Planificatorare de mişcare pentru multiple interogări bazate pe hărţi compacte pentru medii dinamice şi dificile

-Rezumat-

De la mijlocul anilor 1990 încoace, planificarea mişcării a fost făcută în principal cu ajutorul metodelor bazate pe eşantionare. Acestea colectează un număr de eşantioane din spaţiul de configuraţii al unui sistem şi încearcă să le conecteze prin traiectorii simple într-un graf numit "roadmap". Ce se întâmplă cu acest roadmap după ce o problemă de planificare este rezolvată defineşte două categorii de planificatorare: "single query" (roadmap-ul este şters) şi "multi-query" (roadmap-ul este păstrat pentru probleme viitoare).

Deoarece construcţia unui roadmap este un proces costisitor din punct de vedere computaţional, s-ar părea că planificatoarele multi-query au un avantaj de eficienţă. Cu toate acestea, majoritatea lucrărilor publicate în ultimii ani în domeniul planificării mişcării se referă la planificatoare single-query şi aplicaţiile acestora. Biblioteci şi pachete de software pentru robotică, cum ar fi MoveIt! sau OpenRAVE, implementează nativ planificatoare single-query; pentru planificatoare multi-query sau nu există suport sau acesta este limitat.

Principalul motiv pentru această preferinţă pentru single-query este că aceste planificatoare fac mai puţine presupuneri asupra structurii unei probleme de planificare şi deci sunt mai uşor aplicabile unor clase largi de probleme interesante pentru cercetarea curentă. Spre deosebire de ele, planificatoarele multi-query funcţionează cel mai bine atunci când mediul de lucru nu se schimbă; eventuale schimbări pot invalida secţiuni din roadmapul construit. De asemenea, în general planificatoarele multi-query lucrează cu grafuri neorientate şi deci nu pot reprezenta sisteme cu manevre nereversibile sau sisteme în care inerţia este un factor ce determină efortul necesar pentru traiectorii.

În ciuda celor afirmate mai sus, s-ar părea că efortul depus la construcţia unui roadmap nu ar trebui risipit prin pierderea acelui roadmap. Această teză investighează moduri prin care planificatoarele multi-query să devină competitive cu cele single-query. Teza investighează şi probleme pentru care stocarea unui set util de configuraţii, deci o abordare multi-query, este singura opţiune practică deoarece configuraţiile utile sunt prea valoroase şi greu de găsit.

Vom insista în teză pe roadmapuri compacte construite cu ajutorul unor variante ale euristicii de vizibilitate. Căutările într-un roadmap de dimensiune redusă sunt mai rapide, şi de asemenea eventuali paşi de procesare (cauzaţi, de exemplu, de schimbări în mediul de lucru) sunt mai eficienţi. Trebuie păstrat un echilibru între dimensiunea unui roadmap şi capacitatea lui de a reprezenta conectivitatea spaţiului de configuraţii, iar euristica de vizibilitate oferă acest echilibru. Euristica poate fi modificată spre a accepta şi redundanţe în roadmap dacă acestea permit traiectorii mai scurte şi garanţii de aproximativă optimalitate.

Capitolul 1 oferă un scurt rezumat al istoricului domeniului planificării mişcării şi al stării de fapt. Sunt prezentate şi argumentele pentru care planificatorarele single-query sunt preferate în ultimii ani, cât şi se enunţă teza că cele multi-query pot deveni mai competitive.

Capitolul 2 prezintă suportul teoretic şi conţine scurte introduceri în teoria grafurilor, planificatoarelor cu eşantionare, roadmap-urilor compacte, logicii temporale şi teoria controlului. Se prezintă defiţiile câtorva noţiuni importante şi se citează câteva lucrări de referinţă în domeniile enumerate.

În capitolul 3, extindem demonstraţia proprietăţii "probabilistic completeness" a euristicii de vizibilitate. Clarificăm presupunerile implicite din demonstraţia existentă în literatură şi îi lărgim domeniul de aplicabilitate la orice sistem reversibil care respectă acele presupuneri.

Tot în capitolul 3, extindem această demonstraţie şi la sisteme cu manevre nereversibile şi ajustăm euristica de vizibilitate pentru a face posibil lucrul cu grafuri orientate, ceea ce permite modelarea sistemelor cu manevre nereversibile şi inerţie.

