Universitatea Transilvania din Brașov

ȘCOALA DOCTORALĂ INTERDISCIPLINARĂ
Facultatea de INGINERIE ELECTRICĂ ȘI ȘTIINȚA CALCULATOARELOR

Cosmin George GINERICĂ

Conducerea roboților autonomi utilizând tehnici de inteligență artificială
Navigation of autonomous robots using artificial intelligence techniques

REZUMAT

Conducător ştiinţific
Prof.dr.ing. Sorin Mihai GRIGORESCU

BRAȘOV, 2024

Cuprins

Tema tezei de doctorat

Lucrarea de față propune un sistem de navigare autonomă pentru roboți mobili denumit ObserveNet Control. Acesta se bazează pe paradigma perception-plan-control pentru realizarea sarcinilor specifice ale navigației autonome: percepția și înțelegerea mediului înconjurător, planificarea traiectoriei pentru robot, respectiv controlul acestuia astfel încât să urmărească traiectoria generată. Domeniul fundamental în care se încadrează această teză este cel de *Mecatronică și Robotică*. De asemenea, sunt utilizate noțiuni și tehnici specifice domeniilor de Vedere Artificială, respectiv Inteligență Artificială.

În această lucrare, este propus un algoritm de inteligență artificială denumit ObserveNet Control, bazat pe o rețea neuronală recurentă (RNN), care primește ca intrări secvențe de observații senzoriale provenite de la un vehicul și pe baza cărora modelează și prezice mediul de lucru. Folosind acest model, un robot mobil navighează în mod autonom în mediul său înconjurător. Au fost definite toate trei submodulele necesare navigării autonome: modulul de percepție, cel de planificare a traiectoriei, precum și cel de control.

Modulul de planificare al traiectoriei locale, responsabil de generarea traseului pe care robotul mobil trebuie să navigheze, totodată evitând coliziuni cu obstacolele din scenă, este bazat pe Dynamic Window Approach, modificat astfel încât să țină cont de predicțiile observațiilor senzoriale provenite de la arhitectura ObserveNet.

Controlul propriu-zis al robotului este realizat folosind un controller NMPC (Nonlinear Model Predictive Control).

Algoritmul a fost testat atât cu ajutorul unui simulator utilizat pentru validarea diferitelor tehnologii de navigare autonomă, cât și într-un mediu real, folosind un robot patruped și respectiv un vehicul autonom cu șasiu diferențial.

Datele de localizare (starea curentă a vehiculului autonom), respectiv observațiile înregistrate de către senzorii aflați pe vehicul, sunt stocate într-o memorie augmentată. Acestea sunt apoi partiționate în secvențe de lungime prestabilită, urmând a fi folosite atât pentru inferența rețelei neuronale recurente, care modelează dinamica mediului înconjurător, cât și ca intrări pentru controllerul NMPC, care generează traiectoria dorită.

Deși există sisteme care pot realiza sarcina de conducere autonomă deja echipate pe vehicule aflate pe drumurile publice, acestea nu sunt complet funcționale. Pe de altă parte, conducerea autonomă se referă, adițional scenariului vehiculelor care navighează pe șosele, și la domeniul de robotică, unde este nevoie de îndeplinirea acestei sarcini deopotrivă.

La nivel înalt, atât pentru roboții mobili, cât și pentru vehicule, problema navigării autonome se referă la capacitatea agentului inteligent de a naviga din punctul de start până în punctul de destinație, respectând un set de constrângeri fizice, respectiv impuse, simultan evitând coliziuni cu obstacole statice sau dinamice prezente în mediul de lucru al agentului.

În mod evident, sarcina de navigare autonomă presupune o serie de alte sarcini care trebuie rezolvate de către agentul inteligent, grupate în trei componente fundamentale: percepția agentului inteligent, care constă în înțelegerea mediului înconjurător în care acesta operează, planificarea traiectoriei, atât a celei globale, de referință, cât și a celei locale, care permite încorporarea unui anumit nivel de logică necesară deplasării, precum evitarea obstacolelor și

componenta de control, care are rolul de a efectua deplasarea agentului pe baza unor semnale de comandă generate de către componenta inteligentă a agentului.
Cele trei componente principale ale unui sistem autonom trebuie să fie capabile să își îndeplinească funcționalitățile aferente în condiții variate de operare, în funcție de aplicația pentru care este implementat sistemul de navigare autonomă. Spre exemplu, un vehicul autonom care navighează în trafic este supus multor factori perturbatori, precum condiții de iluminare variate, condiții meteorologice precare. Totodată, acest sistem trebuie să țină cont de foarte multe variabile din mediul înconjurător, precum alte vehicule, pietoni, bicicliști, semafoare, marcaje rutiere șterse, ș.a. Pe de altă parte, un robot mobil care navighează într-un mediu nestructurat are parte de provocări specifice acestui scenariu, precum dinamica imprevizibilă a obstacolelor din mediul înconjurător, schimbări constante ale mediului, respectiv diferite impedimente care împiedică navigarea în siguranță.
Revenind la descrierea de nivel înalt a problemei navigării autonome, aceasta poate fi formulată în felul următor: agentul inteligent trebuie să poată urmări traiectoria globală de referință $z_{ref}^{<t-\infty,t+\infty>}$ cât mai îndeaproape, însă fără a produce coliziuni cu alți agenți sau cu obstacole statice din mediul înconjurător. În scopul evitării coliziunilor, trebuie să fie generată o traiectorie locală pe care agentul să o urmărească -$z^{<t+1,t+\tau>}$.
Situațiile limită pot cauza probleme pentru agentul inteligent, cu atât mai mult dacă dinamica din scenă evoluează puternic. De exemplu, este mult mai dificilă pentru vehiculul autonom situația în care alte vehicule se deplasează cu viteză mare sau manifestă un comportament periculos în trafic. Pentru situații de acest gen, este util pentru un agent inteligent să aibă la dispoziție detalii suplimentare. În acest sens, componenta inteligentă a sistemului de conducere autonomă se poate folosi de informații senzoriale prezise de către un algoritm de inteligență artificială.
Sintetizând problema tratată în această lucrare, agentul inteligent care navighează autonom, fie că este vorba despre un vehicul în trafic, fie despre un robot aflat într-un mediu de lucru nestructurat, trebuie să poată naviga între punctul de start, respectiv punctul de destinație, simultan evitând obstacole dinamice din scenă.
De asemenea, scenariile periculoase pentru un agent care navighează autonom, reprezentate sub forma obstacolelor dinamice din mediul înconjurător sunt de o importanță critică pentru buna desfășurare a sarcinilor acestui agent.

Obiectivele cercetării

Obiectivul principal al activității de cercetare este reprezentat de utilizarea tehnicilor de inteligență artificială pentru a realiza navigarea autonomă a unui robot mobil, avându-se totodată în vedere situații limită generate de evoluția imprevizibilă a mediului înconjurător în care robotul navighează. În scopul realizării acestui algoritm de navigare autonomă, s-au implementat activitățile de studiu al literaturii de specialitate, realizarea conceptelor, respectiv implementarea lor, precum și testarea în mediu simulat, respectiv real, utilizând roboți mobili.
Pornind de la obiectivul principal al lucrării, au fost derivate de asemenea obiective specifice:

- Studiul literaturii de specialitate din domeniul navigării autonome.
- Studiul lucrărilor despre predicția mediului înconjurător.

- Studiul lucrărilor despre controlul inteligent.
- Dezvoltarea arhitecturii ObserveNet Control (algoritm de inteligență artificială folosit în cadrul sarcinii de navigare autonomă).
- Antrenarea algoritmului de inteligență artificială ObserveNet cu date specifice pentru sceneriile simulat, respectiv real.
- Implementarea și testarea algoritmilor de control pentru roboți mobili.
- Implementarea algoritmilor de percepție pentru roboți mobili.
- Implementarea algoritmilor de procesare a datelor senzoriale.
- Implementarea unui modul de localizare pentru robotul mobil.
- Implementarea unui modul de planificare a traiectoriei locale.
- Implementarea interfețelor necesare pentru toate subsistemele responsabile de navigarea robotului: actuatori, comunicație inter-module, achiziție de date senzoriale, etc.

Structura tezei de doctorat

Teza este structurată în șapte capitole, după cum urmează.
În capitolul 1 sunt prezentate câteva idei despre impactul inteligenței artificiale asupra domeniului de navigare autonomă, precum și o descriere de nivel înalt a principalelor componente ale unui sistem de navigare autonomă. De asemenea, se prezintă obiectivul principal și obiectivele secundare ale cercetării, împreună cu structura și conținutul tezei.
Capitolul 2 prezintă o vedere de ansamblu asupra tehnologiilor de inteligență artificială, precum și despre rețelele neuronale și aplicarea acestora în tehnici de control inteligent pentru vehicule autonome. Sunt descrise principalele tipuri de rețele neuronale folosite de către cercetători, este prezentată tehnica folosită pentru antrenarea rețelelor neuronale, sunt descrise câteva funcții de cost folosite în cadrul antrenării, precum și diverse funcții de activare utilizate.
Capitolul 3 prezintă noțiuni generale despre componenta de percepție a unui sistem de navigare autonomă, pornind de la tipurile de senzori folosiți pentru percepție, precum și explicații despre procesarea acestor date senzoriale înainte de a fi trimise către diferiți algoritmi de interpretare. Apoi sunt prezentate noțiuni teoretice despre planificarea traiectoriei unui agent inteligent. De asemenea, se expun tipurile de planificatoare de traiectorii și se discută despre generarea traiectoriilor locale, respectiv globale. Principalii algoritmi de control folosiți pentru agenții inteligenți sunt prezentați.
Capitolul 4 prezintă stadiul actual al cercetărilor algoritmilor de inteligență artificială pentru sarcina de navigare autonomă, descrie componentele principale ale sistemelor de navigare și prezintă algoritmii de percepție, planificare, respectiv control realizați în scopul acestei teze.
Platformele simulate, respectiv robotul mobil diferențial (Agile Scout-X) folosite pentru testarea algoritmului, împreună cu rezultatele experimentale și concluziile aferente sunt prezentate în capitolul 5
Cea de-a doua platformă robotică utilizată (robotul mobil Unitree A1) pentru testare, împreună cu rezultatele experimentale și concluziile aferente sunt prezentate în capitolul 6.
Capitolul 7 prezintă concluziile finale, precum și diseminarea rezultatelor și sumarizarea direcțiilor viitoare de cercetare.

