Koreografija jata bespilotnih letjelica

Glavić Sanković, Paola

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 640

KOREOGRAFIJA JATA BESPILOTNIH LETJELICA

Paola Glavić Sanković

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 640

KOREOGRAFIJA JATA BESPILOTNIH LETJELICA

Paola Glavić Sanković

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 640

Pristupnica:	**Paola Glavić Sanković (0036522290)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentorica:	izv. prof. dr. sc. Tamara Petrović

Zadatak: **Koreografija jata bespilotnih letjelica**

Opis zadatka:

Jata bespilotnih letjelica koje izvode koreografije koriste se sve češće na manifestacijama zbog svoje atraktivnosti. Kako bi se osigurao siguran rad letjelica potrebno je osigurati slijeđenje koreografije bez međusobnog sudaranja. Radom na ovom diplomskom zadatku potrebno je razviti metodu za planiranje i izvođenje trajektorija za jato letjelica koja će osigurati sigurno kretanje prema zadanoj koreografiji. Potrebno je isprobati metodu temeljenu na konsenzusu te metodu s unaprijednim planiranjem te usporediti. Potrebno je uzeti u obzir statičke i dinamičke koreografije te osigurati jednostavno zadavanje različitih koreografija. Za lokalizaciju koristiti Optitrack ili Crazyflie LoCo sustav te isprobati u simulaciji te na stvarnom sustavu s deset Crazyflie robota.

Rok za predaju rada: 28. lipnja 2024.

Sadržaj

1. Uvod

Bespilotna letjelica (engl. *Unmanned aerial vehicle – UAV*), poznatija u svakodnevnom rječniku kao *dron* zrakoplovno je vozilo bez posade koje se izumilo za vojne namjere (npr. izviđanja, napadi), ali njihova uporaba u civilnoj sferi društva sve je veća i popularnija pa se tako koriste u komercijalne (npr. dostava, snimanje terena) i umjetničke svrhe (npr. ples letjelica korišten na smotrama i manifestacijama, fotografiranje iz zraka i sl.) [1][2]. Takvim letjelicama može se upravljati direktno koristeći daljinski upravljač, koreografiranim unaprijed isplaniranim putanjama leta ili složenijim sustavima koje izrađuju plan kretanja u stvarnom vremenu.

Ovaj diplomski rad bavit će se već spomenutom koreografijom jata bespilotnih letjelica te je nastavak na istoimeni završni rad [4]. Koreografija jata može koristiti metodu unaprijednog planiranja gdje je potrebno zadati početne i krajnje pozicije u prostoru, dok algoritam isplanira sve ostalo, kao što je bila riječ u završnom radu ili letenje u stvarnom vremenu gdje je potrebno zadati samo formacije koje letjelice moraju *otplesati* i u kojoj moraju ostati leteći, a sve ostalo računa se dinamički kako koreografija napreduje. Ovakav kasniji sustav, koji će se detaljnije obraditi i izvesti u ovom radu, bolji je u nepredvidljivijim situacijama i vanjskim prostorima.

Jedan od takav sustava naziva se konsenzus algoritam (ili protokol) u kojemu agenti međusobnom komunikacijom nastoje postići dogovor (konsenzus) na temelju određenih zajedničkih interesa (npr. nesudaranje, letenje u nekom smjeru itd.) koji direktno ovise o njihovim trenutnim stanjima [6].

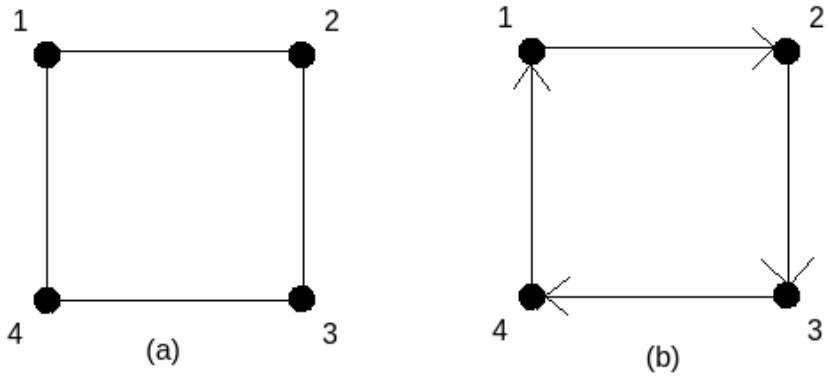
2. Konsenzus protokol

Za dinamičko koreografiranje bespilotnih letjelica koristit će se konsenzus protokoli koji imaju svoje uporište u teoriji grafova i teoriji sustava [5]. Takvi protokoli, želeći postići dogovor (konsenzus), omogućavaju komunikaciju među agentima predstavljenu preko matrice susjedstva koja prikazuje koji agent je povezan s kojim, tj. koji agent će kojemu slati podatke o svojem položaju, translaciji i sl.. Pomoću takvih veza i dijeljenja informacija letjelice postižu željeni oblik ili pak lete dok održavaju određenu formaciju (eng. *flocking*) uz bitan uvjet nesudaranja. Takvi algoritmi mogu se koristiti i za druge primjene, primjerice, za algoritam sastajanja, gdje se sve letjelice nalaze u jednoj točki.

2.1. Grafovi

Konsenzus protokol temelji se na interakciji među agentima pa je tako potrebno definirati komunikacijska pravila. Topologija komunikacijske mreže, koja definira koji agenti međusobno komuniciraju, definirana je na skupu od *n* agenata te modelirana grafom $G_n = (V_n, E_n)$, koji ima *n* vrhova $V_n = \{1, 2, ... n\}$, gdje vrhovi predstavljaju letjelice, a bridovi E_n, definirani na prostoru $V_n \, x \, V_n$, komunikaciju među njima.

Graf može biti neusmjeren i usmjeren. Kod neusmjerenog grafa kao što je lijevi *graf (a)* sa *Slike 1* brid označava slanje informacija u oba smjera, dakle neka dva agenta *i* i *j* povezani bridom međusobno komuniciraju, npr. na *Slici 1* agent *1* prima podatke od agenta *2* i obrnuto. Kod usmjerenog grafa, također prikazanog na *Slici 1* pod slovom *(b),* podatci putuju samo u smjeru kojeg naznačuje strelica, npr. agent *2* dobiva direktno informacije od agenta *1*, dok agent *1* informacije o stanju agenta *2* dobiva eventualno preko ostala dva agenta, ali u kasnijim iteracijama konsenzus protokola – naravno, to vrijedi samo za ovu prikazanu topologiju. U slučaju nepovezanosti npr. Agenta *3* i *4* takav tok informacija se ne bi dogodio - više riječi o posljedicama toga i značaju postojanja *toka* kasnije.



Slika 1. primjer neusmjerenog (a) i usmjerenog (b) grafa

Graf se jednostavno može zapisati matricom susjedstva *A* dimenzija je *n x n*, gdje je *n* broj agenata, a svaki redak *i* predstavlja susjedstvo *i*-tog agenta. Naravno, takva matrica imat će samo nule na dijagonali jer agent sam sebi nije susjed. Nule će postojati i na mjestima gdje *i-ti* agent ne prima informacije od *j-tog* agenta, dakle nisu povezani (usmjerenim) bridom. Sažeto zapisano

$$A = [a_{ij}] \in \mathbb{R}^{nxn} \quad \begin{cases} a_{ij} > 0 & i,j \in E_n \\ a_{ij} = 0 & inače \end{cases} \tag{1}$$

gdje a_{ij} predstavlja težinu brida *(j, i)*. U slučaju da težina nije važna a_{ij} iznosi *1*. Za primjer, matrica susjedstva za usmjereni graf *(b)* sa *Slike 1* izgledala bi

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2}$$

Grafovi još mogu i biti balansirani, što po definiciji znači da vrijedi

$$\sum_{j=1}^{n} a_{ij} = \sum_{i=1}^{n} a_{ji}, \quad \forall\, i \in V_n \tag{3}$$

2.2. Jednadžbe konsenzus protokola

Osnovni zapis konsenzus protokola na koji se kasnije mogu dodavati uvjeti održavanja formacija, nesudaranja i sl. glasi:

$$\dot{x_\iota}(t) = \sum_{j=1}^{n} a_{ij}(t)\big[x_j(t) - x_i(t)\big], \;\; i = 1, 2, \dots n \tag{4}$$

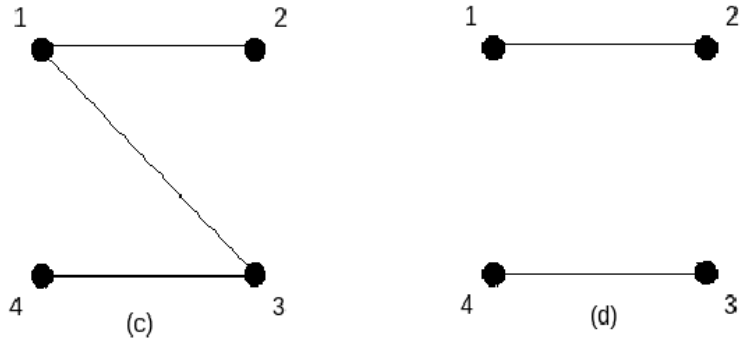
Gdje je dinamika jednog integratora $\dot{x} = u$, a a_{ij} su elementi već prije spomenute matrice susjedstva koja odgovara grafu G_n. Konsenzus je postignut kada vrijedi

$$\big|x_j(t) - x_i(t)\big| \to 0 \;\; t \to \infty, \qquad \forall x_i(0)\; i\; \forall i, j \;\in \{1, 2, .. n\} \tag{5}$$

Tj. kada se svi agenti (koji su povezani) međusobno dogovore oko vrijednosti x.

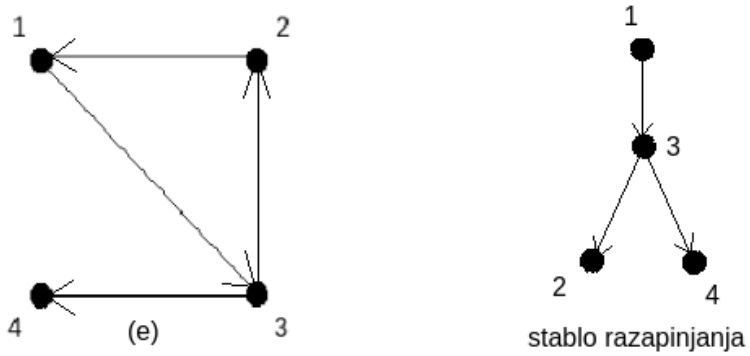
2.3. Uvjeti konvergencije

Agenti konsenzus protokolom izmjenjuju informacije da bi ostvarili nekakav dogovor tj. zajednički cilj, no da bi ga postigli, takav protokol (algoritam) mora konvergirati. Postoje različiti uvjeti ovisno o vrsti grafa [1]. Topologija neusmjerenog grafa konvergirat će ako i samo ako je graf povezan. Graf je povezan ako postoji put između svaka dva čvora (agenta). Neusmjereni graf *(a)* sa *Slike 1* je povezan graf, iz svakog čvora može se doći do ostala tri, dakle za takvu topologiju algoritam će konvergirati. *Slika 2* prikazuje još dva primjera organizacije komunikacija među čvorovima u neusmjerenim grafovima. Graf *(c)* sa *Slike 2,* unatoč jednom bridu manje od već spomenutog grafa *(a)*, također će postići konsenzus jer je povezan. Graf *(d),* sa iste te slike, nema puteve od čvorova *1* i *2* do čvorova *3* i *4*, čime se zaključuje da je graf nepovezan pa se takvom topologijom ne može doći do konsenzusa – algoritam ne konvergira.



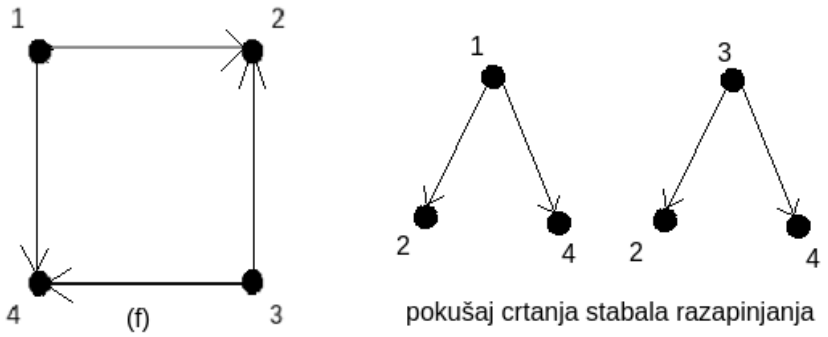
Slika 2. primjer neusmjerenih grafova (c) i (d)

Za razliku od neusmjerenog grafa, za topologiju usmjerenog grafa algoritam će konverigrati samo i samo ako u njemu postoji usmjereno stablo razapinjanja koje je po svojoj definiciji podgraf (V_n^s, E_n^s) grafa (V_n, E_n). Naravno, nije svaki podgraf stablo razapinjanja. Da bi podgraf bio stablo razapinjanja mora imati isti broj vrhova kao originalni graf, tj. mora vrijediti $V_n^s = V_n$, dodatno, barem jedan čvor mora imati usmjeren put do svih ostalih čvorova. Usmjereni graf *(b)* sa *Slike 1* ima stablo razapinjanja, sa čvorom *1* kao početnim čvorom, koji ima put do svih ostalih (1 -> 2 -> 3 -> 4), što znači da algoritam sa ovakvom mrežom komunikacije dolazi do konsenzusa. *Slika 3* prikazuje još jedan primjer usmjerenog grafa za čiju će strukturu mreže algoritam konvergirati što se može zaključiti iz priloženog stabla razapinjanja desno od grafa – za odabrani početni čvor *1* može se doći do ostalih čvorova.



Slika 3. usmjereni graf (e) koji ima stablo razapinjanja

Na *Slici 4* se nalazi usmjereni graf *(f)* čija struktura nema ni jedno stablo razapinjanja – biranjem čvora *1* kao početnog čvora može se doći samo do čvorova *2* i *4,* ali ne i *3*, što se vidi na prvom podgrafu desno od grafa. Isto se dogodi ako se krene graditi stablo iz čvora *3*, također vidljivo na slici kao drugi podgraf. U slučaju izabiranja čvora *2* ili *4* za korijen također se ne može izgraditi stablo jer se iz njih ne može doći ni do jednog čvora. Iz ovoga se zaključuje da graf *(f)* nema mrežnu topologiju koja omogućuje konvergiranje algoritma.