Implementăm planificatoarele noastre bazate pe vizibilitate şi le testăm prin comparaţie cu PRM (în cazul sistemelor reversibile) şi PRM adaptat pentru grafuri orientate (pentru sisteme cu manevre nereversibile). Problemele de test sunt labirinturi prin care un punct (sistem reversibil) sau un vehicul modelat cinematic (sistem cu manevre nereversibile) trebuie să ajungă de la o configuraţie start la una destinaţie. Colectăm statistici pentru numărul de eşantioane folosite pe parcursul mai multor rulări; în cazul planificatoarelor bazate pe vizibilitate, colectăm şi statistici pentru eşantioane rejectate. Constatăm că planificatoarele noastre pot rezolva problemele de planificare cu mai puţine eşantioane decât PRM.

Discutăm şi metode de a optimiza traiectoriile produse de un planificator bazat pe vizibilitate. Enumerăm câteva abordări existente în literatură şi schiţăm o metodă de optimizare locală bazată pe ecuaţia Euler-Lagrange discretă.

În capitolul 4 investigăm aplicarea planificatoarelor multi-query la probleme în medii dinamice: configuraţiile obstacolelor se schimbă în timp. Sistemul de test este un braţ manipulator cu 7 grade de libertate de pe un robot PR2 construit de Willow Garage. Prezentăm o soluţie existentă în literatură numită "lazy PRM" şi o comparăm cu un planificator single-query standard pentru braţe robotice numit RRTConnect. Constatăm că lazy PRM este mai încet decât RRTConnect (uneori cu un factor de doi sau peste), ceea ce explică de ce nu este folosit la planificarea pentru braţe în medii dinamice.

Investigăm ce cauzează această relativă ineficienţă şi constatăm că este cauzată de faptul că lazy PRM caută traiectoria optimă dintre acelea existente în roadmap. Dacă însă traiectoria optimă este invalidă din cauza unui obstacol nou apărut, atunci a doua traiectorie propusă de lazy PRM va fi a doua cea mai bună traiectorie din roadmap, care va fi deseori în apropiere de prima şi probabil tot invalidă deoarece trece prin acelaşi obstacol.

Ca atare, modificăm lazy PRM pentru a depărta căutarea de regiuni invalide. Propunem o funcţie "cost bump" care ajustează costurile vertecşilor din roadmap aflaţi în apropierea unui obstacol. Testăm varianta modificată de lazy PRM prin comparaţie cu RRTConnect şi constatăm că metoda noastră este mai rapidă cu un factor de doi.

Observăm că aceste costuri ajustate pentru vertecşi permit planificatorului să aproximeze forma spaţiului liber din spaţiul de configuraţii. Notă: mediul de lucru al robotului este tridimensional, este perceput cu un senzor Kinect şi reprezentat cu o structură Octomap. Spaţiul de configuraţii al uni braţ însă are şapte dimensiuni şi o formă complexă a regiunii din afara obstacolelor. Obstacole "simple" pot avea efecte foarte dificil de calculat asupra formei spaţiului liber de configuraţii, de aceea nici un planificator nu calculează exact această formă. Noi o aproximăm în schimb, folosind costul unui vertex ca o măsură a probabilităţii ca acesta să se afle într-un obstacol sau în apropierea unui obstacol.

Verificăm planificatorul nostru într-un mediu în schimbare. Observăm că dacă permitem păstrarea costurilor vertecşilor, planificatorul poate rezolva probleme mai repede o dată ce învaţă regiunile ocupate din spaţiul de configuraţii prin intermediul acelor costuri. Este mai rapid decât RRTConnect.

În capitolul 5 studiem o problemă de manipulare: conectarea/deconectarea a două corpuri rigide. Exemple de asemenea probleme sunt agăţarea unor unelte pe un stativ, punerea/scoaterea unor chei pe un breloc, anumite "puzzleuri" etc. Chiar dacă puzzleurile sunt în mod deliberat făcute să fie dificile şi pentru oameni şi deci exemple nu tocmai naturale, ele ilustrează anumite caracteristici interesante ale problemelor de manipulare: necesitatea utilizării unor trăsături fine pentru rezolvarea problemei de manipulare, cât şi nevoia de a schimba modul de apucare asupra obiectelor manipulate în cursul rezolvării. Cu excepţia puzzleurilor, acestea sunt probleme pe care oamenii le întâlnesc zi de zi şi le rezolvă în mod reflex. Dacă un robot operează într-un mediu uman, va trebui să rezolve cu uşurinţă asemenea probleme.