Metodologia de cercetare

Sistemul de navigare autonomă propus în această lucrare este structurat sub forma unei arhitecturi de tipul perception-plan-control, model care presupune separarea componentelor responsabile pentru sarcina de navigare în module dedicate. Aceste module sunt cel de percepție a mediului înconjurător, cel de localizare, cel de planificare a traiectoriei, respectiv cel de control. Acest sistem de navigare a fost implementat conform unei arhitecturi modularizate, astfel încât să poată fi utilizat cu modificări minime pe mai multe platforme, atât în mediu simulat, cât și în cadrul roboților mobili reali.

Un robot mobil care navighează autonom trebuie să posede în orice moment cunoștințe despre mediul său înconjurător, astfel încât să poată evalua anumite situații riscante, respectiv să poată lua anumite decizii în funcție de împrejurări. Componenta de percepție este cea responsabilă de achiziția, respectiv interpretarea datelor senzoriale provenite de la robot. Cel mai important aspect al percepției este abilitatea de evaluare a evoluției dinamice a obstacolelor din mediul de lucru.

Procesarea observațiilor senzoriale

Modulul de percepție al robotului mobil utilizează informații provenite de la o multitudine de senzori. Dintre aceștia, cei mai folosiți sunt camerele RGB, pe baza cărora se realizează cu ajutorul algoritmilor de Vedere Artificială sau Inteligență Artificială înțelegerea scenelor vizualizate. O altă categorie de senzori folosiți frecvent în cadrul sarcinilor de navigare autonomă sunt cei care furnizează informații topologice despre mediul în care navighează roboții mobili, precum LiDAR sau RADAR. Senzorii utilizați de către roboții mobili (simulați sau reali) în cadrul acestei lucrări sunt senzori de distanță: LiDAR, cameră RGB-D, respectiv ultrasonici.

Datele LiDAR vin sub forma unor nori de puncte având o frecvență de achiziție fixă și vor fi referite în continuare utilizând în mod interschimbabil termenul de observație (senzorială). Astfel, o observație senzorială LiDAR va fi descrisă în felul următor:

$$I^{<t>} = \{(X_i, Y_i, Z_i), \forall i \in [0, N]\}$$

Unde I este notația pentru observație, exponentul $< t >$ se referă la momentul de timp curent t, iar (X_i, Y_i, Z_i) sunt coordonatele carteziene tridimensionale ale punctului *i*. *N* este dimensiunea norului de puncte.

Norul de puncte obținut de la senzor trece întâi printr-o transformare de coordonate, astfel încât acesta să aibă punctele relativ la sistemul de coordonate al robotului mobil.

$$I^{<t>} \leftarrow T_{VEH}^{L} \cdot \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}, \forall i \in [0, N]$$

Unde T_{VEH}^{L} este matricea de transformare omogenă din sistemul de coordonate al senzorului în cel al vehiculului. Aceasta are forma:

$$T_{VEH}^{L} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$$

Unde *R* este componenta de rotație, iar *t* este cea de translație.
Următorul pas de procesare al norului de puncte este schimbarea reprezentării acestuia în structură *OcTree*. Aceasta este o structură arborescentă pentru reprezentarea obiectelor tridimensionale prin împărțirea acestora în mod recursiv în subdiviziuni având câte opt descendenți. Astfel, observațiile senzoriale vor deveni:

$$I^{<t>} \leftarrow f_O(I^{<t>}, r)$$

Unde f_O este funcția de conversie nor de puncte – OcTree, iar *r* este rezoluția dorită.
Un alt pas de preprocesare prin care sunt trecute observațiile senzoriale este prăguirea (thresholding) în funcție de distanță:

$$p_r \leftarrow p[i] \leftrightarrow d(s, p[i]) < T_d, \forall i \in [1, N]$$

Unde p_r este unitatea de bază selectată, $p[i]$ este unitatea analizată, $d(s, p[i])$ este distanța euclidiană dintre originea vehiculului și unitate, T_d este valoarea de prag (aleasă empiric), iar *N* este dimensiunea observației.
Următoarea operație de procesare este eliminarea punctelor aflate în planul podelei, respectiv a celor aflate la o înălțime mai mare decât cea a robotului mobil:

$$p_r \leftarrow p[i] \leftrightarrow V_L < p[i]_Z < V_H, \forall i \in [1, N]$$

Unde $p[i]_Z$ este înălțimea unității, V_L și V_H sunt pragurile minim, respectiv maxim pentru ca o unitate să fie luată în considerare.

Punctele aflate în afara câmpului vizual al robotului sunt de asemenea eliminate:

$$p_r \leftarrow p[i] \leftrightarrow |\angle(p[i], V_O)| \leq T_V$$

Unde V_O este originea robotului mobil, iar T_V este valoarea prag pentru definirea limitei câmpului vizual (aleasă empiric).
După realizarea operațiilor de preprocesare descrise anterior, observația senzorială curentă reprezentată în structura OcTree tridimensională este proiectată într-o perspectivă tip vedere de sus (top-down):

$$I^{<t>} \leftarrow f_{2D}^{3D}(I^{<t>})$$

Unde $f_{2D}^{3D}(\cdot)$ reprezintă funcția de proiecție 3D-2D.

$$f_{2D}^{3D}(p_i) = \{(X_{p_i}, Y_{p_i}), \forall i \in N\}$$

Unde (X_{p_i}, Y_{p_i}) sunt coordonatele carteziene ale unității OcTree.

Localizarea robotului mobil

Componenta de localizare folosește date de geolocație înregistrate de către robotul mobil, precum și măsurători de la senzorul inerțial pentru a calcula starea curentă a robotului.
Modelul de tranziție al stărilor este definit în felul următor:

$$s^{<t+1>} = s^{<t>} + v^{<t>} \begin{bmatrix} \cos(\phi^{<t>}) \\ \sin(\phi^{<t>}) \\ 1 \\ \frac{1}{L}\tan(\delta^{<t>}) \end{bmatrix} dt$$

Unde $s^{<t>}$ este starea curentă a robotului, iar $\delta^{<t>}$ este mărimea unghi al direcției. Starea curentă este definită astfel:

$$s^{<t>} = [x, y, v, \phi]^{<t>}$$

Unde x, y sunt coordonatele carteziene alre robotului, *v* este viteza acestuia, iar ϕ este orientarea acestuia. *L* este lungimea vehiculului, iar *dt* perioada de eșantionare.
Procesul de estimare a stării robotului se realizează utilizând Filtrul Kalman, după cum este descris în continuare (pentru cazul unui robot mobil cu două motoare). Întâi se calculează numărul de impulsuri date de encoderul roților în timpul perioadei de eșantionare *dt*:

$$\delta_S^{<t+1>} = E_S^{<t+1>} - E_S^{<t>}$$

respectiv

$$\delta_D^{<t+1>} = E_D^{<t+1>} - E_D^{<t>}$$

Unde δ este numărul de impulsuri înregistrat în timpul perioadei de eșantionare, iar *E* este valoarea returnată de către encoder. Exponenții $< t + 1 >$ respectiv $< t >$ se referă la momentele de timp pentru care se calculează valorile, iar indicii *D* și *S* se referă la roata dreaptă, respectiv stângă.
Ulterior se calculează numărul de rotații pentru roata stângă, respectiv dreaptă:

$$R_S^{<t+1>} = \frac{\delta_S^{<t+1>}}{p_{RS}}$$

respectiv

$$R_D^{<t+1>} = \frac{\delta_D^{<t+1>}}{p_{RD}}$$

Unde R_S și R_D sunt numărul de rotații pentru roata stângă, respectiv dreaptă, iar p_{RS} și p_{RD} sunt numărul de impulsuri per rotație pentru roata stângă, respectiv dreaptă.
Având numărul de rotații, se calculează distanțele parcurse de cele două roți:

$$d_S^{<t+1>} = R_S^{<t+1>} C$$

respectiv

$$d_D^{<t+1>} = R_D^{<t+1>} C$$

Unde d_S și d_D sunt cele două distanțe parcurse, iar *C* este circumferința roților.
Rezultă viteza robotului:

$$v^{<t+1>} = \frac{1}{2}(d_S^{<t+1>} + d_D^{<t+1>})dt$$

Orientarea robotului este dată de busolă:

$$\phi^{<t+1>} = \phi_{IMU}^{<t+1>}$$

Unde ϕ_{IMU} este orientarea dată de busola senzorului inerțial IMU.
Având această estimare a stării robotului, se realizează pasul de predicție al Filtrului Kalman:

$$\hat{s}^{<t+1>} = F^{<t+1>} s^{<t>}$$

Unde $\hat{s}^{<t+1>}$ este starea prezisă, iar $F^{<t+1>}$ este matricea de tranziție a stării.
De îndată ce sunt disponibile măsurători de la GPS, se realizează pasul de update al Filtrului Kalman:

$$s^{<t+1>} = \hat{s}^{<t+1>} + K^{<t+1>} \hat{y}^{<t+1>}$$

Unde $\hat{y}^{<t+1>}$ depinde de vectorul măsurătorilor provenite de la GPS, *K* este amplificarea Kalman, iar $s^{<t+1>}$ este starea estimată post măsurătoare.
Pentru situația în care măsurătorile GPS nu sunt disponibile (navigație în spații închise), se utilizează Filtrul Kalman împreună cu măsurători provenite de la sistemul de odometrie vizuală bazat pe detecția markerilor ArUco.