Slika 4. Slika 4. - usmjereni graf (f) koji nema stablo razapinjanja

2.4. Primjena na problemu sastajanja

Najjednostavniji primjer primjene konsenzus protokola je na rješavanju tzv. problema sastanka (engl. *rendezvous problem*). Agenti tada imaju jedan cilj – da se svi nađu u jednoj nepoznatoj točki. Da bi to postigli, agenti razmjenjuju informacije o svom položaju *r* u prostoru te primjenjuju konsenzus protokol. Algoritam razmjene podataka, sličan izrazu (4) onda glasi

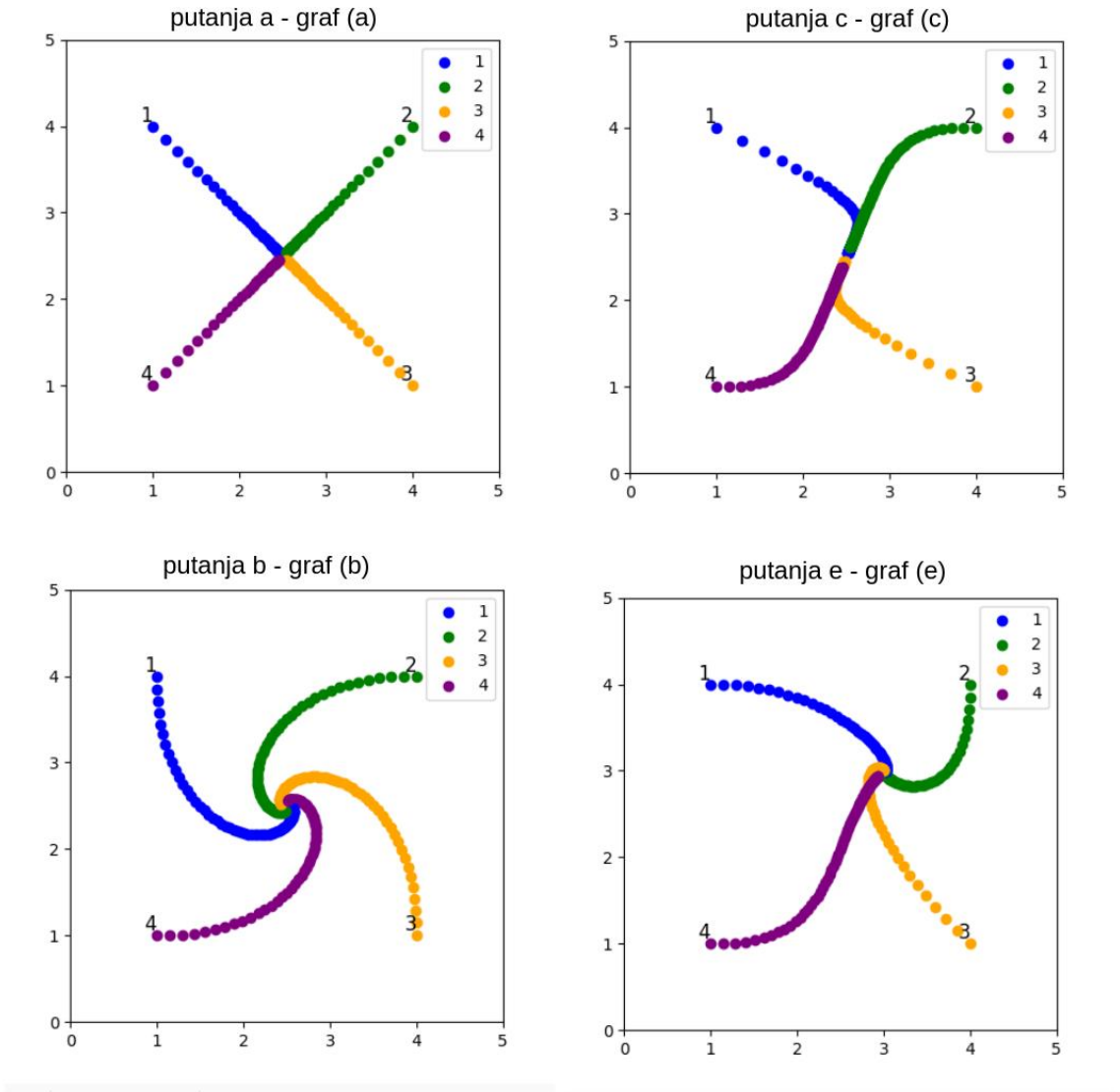
$$\dot{r}_i(t) = \sum_{j=1}^{n} a_{ij} \left(r_j(t) - r_i(t) \right) \tag{6}$$

Gdje je $r_i = [x_i, y_i]^T \in \mathbb{R}^2$, a a_{ij} su elemetni već spomenute matrice susjedstva. Prostor je radi jednostavnosti dvodimenzionalan, ali isto vrijedi i za veće dimenzije. U svakoj iteraciji algoritma dobiju se brzine ($\dot{r}_i$) iz kojih se na klasičan način računa sljedeća pozicija kao

$$r_i(t + dt) = \dot{r}_i(t) \cdot dt \tag{7}$$

Gdje je *dt* konstantni vremenski korak.

Konvergira li algoritam konsenzusa ili ne za zadanu matricu susjedstva tj. za neku komunikacijsku mrežu najlakše je pokazati upravo na problemu sastajanja – ako su se svi agenti našli u istoj točki očito je da je njihov cilj postignut, dakle algoritam je konvergirao. Već je rečeno da će za grafove sa *Slike 1* (*(a)* i *(b)*), graf *(c)* sa *Slike 2* i graf *(e)* sa *Slike 3* algoritam konvergirati. Ubacivanjem matrica susjedstva navedenih grafova u algoritam opisan izrazom (5) dobiju se putanje četiriju agenata (prvobitno smještenih u formaciju kvadrata) na *Slici 5*.

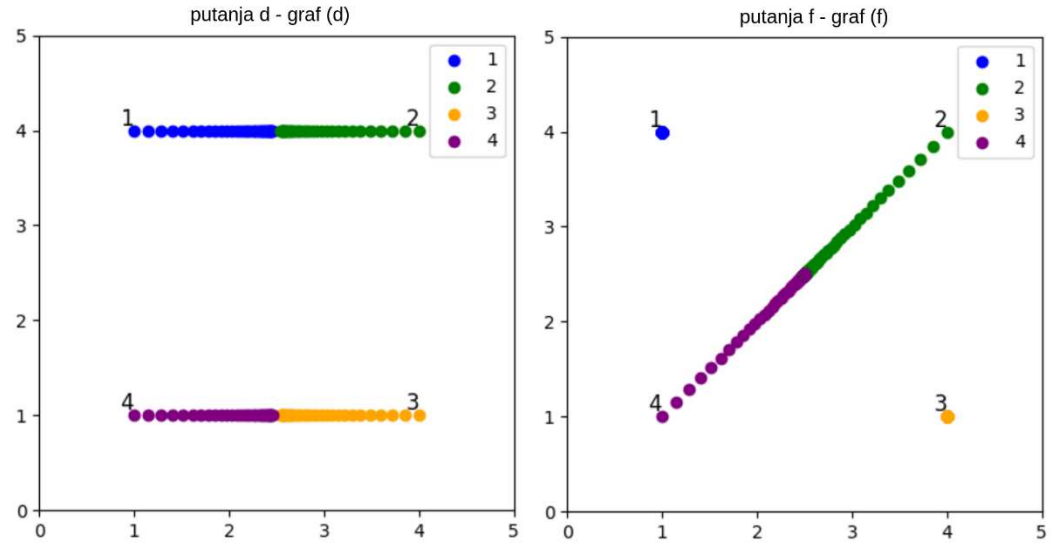


Slika 5. Putanje agenata za komunikacijske topologije koje omogućuju konvergenciju

Jasno vidi da su se sva četiri agenta našla u jednoj zajedničkoj točki za komunikacijske strukture definirane grafovima *(a), (b), (c)* i *(e)*, tj. za one strukture za koje je i očekivano konvergiranje algoritma. Naravno, sa *Slike 5* se još može i uočiti da o mjestu sastajanja i izgledu svih putanja kretanja odlučuje topologija matrice susjedstva tj. dizajn grafa koji prikazuje pravila komuniciranja među agentima. Tako će se, kad agent prima informacije o svojim susjedima (graf *(a), Slika 1*) svi gibati pravocrtno (putanje *a, Slika 5*), dok kad prima informacije od samo jednog susjeda desno sebi (graf *(b), Slika 1*, graf *(c), Slika 2*) putanja agenata nalikuje svrdlu (putanje *b* i prva tri agenta u Slici 5 putanja *e* - agent *4* pridružit će se mjestu sastanka jer prima podatke o agentu *3*).

Izraz (4) može se još pomnožiti koeficijentom *k* koja skaliranjem brzina omogućuje kontroliranije gibanje i glađe putanje. Sve putanje sa *Slike 5* koriste isti *k* iznosa *0.5* i isti fiksirani vremenski korak *dt = 0.1*. Ako se, za primjer, uzme samo putanja agenta *1* (označen plavom bojom, *Slika 5*), može se lako uočiti da negdje napravi kraći put (putanja *a*), a negdje dulji put (putanja *b*). Time se, iako je poprilično očito, da zaključiti da za različite matrice susjedstva (grafove) postizanje konsenzusa različito traje, pa se, osim pažnje na konvergenciju, u slučaju većeg broja agenata, treba skrenuti i pažnja na samu konstrukciju matrice.

Agenti koji svoju komunikaciju definiraju grafom *(d)* sa *Slike 2* i grafom *(f)* sa *Slike 4* neće se naći u istoj zajedničkoj točki jer koriste topologije za koje, kao što je već objašnjeno, algoritam neće konvergirati. *Slika 6* upravo to i prikazuje. mirovanje ostalih agenata, razlog tomu opet leži u izgledu grafa *(f)* sa *Slike 4* - agenti *1* i *3* ne primaju informacije od nikog (pa stoje na mjestu), ali ih zato šalju agentima *2* i *4* koji se onda kreću po dijagonali kvadrata, kojeg čini početna formacija agenata, nastojeći tako biti jednako blizu i agentu *1* i agentu *3* time se sudarajući jedan s drugim.



Slika 6. Putanje agenata za komunikacijske topologije koje ne omogućuju konvergenciju

Za razliku od putanja sa *Slike 5*, putanje sa *Slike 6* ne koriste matrice susjedstva koje dovode do konvergencije algoritma.

3. Izvođenje koreografija primjenom konsenzus protokola

Konsenzus protokol, osim za već spomenuti problem sastajanja, primjenjuje se i za postavljanje agenata u određenu formaciju, mijenjanje formacija i njihovo kretanje u prostoru. S obzirom da je koreografija bespilotnih letjelica ništa drugo nego izmjena niza različitih formacija u vremenu te njihovo pomicanje, u ovom radu koristi se upravo konsenzus protokol.

3.1. Definiranje formacije

Prije samog leta tj. pokretanja algoritma potrebno je osmisliti formacije koje letjelice trebaju odraditi. Takve formacije trebaju zadovoljiti određene kriterije. Jedan od uvjeta je da neka formacija *D* mora biti izvediva tj. Mora vrijediti

$$D = \{d_{ij} \in \mathbb{R} \,|\, d_{ij} > 0, , i, j = 1, \dots, n \;\; i \neq j\} \tag{8}$$

Takvi da

$$. \exists\, \xi_1, \dots, \xi_n \in \mathbb{R}^n;\; \|\xi_i - \xi_j\| = d_{ij} \,\forall\, ij \tag{9}$$

Gdje je d_{ij} udaljenost između agenta *i* i *j*, a ξ_i željena pozicija agenta *i* u formaciji. Uz sve to formacija još može biti invarijantna na skaliranje bilo kojim realnim brojem *a*

$$D' = aD \tag{10}$$

Tako da vrijedi

$$x_i = \xi_i + \tau \tag{12}$$

x_i je naravno trenutna pozicija *i*-tog agenta pa je tako translacija zapravo razlika između nje i željene pozicije Formacija ne mora biti i nije rotacijski invarijantna.

3.2. Postizanje formacije

Za formaciju zadanu izrazom (11), uz prethodno spomenute uvjete može se definirati izraz (13) za translacijsku varijablu τ agenta i koji slijedi iz izraza (12):

$$\tau_i(t) = x_i(t) - \xi_i(t) \tag{13}$$

S obzirom da se neka početna formacija nastoji približiti željenoj formaciji (11), translacijska varijabla se, za svakog agenta mijenja kroz vrijeme. Cilj je da se svi agenti slože, tj. postignu *konsenzus* oko vrijednosti translacije odnosno $\tau_i(t) = \tau_j(t)\ \forall\ i, j$. To znači da, u ovom slučaju uobičajena jednadžba konsenzus algoritma (1), sada glasi:

$$\dot{\tau}(t) = \sum_{j=1}^{n} a_{ij}\left(\tau_j(t) - \tau_i(t)\right) \tag{14}$$

Ovakav izraz omogućuje postizanje i ostajanje letjelica u formaciji. Lijeva strana jednadžbe (14) može se zamijeniti desnom stranom izraza (13) ako se on derivira pa slijedi

$$\dot{x_\iota}(t) - \dot{\xi_\iota} = \sum_{j=1}^{n} a_{ij}\left(\tau_j(t) - \tau_i(t)\right) \tag{15}$$

Raspisivanjem $\tau_j(t)$ i $\tau_i(t)$ potom se dobije

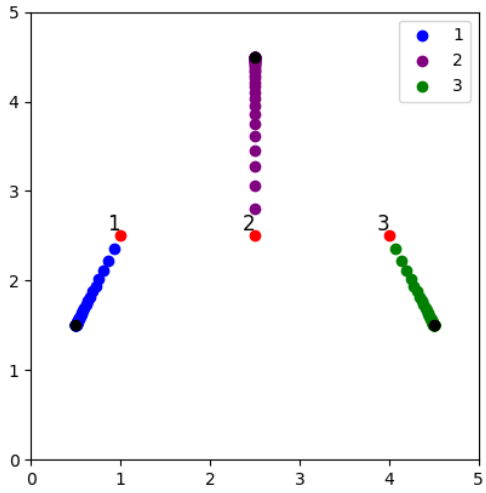
$$\dot{x_\iota}(t) - \dot{\xi_\iota} = \sum_{j=1}^{n} a_{ij}\left(\left(x_j(t) - \xi_j(t)\right) - \left(x_i(t) - \xi_i(t)\right)\right) \tag{16}$$

Izraz se još dodatno može preurediti uz zanemarivanje $\dot{\xi_\iota}$ jer on slabo varira (ako se izmjenjuju formacije) ili je iznosa 0 (ako se želi postići samo jedna formacija) pa se tako dolazi do konačnog izraza:

$$\dot{x_\iota}(t) = \sum_{j=1}^{n} a_{ij}\left(\left(x_j(t) - x_i(t)\right) - \left(\xi_j(t) - \xi_i(t)\right)\right) \tag{17}$$

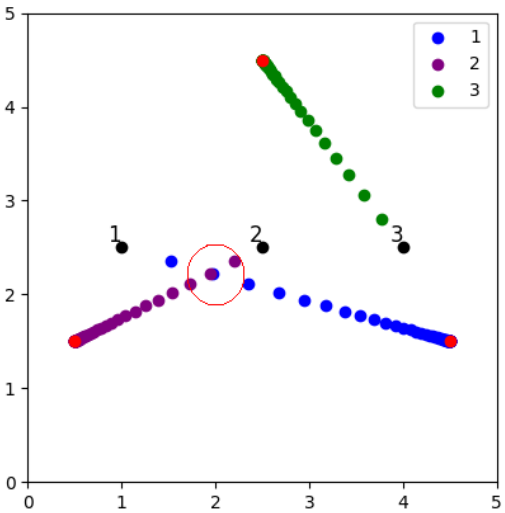
Ovakav izraz (17) izveden iz početnog izraza (14) koristi se kao konsenzus algoritam za postizanje formacija. Implementacija algoritma temeljenog na izrazu (17), nalazi se u prilogu u skripti *postizanje_formacije.py*, u metodi *calculate_dx*. Njenim pokretanjem može se, na *Slici 7* vidjeti primjer putanja tri agenta. Agenti u početku stoje u liniji (startne pozicije označene brojem agenta i crnim točkama), a kroz vrijeme napreduju i oblikuju formaciju

trokuta, na slici označen crvenim točkama. Algoritam koristi anti-identitetske matrice susjedstva (svugdje jedinice, osim na dijagonali, gdje su nule).