Pasajele înguste şi trăsăturile fine sunt însă dificil de descoperit şi folosit de planificatoarele bazate pe eşantionare. Am încercat să folosim RRTConnect pentru a rezolva o problemă test pe care am folosit-o la verificarea prin simulare (un inel cu un mic arc lipsă ce trebuie mişcat în jurul unei plăci cu două găuri). Am încercat o variantă simplificată a problemei în care permitem inelului să se mişte singur, fără a avea nevoie de manevre ale robotului. Cu toate acestea am constatat că nici după cinci minute de calcul RRTConnect nu a găsit o soluţie.

Problema de manipulare devine mai dificilă din cauza numărului de grade de libertate: un corp rigid are doar 6, pe când cele două braţe ale unui PR2 au împreună 14, plus încă 2 pentru cele două grippere. Acesta este însă spaţiul în care trebuie căutată o soluţie care descrie şi ce manevre trebuie să facă robotul.

Există în literatură unele euristici pentru identificarea pasajelor înguste, dar pentru acest capitol am decis să oferim robotului un ajutor oferit de un operator uman. Acesta oferă robotului câteva configuraţii utile ale inelului în jurul plăcii şi enunţă probleme de planificare pentru un planificator bazat pe vizibilitate care construieşte un roadmap pentru spaţiul de configuraţii al inelului în jurul plăcii.

Am decis să folosim o abordare multi-nivel pentru planificarea manipulării: mai întâi planificare pentru inel, cosiderat capabil de mişcare autonomă, iar apoi folosirea acestui plan pentru a ghida selectarea modurilor de apucare a inelului de către robot şi planificarea pentru mişcarea braţelor, în aşa fel încât robotul să conducă inelul de-a lungul traiectoriei planficate. Planificăm mişcări pentru braţe numai o dată ce avem o secvenţă completă de moduri de apucare. În acest mod îmbunătăţim eficienţa metodei prin reducerea dimensionalităţii spaţiilor în care planificăm, cât şi reducem numărul de probleme de planificare pentru braţe care trebuie rezolvate.

Constatăm că o euristică "greedy" pentru selectarea modurilor de apucare produce soluţii cu puţine schimbări de apucare: foloseşte acel mod de apucare ce permite urmărirea traiectoriei planificate pentru inel pe cea mai mare distanţă. Presupunem că există o colecţie finită de moduri de apucare cunoscute de robot. Testarea tuturor este însă costisitoare din punct de vedere computaţional, deci propunem o euristică de ierarhizare a modurilor de apucare numită "grasp suggestion strength": în roadmapul pentru inel stocăm şi informaţii cu privire la ce moduri de apucare sunt fezabile şi pe ce distanţe în jurul configuraţiilor corespunzătoare vertecşilor din roadmap.

Verificăm metoda noastră pe problema inel şi placă şi constatăm că produce soluţii în intervale rezonabile de timp (câteva zeci de secunde). Observăm că folosirea "grasp suggestion strength" înjumătăţeşte timpul necesar pentru selecţia modurilor de apucare.

Combinăm apoi metoda descrisă aici cu "cost bumps" din capitolul 4, spre a permite lucrul în medii dinamice cât şi pentru a îmbunătăţi robusteţea metodei. O problemă ce poată să apară dacă planificarea se face mai întâi pentru inel şi apoi pentru braţe este ca traiectoria pentru inel să nu poată fi urmărită prin nici o secvenţă de manevre ale robotului. O traiectorie imposibil de urmat crează situaţii "dead end": de la configuraţia curentă, nici un mod de apucare nu este fezabil (de exemplu, deoarece în mediu a apărut un obstacol).

Verificăm această metodă pe problema inelului şi plăcii într-un mediu dinamic. Observăm că robotul este capabil să rezolve problemele de planificare enunţate, iar costurile aplicate vertecşilor îi permit să înveţe ce regiuni să evite, planificând mai rapid.

