);
    set_color(RGB(1,1,1));
    transmit_message.data[1] = 1;
    transmit_message.crc = message_crc(&transmit_message);
    state = 6;
}
}

```

În starea inițială (starea 1) robotului îi este impus să se miște aleator și să găsească un robot ce formează linia. În starea 2 se verifică dacă robotul de la care s-a primit mesaj este cel care are ID-ul mai mic cu unu, în cazul care este acesta robotul se va opri și va

aștepta ca acesta să se poziționeze, dacă robotul este deja poziționat se va trece în starea 3. Starea 3 este cea în care se va apela funcția `goTo` și îi se va impune robotului să se deplaseze spre linie. Se va rămâne în această stare până când robotul se află la o distanță mai mică de `40mm` (`#define GO_TO_DISTANCE 40`) de un robot ce se află deja pozitionat în sir. În starea 4 robotul efectuează deplasarea pe conturul formei și ține mereu cont de robotul lângă care se află prin variabilele `nextToId` și `nextToIdDistance`. În momentul în care distanța față de alt robot este mai mică decât `nextToIdDistance`, valoarea variabilei `nextToId` va fi reactualizată astfel încât să reprezinte ID-ul robotului față de care distanța este mai mică. Această stare persistă până când robotul este în poziția corectă, adică respectă cele două condiții ce țin de distanțe prezentate la început. În acest moment se va trece în starea 5, iar robotul va emite un mesaj prin care anunță că a ajuns cu succes în linie. Robotul va semnala faptul că a ajuns în această stare prin aprinderea LED-ului RGB cu culoarea albă.

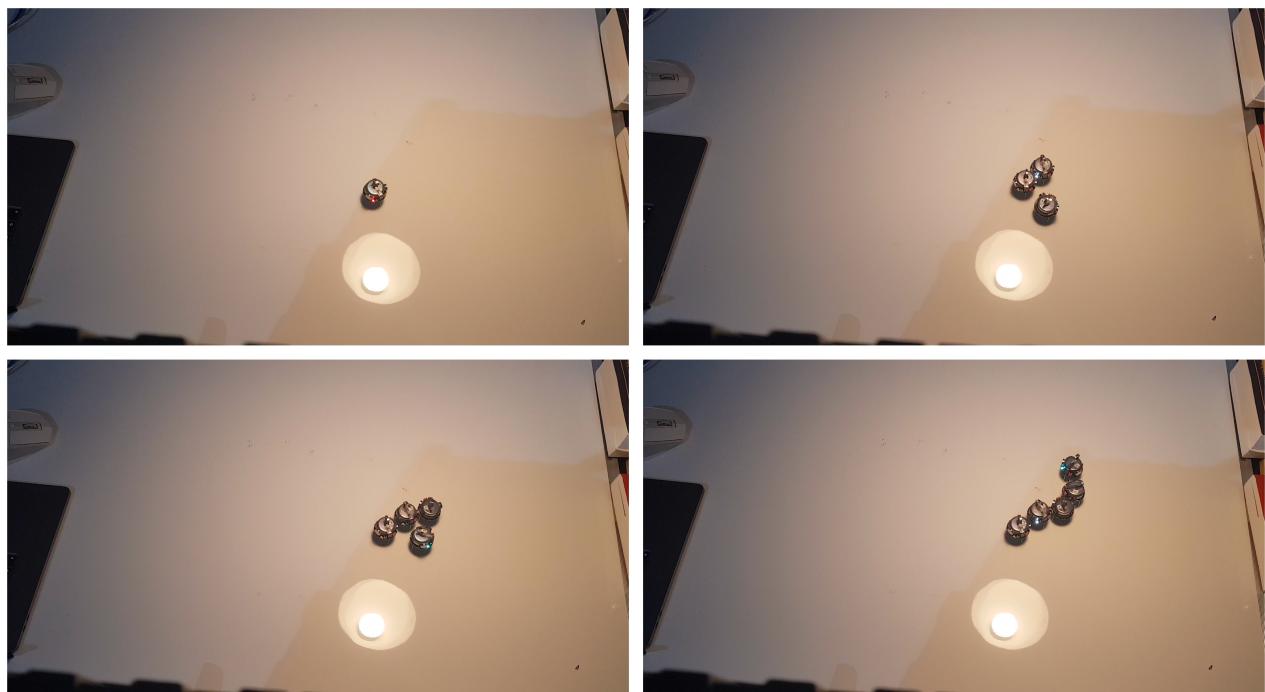


Figura 3.6: Algoritmul ce aliniaza 5 roboți într-un sir indian.