Astfel, problema determinării poziției, respectiv a orientării agentului inteligent se transformă într-o problemă de Vedere Artificială. Markerii vizuali sunt șabloane generate pe calculator, având proprietatea de a fi ușor de detectat în imagini. Un asemenea marker prezintă suficiente puncte cheie (colțuri) astfel încât să se poată realiza estimarea poziției și a rotației sale în mediul înconjurător. Adițional faptului că sunt ușor de detectat în imagini datorită construcției lor, acești markeri codifică un identificator unic, în funcție de șablonul generat pentru fiecare în parte facilitând astfel identificarea lor în scena vizualizată.
Metoda de localizare bazată pe ArUco funcționează în felul următor:

1. Se aleg anumite puncte de referință în harta unde navighează robotul mobil.
2. Se plasează markeri vizuali orientați către direcția generală din care se va apropia robotul în acele puncte alese anterior.
3. Modulul de localizare al agentului stochează într-un dicționar corespondențele dintre identificatorul codificat de către fiecare marker, respectiv coordonatele în sistemul global de referință ale markerului.
4. În timpul navigării, agentul inteligent detectează utilizând camera câte un marker rând pe rând apoi realizează odometria vizuală bazată pe acești markeri.

În continuare este descris procesul de odometrie vizuală bazată pe ArUco. Pornind de la matricea parametrilor intrinseci ai camerei atașată vehiculului, totodată cunoscându-se coeficienții de distorsiune, respectiv dimensiunea fizică a markerilor ArUco, se construiește transformarea omogenă de coordonate pentru fiecare marker:

$$T_M^W = \begin{bmatrix} R_M & T_M \\ O_{1\times 3} & 1 \end{bmatrix}$$

Unde T_M^W este transformarea omogenă care descrie poziția markerului în sistemul de coordonate global, R_M este componenta de rotație, T_M este translația, iar $O_{1\times 3}$ este un vector de o linie și trei coloane cu valori nule.
Apoi se detectează utilizând funcția *detectMarkers* din librăria OpenCV markerul, dacă acesta este prezent în scena vizualizată. Folosind această funcție se obține matricea omogenă de

transformare care descrie coordonatele markerului relativ la cameră T_M^C. Ulterior se determină matricea de transformare care descrie coordonatele camerei relativ la marker:

$$T_M^C = {T_M^C}^{-1}$$

Întrucât sistemul de coordonate al camerei nu se află în sistem ISO, trebuie realizate două rotații: una în jurul axei *X*, cu $90°$, respectiv una în jurul axei Z, cu $90°$:

$$T_{YZ} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(90°) & -\sin(90°) & 0 \\ 0 & \sin(90°) & \cos(90°) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

respectiv

$$T_{XY} = \begin{bmatrix} \cos(90°) & -\sin(90°) & 0 & 0 \\ \sin(90°) & \cos(90°) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Avându-se acestea, se definește matricea omogenă pentru schimbarea coordonatelor din sistemul de coordonate al camerei în cel global:

$$T_{CM}^{W} = T_{XY} \cdot T_{YZ} \cdot T_{C}^{M}$$

Apoi se obțin coordonatele camerei în sistemul de coordonate global:

$$T_{W}^{C} = T_{M}^{W} \cdot T_{CM}^{W}$$

Obținând această matrice omogenă, se pot extrage coordonatele carteziene ale camerei în sistemul de coordonate global, precum și rotațiile pe cele trei axe. Acestea pot fi apoi integrate în vectorul măsurătorilor al Filtrului Kalman:

$$z_{A}^{<t+1>} = [x_A, y_A, 0, \phi_A, a_x, a_y, r_z]$$

Pasul de predicție al Filtrului Kalman se realizează având estimările stării date de modulul IMU , împreună cu modelul de mișcare în sine, pe când pasul de update se realizează cu ajutorul componentelor de poziționare, în funcție de disponibilitatea lor. Măsurătorile fie sunt calculate utilizând odometria vizuală, bazată pe camera montată pe robotul mobil, plus algoritmul de detecție al markerilor vizuali ArUco, fie de către modulul GPS cuplat cu busola.
Alternanța celor două sisteme de localizare se face atât la pasul de inițializare al sistemului, cât și pe parcursul desfășurării sarcinii de navigare autonomă. Acest lucru se face din pricina faptului că uneori robotul mobil pătrunde în anumite zone în care nu este semnal suficient de stabil al sistemului GPS, caz în care există sistemul de odometrie vizuală ca sistem redundant.

Modelul observațiilor senzoriale

Componenta principală a algoritmului propus în cadrul acestei teze se numește ObserveNet Control și este compusă dintr-un algoritm de inteligență artificială menit să prezică un model al observațiilor senzoriale ale unui robot mobil care navighează autonom. Aceasta utilizează date provenite de la componenta de percepție a sistemului, respectiv cea de localizare pentru a realiza predicțiile observațiilor senzoriale din viitor.
Observațiile senzoriale înregistrate de către robot sunt stocate în memoria augmentată a modulului de inteligență artificială, împreună cu traiectoriile locale generate, respectiv traiectoria globală de referință:

$$f_{MA}(z_{ref}, z, I) = \{I^{<t-i>}, \forall i \in [0, \tau_0]; z^{<t-i>} \forall i \in [0, \tau_i]; z_{ref}\}$$

Unde z_{ref} este traiectoria globală de referință, iar $z^{<t-i>}$ sunt traiectorii generate pentru secvențele din istoric.
Observațiile senzoriale istorice $I^{<t-\tau_i,t>}$ sunt trecute succesiv prin două straturi convoluționale. Astfel sunt extrase caracteristici de nivel înalt din secvențele de hărți de ocupare returnate de către memoria augmentată a modulului.
Matematic vorbind, straturile convoluționale realizează următoarea operație:

$$G(m,n) = f * h = \sum_{j} \sum_{k} h[j,k] f[m-j, n-k]$$

Unde *G* este rezultatul convoluției, f este structura de date peste care se aplică operația de convoluție, iar *h* este kernelul.

$$I_{CONV}^{<t-\tau_i,t>} = I^{<t-\tau_i,t>} * K_i, \forall i \in [0, M]$$

Unde M este numărul de kernels pentru stratul convoluțional respectiv.

Ulterior convoluțiilor, observațiile senzoriale sunt trecute prin două straturi complet conectate, respectiv un strat de flatten.

$$I_f^{<t-\tau_i,t>} = f_f(f_{FC}(I^{<t-\tau_i,t>}))$$

Observațiile senzoriale rezulte în urma acestor straturi sunt trecute prin straturile recurente LSTM:

$$\hat{I}^{<t+1,t+\tau>} = f_{LSTM}\left(I_f^{<t-\tau_i,t>}\right)$$

Adițional observațiilor istorice, ObserveNet procesează traiectoriile locale generate la momentele de timp $[t-\tau_i, t]$ corespunzătoare observațiilor înregistrate la acele momente de timp, precum și traiectoria globală de referință z_{ref}.

În afara straturilor diferențiabile care pot fi antrenate (complet conectate, convoluționale,

respectiv LSTM) aflate în componența algoritmului ObserveNet, acesta înglobează și anumite straturi non-diferențiabile, menite să asocieze fiecare observație prezisă cu cea mai probabilă stare viitoare a agentului inteligent. Stratul de Planificare și Simulare utilizează în mod recursiv planificatorul temporal pentru a simula stări viitoare ale robotului, în același timp replanificând traiectoria țintă *z*. Traiectoriile generate cu ajutorul observațiilor prezise simulate în cadrul stratului de planificare și simulare sunt filtrate folosind Filtrul Kalman.

Rețeaua neuronală aflată în centrul algoritmului ObserveNet a fost antrenată timp de 10000 de epoci în manieră self-supervised, folosind optimizatorul Adam și o rată de învățare de 0.0003. Funcția de cost minimizată în cadrul antrenării a fost Mean Squared Error.

Planificarea traiectoriei locale

Planificatorul temporal al traiectoriei este bazat pe DWA, modificat pentru a calcula iterativ traiectorii pentru vehiculul controlat în funcție de observații curente, respectiv istorice ale senzorilor. DWA este o strategie de evitare a coliziunilor pentru roboți mobili, care utilizează mișcările dinamice ale acestora, precum și diferite constrângeri impuse asupra vitezelor, respectiv accelerațiilor, pentru a calcula ruta optimă în planul 2D.

DWA a fost implementat pe baza planificatorului care se regăsește în ROS. Acesta primește ca intrări informații despre obstacolele prezente în scenă, sub forma distanței dintre vehiculul controlat și obstacole. În această lucrare, DWA a fost modificat să poată accepta atât observații curente ale vehiculului, cât și predicții.

Starea dorită a vehiculului este prezisă iterativ la fiecare perioadă de eșantionare din viitor, bazat pe dinamica temporală a vehiculului, respectiv pe observații. Datele de intrare sunt atât date istorice $< t-\tau_i >$, cât și date prezise $< t, t+\tau_0 >$:

$$z^{<t+i>} = f(\hat{I}^{<t-\tau_i,t+i>}, z^{<t+i-1>})$$

Unde *i* este pasul de timp pentru care s-a realizat predicția și $f(\cdot)$ este funcția de planificare temporală. Funcția este folosită pentru a interpola traiectoriile rezultate în urma calculului DWA de-a lungul orizontului de timp $[t-\tau_i, t+\tau_0]$ și poate fi descompusă în:

$$f(\cdot) = \sum_{t=-\tau_i}^{0} DWA(I^{<t>}, z^{<t>}) + \sum_{t=1}^{\tau_0} DWA(\hat{I}^{<t>}, \hat{z}^{<t>})$$

Unde primul termen reprezintă traiectoriile interpolate de către DWA pentru observațiile istorice, iar al doilea reprezintă interpolarea traiectoriilor bazate pe observații prezise. Stările sunt integrate în cadrul traiectoriei locale generată, iar aceasta este folosită ca intrare pentru modulul de control al algoritmului.