De asemenea în capitolul 5, investigăm o metodă de a construi şi apoi refolosi un roadmap pentru o pereche de corpuri rigide pentru care spaţiul liber este alcătuit din pasaje înguste şi regiuni mici în care sunt disponibile mai multe grade de libertate ("junctures"). Definim o structură de date "DoF map" care este un graf în care vertecşii sunt regiuni din spaţiul de configuraţii (pasaje şi "junctures"). Propunem DoF map ca un criteriu de clasificare al obiectelor, util pentru planificarea mişcării: două perechi de corpuri rigide sunt 'la fel' dacă au DoF map-uri izomorfe.

Propunem acest criteriu deoarece geometria unei perechi de corpuri conţine prea multă informaţie şi nu se generalizează bine. Există multe forme de ceşti, de exemplu, dar oamenii le pot recunoaşte pe toate drept ceşti. Topologia unui obiect oferă prea puţină informaţie: o ceaşcă şi un covrig sunt topologic echivalente, dar într-un covrig nu se poate pune cafea.

Abordarea noastră este să reprezentăm printr-un DoF map interacţiunile cinematice dintre obiecte, similar cu conceptul de "affordances" existent în literatură: contează mai puţin formele, sau chiar primitivele geometrice, ale unor obiecte. Contează în schimb ce pot face sau ce se poate face unor obiecte.

Descriem o metodă de a construi în mod automat un DoF map folosind senzori tactili. Apoi descriem o metodă de refolosire a unui DoF map folosind informaţia de la senzori tactili şi estimări a priori ale stării curente pentru a obţine estimări actualizate pentru starea curentă. Nu refolosim informaţii cu privire la forma ori configuraţiile unei perechi de obiecte pentru alta; folosim doar informaţii despre numărul de grade de libertate disponibil la momentul curent.

Verificăm metoda noastră pe o problemă de planificare în două spaţii cu DoF map-uri izomorfe. Constatăm că prin construcţia unui DoF map putem planifica, în timp ce RRTConnect nu poate rezolva problemele date nici după minute de calcul. Constatăm şi că refolosirea unor DoF map-uri face mai rapidă planificarea decât reconstrucţia lor.

În capitolul 6, extindem planificatoarele de vizibilitate pentru probleme enunţate într-un fragment de logică temporală ce permite construcţia unor predicate tip "există o traiectorie". Ne limităm la acest fragment, deoarece presupunem că dacă o traiectorie există, atunci poate fi aleasă de planificator. Asta corespunde unei situaţii în care planificatorul este singurul agent care alege, sau toţi agenţii existenţi colaborează. Nu am considerat situaţii adversariale ori predicate tip "oricare traiectorie", deoarece acestea nu pot fi reprezentate pe un roadmap compact construit prin vizibilitate; singura garanţie pentru rezolvarea unui predicat tip "orice traiectorie" poate fi oferită de reprezentarea tuturor traiectoriilor.

Logica temporală permite problemelor de planificare să includă în mod natural secvenţe sau condiţionări pe secvenţe de manevre şi permite o integrare mai strânsă între un nivel simbolic de planificare al unor sarcini şi nivelul geometric de planificare al mişcării care poate decide dacă o sub-secvenţă de sarcini este fezabilă sau nu.

Constatăm că euristica de vizbilitate trebuie modificată spre a putea rezolva probleme în logica temporală tip "există traiectorie" şi propunem o modificare. Demonstrăm că este probabilistic completă presupunând un set rezonabil de condiţii. Descriem o metodă de a extrage un plan dinstr-un roadmap pentru logică temporală tip "există traiectorie".

Observăm că planificatorul nostru poate fi îmbunătăţit printr-un pas de închidere al traiectoriei obţinute ("gap reduction") ceea ce îi permite să genereze şi traiectorii tip buclă închisă.

Verificăm metoda propusă prin comparaţie cu un alt planificator pentru logică temporală: RRG. Constatăm că metoda noastră poate rezolva probleme de planificare cu mai puţine eşantioane şi în timpi mult mai scurţi decât RRG.

În fine, capitolul 7 oferă un sumar al contribuţiilor, o listă a lucrărilor publicate de autorul tezei şi indică direcţii viitoare de cercetare.