În figura 3.6 se poate observa comportamentul descris mai sus aplicat pe un grup de 5 kilobti. Durata de timp necesară execuției algoritmului este foarte mare, deoarece mișcarea robotilor este intenționat redusă petru ca aceștia să nu omită momentul în care ajung în punctul potrivit, dar și pentru a crește precizia în luarea deciziilor.

Concluzii

Concluzii generale

Pe baza studiului realizat în cadrul elaborării acestei teze se poate conchide că sistemele colaborative au o aplicabilitate vastă și pot aduce un beneficiu real în multe domenii.

În cadrul lucrării s-au îndelinit toate obiectivele inițiale și s-a construit o platformă pe baza căreia se pot elabora versiuni scalate și mai complexe ale aplicațiilor implementate în lucrare. Kiloboții reprezintă poate cea mai bună variantă disponibilă pentru studiul sistemelor colaborative (atât timp cât spațiul de lucru este corespunzător), datorită construcției simple și a metodei de interacționare cu alți agenți. Aplicațiile ce au fost gândite pentru kiloboți pot fi portate atât pe platforme microscopice, cât și macroscopice și pot reprezenta alternative viabile la sistemele centralizate.

În opinia mea, rezultatele obținute în lucrare sunt mulțumitoare, dar există nenumărate alte aplicații care pot fi implementate cu astfel de sisteme. Direcția în care consider că aceste sisteme ar putea face un impact în momentul de față este gestionarea traficului rutier, ducând astfel la emisii mai reduse și la mai puțin timp pierdut în ambuteiaje. Pe termen lung, o dată cu dezvoltarea tehnologiei, roboții pot ajunge să aibă dimensiuni microscopice și să acționeze în corpul uman, devenind indispensabili în aplicații medicale precum tratarea tumorilor.

Contribuții personale

Lucrarea prezentată conține atât contribuții documentate, cât și contribuții personale. Toate contribuțiile documentate au fost marcate cu referințe la diversele lucrări de unde a fost preluată informația. Contribuțiile personale sunt:

- **Construirea Overhead Controler-ului:** În cadrul lucrării a fost creat schematic-ul Overhead Controler-ului, pe baza căruia s-a implementat varianta fizică pe perfboard și trasee din fludor. Codul încărcat este în mare parte oferit deja de către K-Team, dar conține și contribuții proprii.
- **Construirea plăcilor cu LED-uri IR:** Pe baza schematic-ului creat, s-a implementat varianta fizică pe perfboard și trasee din fludor a plăcilor cu LED-uri IR. În cadrul proiectului s-au construit 6 astfel de plăci.
- **Încărcătorul kiloboților:** Design-ul folosit pentru încărcător este inspirat din încărcătorul oferit de K-Team, dar implementarea lui fizică reprezintă o contribuție proprie.

- **Organizarea suprafetei de lucru și elaborarea studiului privind influența iluminării asupra distanței de comunicare dintre kiloboți:** S-a arătat cum prin folosirea unei surse de iluminat auxiliare putem mări raza de transmisiune a mesajelor dintre kiloboți, crescând astfel eficiența sistemului.
- **Aplicațiile:** Toate aplicațiile descrise reprezintă contribuția proprie și arată modul în care funcționează un sistem colaborativ ce are ca actorii kiloboți.

Toate resursele software implementate în cadrul acestui proiect sunt disponibile online¹.

Dezvoltări ulterioare

Domeniul sistemelor colaborative este unul vast și există o multitudine de aplicații ce subliniază utilitatea acestora. La nivelul sistemelor de kiloboți, consider că pe baza algoritmilor descriși mai sus se pot dezvolta și alte aplicații mai complexe.

O aplicație ce prezintă interes este crearea de forme geometrice. Pe baza algoritmilor de urmărire de contur și prin măsurarea distanțelor față de mai mulți roboți se poate crea aproape orice formă geometrică. Desigur, pentru o asemenea aplicație numărul de kiloboți implicați trebuie să fie substanțial mai mare.

O altă aplicație ce ar putea face subiectul unei lucrări ar fi crearea a două grupuri de kiloboți care să simuleze o bătălie între mai mulți indivizi, alcătuită din mai multe lupte mici. Luptele mici ar fi între grupuri de 2-5 kiloboți, câștigătorul fiind desemnat pe baza numărului de indivizi ce iau parte la acțiune. Roboții ar putea observa numărul de kiloboți de care dispune echipa adversă și chiar să solicite ajutor în cazul în care sunt depășiți numeric. Aplicația ar putea avea 2 sau mai multe echipe care iau parte la luptă, fiecare echipă având o strategie diferită, iar prin simularea luptei se poate vedea care strategie este mai eficientă.

