SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Antonella Barišić, Marko Križmančić

Decentralizirano upravljanje formacijom višeagentskog sustava autonomnih sfernih robota

Zagreb, 2019.

Ovaj rad izrađen je u Laboratoriju za robotiku i inteligentne sustave upravljanja, na Zavodu za automatiku i računalno inženjerstvo pod vodstvom prof. dr. sc. Stjepana Bogdana i predan je na natječaj za dodjelu Rektorove nagrade u akademskoj godini 2018./2019.

SADRŽAJ

1. Uvod

Razvoj tehnologije kroz stoljeća često je bio motiviran promatranjem svijeta oko nas, a posebice prirodnih sustava i zakona. Inspiracija prirodom i pronalaženje rješenja promatrajući njezine strukture, zakone i sustave naziva se biomimikrija. U području robotike biomimikrija se primjenjuje na dizajn sustava [1], [2] i algoritme ponašanja [3], [4], [5]. Ponašanja životinja u skupinama kao što su jata ptica, rojevi pčela i kolonije mrava su nam izrazito zanimljiva jer su prilagođena za probleme potrage i optimizacije procesa (potraga za hranom, preseljenje kolonije, izbjegavanje grabežljivaca, itd.).

Ponašanje životinja u prirodi je vrlo robusno i adaptivno na nepredviđene promjene u okolini što ga čini pogodnim za primjenu u robotici. Promatranjem jata ptica ili roja pčela [6] uočavamo ponašanja koja se mogu koristiti za robotske operacije kao što su grupno nadziranje i istraživanje, misije traganja i spašavanja, prikupljanje podataka i mapiranje nepoznatih područja. Logika takvog ponašanja može se predočiti u skup matematičkih i fizikalnih pravila na kojima temeljimo algoritme. Jedan skup takvih pravila o gibanju životinja u jatima definirao je Craig Reynolds te ih po autoru nazivamo Reynoldsova pravila. Primjenom osnovna tri Reynoldsova pravila možemo ostvariti kompleksna vladanja distribuiranog višerobotskog sustava kao što je opisano u 2. poglavlju. Cilj ovoga rada je pokazati kako primjenom pravila ponašanja koja primjećujemo u prirodi možemo ostvariti decentralizirano upravljanje formacijom višeagentskog robotskog sustava.

Za razumijevanje ovih bihevioralnih algoritama, važno je definirati pojam "autonomni agent". Ovaj pojam uglavnom se koristi za opis jedinke koja samostalno odlučuje kako se ponašati u svom okolišu bez utjecaja vođe ili globalnog plana. U primjeru ovog rada, jedinke odlučuju kako se kreću u odnosu na ostale članove jata. Cilj algoritma je simulirati živi svijet, tj. dobiti živopisno ponašanje. Autonomni agenti posjeduju sljedeća tri ključna svojstva [7]:

1. Imaju ograničenu mogućnost percipiranja okoliša - agent može promatrati druge agente samo ako su unutar neke ograničene udaljenosti

2. Samostalno obrađuju informacije primljene iz okoliša i na temelju njih donose odluku

3. Ne postoji vođa koji će definirati ponašanje grupe

Autonomnost agenata nam omogućuje robusniji sustav koji se s lakoćom prilagođava okruženju na temelju informacija o susjednim agentima i preprekama. Implementirano uprav-

ljanje formacijom autonomnih agenata ne definira unaprijed određenu konfiguraciju jata već naprotiv, dopušta agentima da autonomno odrede svoju poziciju unutar formacije. Model samoorganizirajućeg višeagentskog sustava i rezultati simulacije navedenog ponašanja opisani su u poglavlju 3 ovog rada.

Kao stvarne autonomne jedinke jata korišteni su sferni roboti Sphero opisani u poglavlju 4. Sphero roboti ostvaruju holonomično kretanje, a željenu brzinu i smjer kretanja određuju autonomno na temelju informacija o susjednim agentima. Za ostvarenje ideje ovog rada u stvarnom okruženju potrebno je uspostaviti komunikaciju s agentima te osigurati povratnu vezu po poziciji i stabilnost informacija polja percepcije što je ostvareno primjenom Kalmanovog filtra. Kompletno programsko okruženje opisano je u poglavlju 5, a eksperimentalni rezultati koji potvrđuju cilj ovog rada predstavljeni su u poglavlju 6.

2. Upravljanje formacijom primjenom Reynoldsovih pravila

2.1. Upravljanje formacijama

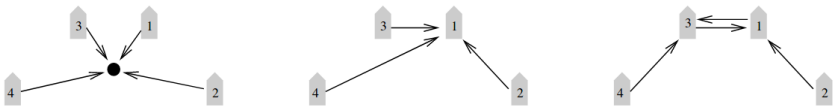
Napredak robotike u zadnjih nekoliko godina doveo je do pojeftinjenja robotskih sustava. Sukladno tome otvorio se novi pristup za rješavanje već poznatih problema korištenjem većeg broja robota. Višerobotski sustavi omogućuju energetski i vremenski efikasnije obavljanje zadataka. Kontrola i koordinacija takvih sustava, takozvano upravljanje formacijama, omogućuje pojedinim agentima da se fokusiraju na prikupljanje informacija sa senzora u ograničenom području percepcije dok ostali agenti pokrivaju preostalo okruženje [8]. Upravljanje formacijom definiramo kao upravljanje relativnom pozicijom i orijentacijom robota u skupini tako da se skupina giba kao cjelina. Neki od glavnih pristupa rješavanju ovog problema su [9]:

- Linearizacija po povratnoj vezi
- Teorija grafova
- Teorija jata
- Razni oblici prediktivnog upravljanja
- Virtualni potencijali

Slika 2.1: Konfiguracije formacija s četiri robota [8]

Odabir pristupa ovisi o postojanju prepreka u okruženju, poznavanju rasporeda u okruženju, mogućnosti detekcije pozicije drugih robota i mogućnosti komunikacije s drugim robotima. Upravljanje formacijama možemo podijeliti na algoritme s unaprijed određenom konfiguracijom formacije i algoritme s formacijama određenim ograničenjima. Najčešće korištene konfiguracije formacija su linija, stupac, dijamant i klin te su prikazane na slici 2.1.

Formacije mogu biti varijabilne kada su određene ograničenjima udaljenosti i područjem pretraživanja. Za održavanje formacije koriste se tri pristupa: referenciranje prema središnjem agentu (engl. *unit-center-referenced*), referenciranje prema vođi skupine (engl. *leader-referenced*) i referenciranje prema susjednim agentima (engl. *neighbor-referenced*). Na slici 2.2 s lijeva na desno su prikazani navedeni pristupi održavanju formacije.

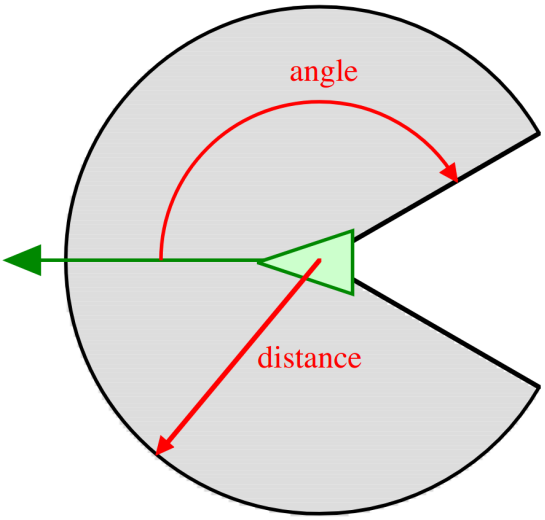


Slika 2.2: Održavanje formacije s četiri robota [8]

2.2. Reynoldsova pravila

Reynoldsova pravila možemo svrstati u skupinu algoritama s formacijama određenim ograničenjima koji koriste referenciranje prema susjednim agentima. Osmislio ih je Craig Reynolds, američki računalni inženjer specijaliziran za računalnu grafiku i modeliranje "umjetnog života". Većina njegovog rada bazira se na proceduralnim računalnim programima za simulaciju različitih aspekata ljudskog i životinjskog ponašanja, ponajviše simulacijama ponašanja u velikim grupama. 1986. došao je na ideju izrade računalnog modela za animaciju kretanja životinja u jatima (poput ptica ili riba). Generičke jedinke koje se koriste kao autonomni agenti u simulaciji nazvao je *boids* (*bird-oid object*, "objekt nalik ptici"). Nedugo nakon začetka ideje, napravio je i prvu proceduralnu animaciju koja je koristila razvijeni algoritam pod imenom *Stanley and Stella in: Breaking the Ice*. Razvijeni algoritam koristi se za animaciju gibanja jata riba i ptica u dvije polusfere ispunjene vodom i zrakom odvojene slojem leda na sredini. Kratki animirani film i članak o ovim umjetnim pticama prvi puta su objavljeni na konferenciji SIGGRAPH 1987 [10].

Svaka jedinka (*boid*) je autonomni agent koji samostalno donosi odluke o svom ponašanju na temelju okoline koju može percipirati. Sposobnost percepcije u prirodi nije savršena: životinje imaju ograničenu širinu vidnog polja, a vidljivost može biti dodatno narušena nepovoljnim uvjetima kao što su mrak, magla ili mutna voda. Zbog toga se i okolina virtualnih jedinki modelira određenim radijusom i kutem unutar kojega jedinka može "vidjeti" druge jedinke. Takav model vidnog polja prikazan je na slici 2.3. Važno je napomenuti da naglasak nije na simuliranju načina percepcije, dakle točno onoga što jedinke mogu vidjeti. Bitno je samo da model pruža sve informacije koje su neophodne za donošenje odluka o gibanju u jatu. U ovome slučaju to su relativan položaj susjednih agenata i njihova trenutna brzina.



Slika 2.3: Model vidnog polja agenta [11]

Kako bi se osiguralo realistično i živopisno kretanje u jatu, Reynolds je definirao pravila usmjeravanja (engl. *steering*) autonomnih agenata. Osnovna pretpostavka za koju će vrijediti sva opisana pravila je da se autonomni agenti promatraju kao točkaste mase. Iz toga proizlazi da će model agenta biti opisan njegovom masom, pozicijom u prostoru i brzinom. Brzina se mijenja djelovanjem sila na agenta. Sile mogu dolaziti iz pravila ponašanja u jatu te su najčešće posljedica djelovanja samih agenata, a ne vanjskih utjecaja. Zbog toga je maksimalna sila koju agent može proizvesti ograničena nekim konačnim iznosom te postaje još jedan parametar za opis modela agenta. Budući da se ništa ne može gibati neograničenom brzinom, posljednji parametar modela agenta čini njegova maksimalna ostvariva brzina [11].

Realističnost gibanja i ponašanja agenata ovisi o određivanju vektora sile usmjeravanja i točnosti samog fizikalnog modela. Reynolds je svoje osnovne ideje iz prvog rada dodatno proširio u [11] definirajući različita ponašanja koja se mogu pojaviti pri animaciji autonomnih agenata, a ona potrebna za modeliranje gibanja u jatu navedena su u nastavku.

Traženje (engl. *seek*) pokušava usmjeriti agenta prema određenoj točki u prostoru tako da njegova brzina bude radijalno usmjerena prema meti. Iznos te željene brzine (duljina vektora) može se, primjerice, postaviti na maksimalnu brzinu određenu parametrom agenta. Time se predstavlja želja da agent dođe u zadanu točku u najkraćem mogućem vremenu. Vektor sile usmjerenja $\vec{F}_{steer}$ za ovo ponašanje tada je razlika između željene brzine $\vec{v}_d$ i trenutne brzine agenta $\vec{v}_a$:

$$\vec{v}_d = \frac{\vec{p}_a - \vec{p}_d}{\|\vec{p}_a - \vec{p}_d\|} \cdot v_{max}, \tag{2.1}$$

$$\vec{F}_{steer} = \vec{v}_d - \vec{v}_a, \tag{2.2}$$

gdje su $\vec{p}_d$ i $\vec{p}_a$ vektorski prikazi pozicije mete, odnosno agenta, a v_{max} maksimalna brzina agenta.

Uz ovako definirano ponašanje, agent će dostići metu te velikom brzinom proći kroz nju,

nakon nekog vremena se okrenuti i ponovno početi približavati meti. Proces se ponavlja te dolazi do oscilacija. Kako bi se to spriječilo, ponašanje se modificira faktorom za usporavanje prije dolaska do mete. Definira se udaljenost od mete unutar koje agent linearno smanjuje željenu brzinu prema nuli kako joj se približava. Izvan definirane udaljenosti, željena brzina je ograničena na maksimalnu kao i prije ove modifikacije:

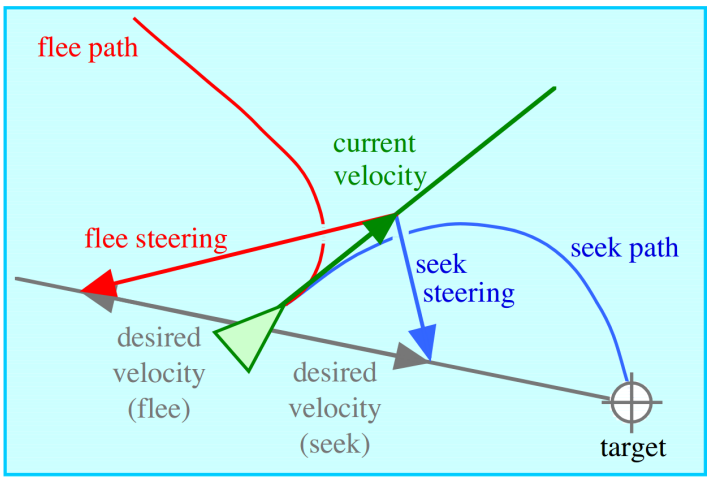
$$\vec{p}_\Delta = \vec{p}_a - \vec{p}_d, \tag{2.3}$$

$$\vec{v}_d = \begin{cases} \hat{p}_\Delta \cdot v_{max}, & \|\vec{p}_\Delta\| > d_{max} \\ \hat{p}_\Delta \cdot \frac{\|\vec{p}_\Delta\|}{d_{max}} \cdot v_{max}, & \|\vec{p}_\Delta\| \leq d_{max} \end{cases}, \tag{2.4}$$

$$\vec{F}_{steer} = \vec{v}_d - \vec{v}_a, \tag{2.5}$$

gdje je $\hat{p}_\Delta$ jedinični vektor koji pokazuje od agenta do mete, $\|\vec{p}_\Delta\|$ udaljenost do mete, a d_{max} je udaljenost do koje se agent giba maksimalnom brzinom.

Bježanje (engl. *flee*) je ponašanje suprotno od traženja. Agent se pokušava gibati tako da se odmiče od mete. Proračun vektora usmjeravanja isti je kao i kod traženja, a jedina razlika je da vektor željene brzine invertiran, tj. da pokazuje u suprotnom smjeru. Ova dva ponašanja vizualno se mogu predočiti slikom 2.4.



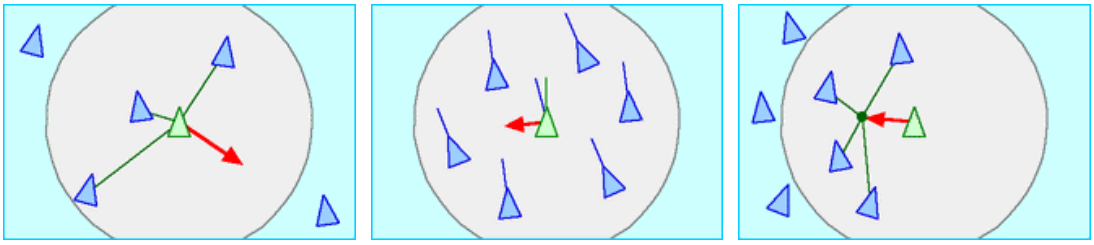
Slika 2.4: Ponašanja traženje i bježanje [11]

Naizgled vrlo kompleksno ponašanje jata kao cjeline proizlazi iz samo tri jednostavna pravila koja se zasnivaju na traženju i bježanju:

1. Odvajanje (engl. *separation*): izbjegavanje gužve i održavanje razmaka.
2. Poravnanje (engl. *alignment*): usklađivanje smjera i iznosa brzine.
3. Privlačenje (engl. *cohesion*): formiranje grupa s jedinkama u blizini.

Vizualna reprezentacija ovih pravila prikazana je na slici 2.5.

Odvajanje osigurava minimalan razmak između jedinki u jatu kako bi se izbjegla gužva i smanjila mogućnost međusobnih sudara. U implementaciji ovog pravila prvo se traže



(a) Odvajanje **(b)** Poravnanje **(c)** Privlačenje

Slika 2.5: Prikaz Reynoldsovih pravila gibanja [11]

sve jedinke koje se nalaze unutar definiranog susjedstva. Radijus koji opisuje susjedstvo u pravilu je manji od radijusa percepcije agenata te će u ovome radu biti označen s r_{crowd}. Za svaku jedinku u susjedstvu računa se vektor koji pokazuje od promatrane prema susjednoj jedinki, taj vektor se normalizira te množi s inverzom njihove međusobne udaljenosti kako bi se dobila odbojna sila nalik opruzi između dvije jedinke. Osim inverza udaljenosti, može se koristiti i inverzna kvadratna ovisnost koja više nalikuje sili gravitacije. Time se postiže prirodnije i bolje prigušeno ponašanje jata [10]. Zbroj odbojnih sila između promatrane jedinke i svih susjeda na kraju se uprosječi kako broj susjeda u okolini ne bi imao utjecaj na iznos sile. Ako je potrebno, sila se može i ograničiti na maksimalni iznos kao što je opisano ranije.

Privlačenje djeluje obrnuto i pokušava približiti i spojiti jedinke u grupe. Kao i kod odvajanja, pronalaze se sve jedinke u susjedstvu te se računa njihova prosječna pozicija na što se može gledati i kao centar gravitacije susjedstva. Sila se tada može primijeniti u tom smjeru ili se centar gravitacije susjedstva može postaviti kao meta za pravilo traženja. Radijus susjedstva u ovom slučaju najčešće je jednak radijusu percepcije agenta. Time je definirano područje pretraživanja svakog agenta pa će se u ovom radu označavati s r_{search}.

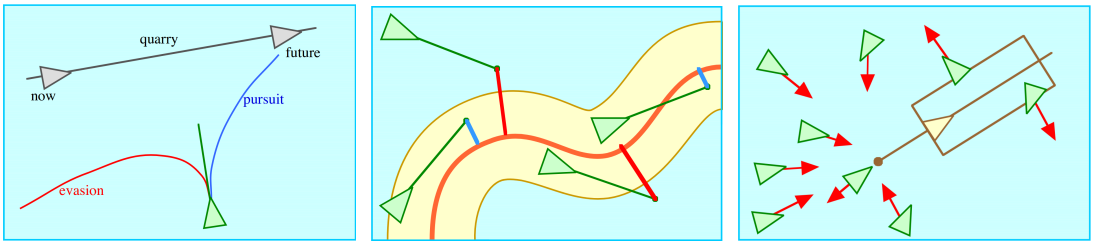
Cilj **poravnavanja** je da sve susjedne jedinke u nekom jatu imaju približno jednak smjer i brzinu kretanja. Računa se prosječni smjer i brzina gibanja susjeda, a dobiveni vektor predstavlja željenu brzinu koja se koristi u fizikalnom modelu prema pravilu $\vec{F}_{steer} = \vec{v}_d - \vec{v}_a$.