Controlul robotului mobil

Traiectoria generată de către planificatorul temporal este cea folosită de controlerul MPC pentru a executa mișcarea robotului mobil. MPC calculează o soluție iterativă pentru probleme de control optim pe un orizont de predicție finit.
Pentru a putea rula MPC, întâi se definesc mărimile necesare, începând cu intrările sistemului (mărimile de comandă):

$$u = [a^{<t>} \quad \delta^{<t>}]$$

Unde $a^{<t>}$ este accelerația, iar $\delta^{<t>}$ este unghiul direcției.
Apoi se definește numărul de pași pentru care se face optimizarea (e.g. *T=10*). Variabilele diferențiale pentru care se construiește sistemul sunt definite de starea vehiculului:

$$s^{<t>} = [x, y, v, \phi]^{<t>}$$

În continuare se aleg mărimile pe baza cărora se construiește funcția de cost, care va fi optimizată cu ajutorul algoritmului MPC. Acestea sunt viteza de referință, obținută din

traiectoria globală de referință (v_{ref}), eroarea de orientare (e_ϕ), respectiv abaterea de la traseul de referință (e_{ct}). Apoi se definește un set de ponderi pentru aceste mărimi de eroare:

$$w = [1.0 \quad 3.0 \quad 10.0]$$

Unde aceste ponderi au fost alese empiric, astfel încât MPC să aibă un comportament considerat satisfăcător.
Ulterior se inițializează matricea stărilor X_{T,N_S}, respectiv a mărimilor de control U_{T,N_U}, unde N_S este dimensiunea vectorului de stare al robotului, iar N_U este dimensiunea vectorului mărimilor de control.
După pasul de inițializare a variabilelor, urmează pasul de optimizare a sistemului. Se aleg cele mai apropiate T stări de pe traiectoria de referință, față de robotul mobil, din punct de vedere al distanței euclidiene:

$$P_R = \{z_i, z_i \in z_{ref}^{<t-\infty, t+\infty>}, \forall i \in [0, T]\}$$

Unde z_i este starea curentă de pe traiectoria de referință, iar z_{ref} este traiectoria globală de referință.
După ce au fost selectate coordonatele carteziene aparținând stărilor selectate din cadrul traiectoriei de referință, acestea au fost translatate în originea vehiculului și respectiv rotite cu unghiul acestuia:

$$P_R = \{(x_i - x_{veh}, y_i - y_{veh}), \forall i \in [0, T]\}$$

respectiv:

$$P_R = R_{-\phi} P_R$$

Unde x_i, y_i sunt coordonatele carteziene ale stării i aleasă de pe traiectoria de referință, x_{veh}, y_{veh} sunt coordonatele carteziene ale robotului mobile, iar $R_{-\phi}$ este matricea de rotație necesară pentru a roti punctele cu inversul orientării vehiculului.
Punctele translatate și rotite sunt apoi folosite pentru a realiza regresia polinomială pentru un polinom de gradul 4, prin metoda celor mai mici pătrate. Astfel se definește polinomul:

$$\hat{y} = X\hat{\beta}$$

Unde $X = [1 \quad x \quad x^2 \quad x^3 \quad x^4]$
Iar $\hat{\beta}$ este matricea coeficienților polinomului. Aceasta se obține așadar:

$$\hat{\beta} = (X^T X)^{-1} X^T \hat{y}$$

Având acestea, se pot defini mărimile de eroare e_{ct}, e_{ϕ}, respectiv funcția de cost MPC:

$$J_{MPC} = K_{ct} e_{ct} + K_{\phi} e_{\phi} + K_v v_{ref}$$

Unde K_{ct}, K_{ϕ}, K_v sunt ponderile asociate mărimilor de eroare definite anterior.
În continuare au fost definite constrângeri pentru mărimile de comandă:

$$a_{min} < a < a_{max}$$

respectiv:

$$\delta_{min} < \delta < \delta_{max}$$

Având de asemenea aceste constrângeri, se asignează ca stare inițială a algoritmului MPC starea curentă a vehiculului și se execută optimizarea, utilizând algoritmul IPOpt.

Particularități ale algoritmului ObserveNet pentru informații RGB-D

Deși structura generală a algoritmului ObserveNet Control este similară atât pentru cazul datelor senzoriale provenite de la LiDAR, cât și pentru cele provenite de la camera RGB-D, există totuși anumite diferențe.
O altă implementare a algoritmului ObserveNet Control care realizează predicția pentru observații senzoriale ale unui agent inteligent se bazează pe imagini de adâncime achiziționate cu ajutorul unui senzor RGB-D, spre deosebire de implementarea prezentată anterior, care utilizează date provenite de la senzorul LiDAR. În continuare vor fi descrise particularitățile algoritmului de predicție al datelor senzoriale provenite de la senzorul RGBD, precum și particularitățile sistemului complet de navigare autonomă construit în jurul acestui algoritm de predicție.
Din punct de vedere principial, arhitectura ObserveNet Control care operează folosind date RGB-D este similară cu varianta LiDAR. Așadar, problema pe care o tratează această arhitectură este următoarea: dându-se o serie de observații istorice $\Theta^{\{<t-\tau,t-1>\}}$, observația curentă $\theta^{<t>}$, toate provenite de la senzorul RGB-D, starea curentă a agentului inteligent $s^{<t>} = [x, y, v, \phi]^{<t>}$, precum și traiectoria globală de referință $z_{ref}^{<t-\infty,t+\infty>}$, obiectivul este de a construi o rețea neuronală recurentă care poate codifca în cadrul straturilor sale ascunse dinamica mediului înconjurător și bazat pe observațiile istorice prezentate rețelei, aceasta poate prezice observații din viitor pentru o fereastră de timp finită $[t+1, t+n]$. Acest model de inteligență artificială are rolul de a construi o *părere* a agentului despre mediul înconjurător, urmând ca această părere să fie utilizată de către algoritmul de planificare a traiectoriei locale pentru a genera o traiectorie optimă lipsită de coliziuni pentru agent $z_L^{<t+1,t+n>}$.

Codificarea dinamicilor prezente în scenă implică procesarea datelor provenite de la senzorul RGB-D, pentru a crea o reprezentare a scenei curente, de asemenea pentru a prezice evoluția acestor dinamice de-a lungul unui orizont finit de timp. Rețelele neuronale LSTM sunt potrivite pentru codificarea dinamicilor scenei datorită abilității lor de a învăța dependențe temporale pentru datele de intrare și de a folosi aceste dependențe pentru a prezice stări viitoare ale sistemului.

Modelul de codificare a mediului înconjurător este bazat pe o rețea neuronală recurentă tip LSTM, care conține mai multe celule destinate procesării seriilor temporale de date sub forma observațiilor istorice de lungimeτ. Ieșirea acestui model de inteligență artificială este o secvență de observații prezise din viitor de lungime n.

Informațiile de adâncime provenite de la sistemul RGB-D nu sunt introduse direct în rețea, ci sunt înainte de toate preprocesate. Astfel, datele achiziționate utilizând senzorul RGB-D trebuie utilizate pentru a recrea structura lor tridimensională din sistemul de coordonate global.

$$P_{3D} = \{[X_i, Y_i, Z_i], \forall i \in D_{img}\}$$

Unde X_i, Y_i, Z_i sunt coordonatele tridimensionale ale punctelor P, iar D_{img} este imaginea de adâncime. Pentru fiecare punct din imaginea de adâncime este calculat punctul în sistemul de coordonate global:

$$\begin{cases} X = (j - c_x)\dfrac{z}{f_x} \\ Y = (j - c_y)\dfrac{z}{f_y} \\ Z = z \end{cases}$$

Unde j, i sunt coordonatele unui pixel din imaginea de adâncime, f_x, f_y sunt coordonatele distanței focale, iar z este valoarea pixelului din imaginea de adâncime de la coordonatele (i, j).

Apoi, similar cazului de procesare a informațiilor LiDAR, se realizează operația de *thresholding*:

- Se elimină punctele aflate la o distanță prea mare sau prea mică față de originea vehiculului.
- Se elimină punctele aflate la o înălțime mai mare decât cea a robotului, respectiv cele aflate în planul drumului.
- Se schimbă reprezentarea din structură tip nor de puncte în structură tip Voxel Grid.
- Se realizează proiecția top-down a structurii obținute la pasul anterior.
- Datele sub formă de voxel grid bidimensionale se acumulează în secvențe de date istorice, care vor fi procesate ulterior de către rețeaua neuronală recurentă.

Aceste secvențe de date sunt apoi restructurate în forma necesară pentru a fi prelucrate de către rețeaua LSTM. Aceasta are ca ieșiri secvențe de date de lungime n, reprezentând predicțiile observațiilor senzoriale pentru cei n pași în viitor.

Predicțiile rețelei LSTM sunt apoi trecute prin straturi succesive complet conectate, respectiv trimise spre modulul de *clustering*, pentru a fi grupate în *clustere*, utilizând metoda DBSCAN. Modulul de clustering are rolul de a transforma predicțiile rețelei sub forma voxelilor la fiecare pas de timp $t + i$ în liste de centroide ale obstacolelor plus dimensiunile acestora. De asemenea, se folosește metoda *Log Odds* pentru filtrarea acestor centroide.