Slika 7. Koreografija 1 (početni položaji označeni crnim, a krajnji crvenim točkama)

Ono što ovaj algoritam ne osigurava je nesudaranje agenata, već samo postizanje formacija. Primjer za to može se vidjeti na *Slici 8* koja uzima istu koreografiju kao primjer sa *Slike 7* (iste početne pozicije i istu formaciju), ali zamjenjuje ciljane pozicije u formaciji tj. agent *2* više nije gornji vrh, već lijevi, dok je agent *1* sad desni vrh umjesto lijevi, drugim riječima, formacija sa *Slike 7*, $\Xi = \{\xi_1, \xi_2, \xi_3\}$, na *Slici 8* je preoblikovana u $\Xi = \{\xi_3, \xi_1, \xi_2\}$. Takvom malom i naizgled nebitnom modifikacijom događa se sudar između agenata *1* i *2*, na *Slici 8* zaokružen crvenim kružićem.



Slika 8. Koreografija 2, sa sudarom (početni položaji označeni crnim, a krajnji crvenim točkama)

3.3. Izbjegavanje sudara

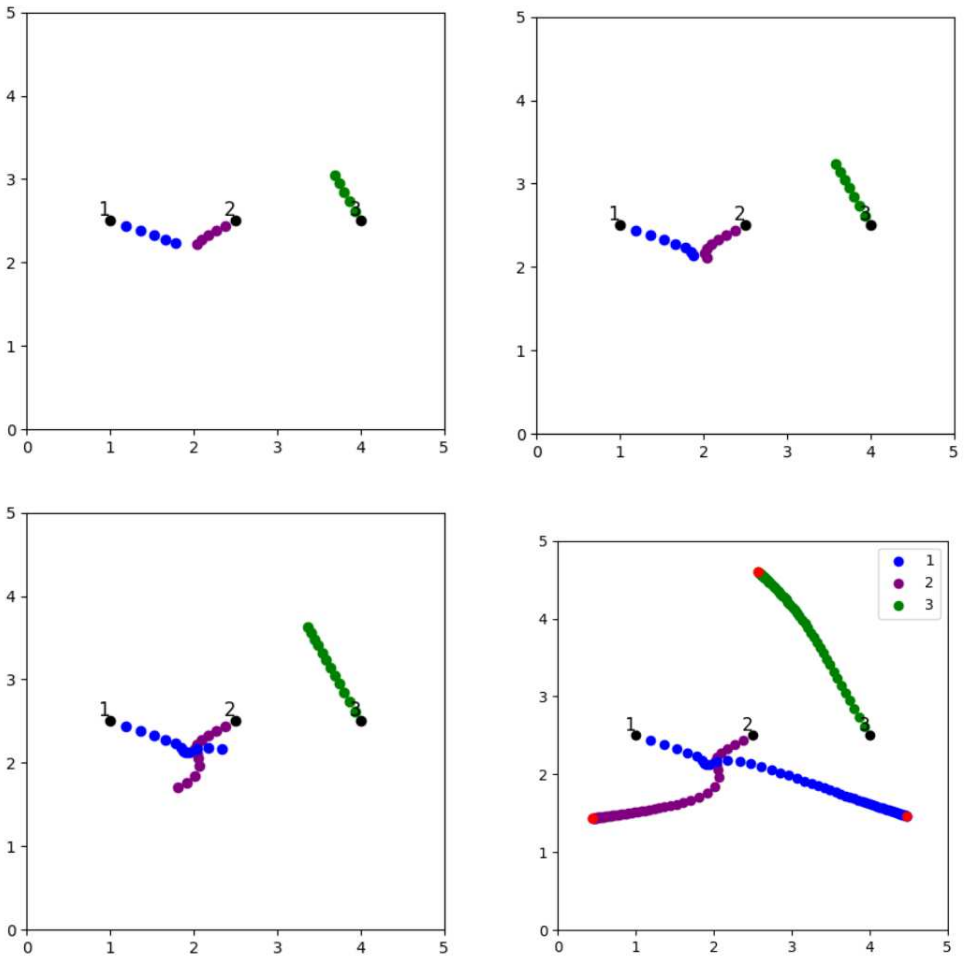
Bitan i nezanemariv zahtjev kod koreografiranja bespilotnih letjelica je osigurati izbjegavanje sudara. Korištenjem običnog konsenzus protokola (17), kao što je pokazano u prijašnjem potpoglavlju, jamči se samo dovođenje agenata u formaciju, ali ne i njihov neometan let bez sudara (*Slika 8*). Kako bi se postigao rad bez kolizija, potrebno je uvesti neku minimalnu udaljenost koja definira koliko se agent smije približiti ostalima. Ideja je da svaki agent djeluje odbojnom silom na svoje susjede. Bliži agenti odbijaju se međusobno jače, a dalji manje, otuda potreba za korištenjem kvadratnog ili kubičnog, a ne linearnog izraza. Separacijska komponenta, koja će biti uključena u konsenzus protokol, glasi:

$$c\sum_{j=1}^{n}\frac{(x_j - x_i)}{\|x_j - x_i\|^2} \tag{18}$$

Gdje je c varijabla koja definira bitnost separacije, veći c znači i veću udaljenost među agentima. Sada, kako bi se riješio problem kolizija, separacija (18) se oduzima od sume u konsenzus protokolu, pa se tako izraz (17) transformira u

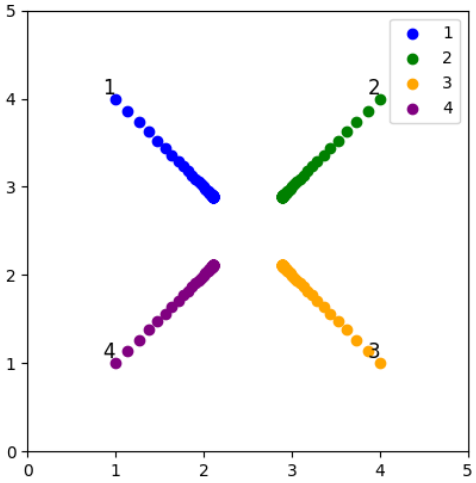
$$\dot{x_i}(t) = k\sum_{j=1}^{n} a_{ij}\Big(\big(x_j(t) - x_i(t)\big) - \big(\xi_j(t) - \xi_i(t)\big)\Big) - c\sum_{j=1}^{n}\frac{(x_j - x_i)}{\|x_j - x_i\|^2} \tag{19}$$

Jednadžba (19) je konačni oblik konsenzus protokola za postizanje formacija koji osigurava neometanu koreografiju bez sudara. Implementacija takvog izraza također se nalazi u skripti *postizanje_formacija.py*. Kod koji odgovara algoritmu (19) nalazi se u metodi *calculate_dx_sep (github).* Pokretanjem *postizanje_formacije.py* na istoj koreografiji kao sa *Slike 8*, ali ovog puta sa metodom koja sprječava kolizije, dobije se prikaz putanja na *Slici 9*. Koeficijen*t k* ovog je puta smanjen sa *0.5* na 0.1. Na *Slici 9* vidi se, kroz četiri različita trenutka u vremenu, simulacija leta. Na prvoj slici agent *1* i *2* lete jedan prema drugome u susret želeći doći u svoj ciljni položaj. Na slici pokraj (desno, gore) vidi se kako se oba agenta okreću jedan od drugoga nastojeći se zaobići. Treća slika prikazuje daljnji napredak, gdje su se agenti, koji su se na *Slici 8* sudarili, ovdje uspješno mimoišli. Konačno, zadnja slika (desno, dolje) pokazuje agente u ciljnoj formaciji (označeno crvenim točkama).



Slika 9. Koreografija 2 bez sudara

Izbjegavanje sudara korištenjem separacije možda nije lako uočljivo na *Slici 9*, pa se tako na *Slici 10* nalazi prikaz uključivanja separacijske varijable na randevu algoritmu iz koda *randevu_alg.py*. Slika prikazuje četiri agenta koji koriste matricu susjedstva opisanu grafom *(a)* sa *Slike 1*, što znači da je ovo koreografija istovjetna onoj sa *Slike 5* (lijevi, gornji kut), ali uz izbjegavanje kolizije, pa se tako agenti, umjesto da se sastanu (kao na *Slici 5*), nikad neće naći u jednoj točki, već će ostati stajati na nekoj udaljenosti.



Slika 10. algoritam sastajanja, uz separaciju

3.4. Koreografija

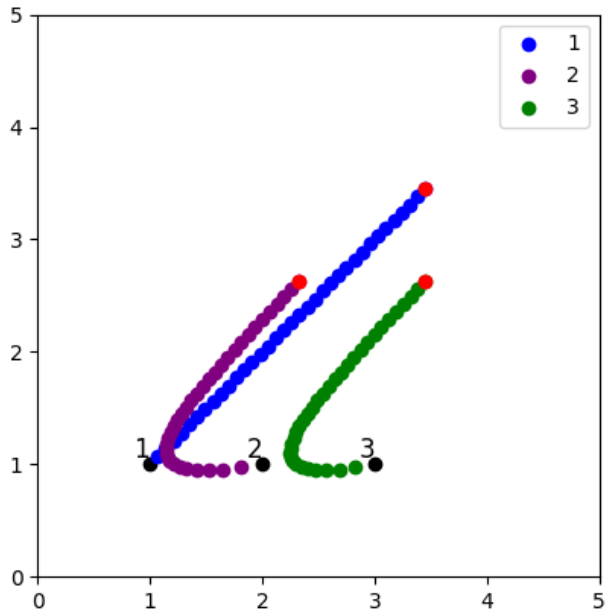
Koreografija letjelica podrazumijeva mijenjanje formacija agenata ili pomicanje formacija kroz prostor. Jednom kada agenti postignu formaciju Ξ_1, vrlo se lako može zadati nova formacija Ξ_2, ili niz formacija $[\Xi_2, \dots, \Xi_n]$ koje agenti moraju izvesti. Izmjena formacija može se događati po isteku određenog perioda vremena ili nakon ispunjavanja određenog uvjeta. Izmjena formacije ostvaruje se na način da se u izrazu (17) ili (19) ξ iz Ξ_1 zamijene pozicijama iz željene formacije Ξ_i. Detaljniji prikaz izmjene formacija obradit će se u slijedećem poglavlju. Drugu način za stvaranje koreografije je pomicanje letjelica u prostoru dok su u nekoj formaciji ili dok ih mijenjaju, što je omogućeno tzv. upravljanjem pomoću pričvršćivanja (engl. *pinning control*).

3.4.1. Pomicanje formacije kontrolom pričvršćenja

Skup letjelica može se pomicati kroz prostor primjenom upravljanja pomoću pričvršćivanja. Takvom metodom odabere se jedan virtualni voditelj (najčešće prvi agent) koji će i dalje, kao do sada, slati svoje podatke o položaju susjedima, ali ih neće primati tj. jedini on neće provoditi konsenzus protokol. Ostale letjelice koje su susjedi voditelja na algoritam konsenzusa (1) dodaju izraz za pričvršćivanje pa se tako dobiva

$$\dot{x}_i(t) = \sum_{j=1}^{n} a_{ij}\Big(\big(x_j(t) - x_i(t)\big)\Big) + g_i(x_0 - x_i) \quad i = 1,..,n \tag{20}$$

Gdje je g_i dobitak pričvršćivanja koji je, ako se radi o susjedu veći od nule. Pomicanje cijele formacije sada se ostvaruje samo pomicanjem voditelja kojem se može, primjerice, zadati da se giba po nekoj unaprijed definiranoj putanji. S obzirom da je voditelj *učvršćen* u formaciju konsenzus protokolom, jer svi ostali i dalje primaju informacije od njega, kako se on pomiče, svi ostali agenti pomicat će se za njim nastojeći ostati u zadanoj formaciji tj. nastojeći održati konsenzus kojeg imaju među sobom, a za kojeg voditelj agent ne zna. Primjer takve koreografije može se vidjeti na *Slici 11* gdje su agenti u početku postavljeni u liniju (crne točke). Agenti *2* i *3* pokušavaju oblikovati trokut zajedno sa agentom *1*, koji ne obavlja konsenzus protokol već se kreće u smjeru vektora $v = [0.8, 0.8]$. Agenti *2* i *3* isprva skreću lagano lijevo i dolje, jer je agent *1* tamo, znajući da on treba biti "njihov" gornji vrh trokuta, ali kako agent *1* s vremenom "bježi" prema gore, ostali agenti mijenjaju putanju i kreću za njim, prateći ga u nadi oblikovanja zadane formaciji, što se na kraju i dogodi – svi agenti oblikuju trokut na kraju (crvene točkice), sa voditeljem kao srednjim vrhom.