Statičko ponašanje odvajanja i dinamičko ponašanje poravnanja su komplementarni. Zajedno osiguravaju da se svaka jedinka može slobodno gibati u prostoru bez međusobnih sudara. Odvajanje se može zamisliti kao odbojna sila između jedinki koja ovisi o njihovoj udaljenosti, ali ne i o brzini (kao između istih polova magneta). S druge strane, poravnanje ovisi o brzini, ali ne o relativnim pozicijama. U neku ruku to je prediktivno izbjegavanje sudara: ako sve jedinke dobro provode pravilo i kreću se otprilike približnim brzinama, malo je vjerojatno da će doći do sudara. Može se reći da pravilo odvajanja služi za osiguravanje minimalnog potrebnog razmaka između pojedinih članova jata, a pravilo poravnanja nastoji taj razmak održati [10].

Tri navedena pravila rezultiraju zasebnim silama koje nastoje ubrzati agenta u različitim smjerovima kako bi bili ispunjeni uvjeti kojima su definirana. U konačnici se sve željene ak-

celeracije moraju kombinirati u jedinstvenu upravljačku veličinu koja će djelovati na agenta. Najlakši način za kombiniranje je jednostavni otežani zbroj gdje težine definiraju važnost pojedinog pravila, a rezultanta sila se dodatno ograničava na vrijednost zadanu parametrom modela agenta. Faktori otežanja određuju se eksperimentalno kako bi se dobilo najbolje ponašanje jata. Problem ovog pristupa je u činjenici da se sile mogu pokratiti u nepogodnim trenucima te tako uzrokovati sudar s preprekom ili drugim agentom. Reynolds u [11] predlaže nekoliko pristupa, a detalji implementacije u ovom radu dani su u sljedećem poglavlju.

Uz tri osnovna pravila ponašanja u jatu, model se može proširiti s brojnim drugima koja opisuju željena ponašanja. Na slici 2.6 prikazana su pravila potjere (engl. *pursuit*), izbjegavanja (engl. *evasion*), praćenja puta (engl. *path following*) i praćenja vođe (engl. *leader following*), a neka od mogućih još su i lutanje (engl. *wander*) te praćenje zida i toka (engl. *wall* i *flow following*). Primjenom nekih od navedenih pravila ili njihovom kombinacijom, moguće je ostvariti ciljano gibanje jata. U ovom radu se ne koristi takav pristup, ali je implementirana metoda za izbjegavanje prepreka te je simulirana sila trenja koja djeluje na svakog agenta prilikom kretanja.



(a) Potjera i izbjegavanje **(b)** Praćenje puta **(c)** Praćenje vođe

Slika 2.6: Neka od dodatnih pravila ponašanja [11]

Važno je još i komentirati algoritamsku složenost navedenog algoritma. U najjednostavnijoj implementaciji ona iznosi $\mathcal{O}(n^2)$ pri čemu je n broj jedinki u jatu. Ova složenost proizlazi iz činjenice da je za svakog agenta potrebno provjeriti nalaze li mu se ostalih $n-1$ agenata u vidnom polju te izračunati sile međudjelovanja. Za mala jata, kao što je slučaj u ovom radu, to ne predstavlja problem. Međutim, povećanjem broja jedinki, vrijeme potrebno za sve proračune može vrlo brzo nadmašiti sposobnosti računala na kojem se izvodi simulacija. Udvostručenjem broja agenata, zahtjevnost algoritma se povećava četiri puta.

U prirodi nije primijećena gornja granica veličine jata što ukazuje na to da složenost može ustvari biti $\mathcal{O}(1)$, odnosno da je vrijeme potrebno za donošenje odluka konstantno i neovisno o broju jedinki u jatu. Čvorci su poznati kao vrsta ptica čija jata tijekom selidbe mogu poprimiti i do desetak tisuća članova [12]. U Danskoj, koja je najveće okupljalište ovih ptica, imaju poseban naziv za te nesvakidašnje prizore: "Sort Sol" ili u prijevodu "Crno Sunce". Jata su toliko velika da imaju sposobnost doslovno zamračiti nebo (slika 2.7).

Slika 2.7: "Sort Sol" ili "Crno Sunce": velika jata čvoraka zamračuju nebo [13].

.

Najefikasnije rješenje problema složenosti je korištenje distribuiranog sustava koji bi oponašao percepciju i mehanizme odlučivanja životinja. Takav sustav bi trebao biti opremljen vlastitim senzorima kojima bi mogao prikupiti informacije o susjednih 7 do 10 agenata koliko je dovoljno za stabilno ponašanje jata [14], [15] te ne bi ovisio o ukupnom broju agenata.

Budući da je takva razina paralelizma vrlo teško ostvariva, češće se koriste pristupi koji nastoje ograničiti prostor pretraživanja susjeda. Autori u [16], [17], [18] koriste podjelu prostora na ćelije i u svakom trenutku simulacije prate u kojim se ćelijama agenti nalaze. Za proračun upravljačkih veličina tada nije potrebno pretraživati cijelo jato, već samo agente koji se nalaze u susjednim ćelijama. U [17] ušteda vremena postiže se tako da se u svakom koraku simulacije ažurira samo $1/8$ jata, a u [19] liste susjeda preračunavaju se samo kada jato značajnije mijenja smjer.

3. Model i simulacija razvijenog sustava

3.1. Razvijeni model

Kao što je navedeno u prethodnom poglavlju, agenti jata modelirani su kao točkaste mase i definirani su maksimalnom silom koju mogu generirati F_{max} te maksimalnom brzinom v_{max}.

Fizika gibanja modela određena je jednostavnom unaprijednom Eulerovom diferencijom. U svakom koraku na točkastu masu modela agenta primjenjuje se sila usmjeravanja dobivena kombinacijom sila proračunatih pravilima gibanja u jatu umanjena za iznos sile trenja. Sila trenja definirana je kao konstantna sila koja djeluje u smjeru suprotnom od smjera gibanja agenta. Rezultat je akceleracija jednaka primijenjenoj sili usmjeravanja podijeljenoj s masom agenta. Akceleracija se integrira u trenutnu brzinu kako bi se dobila brzina u sljedećem koraku te se ona integrira u staru poziciju. Sila usmjeravanja i nova izračunata brzina ograničavaju se na maksimalne dopuštene vrijednosti određene parametrima modela. Ovaj postupak opisan je sljedećim jednadžbama:

$$\vec{F}_k = \max\{\vec{F}_{steer,k}, F_{max}\}, \tag{3.1}$$

$$\vec{a}_k = \frac{\vec{F}_k}{m} - \vec{F}_{tr}, \tag{3.2}$$

$$\vec{v}_k = \max\{\vec{v}_{k-1} + \vec{a}_k \cdot \Delta t, v_{max}\}, \tag{3.3}$$

$$\vec{p}_k = \vec{p}_{k-1} + \vec{v}_k \cdot \Delta t, \tag{3.4}$$

gdje je m masa agenta, $\vec{F}_{tr}$ sila trenja, a Δt vrijeme diskretizacije. U konačnici se dobije model gibanja koji ovisi samo o prošlim vrijednostima i sili usmjeravanja što ga čini izrazito jednostavnim za implementaciju. Izlaz iz modela je zadana brzina $\vec{v}_k$ za svaki korak izvođenja, a proračun nove pozicije odrađuje sam simulator. Ovakav jednostavan model ne uzima u obzir mogućnost proklizavanja, dopušta vrlo nagle i iznosom velike promjene orijentacije te promjenu orijentacije u mjestu.

U razvijenom modelu pretpostavlja se da vidno polje nije ograničeno kutem, odnosno da kut iznosi $360°$. Radijus percepcije je ograničen i definiran parametrom r_{search} kao što je ranije navedeno. Informacije o pozicijama i brzinama agenata prikupljaju se centralizirano, ali se nastoji oponašati lokalni pogled na svijet kao i u prirodi. Agenti tako ne dobivaju apsolutne pozicije svojih susjeda već relativne u odnosu na njih same.

Reynoldsova pravila većinom su definirana kao i u odjeljku 2.2. Sila F_a generirana pravilom **poravnanja** (*alignment*) može se formalnije izraziti na sljedeći način:

$$\vec{v}_{avg} = \frac{1}{|N_s|} \sum_{i \in N_s} \vec{v}_i, \tag{3.5}$$

$$\vec{v}_d = \hat{v}_{avg} \cdot v_{max}, \tag{3.6}$$

$$\vec{F}_a = \vec{v}_d - \vec{v}_a, \tag{3.7}$$

gdje je N_s skup susjednih agenata unutar vidnog polja promatranog agenta, $|N_s|$ je broj agenata u tom skupu, $\vec{v}_a$ trenutna brzina promatranog agenta, a $\hat{v}_{avg}$ jedinični vektor, tj. smjer prosječne brzine jata. Budući da se željena brzina uvijek postavlja na maksimalnu, ova komponenta ima i ulogu P regulatora brzine. Dok se agenti gibaju sporije od maksimalne brzine, sila je veća od nule i nastoji ih ubrzati. Jednom kada se dostigne navedena brzina, sila poravnanja nestaje i agenti se gibaju jednoliko.

Kao što je prije objašnjeno, sila **privlačenja** (*cohesion*) računa se tako da nastoji ubrzati agente prema središtu njihovih susjeda u vidnom polju. Iznos sile ovisi o udaljenosti od prosječne pozicije susjeda, i to tako da je maksimalna na udaljenosti r_{search} te se linearno smanjuje prema nuli kako se agent približava središtu. Formalno, sila $\vec{F}_c$ je definirana jednadžbama:

$$\vec{p}_{avg} = \frac{1}{|N_s|} \sum_{i \in N_s} \vec{p}_i, \tag{3.8}$$

$$\vec{F}_c = \hat{p}_{avg} \cdot \frac{\|\vec{p}_{avg}\|}{r_{search}} F_{max}. \tag{3.9}$$

Vektor relativne pozicije susjeda i označen je s $\vec{p}_i$, vektor pozicije centroida susjeda u vidnom polju tada je $\vec{p}_{avg}$, a $\|\vec{p}_{avg}\|$ je udaljenost promatranog agenta do središta.

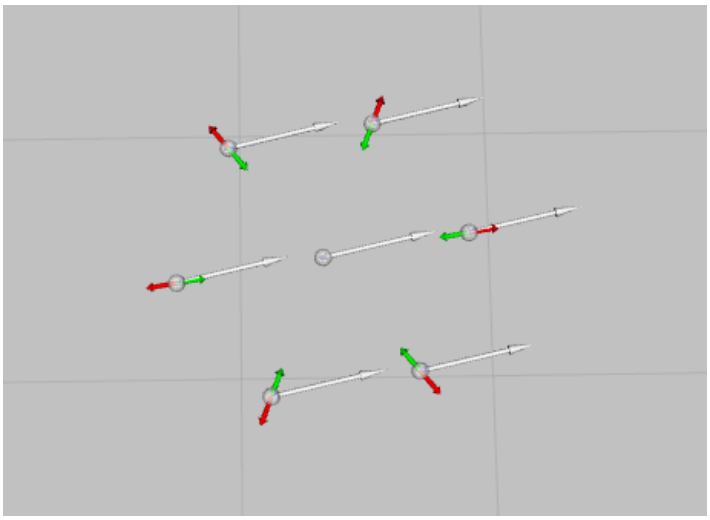
Za **odvajanje** (*separation*) se u ovom radu koristi pristup s inverznim kvadratnim otežavanjem udaljenosti između agenata te se formalno može zapisati sljedećim jednadžbama:

$$\vec{F}_{avg} = \frac{1}{|N_c|} \sum_{i \in N_c} -\frac{\hat{p}_i}{K_s \cdot \|\vec{p}_i\|^2}, \tag{3.10}$$

$$\vec{F}_s = \max\{\vec{F}_{avg}, 2F_{max}\}, \tag{3.11}$$

gdje je N_c skup susjednih agenata udaljenih od promatranog agenta za manje od r_{crowd}, a K_s konstanta čija je vrijednost definirana u nastavku. Generirana sila ograničava se na dvostruki iznos zadane maksimalne sile. Na taj se način ovoj komponenti daje svojevrsni prioritet. Ako se agenti previše približe, sila će nadjačati ostale kako bi se izbjegao sudar.

Očekivano je da će u stacionarnom stanju zbroj svih sila u jatu biti jednak nuli. Jato se u tom slučaju giba jednoliko i, ovisno o trenju, malo sporije od definirane maksimalne brzine. Sila poravnanja jednakog je iznosa i suprotnog smjera od sile trenja, a sila privlačenja obrnuta je sili odvajanja. Na slici 3.1 prikazana je navedena situacija. Vektori zelene i crvene boje



Slika 3.1: Jato u stacionarnom stanju

prikazuju sile privlačenja, odnosno odvajanja. Trenje i sila poravnanja nisu vidljivi zbog malog iznosa. Poželjno je da su u stacionarnom stanju agenti međusobno udaljeni za iznos r_{crowd}. Drugim riječima, želimo da vrijedi $F_s = F_c$ kada je udaljenost između dva agenta $d = r_{crowd}$. Iz ove relacije možemo izračunati iznos konstante K_s:

$$F_c = F_s, \tag{3.12}$$

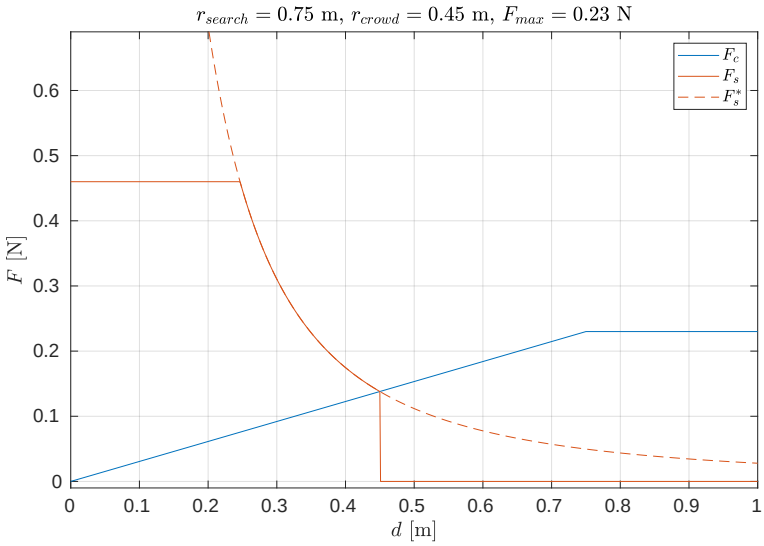
$$F_{max} \cdot \frac{d}{r_{search}} = \frac{1}{K_s \cdot d^2}, \quad d = r_{crowd}, \tag{3.13}$$

$$K_s = \frac{r_{search}}{r_{crowd}^3} \cdot \frac{1}{F_{max}}. \tag{3.14}$$

Na slici 3.2 prikazane su sile F_c i F_s u ovisnosti o udaljenosti između dvaju agenata d. Zadano je $r_{search} = 0.75$ m, $r_{crowd} = 0.45$ m i $F_{max} = 0.23$ N. Oznakom F_s^* prikazano je kako bi izgledala sila odvajanja kada bi bila definirana za udaljenosti veće od r_{crowd} i kada ne bi bila ograničena na maksimalni iznos.

Bitno je napomenuti da se, prilikom zbrajanja odbojnih sila između parova agenata, bočne komponente vektora pokrate te je duljina rezultantnog vektora manja od zbroja duljina vektora pribrojnika. Konačni iznos sile F_s tada je također manji od pretpostavljenog pa se ravnotežno stanje postiže na udaljenosti manjoj od r_{crowd}. Ovaj neočekivani rezultat zapravo pogodno djeluje na glatkoću gibanja jata. Naime, kao što je navedeno, sila odvajanja prema agentima koji su izvan definiranog susjedstva poprima vrijednost nula. Agenti na rubu mogu često izlaziti iz vidnog polja svojih susjeda zbog čega će i sila skokovito mijenjati svoj iznos te unositi trzaje u jato. Uspostavljanje ravnoteže na manjoj udaljenosti stoga ostavlja više prostora za reakciju na smetnje i njihovo ublažavanje.

Osim tri osnovna pravila, u razvijenom modelu implementirano je i pravilo za izbjegavanje prepreka. Sve prepreke i zidovi kojima su definirani rubovi prostora u kojem se jato kreće razloženi su na niz točaka opisanih svojom pozicijom u prostoru. Broj točaka ovisi



Slika 3.2: Ravnoteža sila privlačenja i odvajanja

o rezoluciji mape. U daljnjem tekstu, kao i u razvijenom algoritmu, svaka točka se smatra zasebnom preprekom.

Pravilo za izbjegavanje prepreka sastoji se od dvije komponente. Prva komponenta uzima u obzir sve prepreke koje se nalaze unutar vidnog polja agenta, dakle radijusa r_{search}. Iznos sile proporcionalan je inverzu kvadrata udaljenosti do prepreke te kosinusu kuta pod kojim joj agent prilazi. Kut prilaženja α_i definiran je kao kut između vektora relativne pozicije prepreke u odnosu na agenta $\vec{p}_i$ te vektora brzine agenta $\vec{v}_a$. Ovakvim pristupom se nastoji što vjernije opisati ponašanje u prirodi. Sila je najveća ako se prepreka nalazi točno na putu agenta. Ako agent samo prolazi pokraj prepreke, želja za odmicanjem će biti manja. Time se omogućuje da jato prati zidove prostora u kojem se kreće te da može proći kroz uske prepreke nalik hodniku.

Budući da će za $\alpha_i \sim \pm 90°$ prva komponenta sile biti vrlo mala, jato koje se giba gotovo paralelno uz zid nakon nekog bi vremena ipak moglo doći u kontakt s njime. Zbog toga se uvodi i druga komponenta sile koja ne ovisi o kutu prepreke i služi za održavanje minimalnog razmaka od prepreka. Minimalna udaljenost definirana je parametrom r_{avoid}, a sila linearno ovisi o udaljenosti od prepreke. Pri tome je sila maksimalna kada je udaljenost jednaka nuli.

Opisane komponente matematički se mogu zapisati na sljedeći način:

$$\vec{F}_1 = \frac{1}{|N_o|} \sum_{i \in N_o} -\frac{\hat{p}_i}{K_o \cdot \|\vec{p}_i\|^2} \cdot \max\{\cos(\alpha_i), 0\}, \tag{3.15}$$

$$\vec{F}_2 = \frac{1}{|N_o^*|} \sum_{j \in N_o^*} -\hat{p}_j \cdot \left(1 - \frac{\|\vec{p}_j\|}{r_{avoid}}\right) \cdot 2F_{max}, \tag{3.16}$$

$$\vec{F}_o = \vec{F}_1 + \vec{F}_2, \tag{3.17}$$

gdje je N_o skup prepreka unutar vidnog polja agenta, α_i kut pod kojim se agent približava

prepreci, N_o^* skup prepreka unutar radijusa r_{avoid}, a $\vec{p}_i$ i $\vec{p}_j$ vektori relativne pozicije prepreka u odnosu na agenta. Konstanta K_o određuje se tako da odbojna sila bude jednaka F_{max} na udaljenosti $\|\vec{p}_i\| = 0.85 \cdot r_{search}$.