Rețeaua neuronală recurentă a fost antrenată în manieră auto-supervizată timp de 1000 de epoci, utilizând optimizatorul Adam cu o rată de învățare $\eta = 0.001$, folosind secvențe istorice

de dimensiune $\tau = 50$, având dimensiunea orizontului de predicție $n = 30$, echivalente a 5, respectiv 3 secunde. De asemenea, a fost folosită o funcție de cost non-standard asimetrică, datorită structurii datelor de intrare/ieșire ale rețelei.
Pentru controlul robotului mobil a fost folosit algoritmul Zero Moment Point (ZMP), împreună cu planificatorul de traiectorie locală DWA.
Traiectoriile candidat ale DWA sunt apoi evaluate conform următoarelor criterii de performanță:

- Costul de apropiere de destinație e_G, calculat ca distanța euclidiană dintre ultima stare a treiectoriei candidat și coordonatele punctului destinație din traiectoria globală de referință.
- Costul de orientare e_ϕ, care reprezintă diferența dintre orientarea stării finale de pe traiectoria candidat și cea a stării destinație
- Costul de viteză e_v, care reprezintă diferența dintre viteza de referință înregistrată pe traiectoria globală, respectiv viteza robotului.
- Costul de traiectorie netedă e_S, care reprezintă diferența dintre două seturi succesive de comenzi de accelerație și unghi al direcției.
- Costul de apropiere față de traiectoria globală de referință, care denotă apropierea traiectoriei locale generate de cea globală de referință.

Rezultate experimentale

Din punct de vedere al platformei software, cadrul general care a facilitat dezvoltarea algoritmilor propuși este platforma ROVIS, atât ca sistem de operare pentru platformele hardware folosite, cât și ca mediu de dezvoltare pentru algoritmi de inteligență artificială.
Sistemul de operare pentru roboți ROVIS este construit modular, având componente dedicate pentru senzori folosiți de către vehicule autonome, spre exemplu LiDAR, camere RGB, RGB-D, ultrasonici, etc. Sunt suportați și senzori dedicați pentru localizarea vehiculului autonom, precum IMU, GPS sau busolă.
Acesta rulează nativ în C++, dar permite și achiziția de date prin intermediul programelor terțe, precum ROS sau CARLA, având implementate interfețe pentru acestea.
De asemenea, oferă suport pentru diverși algoritmi de procesare a datelor senzoriale pentru operații de VA sau DL, cum ar fi filtrarea imaginilor, segmentarea acestora, pregătirea datelor pentru algoritmi de inteligență artificială, ș.a.m.d.
ROVIS funcționează atât în timp real, conectat la un simulator precum CARLA, sau la un robot mobil, cât și în mod simulat pe calculator, pentru a testa off-line algoritmii dezvoltați. Astfel, pot fi înregistrate date în timpul operării unui robot, urmând ca apoi aceste date să fie folosite în mod simulat, fără a fi necesară configurarea robotului. De asemenea, sunt suportate baze de date consacrate pentru testarea algoritmilor de inteligență artificială, precum nuScenes sau KITTI.
Componentele dedicate pentru navigarea autonomă au fost implementate folosind arhitectura ROVIS, respectând structura generală, implementând interfețele dedicate pentru achiziția de date, localizarea agentului, planificarea traiectoriei, controlul inteligent și controlul actuatorilor robotului mobil simulat sau real.

Astfel, exceptând variații de implementare specifice fiecărei platforme cu ajutorul căreia s-au realizat experimentele, algoritmii implementați au fost aceiași.
Experimentele au fost realizate în cadrul a trei platforme de test diferite: mediul simulat GridSim, mediul simulat CARLA și platforma robotică Agile Scout-X. În cadrul tuturor celor trei platforme, agentul inteligent controlat cu ajutorul ObserveNet Control trebuia să navigheze autonom, punându-se accent pe situații dificile asociate acestei sarcini.
Astfel au fost definite scenarii de test dedicate pentru fiecare platformă folosită pentru experimente, având roulul de a evidenția performanța agentului inteligent în cadrul situațiilor dificile întâlnite în timpul navigării autonome.

Mediul simulat GridSim

GridSim este un simulator pentru navigare autonomă simplist, în care o arhitectură de tip robot mobil este folosită la generarea hărților de ocupare cu ajutorul senzorilor simulați. Acesta a fost implementat în Python și permite integrarea algoritmilor de inteligență artificială în scopul testării. GridSim permite simularea senzorilor de distanță având un FOV configurabil, putând fi atașați atât în fața vehiculului cât și în spatele acestuia.
Fiind un sistem simplist de simulare a situațiilor de conducere autonomă, GridSim a fost folosit pentru a genera date sintetice ale senzorilor simulați, specifice unui scenariu de navigare pe autostradă. Vehiculul a fost condus manual, în timp ce au fost achiziționate date de antrenare pentru algoritm. Scenariul constă în două vehicule care navighează în paralel în linie dreaptă. Vehiculul principal se deplasează cu viteză constantă, în tip ce al doilea vehicul se deplasează cu viteză variabilă.
Vehiculul principal simulează date senzoriale în timp ce al doilea vehicul intră, respectiv iese din câmpul vizual al vehiculului principal, depășindu-l, respectiv fiind devansat. Aceste date senzoriale sunt stocate și procesate în secvențe istorice de dimensiune$N$, respectiv secvențe de predicții de dimensiune P.
Pentru a aduna date de antrenare, agentul inteligent a fost condus manual, în timp ce un al doilea vehicul a fost controlat automat, variind viteza și distanța față de agent.
Agentul înregistrează date de la senzorii simulați în timp ce alt vehicul trece prin câmpul vizual al acestora, în timp ce depășește sau este depășit. Datele înregistrate sunt apoi procesate în secvențe de lungime N (dimensiunea observațiilor istorice), iar viteza agentului inteligent este stocată în secvențe de lungime P, unde P este dimensiunea orizontului de predicție.
ObserveNet Control a fost antrenat în manieră self-supervised timp de aproximativ 1000 de epoci, având *checkpoints* care conțin ponderile intermediare ale modelului salvate, respectiv având reducerea ratei de învățare, precum și *early stopping*. Pentru a varia datele de antrenare, al doilea vehicul a fost poziționat aleator la începutul fiecărei sesiuni de înregistrare de date.
Pentru antrenarea ObserveNet Control au fost colectate 33000 de secvențe de observații, fiecare având lungimea $N = 150$. Modelul a fost folosit pentru a prezice pentru un orizont $P = 10$ cadre consecutive din viitor, folosind 30 de raze simulate, fiind ulterior evaluat utilizând 6700 de cadre de test, obținându-se mărimi statistice ale erorilor, evaluate per-pixel.
Performanța algoritmului ObserveNet au fost evaluate din punct de vedere al erorii statistice procentuale. Astfel s-a putut observa în cadrul lucrării performanța sistemului pentru acest simulator.

Mediul simulat CARLA

Algoritmul prezentat în teză a fost testat în mediu simulat utilizând platforma destinată evaluării algoritmilor de navigare autonomă denumită CARLA. Acest simulator a fost proiectat pentru a permite cercetătorilor să implementeze, antreneze, respectiv valideze sisteme de navigare autonomă. Adițional, simulatorul pune la dispoziție texturi pentru scene urbane, clădiri și vehicule pentru dezvoltatori. De asemenea, CARLA are implementări pentru diferiți senzori folosiți pentru navigarea autonomă, precum senzori RGBD, LiDAR, ultrasonici, camere RGB și RADAR. Totodată, CARLA are suport pentru localizarea agentului inteligent, generând date GPS simulate.

Din punct de vedere al conectivității cu aplicații dezvoltate de către cercetători, CARLA expune un API care permite atât controlul vehiculelor, cât și al condițiilor de simulare, precum condiții meteo, comportament configurabil al pietonilor și vehiculelor automatizate, ș.a.

Simulatorul a fost construit folosind Unreal Engine și este bazat pe o arhitectură client-server, unde multiple noduri se pot conecta simultan la un server care rulează simularea.

Un agent inteligent poate fi configurat pentru a fi integrat în simulare în mai multe moduri. Simulatorul este flexibil din acest punct de vedere, suportând implementări pentru agenți în C++, precum și Python. De asemenea, pune la dispoziția utilizatorului și un bridge pentru ROS.

Fiind un simulator dedicat pentru algoritmi de navigare autonomă, CARLA a permis rularea întregului lanț de procesare al algoritmului propus. Astfel, au fost realizate interfețe pentru senzorii simulați implementați în CARLA, pentru modulul de localizare din simulator, precum și pentru comenzile vehiculului. Comunicația între ROVIS și CARLA s-a realizat în manieră client-server, unde clientul a fost agentul inteligent.

Agentul autonom a fost implementat urmărind o structură de tip perception-plan-control, în cadrul căreia fiecare sarcină asociată navigării autonome este încapsulată în propriul modul. Astfel, au fost utilizate următoarele module pentru realizarea acestui experiment: modulul de percepție, cel care înglobează algoritmul de inteligență artificială, cel de estimare a stării, cel de planificare, respectiv cel de control.

Modulul de percepție, care în acest caz utilizează date senzoriale provenite de la LiDAR, are rolul de a achiziționa, respectiv procesa secvențele de date organizate sub forma norilor de puncte. Acesta se conectează direct la API-ul expus de CARLA pentru a obține acces la norii de puncte returnați de către simulator.

Modulul de localizare, bazat pe procesarea datelor GPS, conectat la API-ul corespunzător din CARLA este folosit pentru a obține date despre starea curentă a agentului, pe care le transmite mai departe componentei de planificare a traiectoriei.