Slika 11. koreografija sa kontrolom pričvršćivanja

4. Osnovna simulacija

Sve *Python* skripte do sada navedene bile su jednostavne implementacije konsenzus algoritma te su služile samo za prikaz ponašanja agenata, njegovu vizualizaciju, ali ne i simulaciju. Takve skripte računale su nove pozicije agenata u svakoj iteraciji, spremale u strukturu polja, pa ih crtale korištenjem *matplotlib.pyplot* knjižnice, što znači da "stvarnih" agenata u njima nema. Za osnovnu simulaciju u ovom radu, koja će sada naravno, imati agente, koristit će se *sphero_stage*. *Sphero_stage* je okruženje (osigurano od strane predmeta *Višerobotski sustavi* na FER-u) koje sadrži datoteke za konfiguraciju i pokretanje simulatora koji vizualizira dvodimenzionalni prostor i *n* broj agenata u njemu s kojima je moguće upravljati [8].

4.1. Okruženje, instalacija i okruženje

Prije preuzimanja *sphero_stage* simulatora potrebno je imati odgovarajući operacijski sustav, programski jezik i okvir za upravljanje. Simulator zahtjeva *ROS Noetic*, što je *open-source* okvir za kreiranje, programiranje i upravljanje robotima. *ROS Noetic* se može direktno instalirati na računalo, ali je potrebno biti na odabranom Ubuntu operacijskom sustavu, ili se pokrenuti u *Docker* spremniku tako da ne ometa rad neke druge verzije ROS-a, ili ako je operacijski sustav nekompatibilan. Ovaj rad rađen je na operacijskom sustavu *Ubuntu 22.04 Yammy Jellyfish* koji ne podržava *ROS Noetic*, pa je on instaliran tj. Pokrenut preko *Dockera*. Naravno, prvo je bilo potrebno podesiti *Docker* točnije, *Docker engine*, platformu za automatiziranje implementacije i upravljanje aplikacijama na način da ih pakira u spremnike (*Docker containers*) koji se mogu pokrenuti onda u raznim okruženjima [9]. U slučaju da računalo na kojem se želi postaviti *Docker engine* ima *NVIDIA*-inu grafičku karticu, kao što računalo na kojem je ovo izvedeno ima, potrebno je prvo instalirati *NVIDIA Container Toolkit* preko uputa sa službene stranice [11]. Potom se može podesiti *Docker engine*, također prateći upute sa službenih stranica [9]. Jednom kad je to gotovo, github repozitororij stranice predmeta *Višerobotski sustavi* pruža ostale korake podešavanja *Docker* okruženja i spremnika u kojem će se moći koristiti *ROS Noetic*. Konačno, jednom kad se pokrene *container* u njemu se može preuzeti *sphero_stage* paket sa repozitorija [8]. Ukratko, koraci namještanja glase

1. Preuzimanje NVIDIA Container Toolkit-a (ako računalo ima NVIDIA-inu grafičku karticu) i namještanje NVIDIA-inih driver-a
2. Preuzimanje Docker-a (može i samo Docker engine-a) *https://docs.docker.com/engine/install/ubuntu/*
3. Praćenje ostalih uputa za namještanje ROS Noetic containera sa *https://github.com/larics/mrs_course*
4. Za vrijeme rada containera, preuzeti, u njemu, sphero_stage simulator naredbom "git clone https://github.com/larics/sphero_simulation.git"
5. Svaki se ROS paket treba izgraditi pa je potrebno nakon preuzimanja izvesti naredbu "catkin build"
6. Zbog lakšeg rada, može se još i instalirati tmuxinator, alat za pokretanje i upravljanje više terminala (https://github.com/tmuxinator/tmuxinator)

S obzirom da Ubuntu OS dolazi sa *Python* programskim jezikom njega nije potrebno podešavati. Dodatno, radi lakšeg snalaženja u okruženju *Visual Studio Code,* ako je instaliran na računalu, može se dodati ekspanzija za rad sa containerima [10].

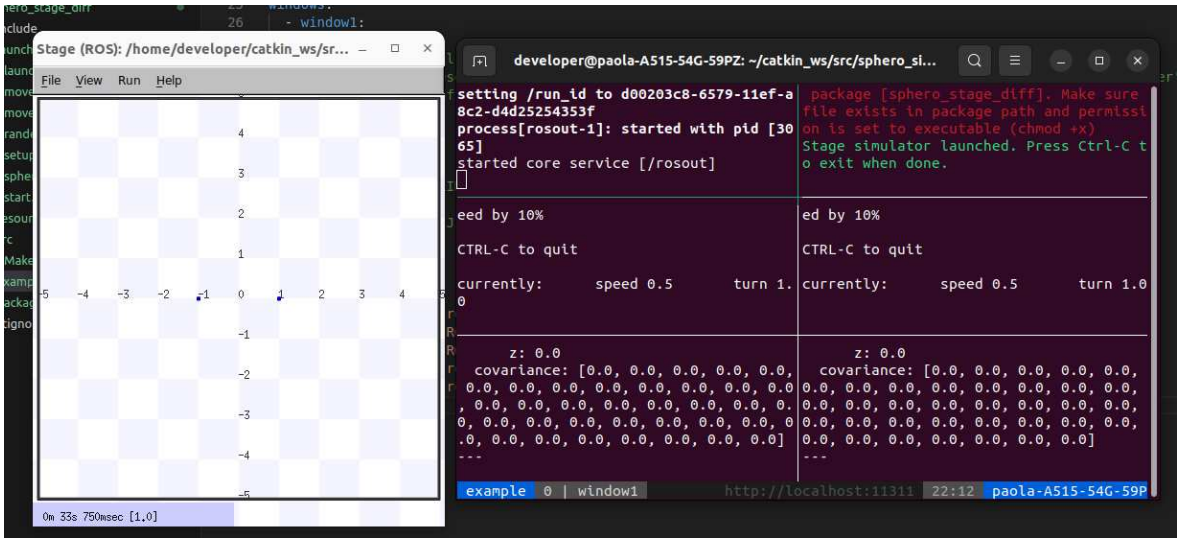
4.2. Opis i prikaz sphero_stage-a

Prije pokretanja sphero_stage-a potrebno je smjestiti se u mapu projekta i pokrenuti container naredbom "*(sudo) docker start -i <ime containera>*" te se zatim opet pozicionirati u okruženje simulatora "r*oscd sphero_stage*" naredbom. U launch mapi *sphero_stage*-a nalazi se yaml datoteka *launch_params.yaml* u kojoj se može birati izgled prostora (koji se nalaze u *resources/maps* mapi), broj agenata i njihova početna formacija – vodoravna linija ili krug. Sve te informacije iz yaml datoteke služe *Python* skripti *start.py*, koja generira svijet na temelju mape, čita broj agenata i željenu formaciju te ih tako postavlja na karti te pokreće simulaciju. Primjer simulacije dva agenta kojima se može upravljati tipkovnicom može se vidjeti pokretanjem naredbe "t*muxinator start -p example.yml*" gdje će tmuxinator, umjesto korisnika, otvoriti nekoliko terminala, navedenih u danom *example.yml file*-u. *Example.yml* u sebi sadrži upute za otvaranje šest "prozora":

roscore, start.py, dva *teleop_twist_keyboard.py* prozora i dva *rostopic echo* prozora.

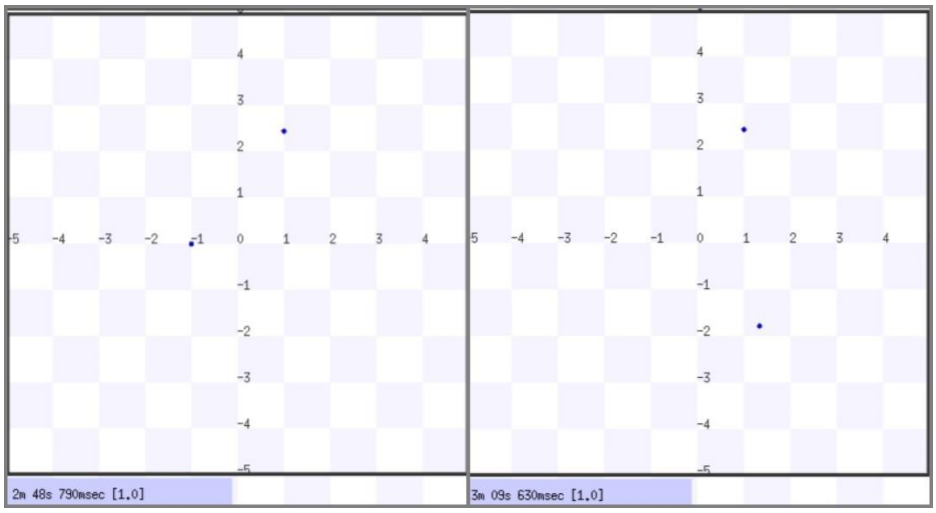
Roscore je osnova ROS-a koja omogućuje komuniciranje među ostalim ROS čvorovima, ovdje, omogućuje povezivanje agenata u simulaciji sa dva teleop_twsit_keyboard čvora, tj. čvora za upravljanje robotima tipkovnicom. Naravno, navedeno je i pokretanje čvora za

simulaciju (“r*osrun sphero_stage start.py”*). Sve ove naredbe iz example.yml mogu se pokrenuti bez *tmuxinatora*, slijedno, jedna po jedna, ali svaka u svom terminalu. Jednom kad se simulacija pokrene mogu se vidjeti dva agenta (prikazana dvijema crnim točkicama) u simulatoru i šest terminala koji odgovaraju pokrenutim čvorovima i *rostopic echo*-ima (printaju informacija o poziciji oba robota) na *Slici 12*.



Slika 12. izgled *sphero stage* okruženja

Sa srednja dva terminala (*Slika 12*) mogu se tipkovnicom pomicati agenti. Lijevim terminalom se tako pokreće samo prvi agent *robot_0*, vidljivo na lijevoj slici *Slike 13*, gdje drugi agent *robot_1* stoji na istoj poziciji kao na početku (*Slika 12*). Desnim terminalom pomiče se drugi agent *robot_1*, vidljivo na desnoj slici *Slike 13*, gdje se agent premjesti u četvrti kvadrant iz trećeg, dok *robot_0* miruje.



Slika 13. pomicanje agenata tipkovnicom

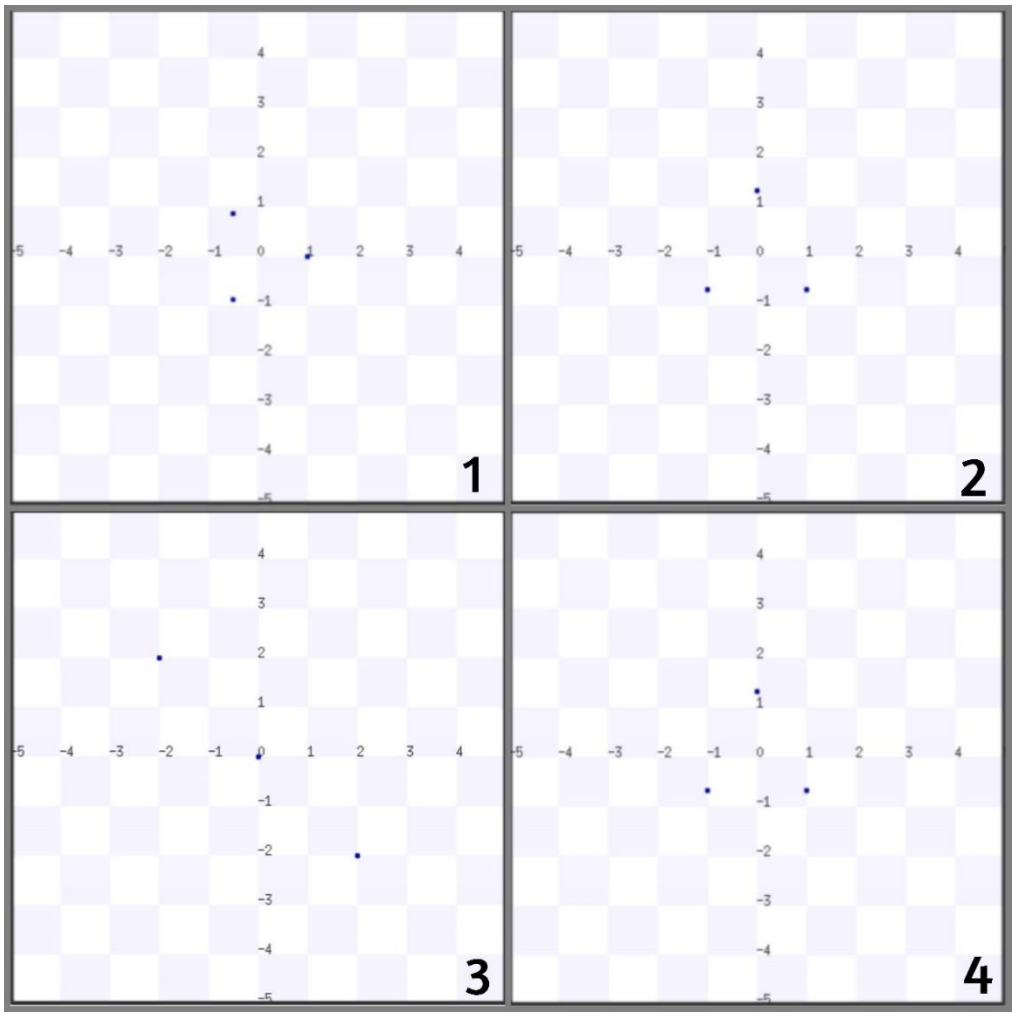
4.3. Rezultati – izmjena formacija

U ovom poglavlju prikazani su rezultati i odgovarajuće koreografije osnovnog algoritma, algoritma koji uključuje separaciju i algoritma koji dodatno pomiče cijelu formaciju u prostoru.

4.3.1. Osnovni algoritam

Umjesto upravljanja robota *teleop_twist_key* paketom kao na *Slici 13* kreirat će se novi *ROS* čvor pomoću skripte mijenjanje_formacije.py koji se primati podatke o agentima (pozicije), te na temelju njih i konsenzus protokola iz izraza (18) računati brzine svakog agenta. Čvor će potom svakom agentu *zadavati* brzine, koje su izračunate tako da agenti postignu željenu formaciju. U idućoj iteraciji proces se ponavlja. Kod koji radi opisano, (bez dijela izmjena više formacija) može se pronaći u funkciji consenus_protocol na github-u.