Nakon što su izračunate sile svih pravila, potrebno ih je kombinirati u jedinstvenu upravljačku veličinu. U ovom radu, konačna sila je jednostavna linearna kombinacija dosad navedenih komponenti. Njihove težine definirane su konstantama $C_a,\ C_s,\ C_c$ i C_o. Kao što je već navedeno, kod ovakvog pristupa moguće je da se sile neočekivano pokrate što može rezultirati sudarom. Zbog toga se najveći prioritet daje izbjegavanju prepreka i to tako da se sili dopušta poprimanje neograničenog iznosa. Time se efektivno zanemaruje utjecaj ostalih pravila i omogućava pojava kompleksnih ponašanja poput razdvajanja jata prilikom nailaska na prepreku. Fizikalna svojstva agenta ovim postupkom nisu narušena jer se konačna upravljačka veličina i dalje ograničava na iznos F_{max}:

$$\vec{F} = C_a \cdot \vec{F}_a + C_s \cdot \vec{F}_s + C_c \cdot \vec{F}_c + C_o \cdot \vec{F}_o, \tag{3.18}$$

$$\vec{F}_{steer} = \max\{\vec{F}, F_{max}\}. \tag{3.19}$$

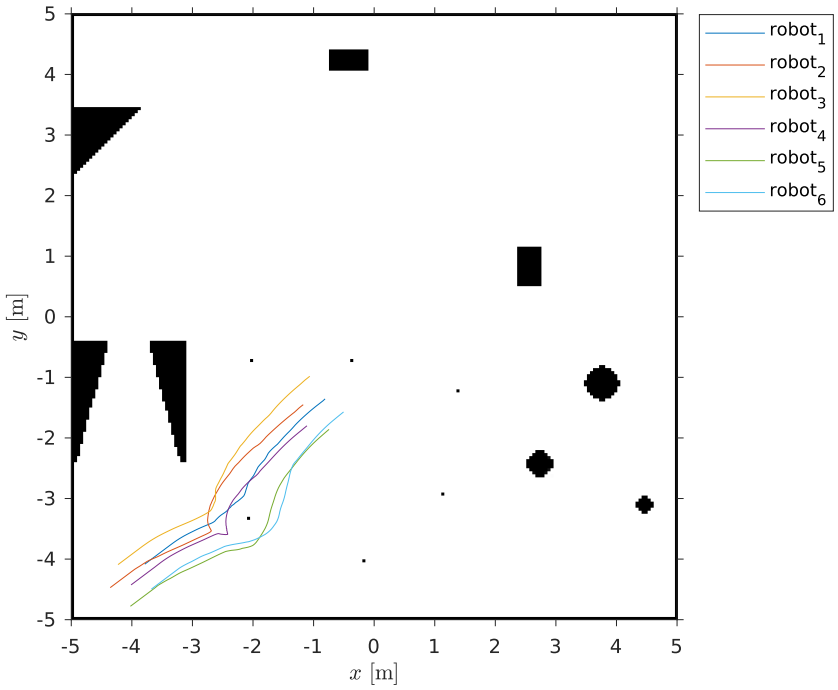
3.2. Simulacija jata upravljanog Reynoldsovim pravilima

U ovom odjeljku prikazani su rezultati simulacije gibanja jata dobiveni opisanom implementacijom algoritma Reynoldsovih pravila. U simulacijama se radi jasnoće prikaza rezultata i jednostavnosti izvedbe koristi šest virtualnih agenata. Mapa prostora kojim se jato kreće posebno je dizajnirana kako bi određena zanimljiva ponašanja jata lakše došla do izražaja. Nedostatak pravila koje bi definiralo putanju pojedinih agenata ili jata u cjelini rezultira nasumičnim gibanjem koje je većinom posljedica interakcija s preprekama. Međutim, u implementiranoj programskoj podršci postoji mogućnost zadavanja početne brzine svakog agenta. Ova značajka je korištena kako bi se jato usmjerilo na željenu stranu u početku eksperimenta.

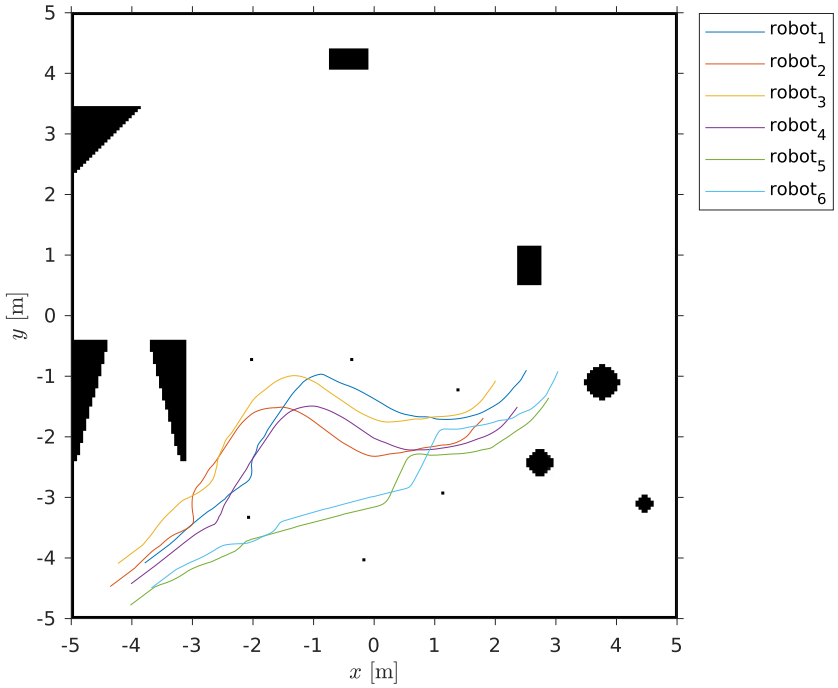
Vjerojatno najzanimljivije svojstvo jata je njegova sposobnost fluidnog zaobilaženja raznovrsnih prepreka. Pri nailasku na prepreku, neki agenti u jatu možda je neće ni vidjeti jer je izvan njihovog vidnog polja. Međutim, agenti koji su bliže počinju usporavati i mijenjati smjer. Kako bi se izbjegli međusobni sudari, i agenti koje su izvan direktne opasnosti od prepreke počinju raditi isto. Time se ostvaruje kompleksna grupna dinamika bez eksplicitne razmjene informacija među agentima.

U slučaju kada je prepreka vrlo uska, može se dogoditi da se jato privremeno razdvoji kako bi je obišlo nakon čega nastavlja dalje kao cjelina. Ovaj slučaj prikazan je na slici 3.3.

Osim kratkotrajnog razdvajanja, moguće je i trajnije razdvajanje na dva manja jata koja će nastaviti zasebnim putevima. Ovisno o veličini mape i rasporedu prepreka, ta dva jata mogu se ponovno spojiti nakon određenog vremena. Vjerojatnost razdvajanja i spajanja jata izravno ovisi o radijusu vidnog polja i zadanoj udaljenosti između susjednih agenata. Kada



Slika 3.3: Privremeno razdvajanje jata radi obilaska prepreke

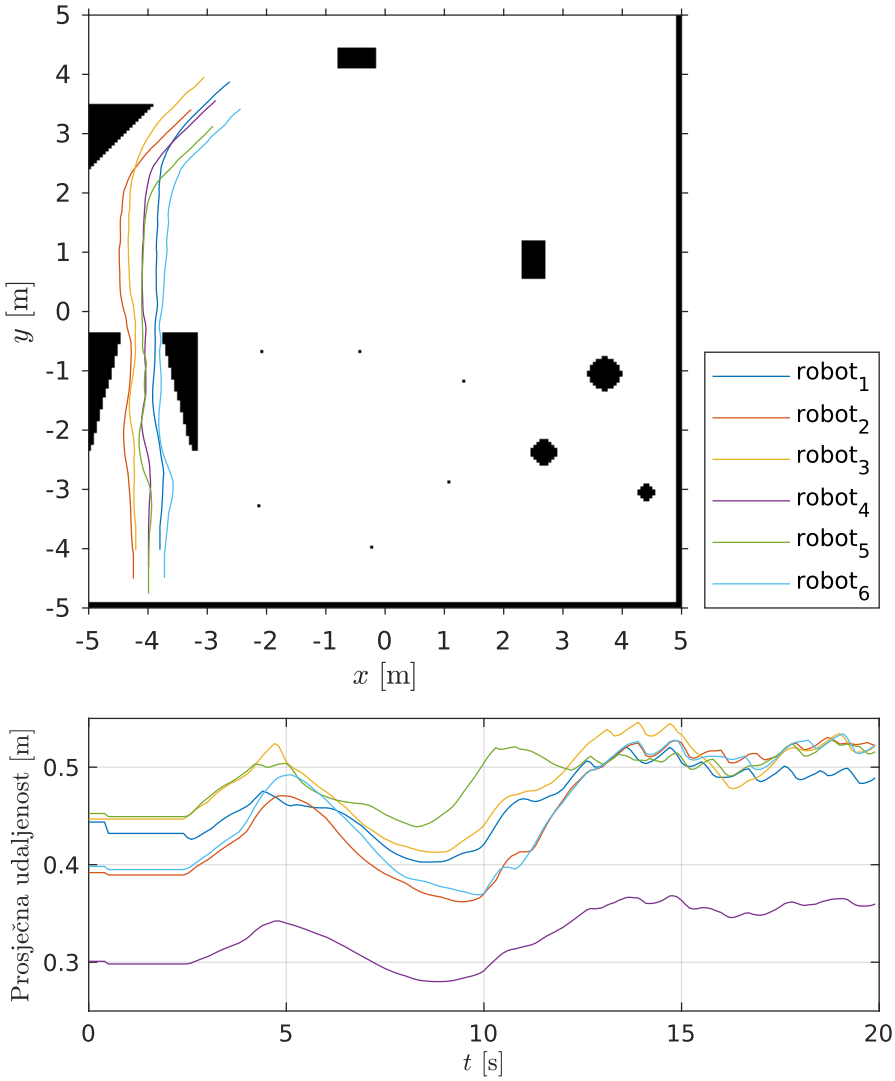


Slika 3.4: Razdvajanje jata prilikom nailaska na prepreku i ponovno spajanje

je vidno polje manje, lakše se dogodi da prilikom nailaska na prepreku polovica jata iz vida izgubi drugu polovicu što dovodi do razdvajanja. Također, ako jato zauzima veću površinu što je posljedica većih međusobnih udaljenosti, veća je i mogućnost razdvajanja. Za šire

vidno polje i manji razmak između susjeda, veća je vjerojatnost spajanja dva bliska jata ako se sretnu u praznom prostoru. Na slici 3.4 jato nailazi na istu prepreku kao i u prethodnom slučaju, ali se zbog većeg međusobnog razmaka dijeli na dva manja jata. Igrom slučajnosti, jata se uskoro ponovno spajaju i nastavljaju put zajedno.

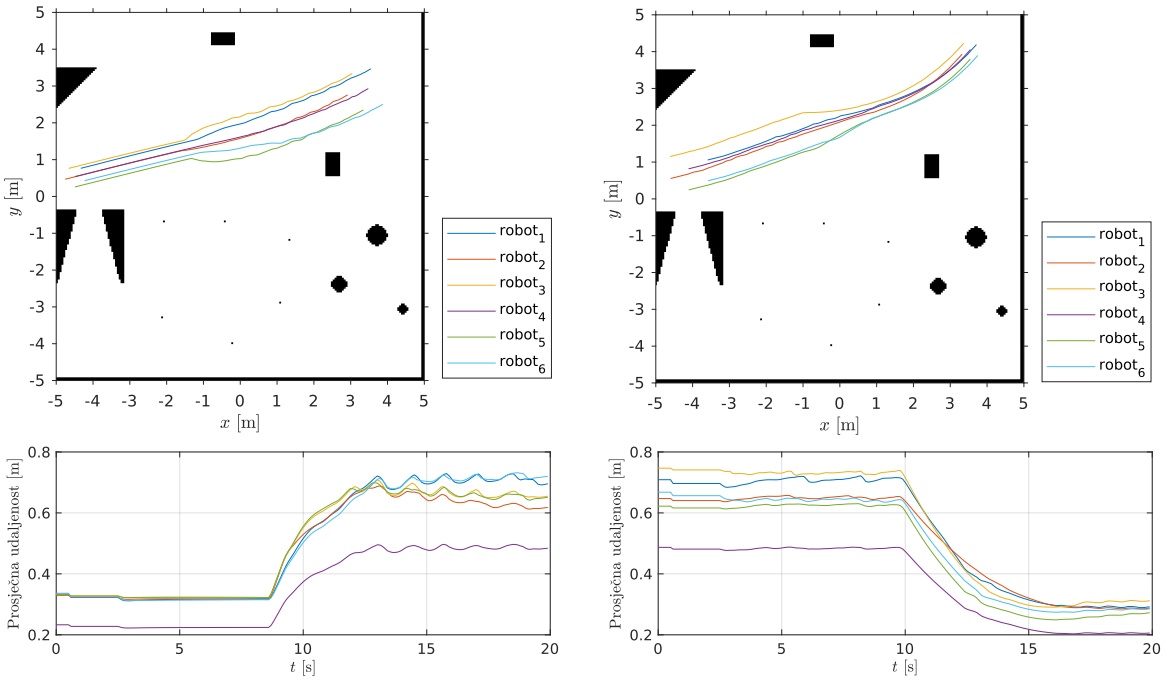
U slučaju prepreka koje tvore uski prolaz, ponašanje jata je obrnuto. Održavanje razmaka između susjeda gubi prioritet zbog opasnosti od vanjskih sudara pa se jato sužava ili mijenja konfiguraciju tijekom prolaska. U lijevom dijelu mape napravljeno je suženje za demonstraciju ovog efekta, a rezultati su prikazani na slici 3.5. Nakon što je jato sigurno prošlo kroz prepreku, vratilo se u prvobitno stanje. Kako bi se lakše uočila promjena razmaka između susjeda, na sljedećim slikama je prikazan i vremenski dijagram prosječne udaljenosti svakog agenta od ostatka jata.



Slika 3.5: Privremeno skupljanje jata radi prolaska kroz usku prepreku

Prilikom pokretanja simulacije, prvobitno ponašanje jata je reakcija na zadane početne uvjete. Ako su agenti postavljeni preblizu u odnosu na parametar željenog međusobnog raz-

maka, dogodit će se nagla ekspanzija iz središta jata kao rezultat pojedinačnih nastojanja da se izbjegne sudar. Analogno, ako su agenti u početnim pozicijama međusobno jako udaljeni, a da se još uvijek nalaze unutar radijusa vidnog polja, u početku će se skupiti zbog pravila privlačenja. Jato također vrlo brzo reagira na promjenu parametara tijekom trajanja simulacije. Posebno se jasno vidi efekt zadane minimalne udaljenosti među susjednim agentima. Rezultati su prikazani na slici 3.6.



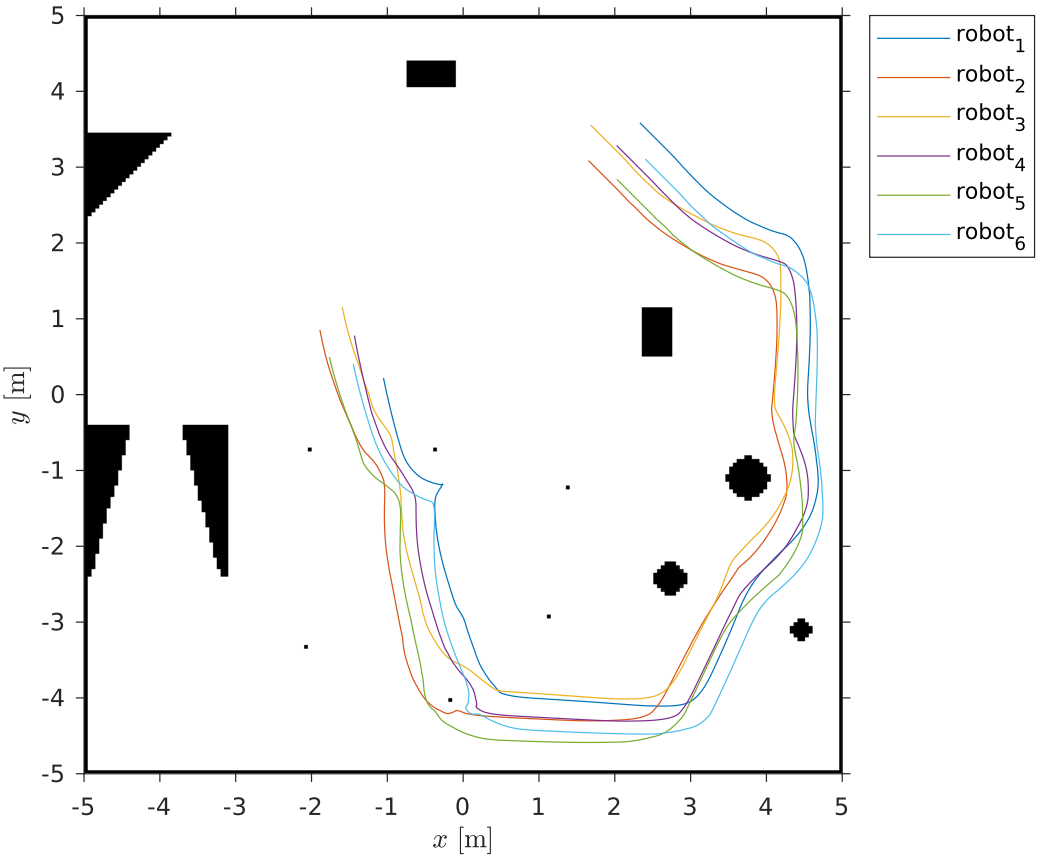
(a) Povećanje razmaka

(b) Smanjenje razmaka

Slika 3.6: Reakcija jata na promjenu parametra r_{crowd}

Na posljednjoj slici vide se pojave navedenih kompleksnih ponašanja u jednom, kontinuiranom gibanju. Agenti se fluidno kreću u slobodnom prostoru, na okupu i održavajući zadanu međusobnu udaljenost. Nailaskom na zid mijenjaju smjer te se nastavljaju kretati paralelno do iduće prepreke gdje se skupljaju kako bi ju prošli. Kasnije se privremeno razdvajaju kako bi obišli usku prepreku te nastavljaju dalje.

Simulirano jato vjerno oponaša prizore pravih jata u prirodi dok se nasumično kreće prostorom i prilagođava okolini. Implementirani algoritam pokazao se vrlo stabilnim pa je moguće generirati proizvoljno duge simulacije s različitim brojem agenata. Video snimke spomenutih eksperimenata te dužih simulacija dostupne su na Youtube servisu [20].



