Vodenje žogice po trajektoriji na platformi z uporabo robotskega vida

Adam Hrastnik, Jaka Kovše

Univerza v Ljubljani, Fakulteta za elektrotehniko, Tržaška 25, 1000 Ljubljana, Slovenija
E-pošta: ah1223@student.uni-lj.si
E-pošta: jk4684@student.uni-lj.si

Povzetek. Prispevek predstavlja razvoj demonstracijskega sistema treh robotov, katerih cilj je vodenje žogice po podani trajektoriji na okrogli platformi, s pomočjo nagiba. Prikazana je izdelava komunikacije s krmilnikom, algoritem za določanje pozicije žoge na platformi in uporabniški vmesnik za vnos trajektorije, ki služi kot referenca za vodenje. Robotski vid je bil zasnovan v jeziku Python, s pomočjo knjižnice OpenCV.

Ključne besede: vodenje, trajektorija, kalibracija, robot, robotski vid

Trajectory ball control on a platform using robot vision

The article introduces development of demonstration system of three robots whose goal is to drive the ball along given trajectory on a platform, by utilizing tilt. Development of communication, algorithm for determining position of ball and user interface are presented. Robot vision was designed in Python with use of OpenCV library.

Keywords: control, trajectory, calibration, robot, robot vision

1 UVOD

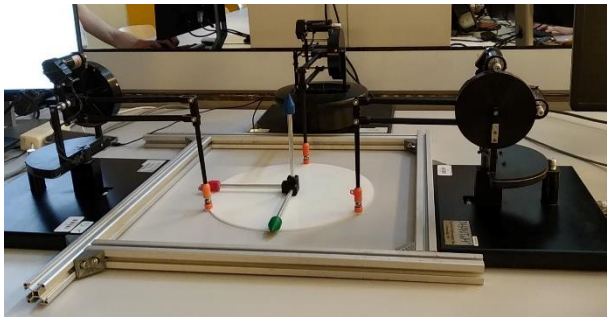
V vsakdanjem življenju se vse pogosteje srečujemo z roboti. Bodisi preko medijev, bodisi jih imamo doma. Posledica vedno večjega povpraševanje po robotih in robotskih sistemih je večanje števila aplikacij in njihove kompleksnosti. Zaradi tega včasih za delovanje nekega sistema ni dovolj zgolj robot, temveč se moramo poslužiti dodatne opreme.

Kadar pridobivanje informacije s fizičnim poseganjem v okolje ni mogoče, največkrat uporabljamo robotski vid, ki robotu omogoča brezkontaktno pridobivanje geometrijskih in kvalitativnih informacij. V najenostavnejših primerih je informacija robotskega vida uporabljena za načrtovanje trajektorij, vendar pa večjo fleksibilnost doprinese uporaba informacij pridobljenih s pomočjo robotskega vida v povratni zanki. Bistvena razlika med običajnimi načini vodenja in vodenja na osnovi robotskega vida je, da so pri uporabi slednjega regulacijske veličine izračunane posredno s pomočjo kompleksnih algoritmov procesiranja slike [2].

Z namenom poglobitve znanja o vodenju robotov na osnovi robotskega vida je bil izdelan demonstracijski projekt, ki predstavlja vodenje žoge po platformi s pomočjo sistema robotov in robotskega vida:

- Sistem je sestavljen iz skupine treh enakih robotov, ki vodijo žogico po okrogli platformi, s pomočjo nagiba [1].
- Žogico vodimo v dveh prostorskih stopnjah [1].
- Vodenje poteka po trajektoriji, ki je povsem poljubna, saj jo določi uporabnik.
- Vsa interakcija s sistemom, je izvedena preko uporabniškega vmesnika.
- Pozicija žogice, zaznana s pomočjo robotskega vida, je uporabljena v povratni zanki vodenega sistema.

2 ROBOTSKI SISTEM

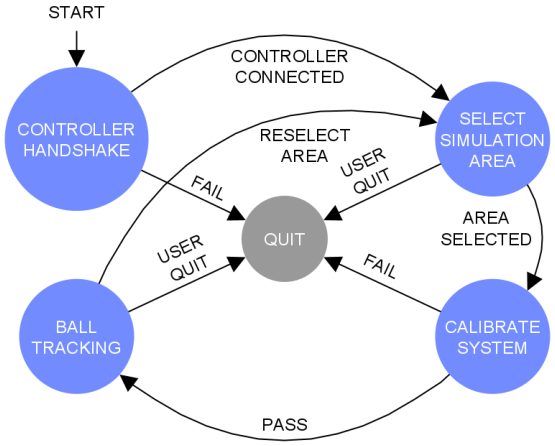


Slika 1: Postavitev robotov

Robotski sistem je sestavljen iz treh strukturno enakih haptičnih Phantom robotov [3]. Posamezen robot ima tri rotirajoče prostorske stopnje. Konec vsakega robota je pritrjen na okroglo platformo, premera 30 cm, ki se nahaja na sredini vseh robotov. Sredina platforme predstavlja vrh robotskega sistema. Kamera se nahaja nad sistemom, tako da ima v vidnem polju celotno platformo.

Na žalost smo bili zaradi pandemije COVID-19, primorani prestaviti celotno izvajanje projekta v simulacijo. Zato naše robote simuliramo s pomočjo poenostavljenih primitivnih modelov, kamera pa je prav tako virtualna.

3 POTEK DELOVANJA APLIKACIJE



Slika 2: Delovanje aplikacije na principu stroja stanj

Krmilnik za robote je načrtan v programskem okolju Matlab-Simulink. Komunikacija s krmilnikom poteka preko mrežnega protokola UDP. Krmilnik deluje kot strežnik, zato je potrebno, da je ta aktiven pred vidom.