O aplicație cu conotație medicală ar fi simularea celulelor canceroase și distrugerea acestora. Aplicația ar debuta printr-un grup de kiloboți care reprezintă celulele, iar unul dintre roboți va fi considerat ca fiind o celulă canceroasă. Celula canceroasă se propagă, iar în timp celulele ce îl învecinează ar deveni și ele afectate. Celulele afectate ar putea cere ajutorul unui grup de roboți care reprezintă un grup de celule ce luptă cu tumoarea și care încearcă să o eliminate.

Tot pe aceeași tematică se poate implementa o aplicație, de actualitate, ce simulează transmiterea virusului SARS-COV-2 pe baza distanței între indivizi. Se poate observa rapiditatea cu care se propagă virusul dacă distanțarea nu este respectată și cât de mult ajută această distanțare. Implementarea ar fi relativ simplă și s-ar putea baza pe o probabilitate de infectare ce poate fi calculată în funcție de distanță.

¹<https://github.com/AndreiDedu/Sistem-inteligent-bazat-pe-roboti-colaborativi>

Bibliografie

- [1] Peter Stone Mark VanMiddlesworth, Kurt Dresner. Replacing the stop sign: Unmanaged intersection control for autonomous vehicles. *Harvard University Elec. Eng. and Comp. Sci. Department , University of Texas at Austin Department of Computer Sciences*, 2008.
- [2] J. Michael Herrmann Calum Imrie. Self-organisation of spatial behaviour in a kilobot swarm. *School of Informatics, Institute for Perception, Action and Behaviour, University of Edinburgh, 10 Crichton St, Edinburgh EH9 8AB, UKP4*, 2017.
- [3] A.A.G. Requicha D.J. Arbuckle. Self-assembly and self-repair of arbitrary shapes by a swarm of reactive robots: algorithms and simulations. *Springer Science+Business Media, LLC*, 2009.
- [4] Heiko Hamann Marco Dorigo Gabriele Valentini, Eliseo Ferrante. Collective decision with 100 kilobots: speed versus accuracy in binary discrimination problems. *Université Libre de Bruxelles, Laboratory of Socioecology and Social Evolution, Department of Computer Science, Heinz Nixdorf Institute, University of Paderborn*, 2015.
- [5] Justin Werfel Golnaz Habibi James McLurkin Radhika Nagpal Michael Rubenstein, Adrian Cabrera. Collective transport of complex objects by simple robots: Theory and experiments. *2013 international conference on Autonomous agents and multi-agent systems*, 2013.
- [6] Manual de utilizare kilobot.
- [7] James A. R. Marshal Andreagiovanni Reina Eleftherios Nikolaidis, Chelsea Sabo. Characterisation and upgrade of the communication between overhead controllers and kilobots. *Department of Computer Science, University of Sheffield, S1 4DP*, septembrie 2017.
- [8] Atmel atmega328p datasheet.
- [9] Alina Gabriela Rusu Manuela Tatiana Nistor. Nanorobots with applications in medicine. *Academia Română, "Petru Poni" Institutul de Chimie Macromoleculară, Iași, România*, 2019.
- [10] Peter Stone Kurt Dresner. Multiagent traffic management: An improved intersection control mechanism. *University of Texas at Austin Department of Computer Sciences*, 2005.
- [11] Gordon D. Friedlander. Computer-controlled vehicular traffic. *BurndlyLibrarY*, februarie 1969.

- [12] Brahim Chaib-draa Simon Halle. A collaborative driving system based on multiagent modelling and simulations. *Departement informatique genie logiciel, Universite Laval, Sainte-Foy, QC, Canada G1K 7P4*, 2005.
- [13] S. Petrovskii V. Volpert. Reaction–diffusion waves in biology. *Institut Camille Jordan, UMR 5208 CNRS, University Lyon 1, 69622 Villeurbanne, France Department of Mathematics, University of Leicester*, 2009.
- [14] Valentini G. Direct modulation of majority-based decisions. in: Achieving consensus in robot swarms. *Studies in Computational Intelligence*, 706, 2017.
- [15] Radhika Nagpal Michael Rubenstein, Christian Ahler. Kilobot: A low cost scalable robot system for collective behaviors. *2012 IEEE International Conference on Robotics and Automation*, 2012.
- [16] Dallas Semiconductor. Fundamentals of rs-232 serial communications. 1998.
- [17] Max232 datasheet.