Slika 3.7: Kompleksno ponašanje tijekom navigacije kroz prostor

4. Sustav sfernih robota Sphero

Autonomne agente, odnosno jedinke jata u ovom radu predstavljaju sferni roboti tipa Sphero SPRK+. Ovaj robot, prikazan na slici 4.1, građen je od plastične prozirne sfere promjera 7.5 cm unutar koje se nalaze svi mehanički i električni dijelovi potrebni za njegov ispravan rad. Sferno kućište je u stalnom kontaktu s podlogom i zbog svog oblika omogućava holonomično kretanje. Holonomično kretanje, poznato kao i omnidirekcionalno kretanje, omogućava kretanje u svim postojećim stupnjevima slobode. U dvodimenzionalnom slučaju kao za robote koji se kreću zemljom postoje tri stupnja slobode: translacija u x i y smjeru te rotacija oko z osi. Drugim riječima, holonomičan robot je u stanju gibati se naprijed-nazad, lijevo-desno, dijagonalno u svim smjerovima te se okretati na mjestu. Ovakav model robota značajno olakšava modeliranje dinamike kretanja te opravdava aproksimaciju točkaste mase korištenu za definiranje pravila kretanja u poglavlju 2.2. Pokretanje vanjske sfere omogućuju dva nezavisno upravljana kotača smještena na dno robota koji su u kontaktu s unutarnjom stranom kućišta. Čvrsto prianjanje uz kućište osiguravaju dva manja pasivna kotača koja su u kontaktu s gornjom stranom kućišta. Radi stabilnosti robota, centar gravitacije postavljen je vrlo nisko pomoću baterije, motora i malog utega.

Slika 4.1: Sphero SPRK+ model i komponente [21]

Sphero SPRK+ opremljen je žiroskopom i akcelerometrom tako da u svakom trenutku zna kako je orijentiran te se ti podaci koriste za održavanje stabilnosti i glatko upravljanje. Komunikacija s mobilnom aplikacijom ili računalom ostvaruje se preko protokola *Bluetooth 4.0 Low Energy* čime se omogućilo da je robot uvijek spreman za povezivanje i primanje naredbi, a u pasivnom stanju troši vrlo malo energije. Na vrhu tiskane pločice koja sadrži sve elektroničke komponente, mikroprocesor, upravljačke elemente za motore i čip za komunikaciju nalaze se dvije RGB svjetleće diode koje se mogu programirati da daju informacije o stanju robota (npr. stanje baterije) te jedna plava svjetleća dioda koja označava stražnji kraj robota i olakšava promjenu orijentacije.

(a) Sphero SPRK+ **(b)** Sphero SPRK+ Power Pack

Slika 4.2: Sustav sfernih robota

Sphero ima vlastiti koordinatni sustav koji nastoji održati tijekom kretanja. Smjer ”ravno” odgovara pozitivnom smjeru y-osi, a ”desno” pozitivnom smjeru x-osi. Praćenje kretanja žiroskopom i akcelerometrom te održavanje koordinatnog sustava stalnim znači da korisnik uvijek može upravljati robotom relativno na svoj položaj, a ne robotov. Kada korisnik zada naredbu za gibanje desno, robot će se gibati desno od korisnika kao što je i očekivano, a ne desno u odnosu na samoga sebe. Zbog zakretanja robota, ta dva smjera ne moraju se poklapati. Međutim, zbog nesavršenosti senzora, koordinatni sustav se nakon nekog vremena može zakrenuti u odnosu na početno stanje (*drift*). To se naročito događa zbog sudara s preprekama pa je potrebno povremeno provesti rekalibraciju. Sphero zahvaljujući svojim senzorima i *Bluetooth* komunikaciji korisniku može javljati svoju poziciju, brzinu i akceleraciju. Bitno je napomenuti da su to odometrijske vrijednosti te su kao takve podložne greškama.

5. Programsko okruženje

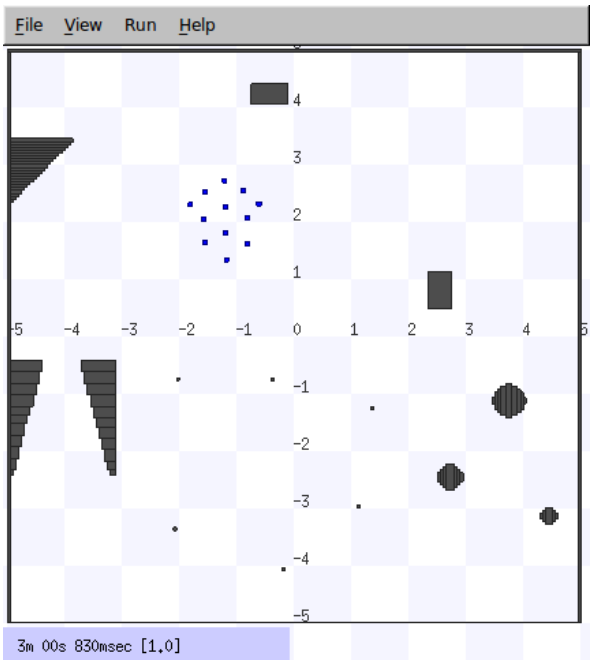
U ovom poglavlju opisana je programska podrška korištena u simulacijama i stvarnom svijetu. Kompletna implementacija ranije navedenih algoritama i pravila temelji se na programskom okviru ROS. Simulacije su provedene u simulatoru *Stage*, a kao povratna veza pozicije i brzine koristi se *OptiTrack* sustav. Za komunikaciju i upravljanje robotima Sphero napisan je upravljački program (engl. *driver*) objašnjen u ovom poglavlju.

5.1. ROS

ROS (*Robot Operating System*) je fleksibilan programski okvir namijenjen specifično za razvijanje programske podrške za rad s robotima. Sastoji se od brojnih alata, biblioteka i protokola koji olakšavaju izgradnju kompleksnih sustava potrebnih za sve zahtjevnije zadaće koje se stavljaju pred robote. Iako mu tako ime kaže, ROS nije pravi operacijski sustav, već oponaša neke njegove karakteristike, a poseban naglasak je na komunikaciji između višebrojnih, heterogenih sustava. Osnovna programska jedinica svakog sustava je čvor (*Node*). *Node* je obično jedna funkcija, modul ili skripta koja obavlja neki podzadatak. Komunikacija između čvorova ostvarena je razmjenom posebne klase poruka (*Messages*) različitih tema (*Topics*). Razlaganje težih problema na manje cjeline povećava preglednost, olakšava dijagnostiku grešaka te povećava modularnost i mogućnost višekratnog korištenja programskog koda za različite zadatke.

5.2. Stage simulator

Za potrebe simulacije koristi se *stage_ros* simulator [22]. To je jednostavan 2D simulator prilagođen ROS okruženju. Omogućava korištenje korisnički definiranih mapa, zadavanje brzine robota preko ROS poruka te šalje poziciju i brzinu svakog robota. Taj opseg funkcionalnosti je točno ono što je potrebno za pravilan rad algoritma. Sphero roboti modelirani su jednostavnim kvadratnim robotima kojima je omogućeno holonomično kretanje. Sustav je postavljen tako da vraća točnu poziciju robota bez odometrijske greške. Izgled simulatora prikazan je na slici 5.1.



Slika 5.1: *stage_ros* simulator s 12 Sphero robota

5.3. OptiTrack

Kao što je opisano u odjeljku 2.2, Reynoldsov algoritam gibanja u jatu zahtjeva poznavanje relativnih pozicija i brzina jedinki u jatu. Budući da su Spherovi senzori podložni nakupljanju grešaka te nemaju zadovoljavajuću preciznost, koristi se *OptiTrack* sustav za lokalizaciju. Usporedba mjerenja sa Spherovih senzora i sustava za lokalizaciju dana je u odjeljku 6.2.

OptiTrack je izrazito precizan sustav za praćenje i pozicioniranje objekata u prostoru (engl. *Motion capture*) u realnom vremenu. Ovakvi sustavi već se dugo koriste u filmskoj industriji za stvaranje digitalnih likova temeljenih na ljudskim pokretima, a zbog svoje preciznosti, sve se češće koriste i za robotiku. Sustav se sastoji od računala, većeg broja posebnih kamera te markera koji se postavljaju na objekt koji se želi pratiti. Kamere su opremljene senzorom osjetljivim na infracrvenu svjetlost te infracrvenim svjetlećim diodama. Diode odašilju svjetlost u prostor te se ona odbija od visoko-reflektivnih markera natrag prema senzoru kamere. Ako dvije ili više kamera vidi refleksiju s istog markera, poznavajući međusobni prostorni odnos kamera, iz više 2D slika se triangulacijom može izračunati 3D pozicija markera. Skup od 3 ili više markera predstavlja jedno kruto tijelo (engl. *rigid body*) za koje se u svakom trenutku može izračunati pozicija, ali i orijentacija u prostoru.

Zbog sfernog oblika robota koji se k tome još i stalno okreće, nije moguće postaviti i koristiti markere. Srećom, Spherove svjetleće diode, kada se postave na bijelu boju, dovoljno su jake da se uz određene postavke kamera prepoznaju kao jedan ili u rijetkim slučajevima kao dva markera.

Kako bi se to ostvarilo potrebno je ugasiti odašiljanje infracrvene svjetlosti iz kamera,

povećati ekspoziciju i smanjiti prag za detekciju markera. Zbog toga što se najčešće prepoznaje samo jedan marker, nije moguće stvoriti *rgid body* pa tako nije ni poznata orijentacija robota. Dodatno ograničenje je da ne postoji mogućnost pridjeljivanja identifikacijskih oznaka markerima, odnosno ne zna se koji marker pripada kojem robotu.

Za vizualizaciju podataka i rad sa sustavom kamera koristi se *Motive* sučelje. Unutar sučelja se također podešavaju postavke kamera. Izračunate pozicije se preko LAN mreže šalju pomoću *NatNet* protokola ugrađenog u *Motive*, a primaju se na računalu preko ROS paketa koji također podržava taj protokol. ROS paket šalje pozicije markera u listi bez identifikacijskih oznaka pa je potrebno razviti vlastiti sustav identifikacije i pridjeljivanja markera robotima. U tu svrhu koristit će se diskretni Kalmanov filtar za estimaciju pozicije i brzine svakog robota opisan u poglavlju 6.1.2.

5.4. Rviz

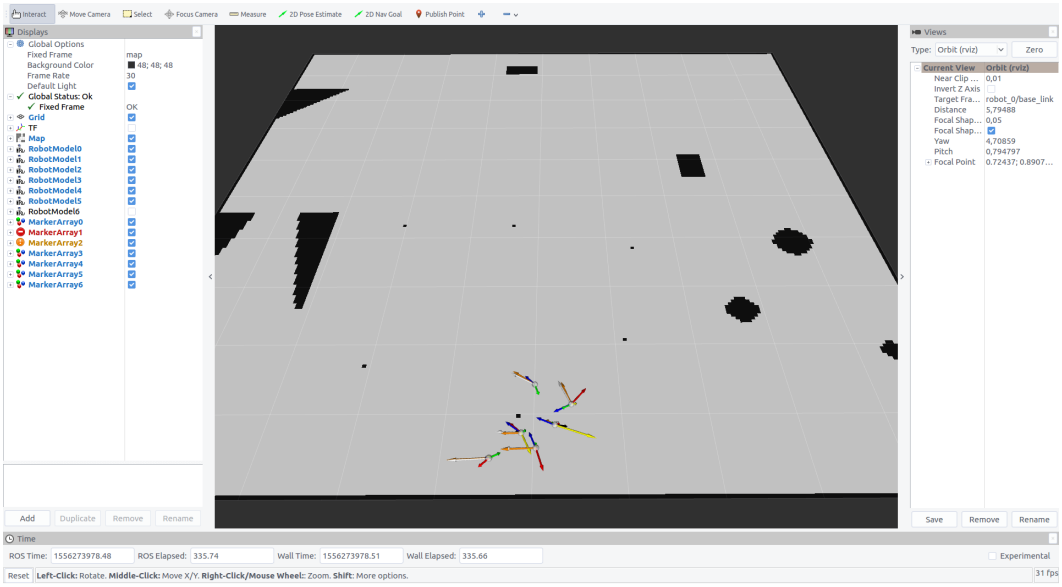
Kako bi bilo što lakše vizualizirati Spheroe u virtualnom prostoru i stvarnom vremenu vidjeti vrijednosti sila Reynoldsovih pravila, koristi se program *Rviz*.

Rviz je 3D vizualizacijski alat razvijen posebno za ROS. U jednostavnom grafičkom sučelju može prikazati model robota, koordinatne sustave, karte prostora, podatke sa senzora i još mnogo toga. Posebno je koristan kao alat za razvoj i uklanjanje grešaka. Vrlo ga je lako povezati s ostatkom razvojnog sustava koristeći ROS poruke, a ima i dobru podršku za pisanje korisničkih ekstenzija za proširivanje funkcionalnosti.

Na slici 5.2 prikazan je prozor *Rviz*-a s modelima Sphero robota i vektorima koji predstavljaju sile Reynoldsovih pravila te iznose brzine i akceleracije. Legenda boja vektora dana je tablicom 5.1.

Tablica 5.1: Legenda oznaka vektora u *Rviz*-u

Boja	**Značenje**
Plava	Sila pravila poravnanja
Crvena	Sila pravila odvajanja
Zelena	Sila pravila privlačenja
Žuta	Sila za izbjegavanje prepreka
Crna	Zadana akceleracija
Bijela	Zadana brzina
Narančasta	Estimirana trenutna brzina



Slika 5.2: *Rviz* prozor

5.5. Upravljački program za BLE komunikaciju

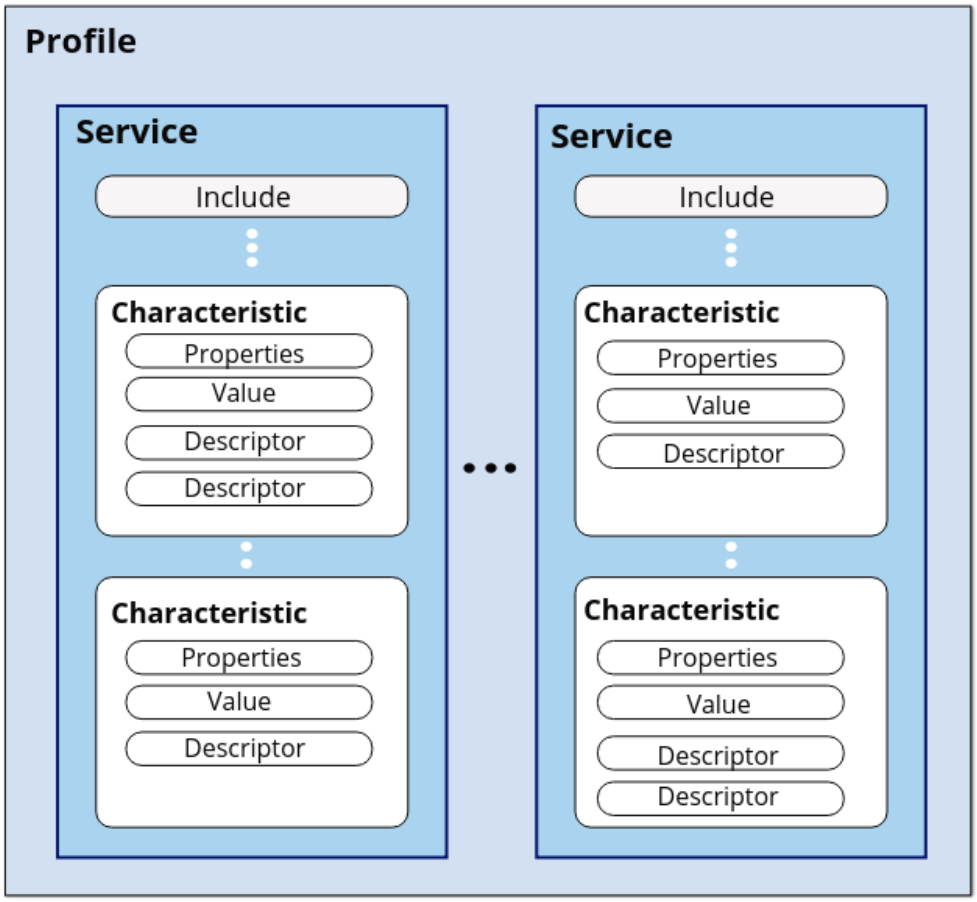
Za razvoj cijelog sustava potrebno je implementirati upravljački program (engl. *driver*) koji je temelj komunikacije i upravljanja robotima. Komunikacija se ostvaruje preko protokola *Bluetooth 4.0. Low Energy* kojeg odlikuje mala potrošnja energije i prijenos podataka na malim udaljenostima. *Bluetooth Low Energy* (BLE), medijski poznat i pod nazivom *Smart Bluetooth*, je tehnologija za bežičnu komunikaciju koja koristi nelicencirani frekvencijski pojas ISM (engl. *industrial, scientific and medical*) koji je globalno usklađen. Sve *Bluetooth* inačice odlikuje niska cijena, mali domet, mala veličina, robusnost te kompatibilnost s mobilnim uređajima i računalima. Posebnost BLE-a je vrijeme povezivanja koje iznosi nekoliko milisekundi i niska potrošnja energije koja omogućuje višegodišnji rad na malom izvoru energije. Za razliku od standardnog *Bluetooth*-a, BLE ostaje u stanju spavanja sve dok nije inicirano povezivanje s nekim drugim uređajem što uvelike doprinosi štednji energije. U većinu mobilnih uređaja i računala ugrađuje se *Bluetooth* mikrokontroler s *dual-mode* načinom rada što znači da podržava BLE i standardni *Bluetooth*, a uređaji s kojima se komunicira, kao što je Sphero, najčešće imaju ugrađen *single-mode* način rada.

Upravljački program koristi *bluepy* modul koji prati strukturu i protokole BLE-a.

5.5.1. Generic Attribute Profile

Bluetooth tehnologija temelji se na profilima koji se koriste pri slanju i primanju podataka. Profili osiguravaju standardizaciju i kompatibilnost između različitih BLE uređaja te optimiziraju upotrebu protokola. BLE koristi *Generic Attribute Profile* (GATT) protokol koji opisuje hijerarhijsku strukturu podataka na temelju profila. Na slici 5.3 je prikazan jedan profil i njegovi elementi:

- Servis (engl. *service*) je skup povezanih svojstava koji zajedno definiraju neku funkciju i odnos prema drugim servisima.
- Svojstvo (engl. *charateristic*) je vrijednost podatka koja se izmjenjuje između klijenta i poslužitelja.
- Opisnik (engl. *descriptor*) pruža dodatne informacije o svojstvima. Opisnik nije obavezan i svako svojstvo može imati proizvoljan broj opisnika.