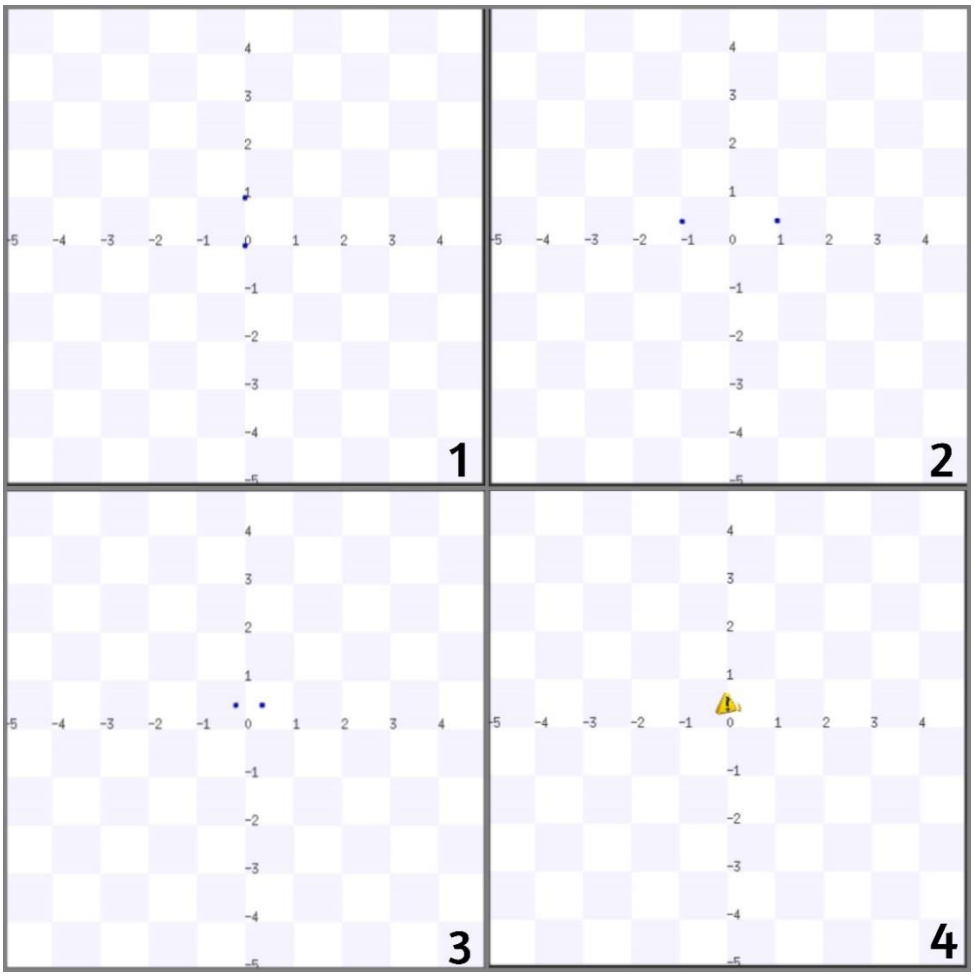
U *funkciji consensus_protocol while* petlja "spava" sve dok program ne skupi podatke o pozicijama svih agenata, zatim se zove funkcija *calculate_dx* koja raspisuje izraz (19). Konačno, svakom agentu šalje se njegova brzina. Već spomenuta skripta mijenjanje_formacije.py također koristi *launch_params.yaml* datoteku, u kojoj je dodana varijabla *choreographies*, koja prima polje imena formacija koje agenti trebaju izvesti. U datoteku je još stavljena i varijabla *alternation* koja, ako je *True*, jednom kad agenti prođu kroz sve formacije, ukazuje programu da krene sa koreografijom iz početka. U slučaju da je *alternation False*, agenti će ostati u zadnje navedenoj formaciji. Prikaz plesa tri agenta, koja su isprva postavljeni u krug i izmjenjuju formacije trokuta i linije može se vidjeti na *Slici 14* i priloženom *videozapisu istog imena*.



Slika 14. izmjenjivanje formacije trokuta i linije

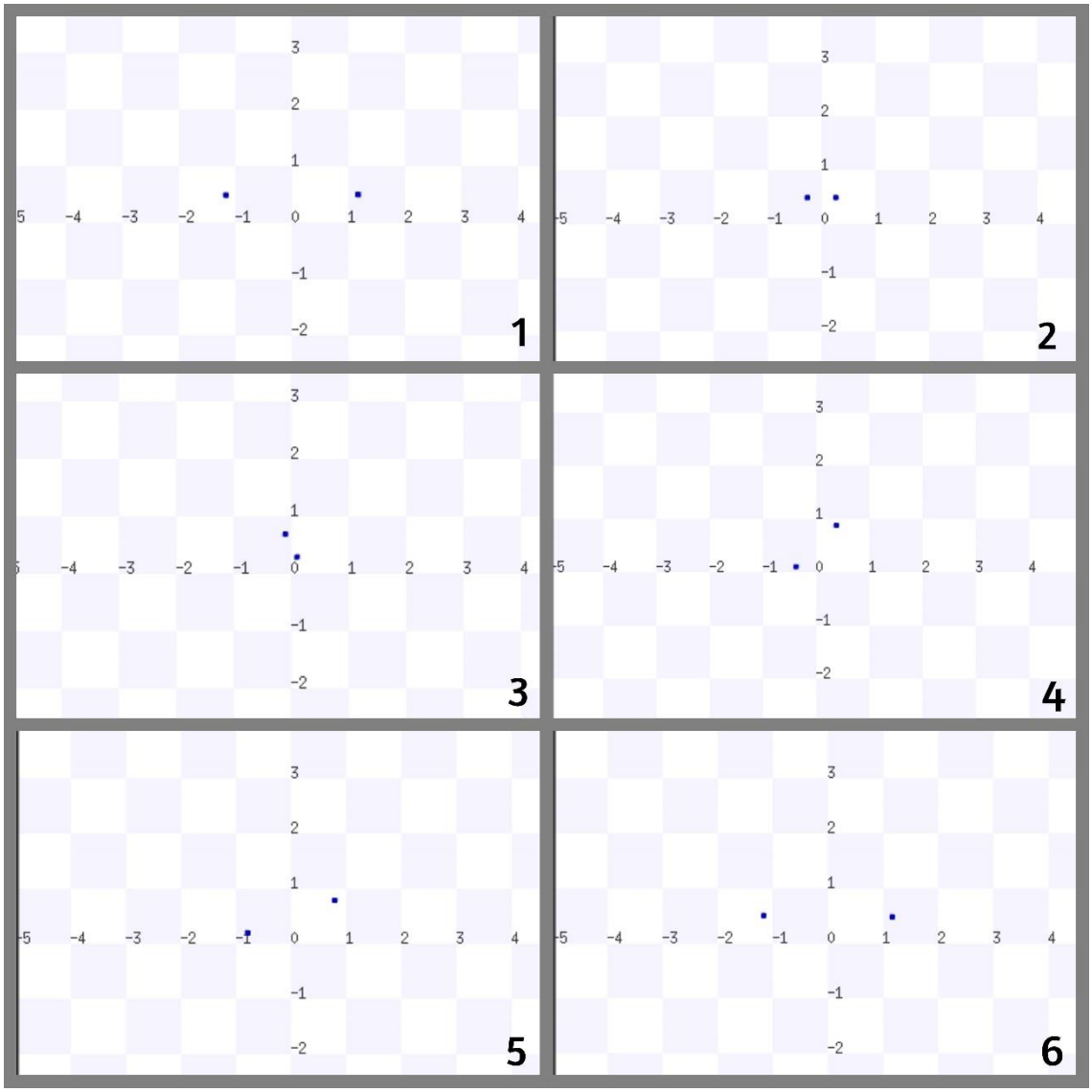
4.3.2. Izbjegavanje sudara

Separacijska varijabla također je uključena u konsenzus izraz, tj. U metodu calculate_dx. Utjecaj separacije ne vidi se na prijašnjem primjeru (jer se radi o slikama) pa je u ovoj sekciji istaknut primjer sa dva agenta koja pokušavaju zamijeniti mjesta (za koreografiju primaju dvije iste formacije, ali sa zamijenjenim pozicijama). Primjer što se dogodi kada je separacija isključena (no_crashing varijabla u *launch_params.yaml* postavljena na False) može se vidjeti na *Slici 15* i priloženom videozapisu, koja prikazuje dva već spomenuta agenta koja se sudare (zadnja slika) u pokušaju da se zamjene za mjesta.



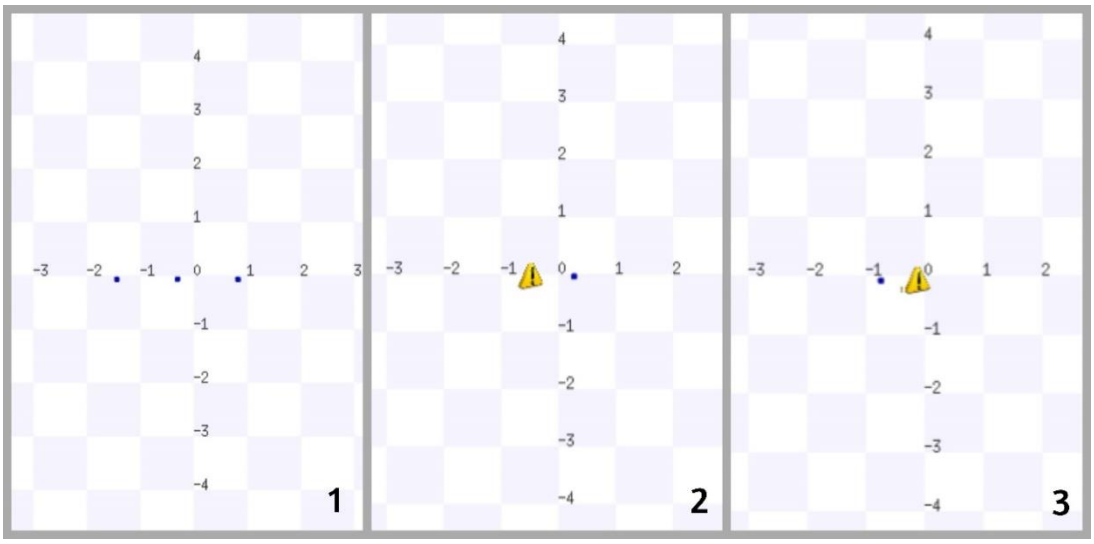
Slika 15. sudar dvaju agenata

Uključivanjem separacije i postavljanjem separacijske varijable na *0.8* dobije se ispravna zamjena mjesta dva agenta na *Slici 16* i danom videozapisu.

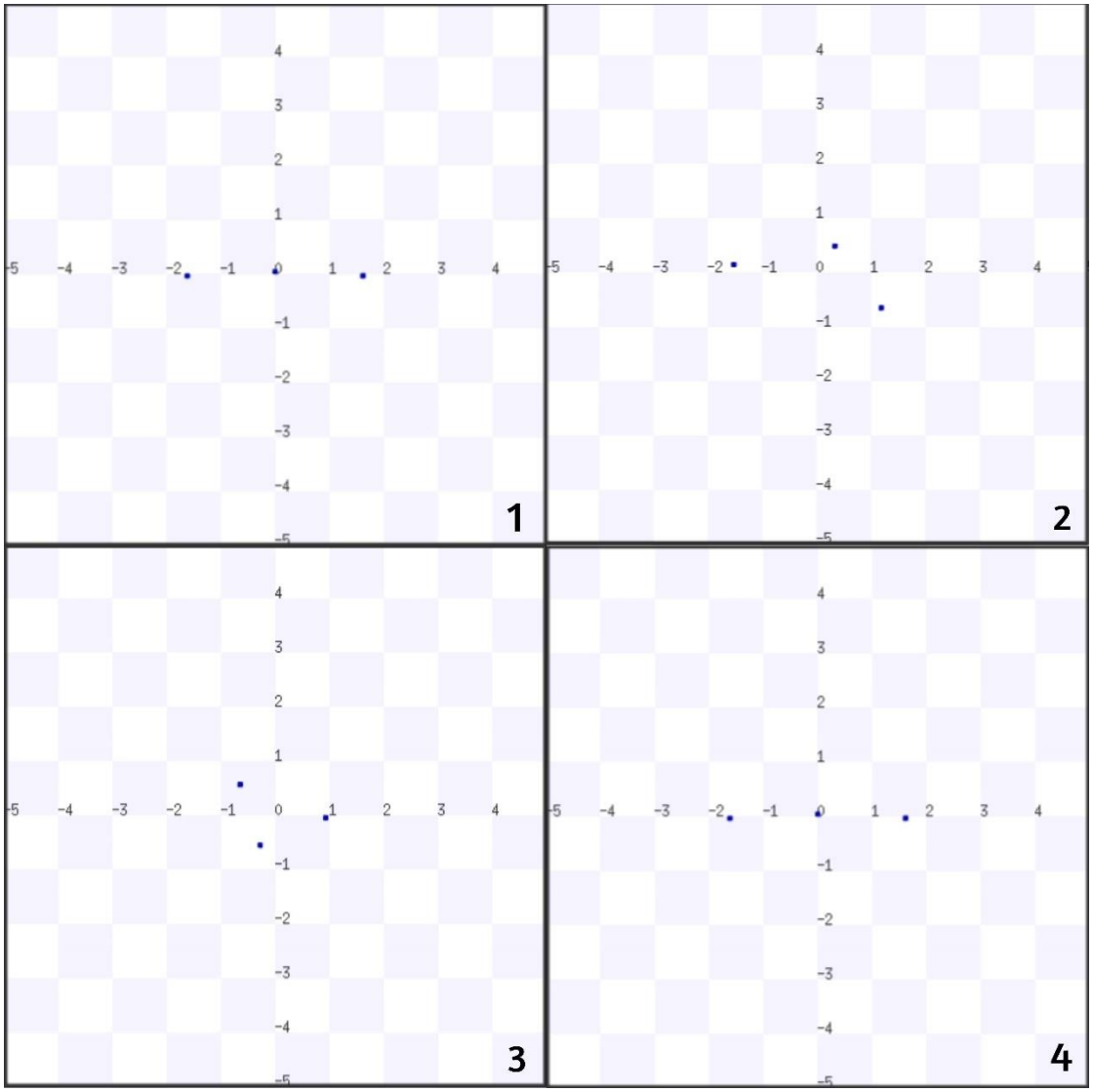


Slika 16. zamjena dvaju agenata uz separaciju

Sa *Slike 16* se jasno vidi kako agenti pravocrtno idu jedan prema drugome, ali kad su dovoljno blizu (slika u kutu desno) krenu se zaobilaziti na način da desni agent produži ravno, a lijevi ga "preskoči" (srednje dvije slike). Na kraju, prva i zadnja slika izgledaju isto, što znači da su agenti uspješno zamijenili pozicije u formaciji. Najmanja separacijska varijabla za koju se agenti ove koreografije neće sudariti iznosi *0.5*. Još jedan primjer koreografije, ovoga puta sa tri agenta, koja pokazuje kako nije bitno samo uključiti separaciju, već i podesiti iznos separacijske varijable može se vidjeti na *Slikama 17 i 18*. Na *Slici 17* (i priloženom videozapisu) svi agenti se međusobno sudare iako je separacijska varijabla postavljena na *0.8*. Podizanjem separacije na *2* agenti se uspješno izbjegnu (Slika 18, videozapis) i obave koreografiju zamjene mjesta (sa tri agenta).