Modulul de inteligență artificială procesează secvențele de nori de puncte furnizate de către modulul de percepție, aranjându-le în secvențe de date istorice, urmând să le aplice rețelei neuronale recurente, care are rolul de a realiza predicția observațiilor senzoriale din viitor. Observațiile prezise, împreună cu observația senzorială curentă, precum și informațiile de localizare sunt utilizate de planificatorul traiectoriei, care are rolul de a genera traiectoria optimă pentru agentul inteligent.

Controllerul primește ca intrare traiectoria generată de către planificator, precum și starea curentă a agentului și generează semnalele de comandă corespunzătoare, astfel încât agentul

să înainteze către destinația sa. Această componentă este conectată prin intermediul API-ului CARLA direct la simulator, pentru a putea controla vehiculul simulat.
În cadrul experimentelor în simulatorul CARLA, s-a pus accent pe modelarea mediului înconjurător al agentului inteligent. Astfel, s-a verificat comportamentul algoritmului de predicție al observațiilor senzoriale sub forma datelor LiDAR, precum și a sistemului întreg de navigare autonomă în cadrul situațiilor limită din trafic.
Astfel, s-au definit mai multe scenarii în care s-a validat comportamentul agentului inteligent: scenariul de depășire, cel cu trafic venind de pe contrasens, cel de apropiere din lateral, cel în care vehiculul este depășit, respectiv cel cu multipli agenți prezenți în scenă.
Scenariul de depășire reprezintă o provocare, în special la viteză mare. Celălalt vehicul accelerează mai puțin decât agentul inteligent, astfel creând necesitatea manevrei de depășire.
Scenariul de trafic de contrasens presupune ca agentul să meargă înainte, iar din sens opus se apropie alt vehicul, circulând lipit de linia de delimitare a sensurilor. Astfel se creează o situație periculoasă pentru agent. Utilizarea predicțiilor senzoriale ar trebui să genereze o traiectorie mai sigură pentru vehicul.
Scenariul de trafic lateral presupune ca agentul să pătrundă într-o intersecție, iar din partea dreaptă, respectiv din cea stângă să pătrundă alt vehicul în intersecție, provocând un accident. Utilizarea predicțiilor ar trebui să facă posibilă evitarea coliziunilor cu ceilalți participanți la trafic.
Pentru situația în care agentul este la rândul său depășit, acesta trebuie să utilizeze predicțiile senzoriale pentru a planifica traiectoria în așa fel încât să se depărteze de vehiculul care depășește.
Scenariul multi-vehicul este cel mai complex mediu de testare din simulator. Vehiculul trebuie să se deplaseze pe un traseu mai lung, timp în care interacționează cu mai multe vehicule, depășind unde este cazul, totodată evitând coliziuni. Acest scenariu replică cel mai bine situații complexe din lumea reală, unde vehiculul autonom are spațiu limitat de manevră.
Algoritmul de navigare autonomă a trebuit să înglobeze pe lângă logica de evitare a coliziunilor bazată pe observațiile senzoriale curente, observațiile senzoriale prezise de către algoritmul de inteligență artificială.
Performanța algoritmului de navigare autonomă a fost comparată cu algoritmul clasic DWA. Evaluarea a constat în încărcarea succesivă a celor cinci scenarii de conducere agresivă descrise anterior, în cadrul simulatorului CARLA. Traiectoria globală de referință a fost înregistrată controlând manual agentul inteligent în mediul simulat.
Totodată au fost înregistrate stările succesive ale agentului pentru a putea fi folosite ulterior.
Ca indicatori de performanță pentru algoritm, s-au înregistrat numărul de coliziuni dintre agentul inteligent și ceilalți participanți din trafic N_C, abaterea medie de la traseul de referință e_{ct}, eroarea de orientare a vehiculului e_ϕ, precum și rata schimbării mărimilor de comandă accelerație, respectiv unghi al direcției: e_a și e_δ. De asemenea, a fost notată cu $C_{100\%}$ abilitatea vehiculului de a finaliza traseul de test.
În primă fază, algoritmul ObserveNet Control a fost comparat cu DWA clasic, acesta obținând performanțe superioare din punct de vedere al criteriilor de performanță descrise anterior.
Similar, a doua comparație cu două metode tip Imitation Learning (Learning by Cheating, respectiv World on Rails) a produs rezultate superioare pentru ObserveNet Control.

Mediul simulat CARLA – Autonomous Driving Learderboard

A doua metodă de evaluare din cadrul simulatorului CARLA s-a realizat utilizând platforma CARLA autonomous leaderboard. Aceasta este o platformă open-source dedicată testării și comparării algoritmilor de navigare autonomă, punând la dispoziția dezvoltatorilor mai multe scene urbane, respectiv mai multe condiții de trafic și condiții meteorologice. Traseele de testare a algoritmilor simulează mai multe scenarii provocatoare, precum pierderea controlului vehiculului, evitarea obstacolelor, schimbarea benzilor de circulație, intersecții semaforizate, pietoni care traversează neregulamentar, etc. Aceste scenarii sunt folosite pentru evaluarea performanței prin constituirea unui scor total al conducerii, procentajul de completare al traseului și penalizări pentru infracțiuni (e.g. coliziuni).
Deoarece algorimul propus are ca obiectiv reducerea numărului de coliziuni, respectiv optimizarea deviațiilor de la traiectoria de referință, dar și optimizarea ratelor schimbării pentru comenzile vehiculului, datorită conceptului folosit pentru controlul vehiculului, bazat doar pe informații senzoriale provenite de la LiDAR, nu s-au luat în considerare scorurile pentru infracțiuni care presupun utilizarea algoritmilor de VA, respectiv a algoritmilor de detecție a obiectelor, precum încălcări ale marcajului rutier sau încălcări ale semnificației luminilor semafoarelor.
Algoritmul propus a fost evaluat în comparație cu trei algoritmi renumiți din domeniu, respectiv DWA, Learning by Cheating și World on Rails. Primul este un algoritm clasic utilizat în special pentru navigarea autonomă în domeniul roboticii, iar următorii doi sunt implementări *state of the art* ale paradigmei imitation learning.
Rezultatele experimentale prezintă îmbunătățiri ale performanței agentului autonom din punct de vedere al numărului de coliziuni, respectiv al calității generării traiectoriei, precum și al modului de control al agentului, față de algoritmi de actualitate folosiți în domeniu.

Robotul mobil AgileX Scout

Platforma robotică AgileX Scout este destinată navigării în medii dificile, având tracțiune integrală și având limitare pentru masa maximă care poate fi transportată de 50 de kilograme. Acest robot poate fi controlat din telecomandă, sau poate fi adaptat pentru a rula algoritmi de navigare autonomă. Viteza maximă de deplasare a sa este de 1.5m/s și este rezistent la intemperii conform standardului IP22. Acesta este echipat cu o baterie de 30Ah, care operează la o tensiune de 24V, având o autonomie de 15km (fără încărcătură) și vine echipat cu patru motoare de curent continuu de 400W.
Din punct de vedere al comunicației, acesta oferă dezvoltatorilor acces la rețeaua CAN, punând totodată la dispoziția lor funcționalități pentru controlul *low-level* organizate sub forma API-urilor dedicate, scrise în limbajul C++.
Pentru implementarea algoritmilor de navigare autonomă, robotului i-au fost atașate următoarele:

- Un senzor LiDAR de 360° Pandar 40 Hesai.
- O unitate IMU VESC.
- Patru camere sincronizate e-CAM130A pentru realizarea perspectivei vizuale de360°.
- O unitate tip GPS receiver pentru localizare.

- O unitate de calcul nVIDIA AGX Xavier care rulează sistemul de operare Linux.

Datorită versatilității sale și a modului de construcție, robotul AgileX Scout a permis montarea sistemului de calcul menționat anterior, precum și a senzorilor necesari pentru algoritmii de inteligență artificială. Astfel, au putut fi rulați diverși algoritmi, de la cei clasici de control, până la algoritmi sofisticați de inteligență artificială.

Sistemul de operare ROVIS, peste care a fost implementat algoritmul de navigare autonomă a fost realizat pe platforma Linux care rulează pe nVIDIA Xavier. Comunicația între agent și platforma robotică este realizată de către sistemul de operare al PC-ului, prin intermediul API-ului pus la dispoziție de către producătorul platformei robotice. Platforma robotică AgileX Scout, împreună cu senzorii, unitatea de calcul nVIDIA și respectiv sistemul de operare ROVIS care rulează pe aceasta poartă denumirea de RovisLab AMTU.

Spre deosebire de experimentele realizate în mediu simulat, descrise anterior, în cazul experimentării folosind platforma RovisLab AMTU, mediul de testare este complet diferit, atât datorită utilizării unei platforme hardware versus cea software folosită în simulator, precum și datorită problemei de navigare în sine. Comparativ cu scenariile simulate anterior, unde mediul înconjurător a fost de tip structurat, sub forma unui scenariu de navigare în oraș, în acest caz robotul mobil trebuie să navigheze într-un mediu nestructurat, în condiții de laborator.

Plecând de la această premisă, problema de navigare autonomă se schimbă datorită mediului înconjurător, cu precădere datorită tipurilor diferite de obstacole la care se poate "aștepta" agentul inteligent.

Astfel, deși fundamental vorbind procesul de navigare autonomă ar trebui să fie identic, particularitățile acestui mediu înconjurător în cadrul căruia s-a realizat navigarea pentru acest experiment creează dificultăți algoritmilor bazați pe inteligență artificială.

Obiectivul principal pentru algoritmul propus este ca agentul inteligent, fie simulat cu ajutorul unui software special, fie implementat pe o platformă hardware reală, să poată naviga între punctul de start, respectiv punctul destinație, evitând coliziuni cu obstacole statice, respectiv obstacole dinamice din mediul înconjurător.