Slika 5.3: Hijerarhijska struktura GATT protokola

Servisi, svojstva i opisnici se kolektivno nazivaju atributima, a identificiraju se pomoću 128-bitnog UUID-a (engl. *Universally unique identifier*). Umjesto cijelog UUID-a koriste se 16-bitni ili 32-bitni identifikatori, a ostatak UUID-a je standardiziran. Standardni profili i njihove specifikacije mogu se pronaći na službenim *Bluetooth* stranicama [23], a moguće je definirati vlastite profile za razne primjene.

Bitno je razlikovati pojam klijenta i poslužitelja u komunikaciji. Klijent je uređaj koji šalje GATT naredbe i zahtjeve poslužitelju te prima odgovore od poslužitelja. U našem slučaju klijent je računalo, a poslužitelj Sphero.

Na nižoj razini GATT koristi *Attribute* (ATT) protokol koji omogućava uređajima da međusobno saznaju koje servise podržavaju i koje parametre trebaju koristiti pri stvaranju podatkovne veze.

5.5.2. bluepy

Za uspostavljanje komunikacije s BLE uređajem na Spheroima korišten je Python modul *bluepy*. Klase modula prate prethodno opisanu strukturu GATT protokola.

Klasa *Pheripheral* označava periferni objekt, tj. server. Za povezivanje je potrebno zadati MAC adresu perifernog objekta i tip adrese. Klasa *Default Delegate* označava klijenta te sve funkcije u *bluepy* modulu prenose podatke i obavijesti ovoj klasi. Prikazani dio koda obavlja povezivanje između Spheroa i računala te definira klijenta i server.

```
self._device=bluepy.btle.Peripheral(self._addr, \
addrType=bluepy.btle.ADDR_TYPE_RANDOM)
self._notifier = DelegateObj(self, self._notification_lock)
self._device.withDelegate(self._notifier)
```

Nakon povezivanja potrebno je omogućiti slanje naredbi na Sphero jer je to temelj svih ostalih funkcionalnosti. Poznavajući UUID od servisa za upravljanje robotom dohvaćamo njegova svojstva (*ResponseCharacteristic* i *CommandsCharacteristic*) s funkcijom *getServiceByUUID*. Koristeći svojstvo *ResponseCharacteristic* možemo slati naredbe na Sphero, a svojstvo *CommandsCharacteristic* nam omogućuje primanje obavijesti sa Spheroa.

```
cmd_service = self._device.getServiceByUUID(RobotControlService)
self._cmd_characteristics = {}
characteristic_list = cmd_service.getCharacteristics()
```

Drugi servis koji koristi upravljački algoritam je BLE servis sa svojstvima *AntiDOSCharacteristic*, *TXPowerCharacteristic* i *WakeCharacteristic* pomoću kojih se omogućuje razvojni način rada i pokreće sekvenca buđenja robota.

5.5.3. Strukture paketa

Za slanje naredbi od klijenta, primanje obavijesti i podatka sa servera koriste se unaprijed definirani podatkovni paketi. Podatkovni paket za slanje naredbi i zahtjeva s klijenta na server prikazan je tablicom 5.2, a uloga pojedinih polja je:

- SOP1 (*start of packet*) označava početak paketa i uvijek iznosi $FF_{(16)}$.
- SOP2 definira želimo li primiti obavijest od servera o izvršenju naredbe. Ako želimo povratnu obavijest onda SOP2 postavljamo u vrijednost $FF_{(16)}$,a u suprotnom on iznosi $FE_{(16)}$.

- DID (*device ID*) je identifikacijski broj za uređaj kojem šaljemo paket. Ovdje koristimo dva DID-a, a to su: $00_{(16)}$ za jezgru i $02_{(16)}$ za Sphero.
- CID (*command ID*) je identifikacijski broj za naredbu koju želimo poslati. Svaka naredba ima svoj unaprijed definiran ID.
- SEQ (*sequence number*) je redni broj paketa i koristi se za sinkroni prijenos podataka, tj. kad očekujemo odgovor od Spheroa na poslanu naredbu od klijenta.
- DLEN (*data length*)je broj bajtova koji slijede do kraja paketa.
- CHK (*checksum*) je kontrolni zbroj bajtova od DID-a pa do kraja paketa. Kontrolni zbroj se računa tako da se na ukupni zbroj primjeni operacija modulo 256 te izračuna komplement.

Tablica 5.2: Struktura podatkovnog paketa za slanje naredbi

SOP1	SOP2	DID	CID	SEQ	DLEN	DATA	CHK

Struktura podatkovnog paketa kojim Sphero potvrđuje naredbe i šalje obavijesti prikazan je tablicom 5.3. MSRP (*message response*) je poruka koju šalje Sphero kao odgovor na naredbu, a u *data* polju se mogu nalaziti dodatne informacije o odgovoru ili zatraženi podaci sa senzora. Poruke omogućuju lakšu detekciju problema pri komunikaciji i sugeriraju potencijalna rješenja.

Tablica 5.3: Struktura podatkovnog paketa potvrdu naredbi

SOP1	SOP2	MSRP	SEQ	DLEN	DATA	CHK

Podatkovni paket kojim Sphero asinkrono šalje podatke klijentu prikazan je tablicom 5.4.

Tablica 5.4: Struktura podatkovnog paketa za asinkrono slanje podataka

SOP1	SOP2	ID CODE	DLEN-MSB	DLEN-LSB	DATA	CHK

ID CODE polje nam govori koje podatke nam Sphero šalje (npr. *data stream* sa senzora ili podaci o napajanju) . Za *data streaming* svih podataka sa senzora u paketu će se slati veća količina podataka pa za zapis duljine paketa koristimo dva bajta.

5.5.4. Funkcionalnosti

Upravljački program pod nazivom `sphero_driver.py` napisan je u programskom jeziku Python. Upravljački program implementira funkcije za slanje podataka, primanje sinkronih i asinkronih podataka, pakiranje paketa podataka, raspakiravanje paketa u smislene

podatke, itd. Funkcionalnosti upravljačkog programa dijele se na funkcionalnosti jezgrenog uređaja i funkcionalnosti Spheroa.

Funkcionalnosti jezgrenog uređaja obuhvaćaju akcije koje su temeljne za sve Orbotix uređaje, a to su:

- $ping(resp)$ funkcija provjerava komunikaciju s robotom.
- $sleep(resp)$ funkcija robota stavlja u stanje mirovanja.
- $set\_power\_notify(enable, resp)$ funkcija omogućuje periodično asinkrono obavještavanje klijenta o stanju baterije ili trenutno obavještavanje ako dođe do promjene stanja. Parametar *enable* indicira želimo li isključiti ili uključiti obavijesti.
- $get\_power\_state(resp)$ funkcija šalje informacije klijentu o stanju baterije, naponu i ciklusima punjenja.

Za slanje naredbi s klijenta na Sphero koristi se funkcija *send(cmd,data,resp)* čiji ulazni argumenti su identifikacijski kod naredbe i podatci koje je potrebno poslati. Za primanje povratnog odgovora ili slanje podataka od Spheroa prema klijentu koristi se funkcija *handleNotification(cHandle,data)* koja provjerava valjanost paketa. Podaci koji pristižu se spremaju u spremnik (engl. *buffer*) dok nije primljen cijeli paket podataka. Funkcija provjerava parametre SOP1 i SOP2 koji označavaju početak paketa te kontrolni zbroj za validaciju podataka u paketu. Ispravni paketi podataka se prosljeđuju funkcijama za raspakiravanje i obradu podataka ovisno o njihovoj vrsti.

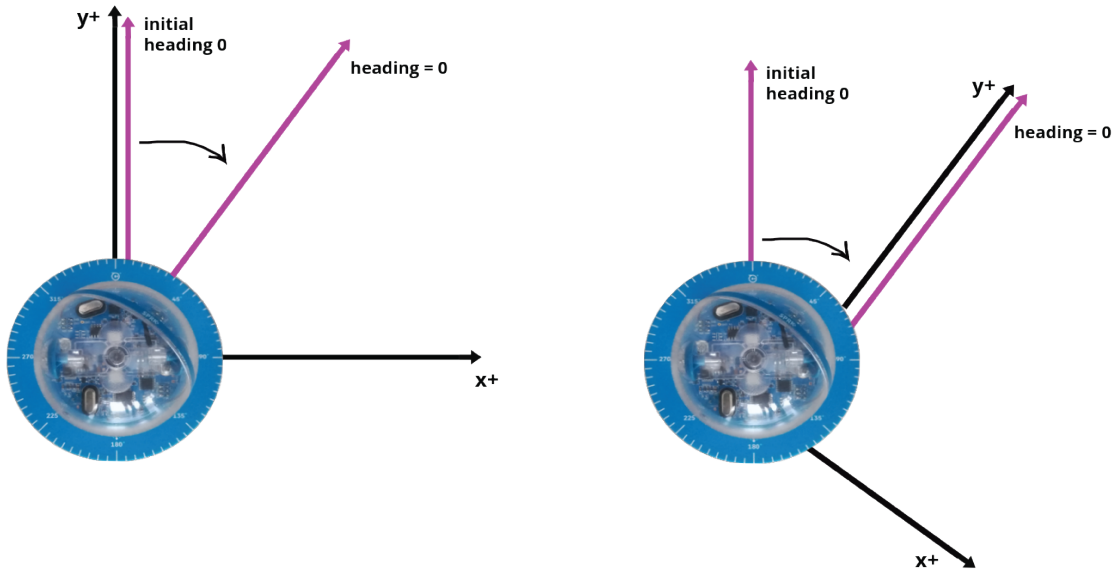
Funkcionalnosti specifične za robota Sphero su:

- $roll(speed, heading, state, resp)$ funkcija za kretanje robota sa zadanom brzinom i smjerom kretanja.
- $set\_heading(heading, resp)$ funkcija definira novi smjer kretanja.
- $set\_stabilization(bool\_flag, resp)$ funkcija za omogućavanje internog stabilizacijskog algoritma robota.
- $set\_raw\_motor\_values(bool\_flag, resp)$ funkcija onemogućuje internu stabilizaciju i postavlja zadane vrijednosti na motore.
- $set\_rotation\_rate(rate, resp)$ funkcija postavlja željenu vrijednost za brzinu rotacije kada robot mijenja svoj smjer kretanja.
- $set\_rgb\_led(red, green, blue, save, resp)$ funkcija postavlja RGB vrijednost boje svjetlećih dioda.
- $set\_back\_led(brightness, resp)$ funkcija postavlja vrijednost svjetline na stražnjoj svjetlećoj diodi.
- $set\_data\_strm(sample\_div, sample\_frames, sample\_mask1,$
  $pcnt, sample\_mask2, resp)$
  funkcija omogućava asinkrono slanje podataka sa senzora. Funkcija redom prima

parametre za djelitelja maksimalne frekvencije slanja podataka, broj okvira po paketu, prvu masku bitova za odabir zatraženih podataka, broj zatraženih paketa (nula za neograničeno slanje) i drugu masku bitova.

- $set\_locator(resp)$ je funkcija za postavljanje parametara lokatora.
- $read\_locator(resp)$ funkcija vraća podatke pozicije i brzine u dvodimenzionalnom prostoru.

Sve funkcije primaju parametar *resp* koji definira tražimo li povratni odgovor od robota o izvršenju naredbe. Za ispravljanje smjera gibanja zbog odstupanja (engl. *drift*) možemo koristiti funkciju $set\_heading$ ili $set\_locator$. Razlika između te dvije funkcije je prikazana na slici 5.4 gdje vidimo da promjenom smjera pomoću funkcije $set\_heading$ koordinatni sustav robota, tj. lokatora je nepromijenjen te se razlikuje od koordinatnog sustava u kojem se zadaju naredbe za kretanje. U funkciji $set\_locator$ možemo istovremeno promijeniti smjer kretanja i poravnati koordinatne sustave. Ovisno o primjeni možemo odabrati način koji nam više odgovara.



(a) $set\_heading$ **(b)** $set\_locator$

Slika 5.4: Postavljanje smjera kretanja i odnos koordinatnih sustava

Programska Python skripta `sphero_node.py` nadograđuje funkcionalnosti upravljačkog programa za korištenje u Robotskom operacijskom sustavu. Podaci sa senzora kao što su pozicija, linearna i kutna brzina se mogu dohvatiti pomoću funkcije set_data_strm ili korištenjem internog algoritma *Locator*. U tablici 5.5 su prikazane teme (engl. *topics*) i vrste poruka skripte `sphero_node` za slanje naredbi i očitavanje podataka sa senzora.

Tablica 5.5: `sphero_node` teme (engl. *topics*)

Subscribed Topics		Published Topics	
Topic	Message Type	Topic	Message Type
cmd_vel	Twist	diagnostics	DiagnosticArray
set_color	ColorRGBA	imu	Imu
set_heading	Float32	odom	Odometry
disable_stabilization	Bool		
manual_calibration	Bool		

6. Eksperimentalni rezultati

6.1. Implementacija programske podrške

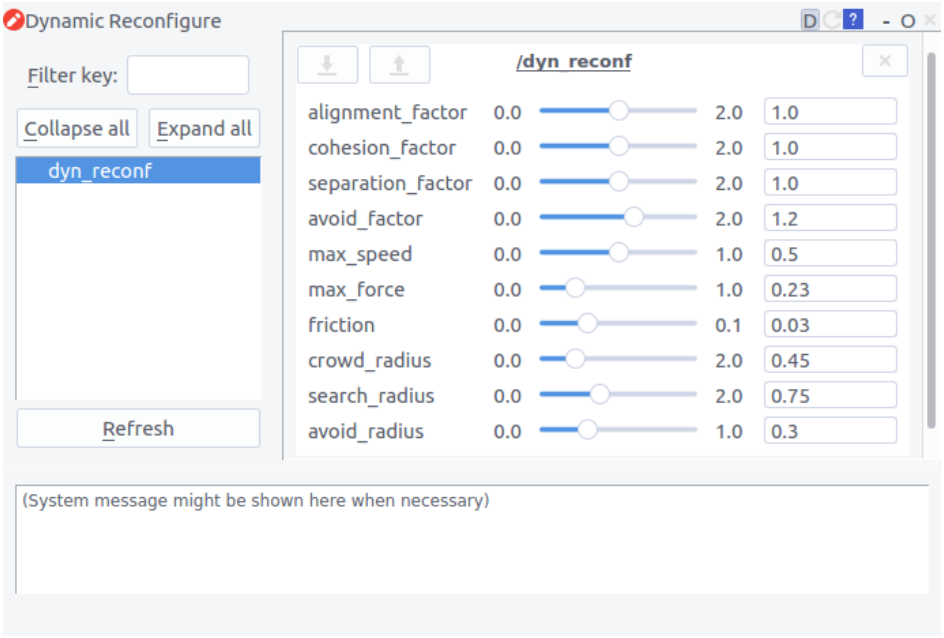
U skladu s tri ključne komponente autonomnog agenta, želja je da svaka jedinka samostalno i lokalno donosi odluke o svom ponašanju. U simulacijskom dijelu to jasno nije moguće budući da se sve komponente rješenja pokreću na istom računalu. Također, budući da Sphero roboti nemaju mogućnost direktnog povezivanja preko ROS-a ili izmjene njihovog upravljačkog programa, ni u praktičnom sustavu nije moguće postići potpunu distribuiranost. Međutim, u nadi korištenja algoritma na nekom drugom sustavu robota u budućnosti, sva programska podrška napravljena je u potpunosti modularno i tako da oponaša distribuiranost.

6.1.1. Centralni dio

Središnji dio sustava čine tri Python skripte. Reynoldsova pravila opisana u odjeljku 3.1 implementirana su kao klasa unutar datoteke `boids.py`. Kako bi se ostvarila komunikacija s ostatkom sustava, ta se klasa koristi unutar ROS *node*-a `reynolds_controller.py`. Za svakog agenta, simuliranog ili fizičkog, skripta se pokreće kao zaseban proces, prima pozicije i brzine samo onih agenata koji su unutar vidnog polja te vraća brzinu kojom se agent mora kretati da bi se gibao u skladu s jatom. Ovako modelirana programska podrška vrlo se lako može primijeniti na sustav s distribuiranim resursima. Jedini centralizirani dio sustava koji posjeduje sve trenutno dostupne informacije nalazi se u datoteci `nearest_search.py`. ROS porukama prenose se informacije o poziciji i brzini svakog agenta, a okruženje u kojem se agenti gibaju je definirano mapom. Ovaj *node* filtrira pristigle podatke te svakom agentu prosljeđuje liste relativnih pozicija njegovih susjeda i prepreka. Ako je zadovoljen format poruka, potpuno je svejedno dolaze li informacije iz simulatora ili nekog od mogućih sustava za lokalizaciju (npr. *OptiTrack* ili obična kamera). U simulaciji je moguće birati koriste li se stvarni podaci koji dolaze izravno iz simulatora ili estimirane vrijednosti iz Kalmanovog filtra.

Dodatno je razvijen i *node* koji omogućuje promjenu parametara algoritma u stvarnom vremenu tijekom izvođenja kako bi se lakše proučavali njihovi utjecaji na ponašanje jata. Promjenjivi parametri uključuju težine za sva četiri pravila (C_a, C_c, C_s, C_o), maksimalnu

brzinu v_{max} i silu F_{max}, iznos sile trenja F_{tr} te radijuse r_{search}, r_{crowd} i r_{avoid}. Konstante opisane u odjeljku 3.1 koje ovise o ovim parametrima također se preračunavaju po potrebi. Grafičko sučelje putem kojeg korisnik može promijeniti parametre prikazano je na slici 6.1:



Slika 6.1: Sučelje za podešavanje parametara

6.1.2. Kalmanov filter

Kao što je ranije spomenuto, zbog ograničenja *OptiTrack* sustava i Sphero robota, bilo je potrebno razviti sustav za identifikaciju robota i estimaciju brzine temeljen na Kalmanovom filtru. Kalmanov filter je rekurzivni matematički algoritam za estimaciju stanja dinamičkih sustava sa zašumljenim mjerenim signalima.