Slika 17. zamjena mjesta 3 agenta, sudar

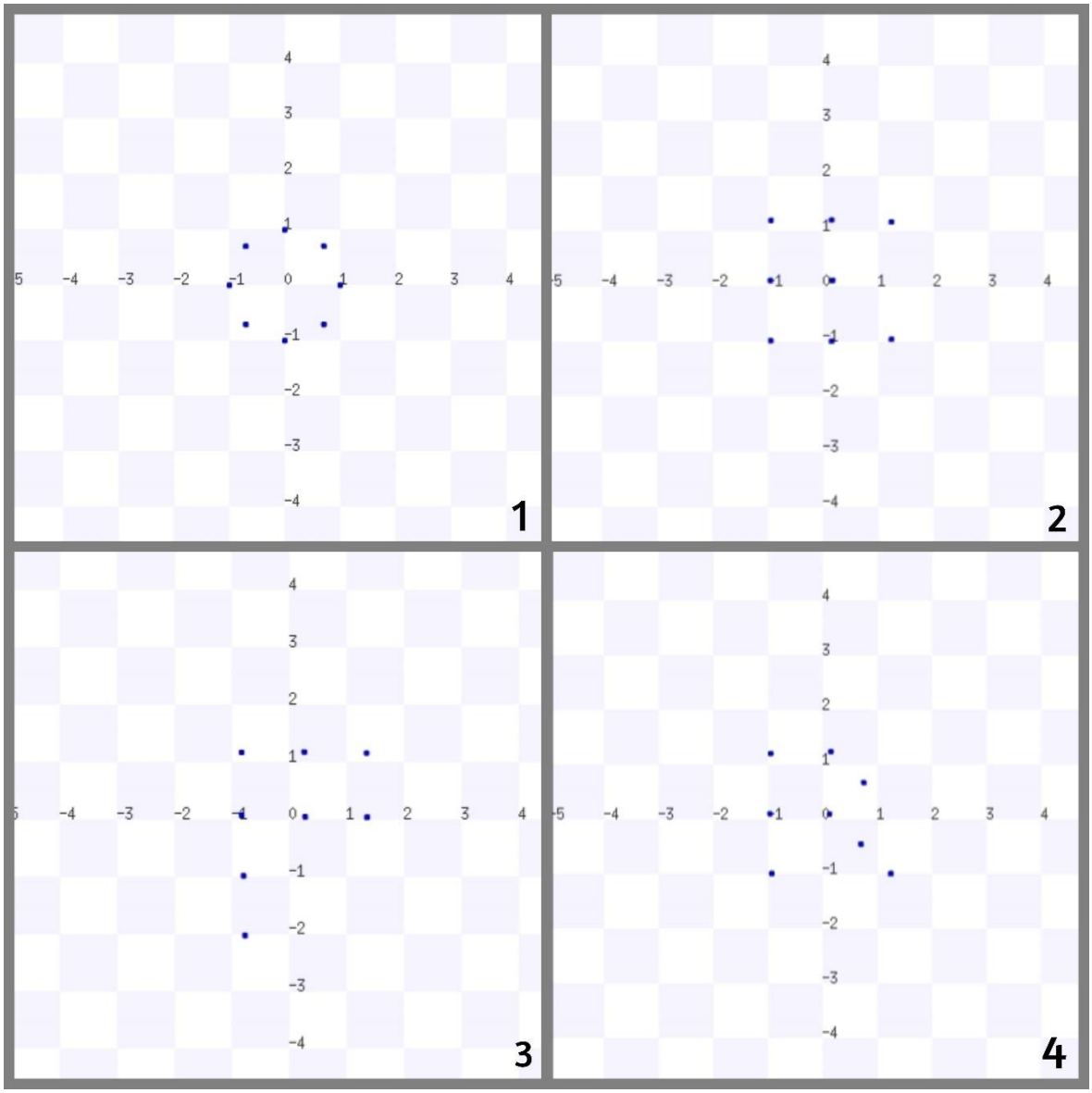


Slika 18. zamjena mjesta tri agenta, bez sudara

Ono što se da primijetiti sa *Slika 17* i *18* je razlika u veličini formacija, ako se pogledaju prve podslike, formacija na *Slici 18* nešto je veća, tj. rastegnutija. Razlog tomu je prisutnost veće separacije. Ovo nije nikakav problem s obzirom da je formacija definirana sa uvjetom invarijantosti na skalu (10), jedino bitno je da je formacija zadržala oblik i da se nije rotirala.

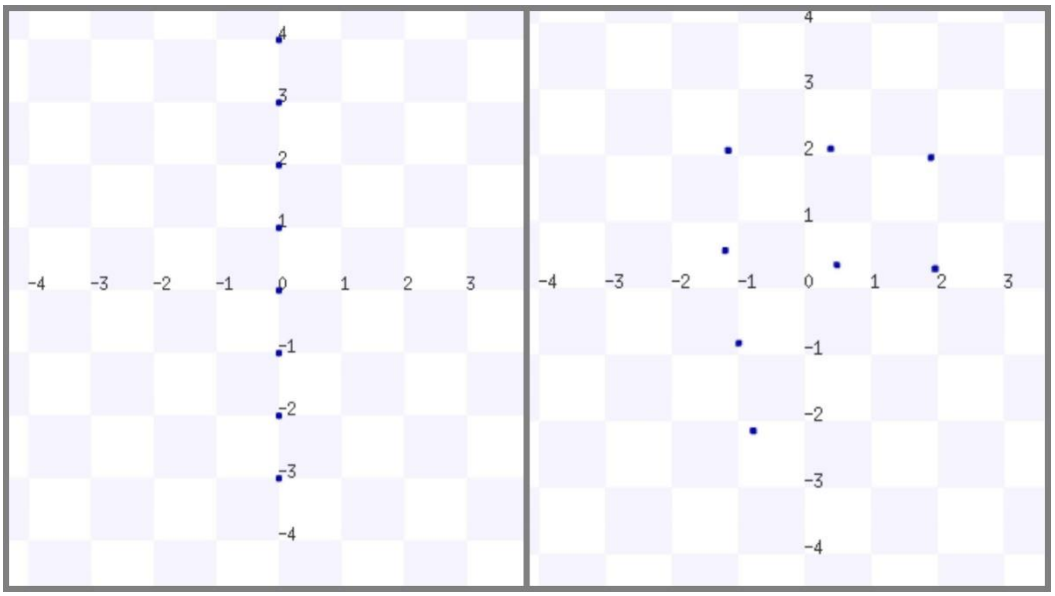
4.3.3. Primjer koreografije F-E-R

Primjer koreografije sa 8 agenata koji su na početku postavljeni u krug, pa kasnije izmjenjuju oblike slova F-E-R može se vidjeti na Slici 19 (i videozapisu). Separacija je postavljena na *0.5*.



Slika 19. F E R koreografija

U slučaju da se agenti u početku postave u liniju (u *launch_params.yaml* varijabla *distribution*) ista koreografija *F-E-R* imat će sudare pa je potrebno povisiti separaciju. Ovo dokazuje da na koreografije imaju utjecaja početni položaji letjelica, te ih je potrebno uračunati u problematiku. Na *Slici 20* (videozapis) vide se početni položaji letjelica (lijevo) i njihova postignuta formacija slova F uz separacijsku varijablu *2.5* koja sprječava sudaranje, ali razvlači slovo F (pa tako i sve ostala slova tj. formacije). Ovime se pokazuje da će početni položaji indirektno utjecati na skaliranje formacije (preko separacije).

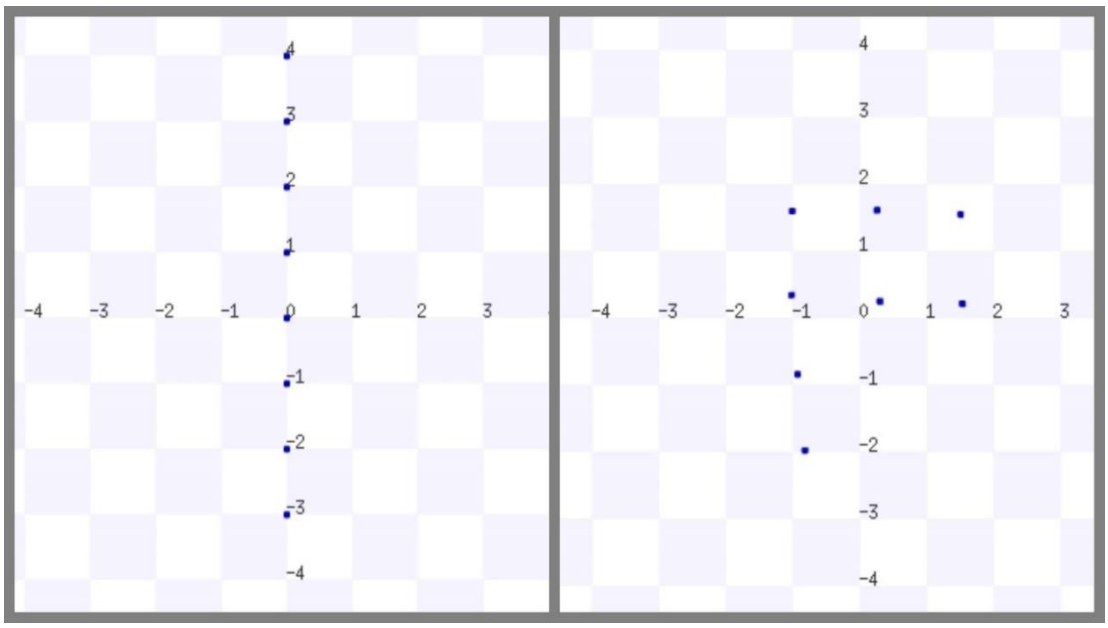


Slika 20. formacija slova F kada su agenti u početku poredani u liniju

4.3.4. Utjecaj skaliranja brzine

Varijabla *control_gain* skalira iznos dobiven konsenzus algoritmom, u ovom slučaju brzine *dx*. Varijabla je do sad, u sphero_stage simulatoru, bila postavljena na 1. Može se odmah u startu pretpostaviti da će skaliranjem brzina na manje vrijednosti agenti gibati sporije, a jednako često će komunicirati kao kad im je brzina bila veća što će dovoditi do točnijeg kretanja i potrebe za manjom separacijom. Postavljanjem *control_gaina* sa 1 na *0.4*, dakle smanjivanjem brzine za 60%, na koreografiju F-E-R koja na početku stavlja svoje agente kao na *Slici 20* separacija se može smanjiti sa *2.5* na *0.5* tj. na istu vrijednost kao i kod koreografije sa *Slike 19*, koja svoje početne agente stavlja u krug. Takva koreografija može se vidjeti na *Slici 21*. *Slika 20* pokazuje iste početne pozicije kao *Slika 20*, ali uz slovo F veličine kao na koreografiji *Slike 19*, dakle manje slovo nego na *Slici 20*. Indirektno se može zaključiti da ako se žele skalirati slova na manje veličine sve što je potrebno je smanjiti

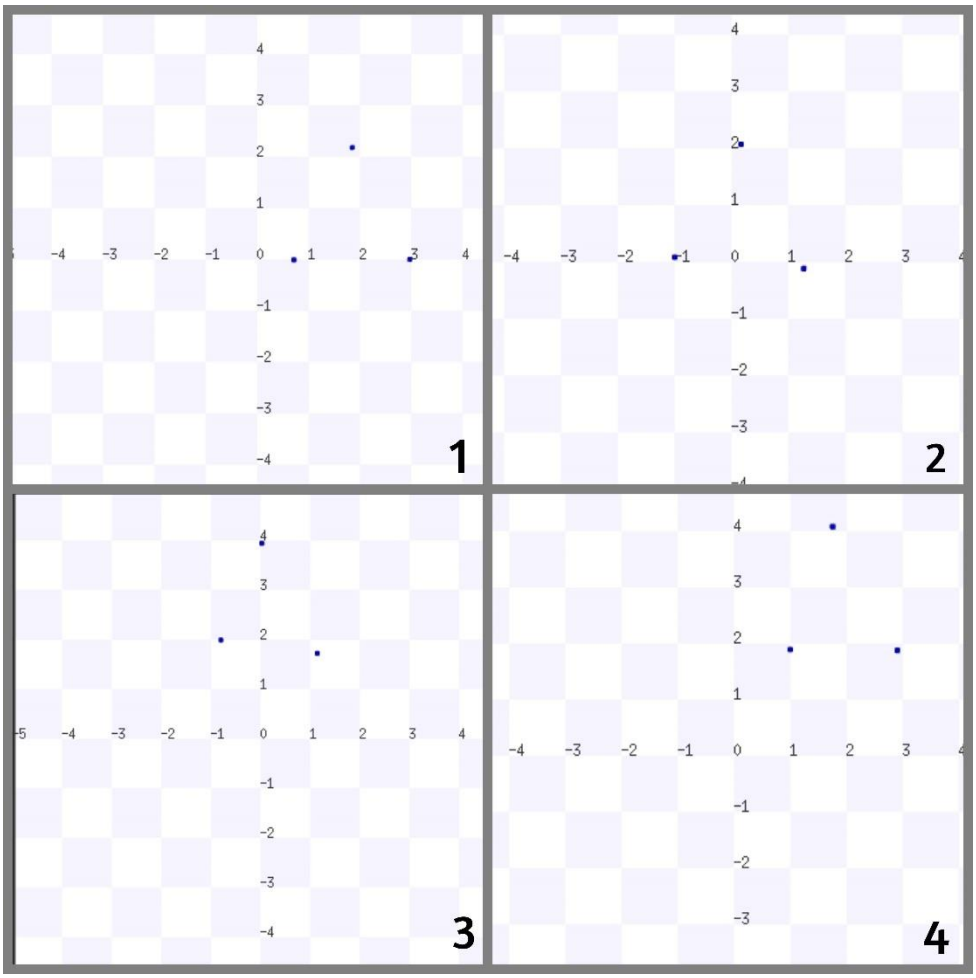
brzine jer će to omogućiti smanjivanje separacijske varijable koja se inače ne bi mogla smanjiti zbog opasnosti od sudara.