În scopul acestui experiment, au fost definite două traiectorii de referință înregistrate utilizând sistemul AMTU controlat manual *în spațiu închis*, respectiv *în spațiu deschis*, pe care agentul inteligent trebuie să le urmărească, fără a realiza coliziuni cu obstacole prezente în mediul înconjurător. Dat fiind faptul că evitarea obstacolelor statice este realizată cu succes și de către algoritmul clasic DWA, se pune accentul pe obstacole care au o dinamică pronunțată, respectiv operatori umani în acest caz.

Au fost achiziționate date de antrenare în număr de aproximativ 38000 de perechi de secvențe LiDAR istorice și din viitor, atât înăuntru, cât și în afara institutului de cercetare al Universității Transilvania.

Similar cu cazul simulat bazat pe CARLA, s-au folosit cinci scenarii pentru a valida comportamentul algoritmului propus în situații limită. Algoritmul a fost comparat cu DWA, iar platforma AMTU a trebuit să navigheze de-a lungul traiectoriilor înregistrate în spațiul închis, respectiv afară.

Rezultatele din punct de vedere calitativ ale evaluării performanței agentului inteligent implementat pe AMTU sunt superioare celor bazate pe metoda clasică DWA.

Robotul mobil Unitree A1

Seria finală de experimente a fost realizată cu ajutorul platformei robotice Unitree A1. Aceasta integrează ca senzori principali pentru percepție o cameră RGB, respectiv una RGBD. Acest robot permite o masă maximă transportată de 5kg, are o viteză maximă a motoarelor atriculațiilor de $21\ rad/s$ și permite controlul individual al fiecărui motor al articulațiilor, acestea putând fi comandate în cuplu, viteză, respectiv poziție. Viteza maximă de deplasare a robotului este de $3.3m/s$. Are o construcție dură și ușoară și poate fi alimentat la o tensiune de 5V, 12V sau 19V. Ca interfețe externe, acesta prezintă patru porturi USB, două porturi HDMI, precum și două porturi Ethernet.

Spre deosebire de experimentele desfășurate pe baza platformei AgileX Scout, în cazul robotului Unitree A1 a fost utilizat senzorul de adâncime pentru modelarea observațiilor senzoriale.

Robotul mobil a navigat în mediu nestructurat, mai exact pe drumuri forestiere, în timp ce situații de coliziuni cu oameni au fost simulate. Între timp, au fost achiziționate date de antrenare sub forma observațiilor provenite de la senzorul de adâncime. Totodată a fost înregistrată și traiectoria de referință z_{ref} pe care ulterior acesta trebuie s-o urmărească.

Pentru evaluarea algoritmului utilizând robotul Unitree A1, au fost folosite secvențe de date înregistrate în timpul navigării robotului pe drumuri forestiere. Modul de desfășurare al experimentelor poate fi sintetizat în felul următor:

- Au fost achiziționate date de antrenare, respectiv testare în timpul controlului manual al robotului. În acest timp, au fost simulate situații de potențiale coliziuni cu oameni aflați în preajma robotului.
- Modelul observațiilor robotului a fost antrenat utilizând datele achiziționate anterior, preprocesate corespunzător.
- Planificatorul traiectoriei locale a fost evaluat în baza datelor de test achiziționate.

A fost definită o distanță de siguranță $s = 0.5m$ pentru robot. Orice încălcare a acestei distanțe, însemnând că robotul s-a apropiat de un obstacol la o distanță mai mică decât s, a fost încadrată drept coliziune. Numărul coliziunilor a fost cuantificat, astfel rezultând prima măsură calitativă a algoritmului: N_C.

A doua mărime evaluativă a fost abaterea medie de la traseu e_{ct}, definită ca diferența dintre traiectoria de referință z_{ref} și traiectoria locală generată.

$$e_{ct} = \frac{1}{n}\sum_{i=0}^{n} |f(x) - y_i|$$

Unde $f(x)$ este aproximarea polinomială a traiectoriei generate, evaluată în punctul x.

A treia măsură a calității a fost eroarea de orientare:

$$e_{\phi} = \frac{1}{n}\sum_{i=0}^{n} |\phi^{<i>} - \phi_d^{<i>}|$$

Unde $\phi^{<i>}$ este orientarea robotului la momentul de timp i, iar $\phi_d^{<i>}$ este orientarea referință, de pe traiectoria globală.

Pentru a simula situații de coliziune pentru robot, s-au realizat experimente cu oameni interacționând cu acesta în timpul navigării sale. Interacțiunile au constat din apropierea

oamenilor față de robot din diferite unghiuri și cu viteză variabilă, pentru a simula obstacole dinamice în scenă.
Au fost realizate patru scenarii de test: coliziune frontală, coliziune laterală stânga, coliziune laterală dreapta, respectiv multiple obstacole.
Pentru primul scenariu, o singură persoană s-a apropiat de robot din fața acestuia, în timp ce pentru coliziunea laterală stângă, respectiv dreaptă, persoana s-a apropiat din respectivele direcții. Pentru scenariul cu multiple obstacole, scenele au conținut câte două persoane.
Utilizând cele patru scenarii descrise în secțiunea anterioară, au fost realizate două teste comparative pentru algoritmul propus. În primă fază, acesta a fost comparat cu implementarea clasică a algoritmului DWA, special adaptat pentru robotul Unitree A1. Rezultatele comparative au fost superioare pentru ObserveNet Control.
Adițional comparației algoritmilor de planificare a traiectoriei, s-a realizat de asemenea comparația acurateții pentru procesul de detecție a coliziunilor între algoritmul de predicție al observațiilor și un detector de obiecte *state of the art*, YoloV7. Această comparație s-a realizat pentru acuratețe A, rata falselor pozitive FP_R și timpului estimat până la coliziune ttc.
Rezultatele acestei comparații arată performanță mai bună pentru ObserveNet Control.

Studiu de ablație privind algoritmul de control

A fost realizat un studiu de ablație referitor la algoritmul de control folosit pe robotul mobil AgileX Scout. Au fost comparate NMPC, respectiv DWA, în cadrul navigării robotului pe cinci trasee de test. NMPC realizează un control mai bun pentru robotul mobil, ca urmare acesta urmărește mai bine traiectoria globală.
Cei doi algoritmi de control au fost comparați din punct de vedere al abaterii medii de la traseul de referințăe_{ct}, precum și al erorii de orientaree_{ϕ}. Din punct de vedere al abaterii medii de la traseul de referință, NMPC a prezentat valori mai bune în toate cele cinci trasee de test, fapt datorat caracterului predictiv al respectivului algoritm de control și al modelării mai bune a sistemului.
De asemenea, a fost realizat un studiu cu privire la impactul dimensiunii orizontului de predicție T asupra performanței algoritmului NMPC, din punct de vedere al celor două mărimi menționate anterior:e_{ct}, respectiv e_{ϕ}. Concluzia studiului este că mărirea orizontului de predicție peste 10 secunde nu aduce îmbunătățiri semnificative ale performanței sistemului de control.

Studiu de ablație privind predicția observațiilor RGB-D

În scopul evaluării performanței componentelor algoritmului, s-au realizat trei studii de ablație. În cadrul primul studiu, au fost comparate trei funcții de cost pentru rețeaua neuronală: Mean Squared Error (MSE), Mean Absolute Error (MAE), respectiv o funcție de cost asimetrică. Rezultă faptul că funcția de cost asimetrică converge cel mai repede dintre cele trei, respectiv la cea mai mică valoare.

$$L(v, \hat{v}) = \frac{1}{n}\sum(\widehat{v_i} - v_i)^2[(1 - 1_+)\alpha + 1_+(1 - \alpha)]$$

Funcția de cost asimetrică a fost utilizată pentru antrenarea rețelei neuronale recurente datorită posibilității modificării ei astfel încât, prin utilizarea unui factor de scalare, costul total

pentru antrenare să poată să penalizeze situații în care valorile prezise pentru observații capătă valoarea logică 1, deși ar trebui să fie 0. Practic, se dorește ca funcția de cost să penalizeze rețeaua mai puternic atunci când aceasta marchează o celulă din harta de ocupare în mod eronat ca fiind ocupată, când ea este de fapt liberă.
În urma experimentării, a fost aleasă valoarea de 0.1 pentru factorul de scalare al funcției de cost asimetrică, aceasta reaalizând cea mai bună performanță.
A doua ablație realizată a fost cea a dimensiunii orizontului de predicție. S-a monitorizat performanța sistemului variind dimensiunea în secunde a orizontului, măsurând totodată timpul până la coliziune ttc, respectiv rata falselor pozitive FP_R. În mod previzibil, mărirea orizontului de predicție produce valori mai bune pentru ttc. Însă simultan cu această îmbunătățire crește și rata falselor pozitive FP_R. Această creștere a ratei falselor pozitive este însă de așteptat deoarece extinderea orizontului de predicție cauzează mai mult zgomot în observațiile prezise, astfel algoritmul semnalează mai multe coliziuni.
Pentru al treilea studiu de ablație, a fost comparată performanța algoritmului propus în funcție de dimensiunea observațiilor istorice. Astfel, au fost folosite următoarele valori (în secunde) pentru dimensiunea istoriei observaționale: $[1, 3, 5, 7, 10]$ (s). Se poate observa faptul că mărirea dimensiunii istoriei nu aduce îmbunătățiri semnificative ale performanței.