U ovom radu se koristi standardni diskretni Kalmanov filter izveden u [24]. Korak predikcije definiran je s:

$$\hat{x}_k^- = A_{k-1}\hat{x}_{k-1}^+ + B_{k-1}u_{k-1}, \tag{6.1}$$

$$P_k^- = A_{k-1}P_{k-1}^+A_{k-1}^T + L_{k-1}Q_{k-1}L_{k-1}^T. \tag{6.2}$$

Korak korekcije je definiran s:

$$K_k = P_k^- H_k^T \left(H_k P_k^- H_k^T + R_k\right)^{-1}, \tag{6.3}$$

$$\hat{x}_k^+ = \hat{x}_k^- + K_k(y_k - H_k\hat{x}_k^-), \tag{6.4}$$

$$P_k^+ = (I - K_k H_k)P_k^-(I - K_k H_k)^T + K_k R_k K_k^T. \tag{6.5}$$

Za varijable stanja sustava uzete su x i y komponente pozicije i brzine, a dostupno je

mjerenje x i y komponenti pozicije:

$$\hat{x}_k = \begin{bmatrix} x_k \\ y_k \\ \dot{x}_x \\ \dot{y}_k \end{bmatrix}, \quad y_k = \begin{bmatrix} x_{m,k} \\ y_{m,k} \end{bmatrix} \tag{6.6}$$

Model agenata koji se koristi u simulaciji i sa stvarnim robotima predstavljen je točkastom masom koja se giba holonomski. U modelu za Kalmanov filter ne koristi se upravljačka veličina u pa je $B = 0$. Sustav je vremenski nepromjenjiv što znači da su matrice A, Q,L, H i R konstantne te ih možemo pisati bez indeksa koraka k. Dakle, model sustava dan je jednadžbom:

$$\begin{bmatrix} x_k \\ y_k \\ \dot{x}_k \\ \dot{y}_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \dot{x}_{k-1} \\ \dot{y}_{k-1} \end{bmatrix} + \begin{bmatrix} \frac{T^2}{2} & 0 & 0 & 0 \\ 0 & \frac{T^2}{2} & 0 & 0 \\ 0 & 0 & T & 0 \\ 0 & 0 & 0 & T \end{bmatrix} \cdot \begin{bmatrix} w_{1,k-1} \\ w_{2,k-1} \\ w_{3,k-1} \\ w_{4,k-1} \end{bmatrix}, \tag{6.7}$$

gdje je T vrijeme diskretizacije, a $w_{1,k-1}$ do $w_{4,k-1}$ su komponente šuma koji na sustav djeluje preko akceleracije. Iz postavljenog modela mogu se iščitati matrice A, L i Q:

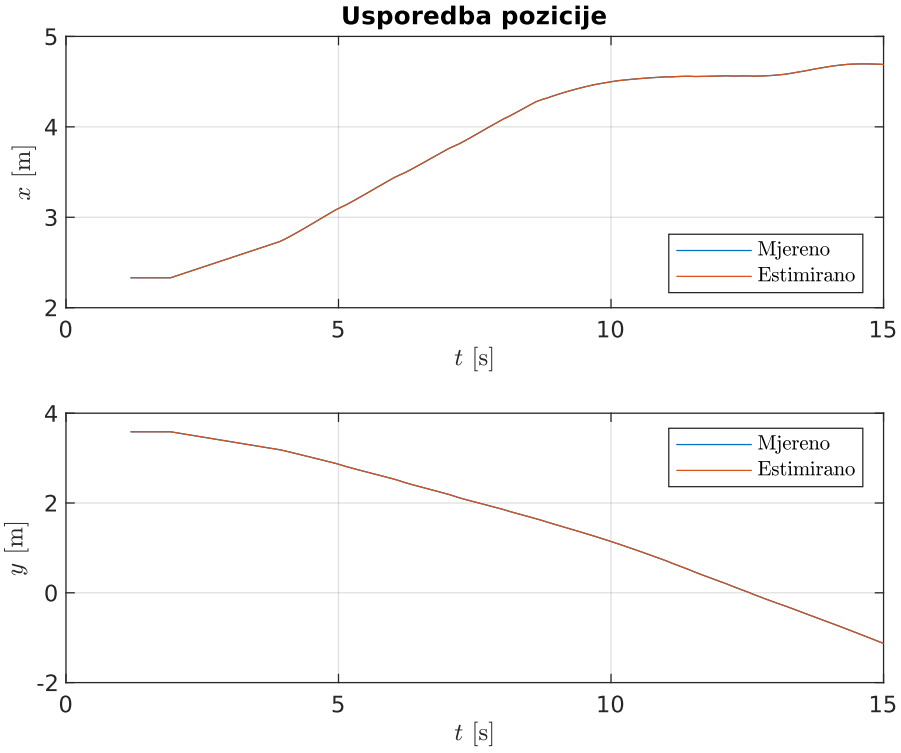
$$A = \begin{bmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad LQL^T = \begin{bmatrix} \frac{T^4}{4}q & 0 & 0 & 0 \\ 0 & \frac{T^4}{4}q & 0 & 0 \\ 0 & 0 & T^2q & 0 \\ 0 & 0 & 0 & T^2q \end{bmatrix}, \tag{6.8}$$

gdje je q iznos varijance šuma modela. Prema varijablama stanja i mjernim veličinama definiranim izrazom 6.6, matrice H i R poprimaju sljedeći oblik:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad R = \begin{bmatrix} 10^{-6} & 0 \\ 0 & 10^{-6} \end{bmatrix}. \tag{6.9}$$

Vrijednost 10^{-6} odabrana je jer su i u simulaciji i u stvarnom sustavu korištenjem *OptiTrack*-a dostupna vrlo kvalitetna mjerenja.

Opisani Kalmanov filter implementiran je u datoteci `kalman_filter.py`, a koristi se u ROS *node*-u koji omogućuje laku razmjenu podataka s ostatkom sustava. Prije početka izvođenja potrebno je odrediti početne pozicije Spheroa. Pokretanjem skripte `sphero_init.py`, roboti će jedan po jedan upaliti svoje svjetleće diode, držati ih upaljenim nekoliko sekundi te ih zatim ugasiti. Može se pretpostaviti da će za to vrijeme *OptiTrack* sigurno barem jednom prepoznati upaljene diode kao marker te poslati njegovu poziciju u sustav koji će je lako pridijeliti robotu. Nakon inicijalizacije Kalmanovog filtra početnim vrijednostima, zadanom frekvencijom se provodi predikcija pozicije i brzine temeljena na fizikalnom modelu sustava te se ažurira mjerenjima s *OptiTrack*-a. Prilikom svakog novog



Slika 6.2: Usporedba mjerene i estimirane pozicije

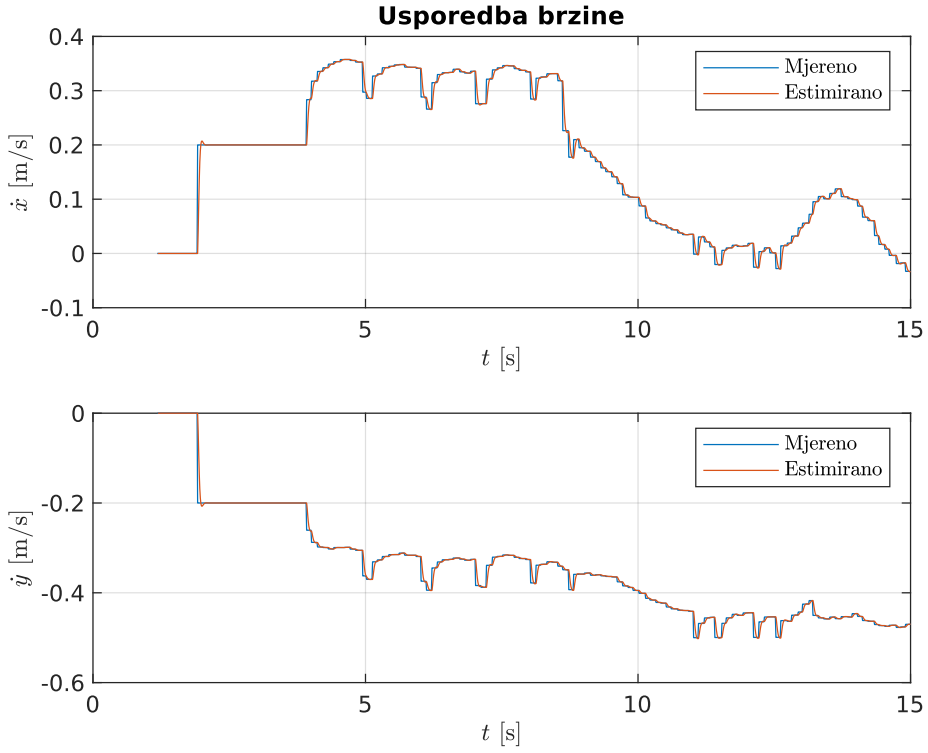
dolaska liste neoznačenih markera, uspoređuje se koji od njih se nalazi na udaljenosti manjoj od radijusa Spheroa u odnosu na zadnju estimiranu poziciju [1] te se, ako zadovoljava ovaj uvjet, pridjeljuje tom Spherou i unosi u Kalmanov filter kao mjerenje.

Kako bi se potvrdilo da je filter pravilno modeliran i implementiran, snimljeni su podaci o poziciji i brzini jednog od robota tijekom simuliranog gibanja u jatu. Na slikama 6.2 i 6.3 prikazane su usporedbe estimiranih podataka i stvarnih podataka dobivenih iz simulatora. Estimirane vrijednosti gotovo se savršeno slažu sa stvarnima. Estimacija pozicije robota očekivano je preciznija od estimacije brzine zahvaljujući dostupnosti kvalitetnih mjerenja. Detaljnijom analizom rezultata eksperimenta utvrđeno je da estimacija brzine u prosjeku kasni samo za 0.05 sekundi, a srednje kvadratne pogreške reda su veličine 10^{-5}. Točni iznosi dani su u tablici 6.1.

Tablica 6.1: Srednje kvadratne pogreške estimiranih vrijednosti

$err(x) =$	$8.0270 \cdot 10^{-6}\ \mathrm{m}^2$
$err(y) =$	$1.0447 \cdot 10^{-5}\ \mathrm{m}^2$
$err(\dot{x}) =$	$7.2744 \cdot 10^{-5}\ \mathrm{m}^2/\mathrm{s}^2$
$err(\dot{y}) =$	$6.1004 \cdot 10^{-5}\ \mathrm{m}^2/\mathrm{s}^2$

[1] Nova mjerenja u sustav pristižu frekvencijom od 100 Hz pa je pretpostavka da se robot između dva koraka neće pomaknuti za više od svog radijusa valjana.



Slika 6.3: Usporedba mjerene i estimirane brzine

6.1.3. Pokretanje, postavke i ručno upravljanje

Radi lakšeg konfiguriranja i pokretanja programske podrške, pripremljene su *launch* datoteke koje postavljaju sve potrebe parametre i pokreću zadani broj instanci ROS *node*-ova opisanih u prethodnim poglavljima.

Za simulaciju su pripremljene datotke:

1. `setup_sim.launch`: Definira vrijednost parametara te pokreće simulator, *Rviz*, Kalmanov filter i *node* za dinamičku promjenu parametara.
2. `reynolds_sim.launch`: Pokreće `reynolds_controller` i `nearest_search` *node*-ove te započinje izvođenje.

U stvarnom sustavu se koriste sljedeće datoteke:

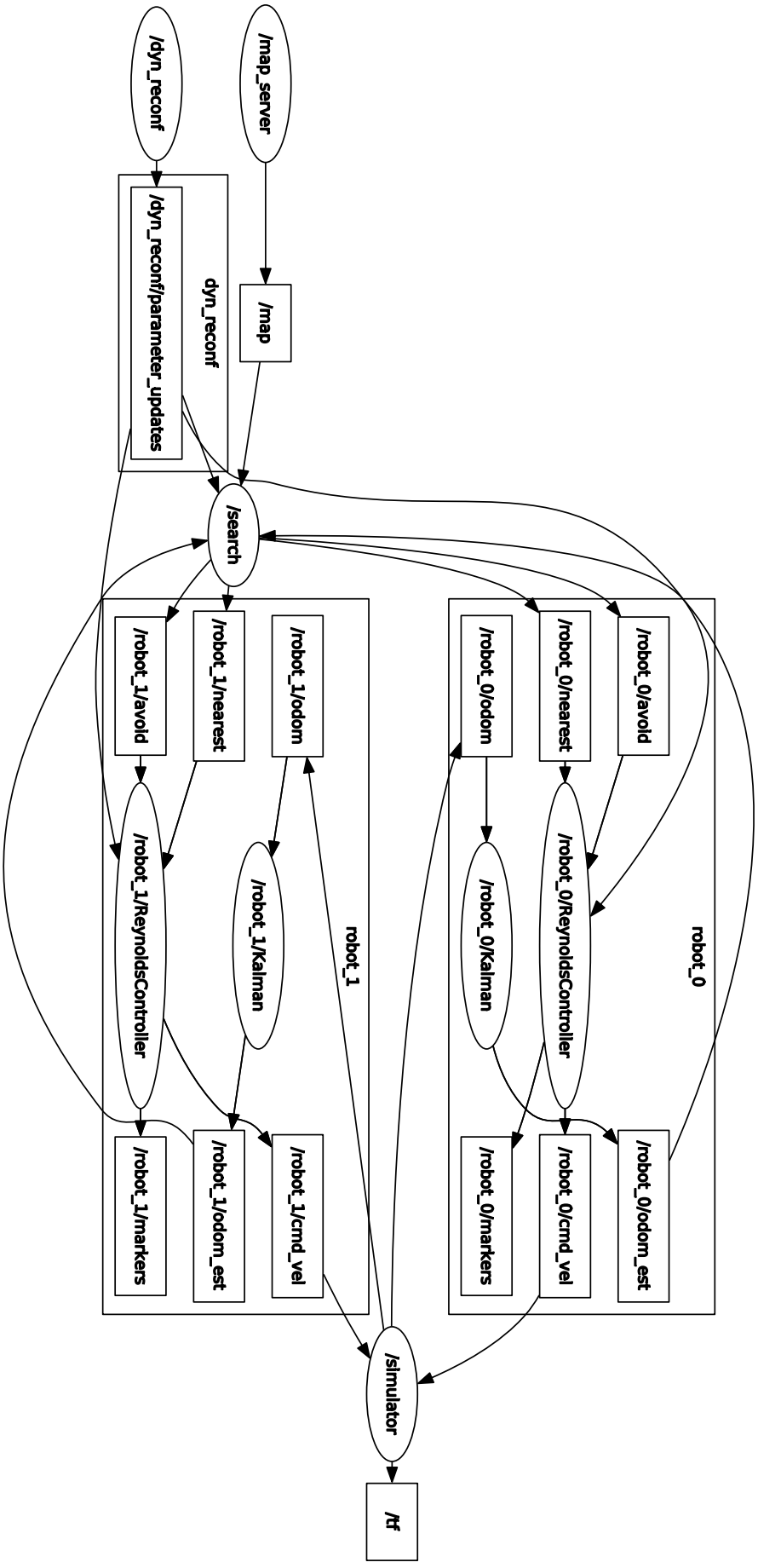
1. `setup_real.launch`: Definira vrijednost parametara te pokreće *Rviz* i *node*-ove za slanje podataka s *OptiTrack*-a, ručno upravljanje i dinamičku promjenu parametara.
2. `drivers.launch`: Pokreće *drivere* za Spheroe.
3. `tracking.launch`: Pokreće Kalmanov filter i skriptu za dohvaćanje početnih pozicija robota.
4. `flocking.launch`: Pokreće `reynolds_controller` i `nearest_search` *node*-ove te započinje izvođenje.

Ručno upravljanje Sphero robotima vrši se pomoću Logitech F710 *joystick*-a, a funkcionalnost tipki objašnjena je slikom 6.4.

Slika 6.4: Logitech F710 joystick

Shematski prikaz simulacijskog sustava u ROS-u za slučaj s dva robota nalazi se na slici 6.5. Na ovom prikazu izravno se vidi sve što je prethodno opisano o sustavu. Simulator šalje podatke o pozicijama svih agenata te se za svakoga posebno Kalmanovim filtrom estimira njegovu brzinu. Podaci o svim agentima skupljaju se u centraliziranom *node*-u za traženje susjeda koji prosljeđuje relevantne informacije o susjedima i preprekama. Na temelju dobivenih informacija, za svakog agenta posebno se računa njegova buduća brzina te se šalje simulatoru.

Shematki prikaz realnog sustava je vrlo sličan. Razlike su jedino u tome što pozicije agenata ne šalje simulator, već *node* koji obrađuje podatke s *OptiTrack*-a, a zadane brzine u temi *cmd_vel* šalju se *driver*-ima koji upravljaju pojedinačnim robotima.

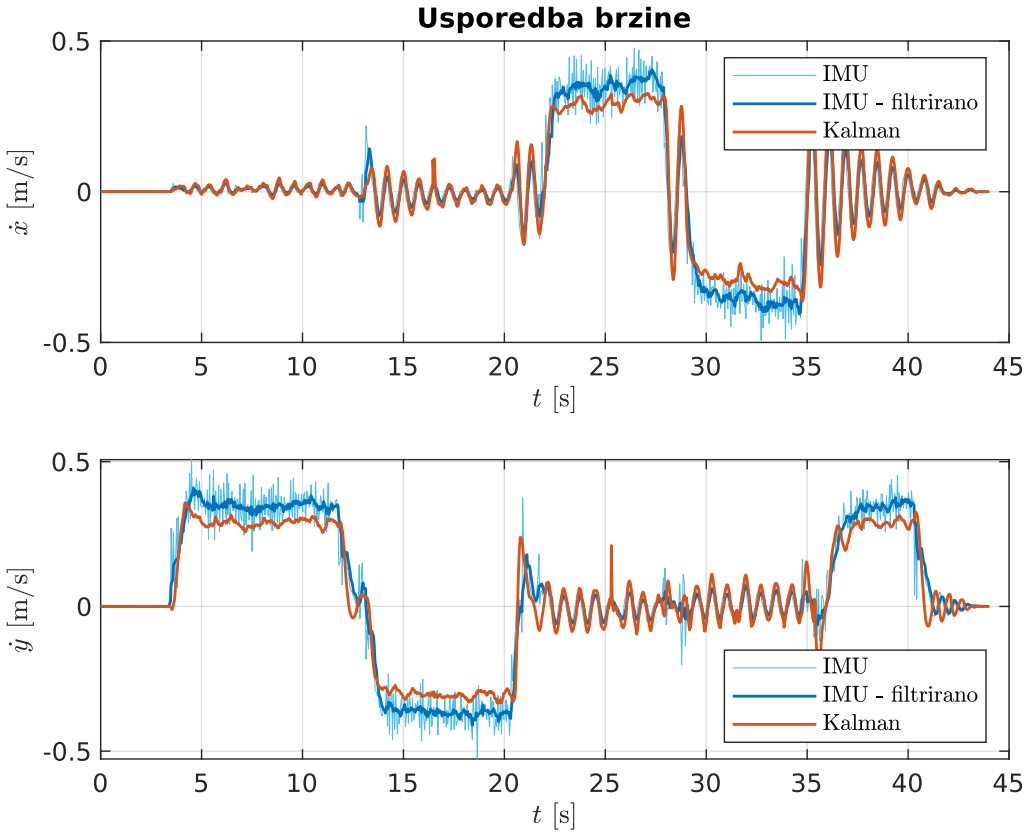


Slika 6.5: Shema simulacijskog sustava

6.2. Analiza Spherovih senzora

Kao što je navedeno, Spheroi su opremljeni IMU (*Inertial Measurement Unit*) senzorom koji uključuje akcelerometar i žiroskop te korisniku mogu javljati svoju poziciju, orijentaciju, brzinu i akceleraciju. Određivanje brzine i pozicije temelji se na integriranju podataka s akcelerometra. Zbog nesavršenosti mjerenja akcelerometra te numeričkih pogrešaka prilikom integracije, tijekom vremena se u izvedenim veličinama nakuplja sve veća pogreška.