Slika 21. formacija slova F kada su agenti na početku poredani u liniju ali uz brzine skalirane na manje vrijednosti

4.4. Rezultati – pomicanje formacije

Koreografiranje agenata podrazumijeva i pomicanje formacija unutar prostora. Skripta *pomicanje_formacija.py* u *sphero_stage* paketu prima sve iste parametre kao i *mijenjanje_formacije.py* navedene u *launch_params.yaml* datoteci, uz dodatne vektore kretanja za agenta voditelja. Tako se u varijabli *formation_control* može izabrati kretanje voditelja u smjeru nekog vektora ili pak u oblik kruga ili kvadrata. Postizanje formacije trokuta i kretanje te iste formacije u obliku kvadrata može se vidjeti na *Slici 22*.

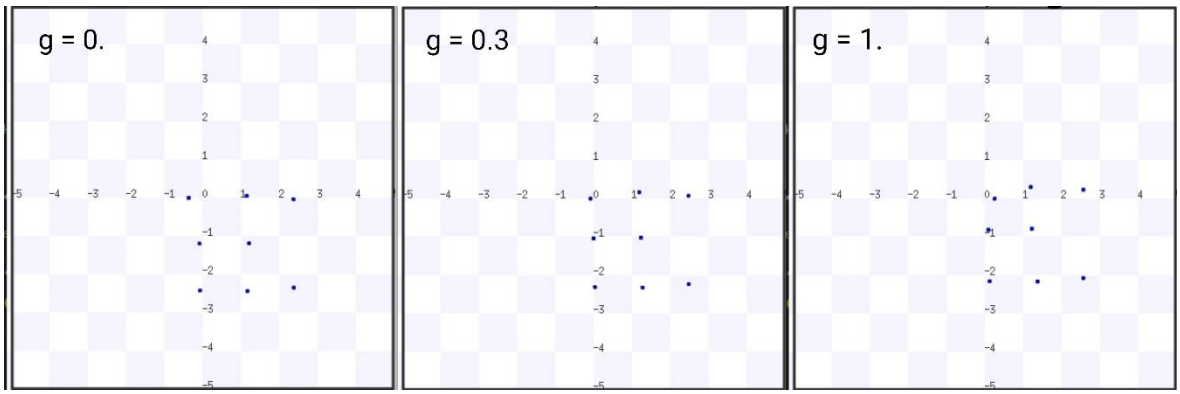


Slika 22. pomicanje formacije trokuta u kvadratni oblik

4.4.1. Utjecaj koeficijenta pričvršćenja g

Koeficijent pričvršćenja g vektor je koji definira kako je voditelj povezan sa drugim agentima u formaciji. Elementi vektora g_i iznose *0* ili više od *0* ako se radi o susjedu voditelja. Kako i koliko točno utječe iznos vektora je prikazano je na *Slici 23*. Na *Slici 23* izvedena je koreografija pomicanja slova *E* u kvadrat u tri različita slučaja. Svi elementi vektora postavljeni su na *0*, a susjedi na *1*, tako je vektor $g = [0,\ 1\ , 0\ , 0\ , 0\ , 0\ , 1\ , 1\ ]$. Za voditelja je izabran nulti agent koji je u formaciji gornji lijevi agent. Prvi slučaj na *Slici 23* prikazuje izgled formacije kada bi se vektor g pomnožio sa 0, dakle kada se dobitak pričvršćenja ne bi uračunavao. Agent voditelj onda se izdvaja od formacije i lagano bježi - nije dobro pričvršćen. Zadnji slučaj (treća slika) pokazuje što bi bilo kad bi vrijednosti g_i ostale na *1*. Agent voditelj onda se nalazi u formaciji, ali je previše povučen između svojih susjeda (tri okolna agenta). Srednji slučaj prikazuje zlatnu sredinu, kada se vektor g pomoži

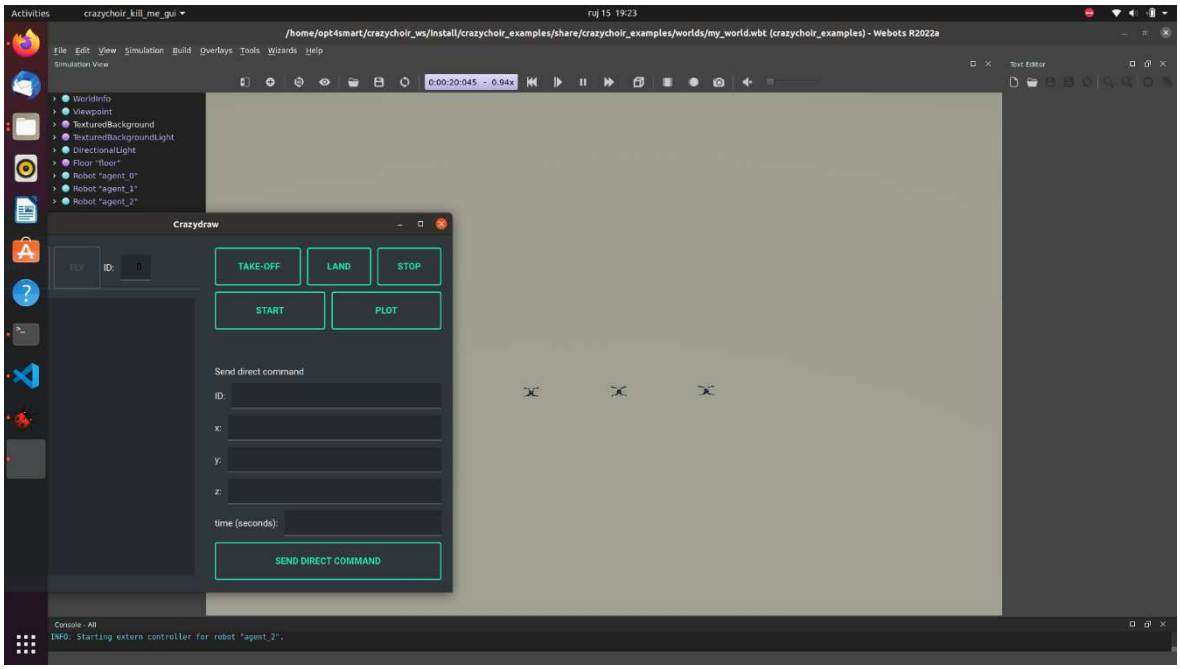
sa *0.3*. Voditelj sada nije ni preblizu ni predaleko. Ovime se pokazalo da je potrebno i odabrati prave vrijednosti za vektor pričvršćenja *g*.



Slika 23. utjecaj vektora pričvršćenja na formaciju

5. Simulacija sa Crazyflie letjelicama

Za agente koji će se koreografirati izabrane su male i lagane *Crazyflie* letjelice [13]. Postoji nekoliko simulacija koje mogu odsimulirati let više od jedne letjelice u trodimenzionalnom prostoru poput *CrazySim* [14] ili *CrazyChoir* simulatora [15]. U ovom radu korišten je *CrazyChoir* simulator za koji postoji verzija instalacije preko *Docker* spremnika korištenjem danih uputa [16]. *CrazyChoir* koristi *webots* i *rviz* za simuliranje letjelica i okruženja te upravlja njima već namještenim ROS2 čvorovima među kojima su esencijalni *gui.py* (kontrolna ploča kojom se upravlja uzlijetanjem, počinjanjem eksperimenta ili slijetanjem letjelica), *controller.py* i *guidance.py* (koji upravljaju letom) i *trajectory.py* koji *publish*-a informacije o trajektorijama svakog agenta. U ovom radu je dodan i čvor koji *publish*-a informacije o poziciji svake letjelice *get_positions.py*. Takav čvor pomaže *guidance.py* čvoru (točnije *ConsensusGuidance.py* čvoru, koji je također dodan) da „vodi“ tj. postavi letjelice u neku formaciju (zadanu u *my_formation_webots.launch.py* skripti) koristeći konsenzus protokol (19). Pokretanjem *launch* skripte naredbom „ros2 launch crazychoir_examples my_formation_webots.launch.py“ pokreće se simulator *webots* sa tri *Crazyflie* letjelice kao na *Slici 24*. Broj letjelica i njihove početne pozicije mogu se također zadavati u *launch* skripti. Sav kod dostupan je na github-u [12].

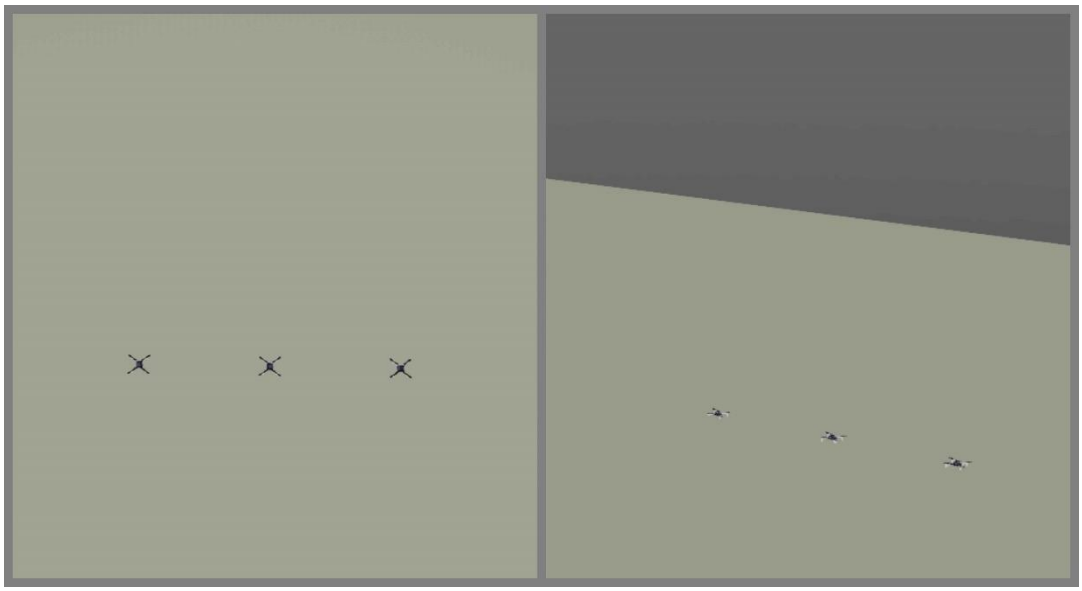


Slika 24. Izgled okruženja

5.1. Rezultati – koreografije u CrazyChoir simulatoru

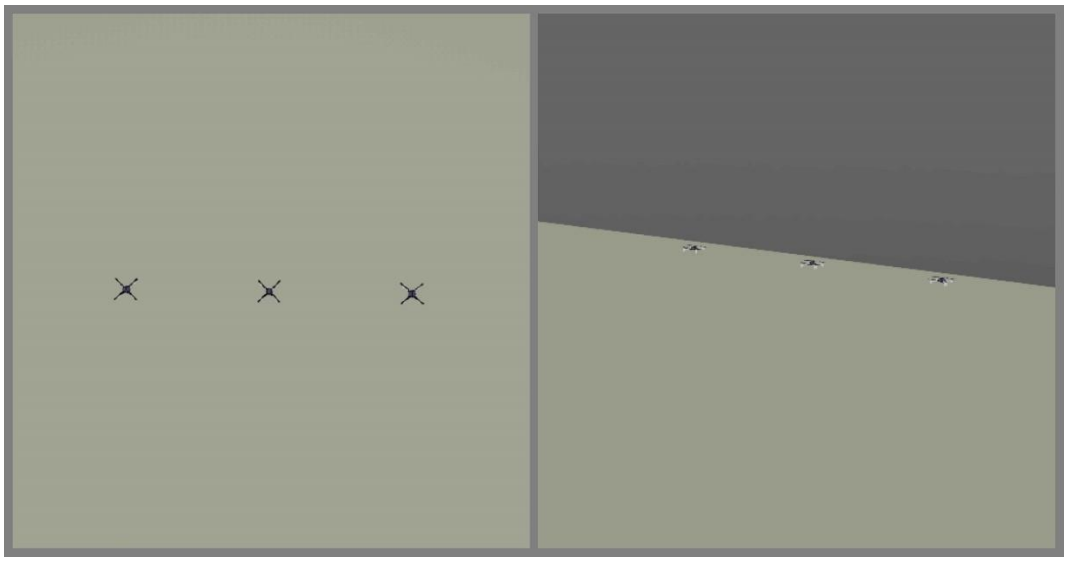
5.1.1. Postizanje formacije trokuta

Pokretanjem *launch* skripte dobiju se letjelice postavljene u formaciju linije kao na *Slici 25*. Lijeva slika prikazuje letjelice odozgo, a desna letjelice sa desne strane.



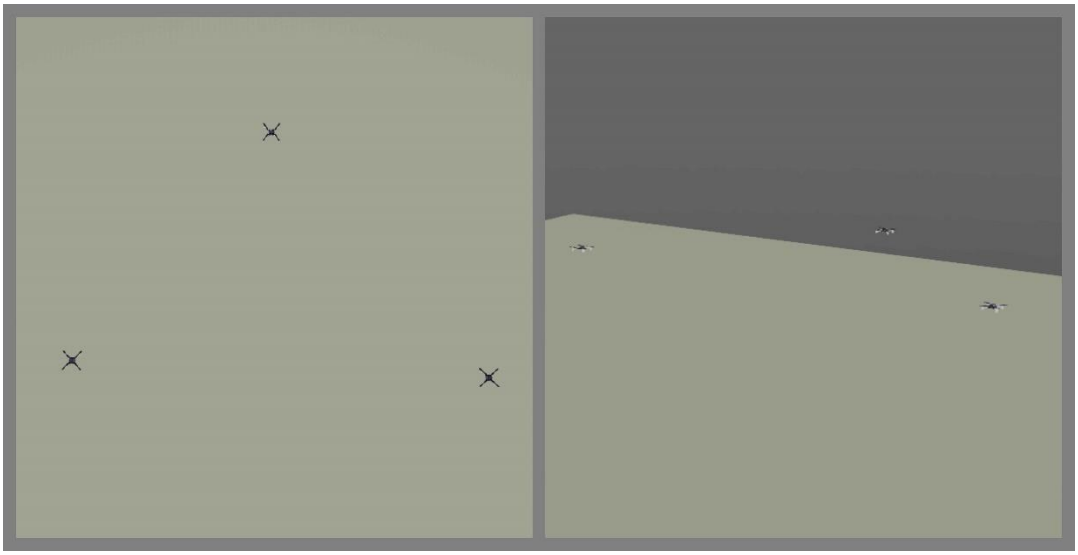
Slika 25. letjelice u početnim pozicijama

Letjelice ne mogu mijenjati svoje formacije dok su u ravnini poda (kao na Slici 25.) pa je potrebno da prije toga budu negdje u zraku. Pritiskom na tipku „*Take off*“ (vidljiva na izborniku lijevo na *Slici 24*) sve tri letjelice uzlete na neku prije zadanu visinu što je vidljivo na *Slici 26* (opet, lijeva slika prikazuje letjelice odozgo, a desna letjelice sa strane).