Concluzii și direcții viitoare de cercetare

Cercetările desfășurate în perioada studiilor doctorale au avut ca scop implementarea unui sistem de navigare autonomă bazat pe tehnici de inteligență artificială, pentru un agent autonom. Un astfel de sistem trebuie să fie capabil să realizeze mai multe sarcini din sfera navigării autonome, cum ar fi percepția mediului înconjurător, extragerea de informații adiționale pe baza percepției, precum informații despre poziția și dinamica obstacolelor din scenă, planificarea traiectoriei pentru agent, astfel încât acesta să se deplaseze înspre destinația sa finală, fără a realiza coliziuni cu obstacole statice sau dinamice din mediul înconjurător, dar și controlul agentului întrucât acesta să urmărească traiectoria locală generată de către planificator.
Noțiunile teoretice fundamentale utilizate în cadrul cercetărilor desfășurate au fost prezentate în prima parte a lucrării. Astfel s-a realizat o sintetizare a conceptelor de Vedere Artificială, cu precădere cele utilizate pentru operarea cu informații provenite de la senzorii care echipează roboții mobili. S-au prezentat procesele de filtrare ale imaginilor digitale, procedeele folosite pentru estimarea tridimensională, precum și anumite metode de extragere a informațiilor de nivel înalt utilizând Vederea Artificială.
Apoi s-au prezentat noțiuni despre inteligență artificială aplicabile în domeniul de navigare autonomă. S-au trecut în revistă câteva concepte de bază ale inteligenței artificiale, punându-se accentul pe rețele neuronale artificiale. S-au prezentat principalele tipuri de rețele neuronale artificiale, aplicabilitatea lor, precum și modul de antrenare. S-a pus accentul pe tipurile de rețele folosite în navigarea autonomă în general, respectiv pe cele folosite în lucrarea de față. Ulterior au fost descriși principalii algoritmi utilizați în lucrările de specialitate pentru planificarea traiectoriei agenților inteligenți, precum și principalele tipuri de controllere utilizate pentru a controla acești agenți.

După aceea, s-a detaliat stadiul actual al cercetărilor din domeniul navigării autonome, cu precădere cele care utilizează inteligența artificială pentru a realiza funcționalitatea de navigare sau anumite sarcini conexe, precum percepția, localizarea sau controlul agentului inteligent.
Au fost prezentate în amănunt particularitățile algoritmului propus în această teză pentru navigare autonomă, trecând în revistă componentele care realizează toate sarcinile specifice necesare procesului de navigare. La final, au fost amintite toate modalitățile prin care s-a realizat validarea algoritmului, respectiv rezultatele experimentale obținute.
Pornind de la scopul lucrării, cum a mai fost menționat anterior, de a realiza un sistem de navigare autonomă pentru roboți mobili, a fost necesară definirea modulelor componente pentru un astfel de sistem. Așadar au fost implementate modulul de percepție pentru agentul inteligent, cel de planificare a traiectoriei locale, cel de planificare al traiectoriei globale, modulul de localizare, precum și cel care realizează controlul agentului.
Planificarea traiectoriei locale pentru agentul inteligent s-a realizat folosind metoda DWA modificată astfel încât să înglobeze predicțiile oferite de către modulul de inteligență artificială. Astfel, planificatorul a beneficiat de disponibilitatea unor informații adiționale sub forma acelor predicții, reușind să genereze traiectorii mai bune atât din punct de vedere al evitării coliziunilor, cât și al mișcării agentului în sine.
Controlul agentului inteligent s-a realizat utilizând metoda MPC, mai specific varianta pentru sisteme neliniare - NMPC. Acest tip de algoritm este potrivit pentru execuția mărimilor de comandă datorită naturii sale care implică aplicarea modelului procesului pentru a prezice evoluția în timp a sistemului controlat (mișcarea agentului inteligent în acest caz).
Toate rezultatele experimentale au fost obținute utilizând sistemul de operare pentru roboți ROVIS, implementând interfețele necesare atât pentru mediile simulate, cât și pentru platformele robotice utilizate.

Contribuții originale. Diseminarea rezultatelor

Lucrarea de față face parte din sfera generală de interes pentru domeniul de Inteligență Artificială (IA), mai specific pentru tehnici de IA aplicate în domeniul navigării autonome. Tema tratată a presupus studiul, respectiv utilizarea unor concepte complexe cu caracter interdisciplinar, precum matematica, Vederea Artificială (VA), IA, robotica și tehnologia informației. Lucrarea de față propune un sistem de navigare autonomă pentru roboți mobili, bazat pe o arhitectură de tipul perception-plan-control, unde componentele de percepție, respectiv planificare a traiectoriei au fost concepute utilizând inteligență artificială. Pe de altă parte, sistemul de navigare autonomă a fost implementat atât pentru mediu simulat, cât și pentru platforme robotice reale.
Astfel, principala contribuție personală rezultată în urma cercetărilor efectuate în cadrul studiilor doctorale este conceptualizarea și implementarea unui algoritm de inteligență artificială care are rolul de a realiza un model al observațiilor senzoriale pentru un robot mobil pe baza observațiilor istorice. Modelul observațiilor senzoriale este folosit pentru a prezice evoluția obstacolelor statice și mai ales dinamice din mediul înconjurător unde navighează robotul, astfel contribuind la o mai bună planificare a traiectoriei, respectiv control superior față de alte tehnici utilizate în domeniu.

Modelul observațiilor diferă de ceea ce s-a mai studiat în domeniu atât prin structura componentei de inteligență artificială, cât și prin modul de funcționare. Spre deosebire de alte arhitecturi utilizate pentru predicția evoluției mediului înconjurător, unde se realizează întâi pasul de detecție a obiectelor, fie 2D sau 3D, urmând ca apoi să fie urmărite respectivele obiecte, algoritmul propus prezice direct observațiile agentului autonom, fără a realiza și pasul de recunoaștere a obiectelor din scenă.
Algoritmul utilizat pentru modelarea observațiilor agentului inteligent se bazează pe o rețea neuronală recurentă, care primește ca intrări secvențe de date istorice sub forma observațiilor anterioare provenite de la senzorul de distanță aflat pe agent și pe baza lor produce estimări pentru observații din viitor, realizând codificarea dinamicii obstacolelor prezente în mediul înconjurător în care acesta navighează.
O altă contribuție rezultată în urma studiilor efectuate este înglobarea modelului acestor observații senzoriale în componenta de planificare a traiectoriei locale pentru robotul mobil. Astfel, componenta de planificare primește pe lângă informații despre starea curentă a vehiculului autonom, predicții ale algoritmului de inteligență artificială. Astfel, traiectoria locală pe care trebuie să o urmeze agentul inteligent ține cont atât de obstacolele apropiate de robot, cât și de cele care pot reprezenta un pericol pentru agent în viitorul apropiat datorită dinamicii lor.
De asemenea, a fost realizată preprocesarea datelor senzoriale utilizate de către algoritmul de inteligență artificială, astfel încât să fie îmbunătățite performanțele sistemului, atât din punct de vedere al acurateții, cât și al vitezei de execuție. Așadar, datele de intrare sub forma norilor de puncte au fost transformate în alte structuri de reprezentare, potrivite pentru a fi procesate de către rețelele neuronale utilizate.
O altă contribuție a venit sub forma algoritmilor de navigare autonomă implementați pe platformele utilizate la validare, respectiv implementarea interfețelor pentru diferiți senzori necesari percepției sistemului de navigare, implementarea planificatorului de traiectorie locală, precum și a controllerului.
Algoritmul propus a fost implementat pentru a fi folosit în două medii simulate, precum și pe două platforme robotice. Astfel, au fost necesare adaptări din punct de vedere al interfețelor pentru senzorii utilizați pentru percepție, cât și modificări ale comenzilor pentru actuatori, precum și pentru sistemul de localizare, acestea fiind diferite pentru fiecare platformă folosită. A fost realizată comparația algoritmului propus cu alte implementări din literatura de specialitate, pentru fiecare platformă hardware sau software folosită. Au fost concepute și implementate scenarii dificile pentru agentul autonom, astfel încât să se pună în valoare adaptibilitatea algoritmului pentru diferite situații care pot apărea în cadrul sarcinilor de navigare. Astfel s-a asigurat prin intermediul experimentelor performanța sistemului complet de navigare autonomă.
Rezultatele obținute pentru algoritmul ObserveNet Control au fost validate prin publicare în următoarele jurnale de specialitate, respectiv conferințe:

- **C. Ginerica**, M. Zaha, L. Floroian, D. Cojocaru, S. Grigorescu, „A Vision Dynamics Learning Approach to Robotic Navigation in Unstructured Environments” in Robotics. 2024; 13(1):15, https://doi.org/10.3390/robotics13010015, IF=3.7.

- **C. Ginerica**, M. V. Zaha, F. Gogianu, L. Busoniu, B. Trasnea and S. M. Grigorescu, "ObserveNet Control: A Vision-Dynamics Learning Approach to Predictive Control in Autonomous Vehicles," in IEEE Robotics and Automation Letters, doi: 10.1109/LRA.2021.3096157, WOS:000678343900031, IF=3.741.
- **C. Ginerica**, D. Cojocaru, S. Grigorescu, „A Vision-Dynamics Learning Approach to Prediction-Based Control in Autonomous Vehicles", International Symposium on Signals, Circuits & Systems - ISSCS 2021, SCOPUS.
- **C. Ginerica**, V. Isofache, S. Grigorescu, „Vision Dynamics: Environment Modelling, Path Planning and Control Based on Semantic Segmentation", International Joint Conference OPTIM-ACEMP 2021, IEEE Xplore.
- S.M. Grigorescu, **C. Ginerica**, M. Zaha, G. Macesanu, B. Trasnea, "LVD-NMPC: A Learning-based Vision Dynamics Approach to Nonlinear Model Predictive Control for Autonomous Vehicles", Advanced Robotic Systems, Sage Journals, 2021, WOS:000660671900001 IF=1.652.
- B. Trasnea; **C. Ginerica**, M. Zaha, G. Măceşanu, C. Pozna, S. Grigorescu, OctoPath: An OcTree-Based Self-Supervised Learning Approach to Local Trajectory Planning for Mobile Robots. Sensors 2021, 21, 3606, IF 3.847, https://doi.org/10.3390/s21113606, Web of Science.