Kako bi se odredilo koliko izračunata pozicija i brzina zaista odstupaju od stvarnosti, proveden je eksperiment u kojem su uspoređeni odometrijski podaci s podacima dobivenim iz *OptiTrack* sustava i Kalmanovog filtra. *OptiTrack* je prethodno kalibriran i osigurava preciznost određivanja pozicije od pola milimetra, a u kombinaciji s Kalmanovim filtrom osigurana je stabilnost i točnost mjerenja kroz vrijeme. Ispravnost implementacije Kalmanovog filtra potvrđena je u odjeljku 6.1.2. Eksperiment se sastojao od ručnog upravljanja jednim robotom u x i y smjeru.

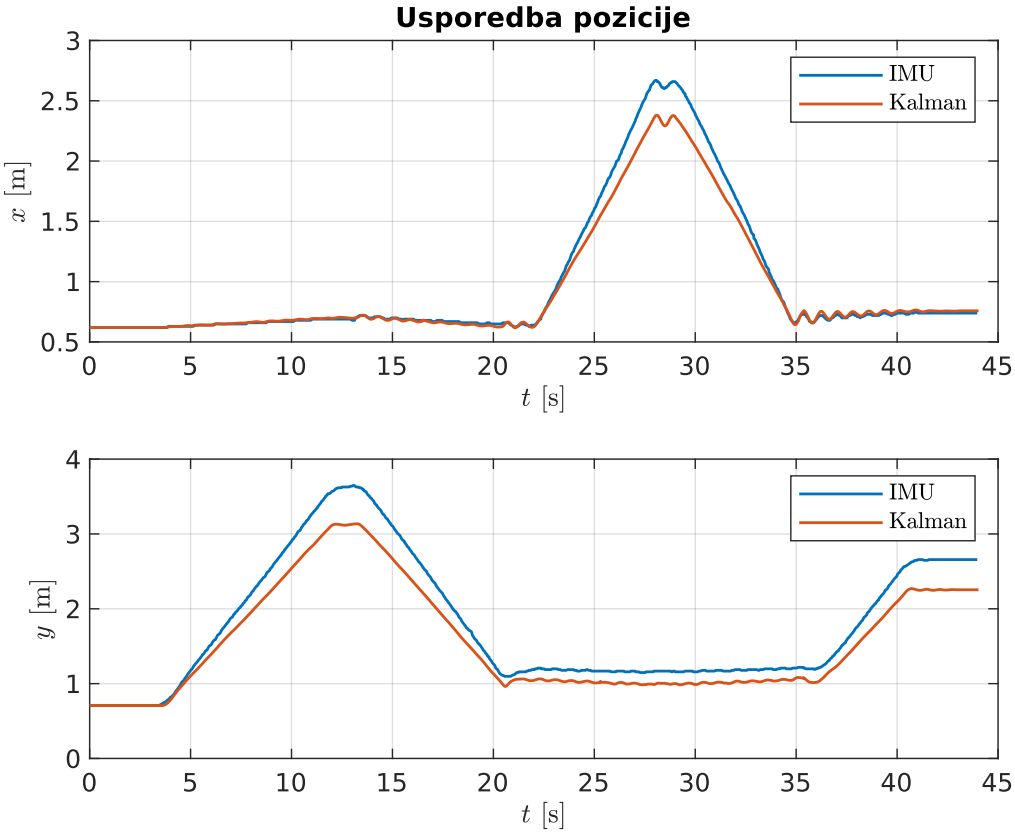


Slika 6.6: Usporedba brzine s IMU-a i iz Kalmanovg filtra

Na slici 6.6 prikazana je usporedba podataka brzine. Prvo što se može uočiti jest činjenica da je brzina koju Sphero šalje vrlo zašumljena. Radi jasnije analize, signal brzine naknadno je filtriran metodom pomičnog prosjeka uz širinu prozora od 5 elemenata. Nakon filtriranja vidljivo je da se dinamike oba signala vrlo dobro slažu. Međutim, izračunata brzina je oko

20 % veća od stvarne. U trenutku $t = 13$ s Sphero se naglo zakreće u mjestu kako bi krenuo u negativnom smjeru y osi. Taj zakret uzrokuje ljuljanje, odnosno oscilatornu brzinu u x smjeru. Zbog daljnjih naglih zakreta u mjestu, ljuljanje se nastavlja i u ostatku eksperimenta.

Usporedba podataka pozicije prikazana je na slici 6.7. U signalu pozicije ne primjećuje se nikakav šum pa dodatna obrada nije bila potrebna. Na ovom se prikazu posebno vidi ranije opisani problem s brzinom. Zbog pogreške u računu brzine, dobivena pozicija se vremenom sve više razlikuje od stvarne. Ipak, čini se da je odstupanje brzine konzistentno: vraćanjem Spheroa na mjesto odakle je eksperiment započeo i pozicija dobivena s njegovih senzora poprima točnu vrijednost. Treba imati na umu da je eksperiment trajao relativno kratko pa nakupljanje integracijskih grešaka ne dolazi do izražaja.



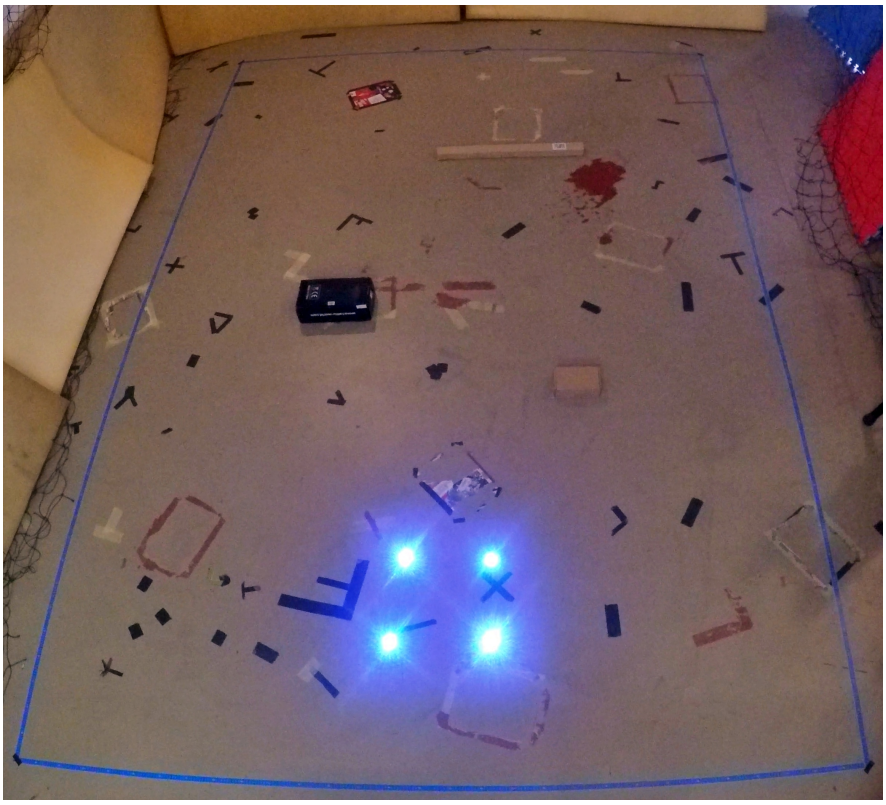
Slika 6.7: Usporedba pozicije s IMU-a i iz Kalmanovog filtra

Iako se Spherov IMU senzor pokazao kao relativno dobro rješenje za grubo određivanje pozicije i brzine, nedostatak preciznosti i nakupljanje greške tijekom vremena čini ga nepovoljnim za korištenje u algoritmu Reynoldsovih pravila.

6.3. Primjena algoritma na jato Sphero robota

Kao što je ranije navedeno, cilj ovoga rada je pokazati kako primjenom pravila ponašanja koja primjećujemo u prirodi možemo ostvariti kompleksna gibanja višeagentskog sustava koja oponašaju jata ptica ili roj pčela. Takvo decentralizirano upravljanje formacijom sfernih robota Sphero ostvareno je u sljedećim eksperimentima.

Duljina korištenog radnog prostora iznosi 3.9 metara, a širina 2.6. Tako je prostor površinom gotovo 90% manji nego u simulaciji zbog čega je istovremeno kretanje više Spheroa bilo izazovno, a i teže je dolazilo do nekih specifičnih ponašanja jata prikazanih u odjeljku 3.2. Eksperimentalni postav prikazan je slikom 6.8, a korištena programska podrška opisana je u poglavlju 5. Prije početka svakog eksperimenta potrebno je kalibrirati orijentaciju jedinki u jatu uz pomoć stražnje plave svjetleće diode kao što je prikazano na slici 6.9.

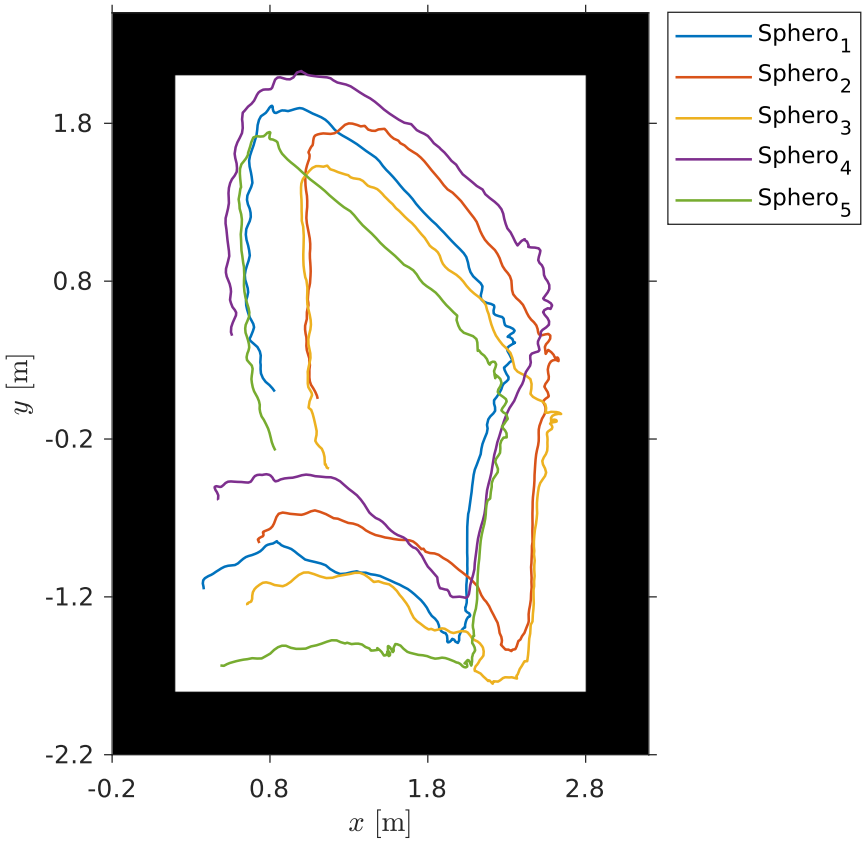


Slika 6.8: Eksperimentalno okruženje

Realni sustav podložan je mnogim smetnjama koje ne postoje u simulacijama. Iako je *OptiTrack* u pravilu iznimno precizan i pouzdan alat, nije prilagođen za praćenje markera nepravilnog oblika i u vidljivom dijelu spektra svjetlosti. Nejednaki uvjeti osvjetljenja, ljuljanje robota i odbljesci mogu značajno otežati pravilnu identifikaciju markera. Primjenom Kalmanovog filtra većina vanjskih utjecaja je eliminirana osim ako mjerenje nedostaje dulje od sekunde. U slučaju duljeg izostanka mjerenja s *OptiTracka* može doći do značajnijeg udaljavanja estimirane od stvarne vrijednosti što u konačnici vodi do trajnog gubljenja pozicije robota. U tom slučaju potrebno je ponovno pokrenuti izvedbu.

Slika 6.9: 11 Spheroa u kalibracijskom načinu rada

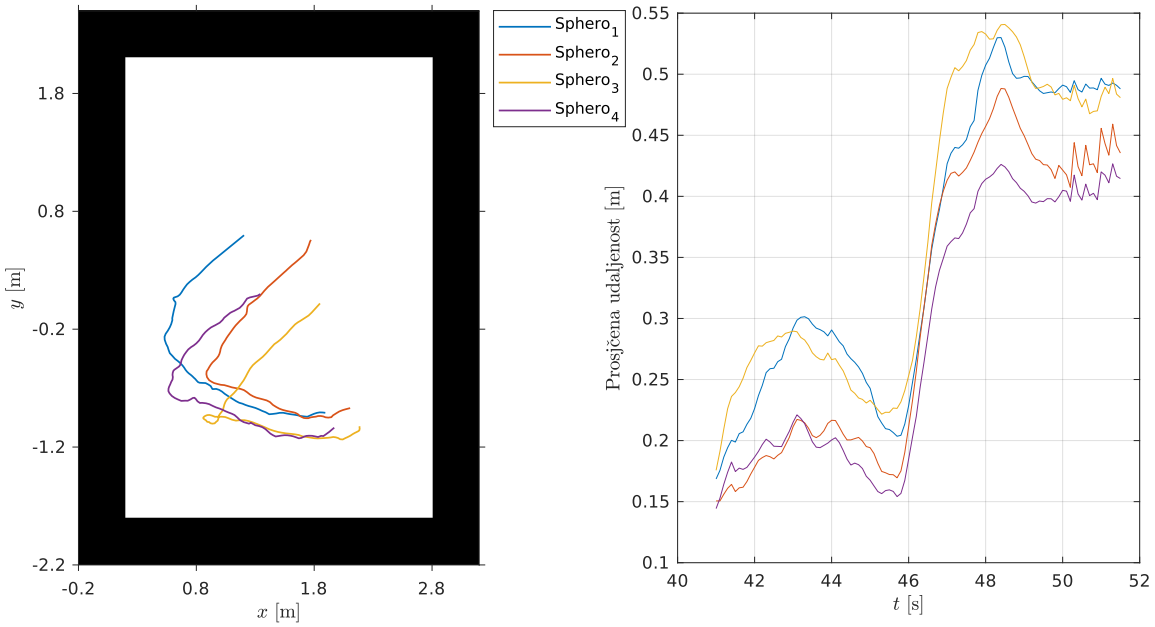
Dodatna poteškoća je sporija dinamika upravljanja Sphero robotima. Masa agenata u simulaciji jednaka je stvarnoj masi Spheroa zbog čega bi oni trebali imati jednaku inerciju. Ipak, dok agenti u simulaciji naredbe o promjeni brzine primjenjuju instantno, u realnom sustavu to nije moguće. Kako bi se jatu dalo više vremena za reakciju na okolinu i unutarnja stanja, maksimalna brzina gibanja smanjena je s 0.5 m/s na 0.4 m/s. Ovo smanjenje pokazalo se dovoljnim za ostvarivanje dobrog ponašanja.



Slika 6.10: Skladno gibanje u formaciji s pet robota

Zbog ograničenja veličine prostora, karakteristike željene dinamike najviše dolaze do izražaja pri eksperimentima s četiri robota. Gibanje u formaciji postignuto je i s veličinom jata od 5 i 6 robota što je prikazano na slikama 6.10 i 6.13. S povećanjem broja robota, tj. jedinki u jatu, dolazi do otežane komunikacije putem bluetooth tehnologije te se pojavljuju problemi u povezivanju i kašnjenje signala. Međutim, zahvaljujući činjenici da je razvijena programska podrška u potpunosti modularna i distribuirana, upravljačke programe za robote moguće je pokretati na više računala istovremeno. Pri tome se mogu koristiti i jeftina minijaturna računala opremljena s *Bluetooth*-om 4.0 kao što je to Raspberry Pi 3. Uz dovoljno velik prostor, sustav se vrlo lako može prilagoditi za rad s većim brojem robota.

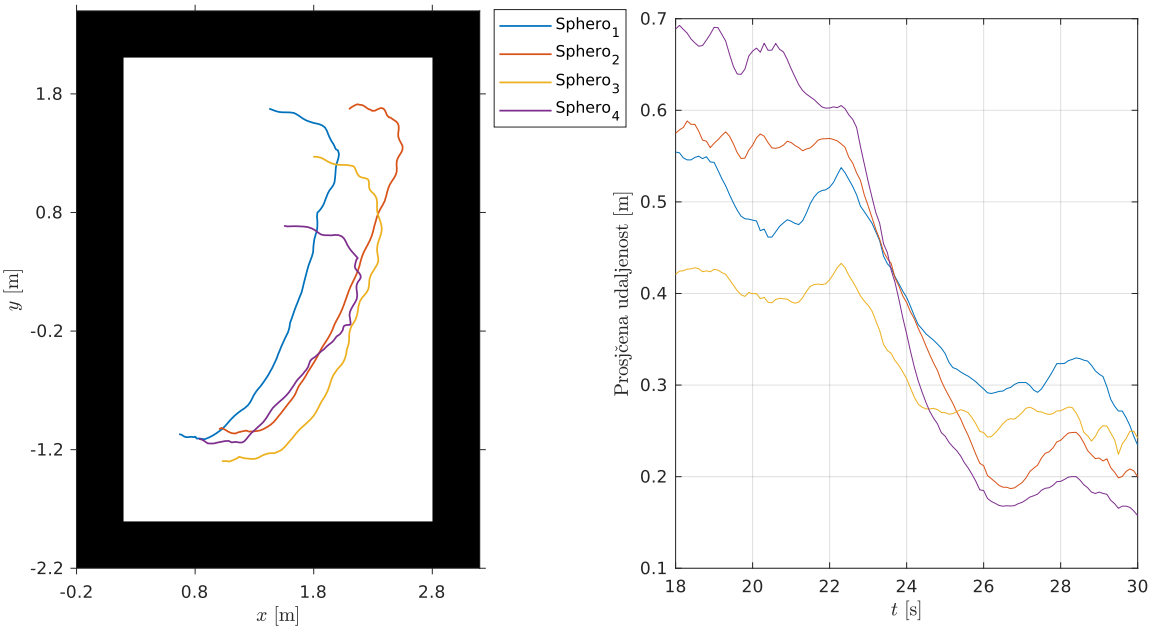
Unatoč navedenim poteškoćama, sustav ostvaruje odlične rezultate i primjećujemo kompleksna ponašanja jata kao i u simulaciji. Agenti se skupno gibaju u prostoru prateći zidove ili se odbijajući od njih. Ubrzavaju i usporavaju kao cjelina te mijenjaju oblik izmjenjujući pozicije u jatu. Pritom je zbog manje dostupnog prostora primjetnije kružno gibanje.