Slika 26. uzlet letjelica

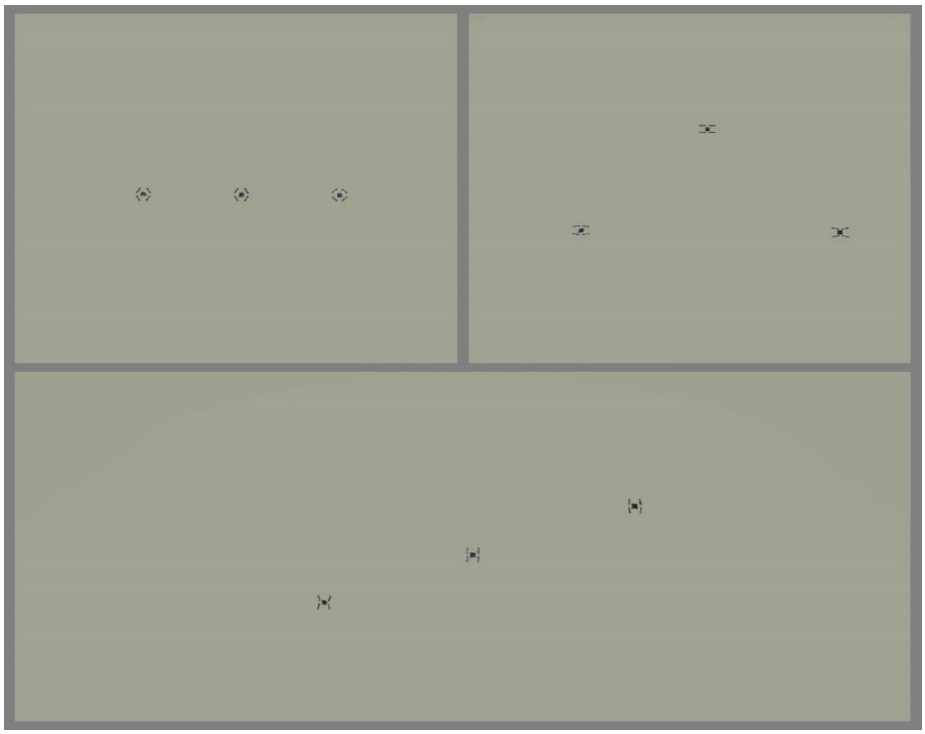
Konačno, sam eksperiment, točnije postizanje formacije trokuta počinje pritiskom na tipku *„Start“* (*Slika 24*.). Izgled formacije odozgo (lijevo) i sa strane (desno) može se vidjeti na *Slici 27*. Cijela koreografija može se vidjeti na slijedećim videozapisima: Video1 i Video2.



Slika 27. Formacija trokuta

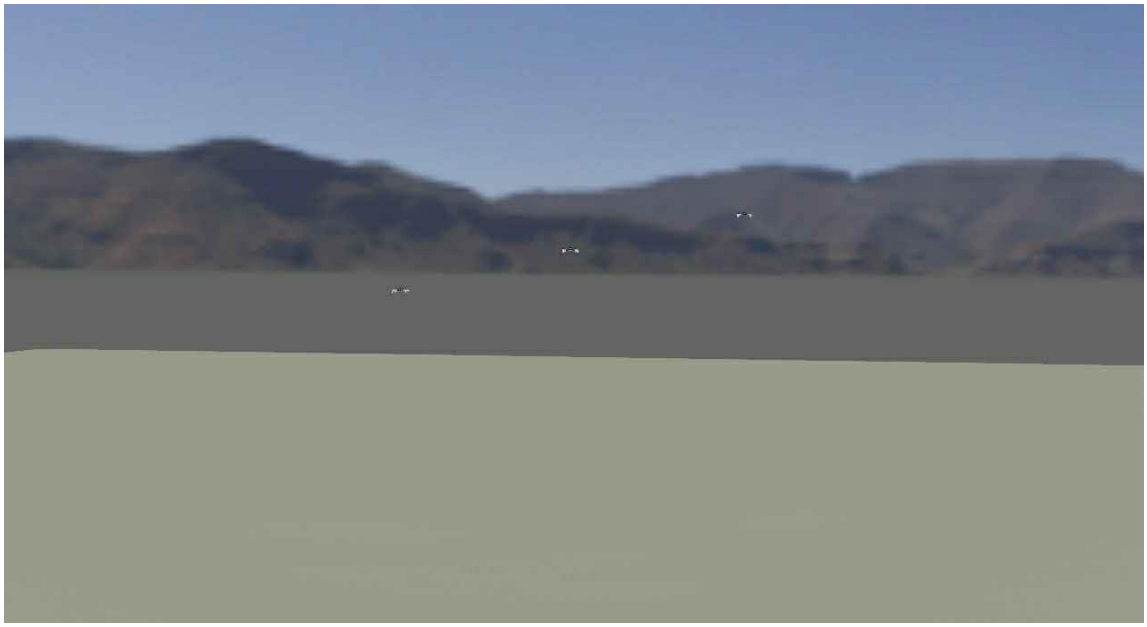
5.1.2. Izmjena formacija – trokut - linija

U simulatoru se mogu i izmjenjivati formacije. Tako nakon uzlijetanja i pritiskom tipke „start“ agenti oblikuju trokut pa formaciju linije prikazano na Slici 28.



Slika 28. izmjena formacija, odozgo

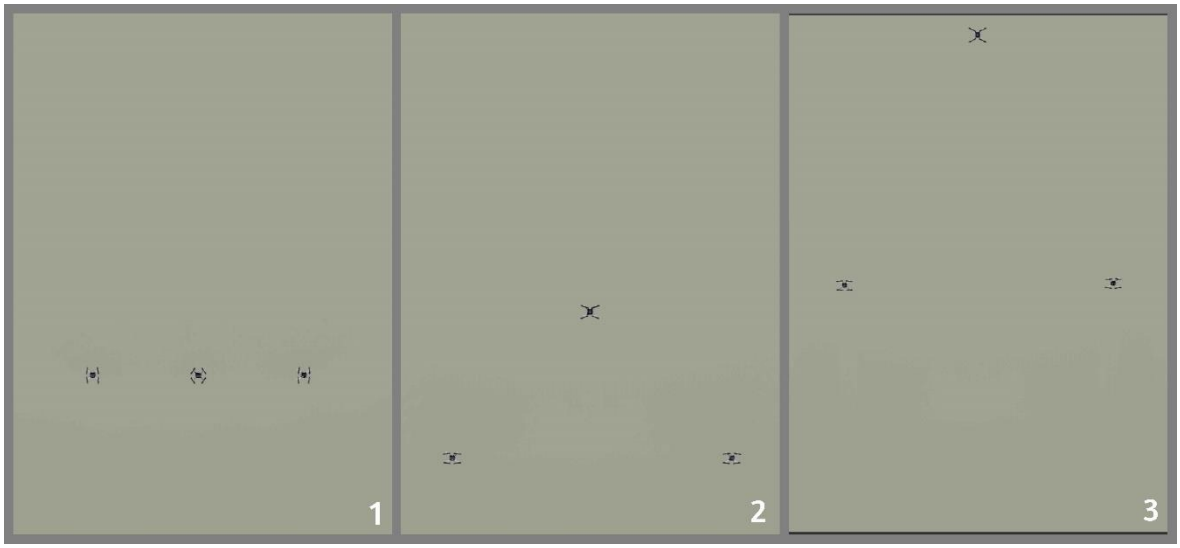
Prve dvije slike sa *Slike 28* prikazuju isto što i *Slike 26* i *27*, a to je postizanje prve zadane formacije – trokut (uzlijetanje se podrazumijeva iako nije prikazano). Formacija trokuta je zadana tako da je paralelna sa podom. Druga formacija, linija, koja se vidi na zadnjoj slici na *Slici 28*. za svaku letjelicu zadaje drugačiju visinu, tj. radi kut sa podom što se, za razliku od *Slike 28*. koja daje prikaz formacija odozgo, bolje može vidjeti na *Slici 29* koja prikazuje formaciju sa bočne strane. Koreografija se može vidjeti na Videu1 i Videu2.



Slika 29. formacija linije sa bočne strane

5.1.3. Pomicanje formacije trokuta

Pomicanje formacije trokuta u smjeru *y* osi može se vidjeti odozgo na *Slici 30*.



Slika 30. pomicanje trokuta, odozgo

Lijeva slika naravno prikazuje letjelice koje su već uzletjele, srednja slika postignutu formaciju trokuta, a desna pomaknutu formaciju trokuta. Prikaz pomicanja može se vidjeti i iz drugog kuta na *Slici 31*. gdje se usporedbom sa okolnim pejzažem (planinama) može primijetiti pomak formacije. Koreografija je prikazana na danom videozapisu.

Slika 31. pomicanje formacije iz drugog kuta

Zaključak

Koreografija jata bespilotnih letjelica, što je u stvari postavljanje letjelica u formaciju, izmjena i pomicanje formacija, postiže se uporabom konsenzus protokola. Konsenzus protokol omogućuje komunikaciju i dogovaranje među agentima (letjelicama) kako bi, u ovom slučaju, oblikovale željenu formaciju. Iako se konsenzus algoritam odvija *online* i dalje je potrebno uključiti separacijski izraz kako ne bi došlo do sudara. Separacijski izraz može se skalirati separacijskom varijablom - što je varijabla veća to je izgled formacije veći, odnosno razvučeniji pa se tako dodatno može kontrolirati izgled formacije. Na veličinu separacijske varijable utječu početni položaji agenata i formacija. Tako može postojati velika opasnost od sudara jer se mnoge putanje sijeku u prelasku u formaciju, bilo radi nezgodnog odabira koji agent će gdje biti u formaciji, bilo radi početnih položaja koji su nezgodni, ali to više ima utjecaja kad je broj agenata velik. Zbog svih tih zahtjeva mora se povećati separacijska varijabla, što znači da broj agenata, početne pozicije i odabir koji agent će ići gdje u formaciji utječe indirektno na razvučenost formacije. Ako se izgled formacije želi suziti, cijelu koreografiju je moguće usporiti (smanjiti brzine), a jednakom frekvencijom obavljati konsenzus algoritam što će za uzrok imati oprezniji i glađi let. Tako se i brzinama može regulirati izgled formacija.

Literatura

[1] *Consensus protocol, https://www.fer.unizg.hr/_download/repository/L6_-_Consensus[2].pdf*

[2] *Bespilotna letjelica* https://hr.wikipedia.org/wiki/Bespilotna_letjelica ; pristupljenjeno 26. Lipnja 2024.

[3] *Unnmaned aerial vehicle* https://en.wikipedia.org/wiki/Unmanned_aerial_vehicle ; pristupljeno 26. Lipnja 2024.

[4] *Koreografija jata bespilotnih letjelica*, Završni rad, P. Glavić Sanković

[5] Consensus dynamics, https://en.wikipedia.org/wiki/Consensus_dynamics ; pristupljeno 26. Lipnja

[6] *Consensus and Cooperation in Networked Multi-Agent Systems*, Reza Olfati-Saber, Member IEEE, J. Alex Fax, and Richard M. Murray, Fellow IEEE

[7] *Flocking [L3], https://www.fer.unizg.hr/predmet/vissus/materijali#%23!p_rep_144165!_-212442*

[8] *Sphero_stage simulator, https://github.com/larics/sphero_simulation/tree/master/sphero_stage*

[9] What is Docker?, *https://docs.docker.com/get-started/docker-overview/*

[10] Developing inside a Container, *https://code.visualstudio.com/docs/devcontainers/containers*

[11] Installing the NVIDIA Container Toolkit, *https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/install-guide.html*

[12] Popratni kodovi i skripte za Diplomski rad, https://github.com/Sankowitch/DIPRAD/tree/main

[13] Bitraze crazyflie, Crazyflie 2.1+ | Bitcraze

[14] *CrazySim: A Software-in-the-Loop Simulator for the Crazyflie Nano Quadrotor*, Christian Llanes, Zahi Kakish, Kyle Williams, and Samuel Coogan, CrazySim: A Software-in-the-Loop Simulator for the Crazyflie Nano Quadrotor (gatech.edu)

[15] *CrazyChoir: Flying Swarms of Crazyflie Quadrotors in ROS 2*, Lorenzo Pichierri, Andrea Testa, Giuseppe Notarstefano, IEEE Xplore Full-Text PDF:

[16] Docker Intallation of CrazyChoir, crazychoir/docker at master · OPT4SMART/crazychoir · GitHub

[17] Videozapisi koreografija SPHERO_STAGE: Sphero stage koreografije (youtube.com)

[18] Videozapisi koreografija CRAZYCHOIR, Crazychoir koreografije (youtube.com)

Sažetak

Koreografija jata bespilotnnih letjelica

Ključne riječi: bespilotne letjelice, koreografija leta, dinamičko koreografiranje leta, konsenzus protokol, upravljanje formacijama konsenzus protokolom

Koreografije jata bespilotnih letjelica koriste se na raznim društvenim smotrama i događanjima. Rastom kompleksnosti koreografije i broja letjelica, raste i opasnost od sudaranja letjelica međusobno, ali i letjelica s preprekama u prostoru. U ovom radu, koristi se konsenzus protokol za ostvarivanje koreografije bespilotnih letjelica u stvarnom vremenu koji i osigurava njihovo nesudaranje. Algoritam je testiran u osnovnoj i naprednoj simulaciji za bespilotne letjelice Crazyflie.

Summary

Coreographing a flock of drones

Key words: Unmanned aerial vehicle – UAV, flight choreography, dynamic flight choreography, consensus protocol, formation control with consensus

Choreographies of flocks of drones are used at various social gatherings and events. As the complexity of the choreography and the number of drones grow, so does the danger of them colliding with each other or colliding with obstacles in space. This paper uses consensus protocol to achieve the choreography of unnmaned aerial vehicles in real time, without danger of them colliding. The algorithm was tested in basic and advanced Crazyflie drone simulation.