Slika 6.11: Ponašanje prilikom povećanja međusobne udaljenosti

Kao i u simulaciji, jato je vrlo responzivno na dinamičku promjenu parametara algoritma. Na slikama 6.11 i 6.12 prikazani su isječci eksperimenta u kojem se 4 agenta gibaju u prostoru bez prepreka te povremeno primaju naredbu za promjenu međusobnog razmaka (r_{crowd}). Na lijevoj strani slika prikazana je korištena mapa s naznačenim dimenzijama prostora te putanje robota tijekom eksperimenta. Na desnoj strani je vremenski dijagram prosječne udaljenosti između svakog od robota i njihovih susjeda. S dijagrama se može uočiti da se međusobne udaljenosti kontinuirano mijenjaju dok se roboti kreću, ali je razlika u zadanom razmaku jasna. Od trenutka zadavanja promjene jatu je potrebno otprilike 2 sekunde za razdvajanje i još 2 za poprimanje stacionarnog stanja na novoj udaljenosti. Situacija kod spajanja je slična, a odziv je malo sporiji jer su se roboti kretali većom brzinom zbog čega

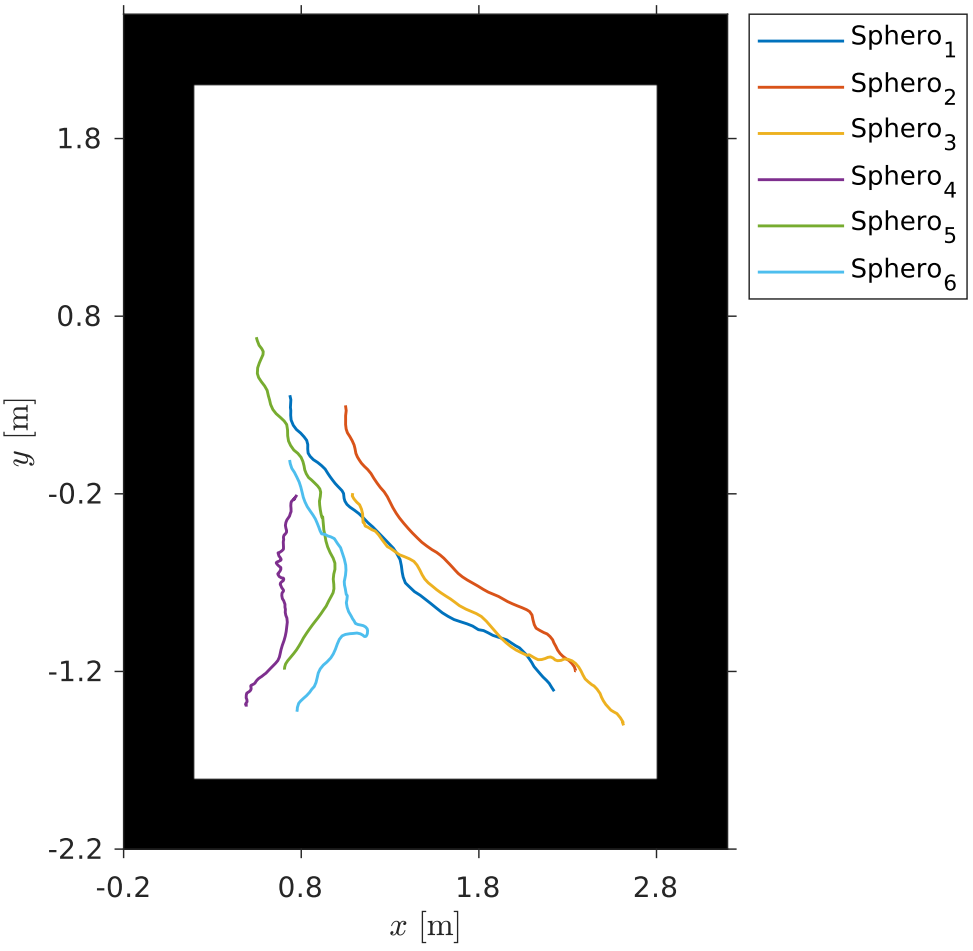
im je bilo teže ubrzati. Demonstrirano ponašanje ukazuje na veliku prednost ovog sustava da se podešavanjem parametara algoritma prilagodi svojoj okolini i zajedničkom cilju jata.



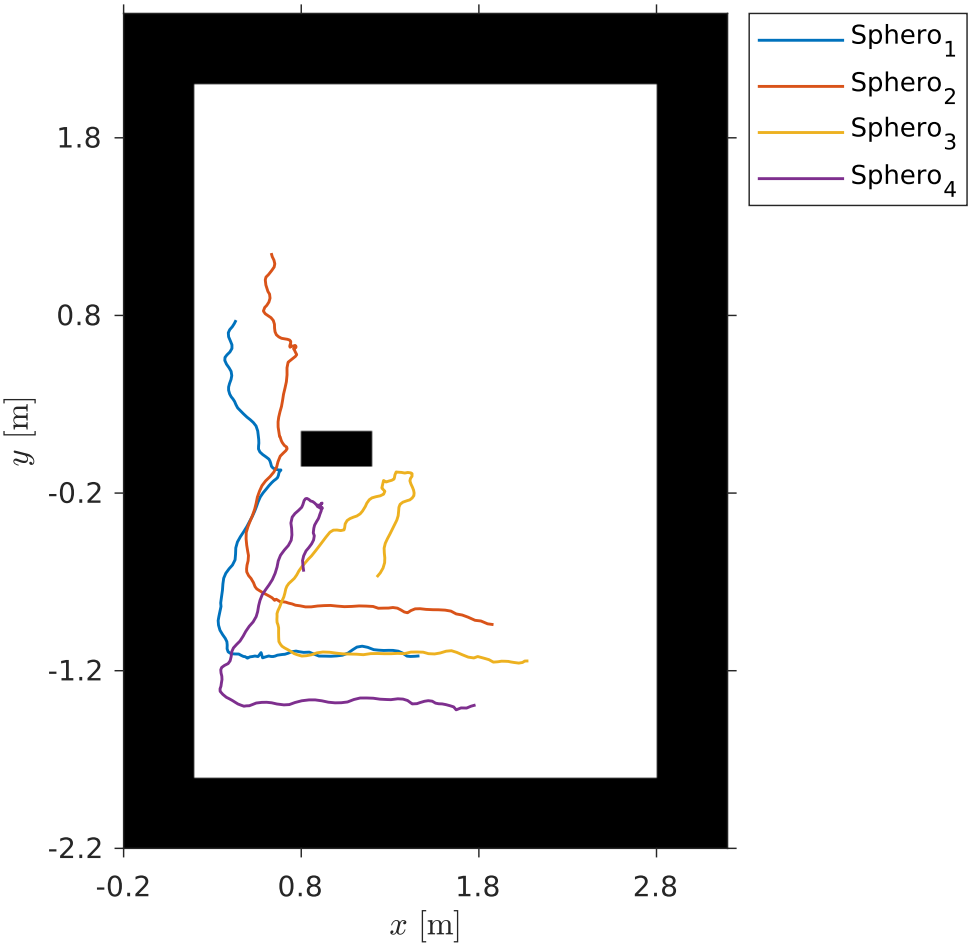
Slika 6.12: Ponašanje prilikom smanjenja međusobne udaljenosti

U drugom eksperimentu u prostoru bez prepreka postavljeno je 6 robota. Roboti su postavljeni u susjedne kutove prostora u dva manja jata od 3 jedinke te su im početne brzine podešene tako da se krenu gibati prema središtu mape. Približavanjem sredini, jata ulaze jedno drugom u vidno polje te zbog sile privlačenja nastoje stvoriti novo veliko jato. Sastajanjem, svih šest robota se nastavlja gibati zajedno. Putanje ostvarene tijekom ovog eksperimenta prikazane su slikom 6.13.

Naposljetku, u prostor je dodana pravokutna prepreka u lijevi središnji dio kako bi se demonstriralo svojstvo samoorganiziranosti jata. Prvih nekoliko minuta trajanja eksperimenta roboti se zajednički gibaju u donjem dijelu prostora. U jednom trenutku dolaze do prepreke s donje strane te se razdvajaju na dva manja jata. Jato koje se uputilo u gornji dio ubrzo se okreće te se vraća istim putem i ponovno spaja s ostatkom nakon čega nastavljaju dalje istraživati prostor. Ovakvo kompleksno ponašanje omogućuje višeagentskim sustavima da nastave svoje gibanje bez obzira na događaje koji mogu oštetiti ili ukloniti dio jata. Putanje ostvarene ovim eksperimentom prikazane su slikom 6.14.



Slika 6.13: Spajanje dvaju jata



Slika 6.14: Razdvajanje prilikom nailaska na prepreku

Provedenim eksperimentima još jednom je potvrđen cilj ovog rada. Jato sfernih robota Sphero uspješno realizira sva karakteristična gibanja ranije opisana u teorijskom dijelu i prikazana u simulacijama. Osim prikaza snimljenih podataka za navedene eksperimente, dostupni su i video uradci svih eksperimenata na Youtube servisu [20]. Programska implementacija upravljačkog algoritma korištena pri eksperimentima dostupna je u Github repozitoriju [25], a implementacija upravljačkog programa za Sphero robote dostupna je u repozitoriju [26].

7. Zaključak

U ovom radu implementirano je decentralizirano upravljanje jatom autonomnih robota korištenjem Reynoldsovih pravila inspiriranih ponašanjem jata ptica i riba u prirodi. Pokazano je da korištenjem samo tri vrlo jednostavno definirana pravila poravnanja, privlačenja i odbijanja jedinki uz ograničenu percepciju okoline ostvarujemo kompleksna ponašanja jata kao cjeline. Kretanje jata posljedica je isključivo kumulativnih efekata lokalnih interakcija između jedinki te prostora u kojem se nalaze. Podesivi parametri bihevioralnog algoritma kao što su međusobna udaljenost agenata i maksimalna brzina gibanja omogućavaju jatu da svoju formaciju prilagodi okolini i pospješi ostvarenje zajedničkog cilja skupine agenata. Autonomni agenti unutar jata samostalno odlučuju o svom gibanju što otvara mogućnost razdvajanja jata pri nailasku na prepreke i ponovnog spajanja pri susretu dvaju ili više jata. Jato tako ostvaruje svoje gibanje neovisno o događajima koji mogu oštetiti ili ukloniti njegove dijelove.

Teorijske pretpostavke i očekivana dinamika ponašanja potvrđeni su u simulaciji, a potom i eksperimentima sa stvarnim sustavom. Višeagentski robotski sustav kojeg predstavljaju točkaste mase u simulaciji ili sferni roboti Sphero u stvarnom okruženju, autonomno ostvaruje kontinuirano gibanje u formaciji za ostvarenje zajedničkog cilja. U sklopu ovog rada uspješno je implementiran i upravljački algoritam (*driver*) za Sphero SPRK+ robote te Kalmanov filtar koji osigurava stabilnost povratne veze po poziciji i brzini.

Na distribuiranim sustavima, računalni zahtjevi za provedbu razvijenih algoritama relativno su skromni te je stoga moguće istovremeno koristiti velik broj agenata u jatu. Time je sustav prilagodljiv različitim zadacima i primjenjiv na razne probleme u području robotike kao što su misije traganja i spašavanja, istraživanje prostora, prikupljanje podataka i sl. Razvijeni sustav također je i izrazito modularan što omogućava vrlo jednostavno podešavanje u budućnosti. Uz minimalan trud moguće je zamijeniti simulator, sustav za lokalizaciju ili čak i vrstu korištenih robota te ga tako prilagoditi specifičnostima postavljenog zadatka.

ZAHVALA

Zahvaljujemo mentoru prof. dr. sc. Stjepanu Bogdanu na ukazanom povjerenju, nesebičnoj pomoći i potpori tijekom izrade ovog rada. Zahvaljujemo i Juraju Oršuliću, mag. ing. na uloženom trudu i požrtvovnosti u pružanju savjeta.

LITERATURA

[1] Onur Ozcan, Han Wang, Jonathan D. Taylor, Metin Sitti. STRIDE II: A water strider-inspired miniature robot with circular footpads. *International Journal of Advanced Robotic Systems*, 11(6):85, Siječanj 2014. doi: 10.5772/58701. URL https://doi.org/10.5772/58701.

[2] Robert Wood, Radhika Nagpal, Gu-Yeon Wei. Flight of the robobees. *Scientific American*, 308(3):60–65, Veljača 2013. doi: 10.1038/scientificamerican0313-60. URL https://doi.org/10.1038/scientificamerican0313-60.

[3] Martin Saska, Jan Vakula, Libor Preucil. Swarms of micro aerial vehicles stabilized under a visual relative localization. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, str. 3570 – 3575. IEEE, 2014. doi: 10.1109/icra.2014.6907374. URL https://ieeexplore.ieee.org/document/6907374/.

[4] R. Olfati-Saber. Flocking for multi-agent dynamic systems: Algorithms and theory. *IEEE Transactions on Automatic Control*, 51(3):401–420, Ožujak 2006. doi: 10.1109/tac.2005.864190. URL https://ieeexplore.ieee.org/document/1605401/.

[5] Dmitrii Pavlov A Kasumyan. Patterns and mechanisms of schooling behavior in fish: A review. *Journal of Ichthyology*, 40:S163–S231, Prosinac 2000.

[6] Frank Bonnet, Rob Mills, Martina Szopek, Sarah Schönwetter-Fuchs, José Halloy, Stjepan Bogdan, Luís Correia, Francesco Mondada, Thomas Schmickl. Robots mediating interactions between animals for interspecies collective behaviors. *Science Robotics*, 4(28), 2019. doi: 10.1126/scirobotics.aau7897. URL https://robotics.sciencemag.org/content/4/28/eaau7897.

[7] Daniel Shiffman. *The Nature of Code: Simulating Natural Systems with Processing*, poglavlje 6. The Nature of Code, 2012. URL http://natureofcode.com/book/.

[8] T. Balch R. C. Arkin. Behavior-based formation control for multirobot teams. *IEEE Transactions on Robotics and Automation*, 14(6):926–939, Prosinac 1998. doi: 10.1109/70.736776.

[9] S. Bogdan Z. Kovačić. *Upravljanje robotskim sustavima - predavanja*. Fakultet elektrotehnike i računarstva, Zagreb, 2009.

[10] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques - SIGGRAPH '87*. ACM Press, 1987. doi: 10.1145/37401.37406. URL https://www.red3d.com/cwr/papers/1987/boids.html.

[11] Craig W. Reynolds. Steering behaviors for autonomous characters. In *Games Developers Conference 1999*, str. 763–782, San Jose, California, 1999. Miller Freeman Game Group. URL https://www.red3d.com/cwr/papers/1999/gdc99steer.pdf.

[12] Peter Friederici. Flight plan. *Audubon*, 2009. URL https://www.audubon.org/magazine/march-april-2009/how-flock-birds-can-fly-and-move-together. Pristupano: 19.4.2019.

[13] Franke de Jong. Large flock of starlings. URL https://www.shutterstock.com/image-photo/large-flock-starlings-496212604. Pristupano: 19.4.2019.

[14] Yilun Shang Roland Bouffanais. Influence of the number of topologically interacting neighbors on swarm dynamics. *Scientific Reports*, 4(1), Veljača 2014. doi: 10.1038/srep04184. URL https://doi.org/10.1038/srep04184.

[15] George F. Young, Luca Scardovi, Andrea Cavagna, Irene Giardina, Naomi E. Leonard. Starling flock networks manage uncertainty in consensus at low cost. *PLoS Computational Biology*, 9(1):e1002894, Siječanj 2013. doi: 10.1371/journal.pcbi.1002894. URL https://doi.org/10.1371/journal.pcbi.1002894.

[16] Nathan Bell, Yizhou Yu, Peter J. Mucha. Particle-based simulation of granular materials. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation - SCA '05*. ACM Press, 2005. doi: 10.1145/1073368.1073379. URL https://doi.org/10.1145/1073368.1073379.

[17] Craig Reynolds. Big fast crowds on PS3. In *Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames - Sandbox '06*. ACM Press, 2006. doi: 10.1145/1183316.1183333. URL https://doi.org/10.1145/1183316.1183333.

[18] Wei Shao Demetri Terzopoulos. Autonomous pedestrians. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation - SCA '05*. ACM Press, 2005. doi: 10.1145/1073368.1073371. URL https://doi.org/10.1145/1073368.1073371.

[19] Rosario De Chiara, Ugo Erra, Vittorio Scarano, Maurizio Tatafiore. Massive simulation using gpu of a distributed behavioral model of a flock with obstacle avoidance. In *VMV*, 2004.

[20] Antonella Barišić, Marko Križmančić. Snimke rezultata projekta za Rektorovu nagradu, 2019. URL https://www.youtube.com/playlist?list=PLPiKSV7BqbsZcMZir4M2gN9oI02NTkEvi. Pristupano: 29.4.2019.

[21] Jose Maria Rodriguez Corral, Ivan Ruiz-Rube, Anton Civit Balcells, Jose Miguel Mota-Macias, Arturo Morgado-Estevez, Juan Manuel Dodero. A study on the suitability of visual languages for non-expert robot programmers. *IEEE Access*, 7:17535–17550, Siječanj 2019. doi: 10.1109/access.2019.2895913. URL https://doi.org/10.1109/access.2019.2895913.

[22] stage_ros wiki page, 2013. URL http://wiki.ros.org/stage_ros. Pristupano: 29.4.2019.

[23] The official website of bluetooth technology. URL https://www.bluetooth.com/. Pristupano: 29.4.2019.

[24] Dan Simon. *Optimal state estimation : Kalman, H [infinity] and nonlinear approaches*, poglavlje 5. Wiley-Interscience, Hoboken, N.J, 2006.

[25] Marko Križmančić. Repozitorij upravljačkog algoritma za Rektorovu nagradu, 2019. URL https://github.com/mkrizmancic/sphero_formation. Pristupano: 29.4.2019.

[26] Antonella Barišić. Repozitorij upravljačkog programa za Sphero robote, 2019. URL https://github.com/antonellabarisic/sphero_sprk_ros. Pristupano: 29.4.2019.

Antonella Barišić, Marko Križmančić

Decentralizirano upravljanje formacijom višeagentskog sustava autonomnih sfernih robota

Sažetak

U radu je implementirano decentralizirano upravljanje formacijom višeagentskog sustava autonomnih sfernih robota korištenjem Reynoldsovih pravila. Algoritam proceduralno proizvodi gibanje robota nalik gibanju jata ptica ili riba u prirodi te omogućava kretanje zatvorenim prostorom s preprekama. Razvijen je i upravljački program (engl. *driver*) za *Bluetooth* komunikaciju s robotima, a sve funkcionalnosti ostvarene su korištenjem ROS programskog okruženja i programskog jezika Python. Rezultati rada prezentirani su eksperimentima u simulaciji te u stvarnosti na Sphero SPRK+ robotima lokaliziranim pomoću sustava *OptiTrack*.

Ključne riječi: višeagentski sustavi, decentralizirano upravljanje, Reynoldsova pravila, upravljanje formacijom, Bluetooth komunikacija

Antonella Barišić, Marko Križmančić

Decentralized formation control for a multi-agent system of autonomous spherical robots

Summary

In this work, a decentralized control algorithm based on Reynolds' rules is implemented on a multi-agent system of spherical robots. The algorithm procedurally generates motion patterns that resemble those characteristic for flocks of birds or schools of fish. Generated motion patterns allow robots to move in a closed space with static obstacles. A Bluetooth driver for controlling the robots has also been developed. The complete system is implemented using the ROS framework and Python programming language. The results of this work are demonstrated with experiments in a simulated environment, as well as in the real world using Sphero SPRK+ robots localized with *OptiTrack*.

Keywords: multi-agent systems, decentralized control, Reynolds' rules, formation control, Bluetooth communication