Aplikacija robotskega vida deluje na principu stroja stanj:

- Program po zagonu prične v stanju »CONTROLLER HANDSHAKE«, kjer poskusi vzpostaviti povezavo s krmilnikom.
- Po uspešni povezavi, program nadaljuje v stanje »SELECT SIMULATION AREA«, kjer zahteva od uporabnika označeno področje simulacije na zaslonu.
- Z določenim področjem, program nato poskuša kalibrirati slikovno pretvorbo koordinat v stanju »CALIBRATE SYSTEM«.
- Ko kalibracija uspe, je program pripravljen za uporabo. Za tem nadaljuje v zadnje stanje, kjer aktivno sledi žogici in krmilniku posreduje njeno lokacijo.
- V istem stanju se tudi upravlja zajem in prenos trajektorije.
- Če uporabnik kadarkoli med delovanjem zapre aplikacijo ali pa je eden od postopkov inicializacije neuspešen, program zaključi z operacijo, tako da zapre vse module in uporabniški vmesnik.

4 KOMUNIKACIJA S KRMILNIKOM

Izbrani protokol za komunikacijo – UDP, omogoča visoko podatkovno pretočnost [4], kar je izredno velikega pomena pri tem projektu, saj je od tega odvisna uspešnost regulacije pozicije žogice.

Zaradi tega razloga je potrebno zagotoviti, kar se da nizek časovni zamik med zajeto fotografijo simulacije, procesiranjem slike, iskanjem pozicije žogice in prenosom podatka na krmilnik.

4.1 Zagotavljanje uspešnega prenosa podatkov

Čeprav protokol UDP omogoča večjo prepustnost, kot protokol TCP, vseeno vsebuje nekaj slabosti.

Glavna slabost je pomankanje preverjanja uspešnosti prenosa. V izogib izgubi podatkov [5], je implementiran preprost mehanizem za rokovanje, ki zahteva povratni prejem poslanega paketa.

Naslednja slabost je, da paketi lahko prispejo v drugačnem vrstnem redu, kot so bili poslani. Preverjanje pravilnega zaporedja ni implementirano, saj v večini primerov ni potrebno. Zaželena funkcionalnost, bi edino prispevala k prenosu trajektorije. Ker je to edini primer, ki zahteva pravili vrstni red prispelih podatkov, se nam implementacija tega ni zdela smiselna.

Dodatna slabost je, da UDP protokol ne določa nobenega preverjanja obremenjenosti mreže, kar lahko privede do visoke izgube podatkov z večjimi količinami poslanih podatkov [4]. V tem primeru je previdnost odveč, saj so podatkovni paketi izredno majhni, njihova frekvenca pa je prav tako relativno nizka.

4.2 Podatkovni paketi

Tabela 1: Vrste podatkovnih paketov

Paket	Tip	Vsebina		
Začetek povezave	0xE0	0x00	0x00	0x00
Konec povezave	0xE1	0x00	0x00	0x00
Pozicija žogice	0xF0	X	Y	0
Začetek trajektorije	0xF1	Dolžina	0x00	0x00
Konec trajektorije	0xF2	0x00	0x00	0x00
Vzorec trajektorije	0xF3	X	Y	0

Podatki se prenašajo v paketih velikosti 4 bajte. Prvi bajt pove tip paketa, ostali pa predstavljajo vsebino. Večina paketov je kontrolne narave, saj ne vsebujejo vsebine, ker njihov *tip* predstavlja zgolj ukaz. Razlikujeta se paketa »Pozicija žogice« in »Vzorec trajektorije«, ki krmilniku povesta koordinate, bodisi žogice ali vzorca trajektorije. »Začetek trajektorije« je prav tako drugačen, saj poleg ukaza krmilniku pred prenosom pove, koliko vzorcev trajektorije naj pričakuje.

5 ZAJEM SLIKE

Aplikacija je bila izdelana izključno v simuliranem okolju, zato je bila potrebna tudi simulacija kamere. Testiranih je bilo več modulov, ki omogočajo zajem zaslona. Izbran je bil modul *mss* [7], ki prepozna

operacijski sistem na katerem funkcijo uporabljamo in izbere najbolj prilagojen *mss_class* za zajem posnetkov zaslona v čim krajšem času. Uporabnik s pomočjo miške označi, kateri del ekrana naj bo zajet, podatki, pa se posredujejo funkciji, ki naredi posnetek zaslona in ga prikaže v novem oknu. Slika ostane v spominu dokler se nad njo izvajajo operacije detekcije žoge in dokler se ne zajame nova slika. Operacija zajema slike se izvaja z najvišjo frekvenco okrog 50 Hz, povprečno pa okoli 30 Hz in je tako primerljiva z različnimi kamerami, ki se navadno uporabljajo pri takšnih aplikacijah.

6 KALIBRACIJA ZAJEMA

Kalibracija je eden pomembnejših postopkov inicializacije programa, saj je potrebna za ustrezno delovanje sistema. Na simuliranem sistemu ni bilo potrebno izvajati geometrijske kalibracije zaradi simulacije kamere. Ravno zaradi tega vplivov optičnih aberacij kamere ni bilo, prav tako pa smo se zaradi računalniške postavitve kamere izognili neenakomernosti absolutnih velikosti slikovnih elementov. Ker se aplikacija izvaja z zajemom posnetka zaslona, je bilo pri kalibraciji potrebno upoštevati predvsem ločljivost ekrana. Navadno postopek kalibracije izvajamo s postavitvijo kalibracijskega objekta. Zaradi uporabe orodja SimMechanics, ki ne omogoča uvoza tekstur, je bilo potrebno ubrati drugačen pristop. V orodju se programsko generirata dva markerja različnih barv, ki jima lahko določimo velikost v slikovnih točkah in predstavljata smer abscisne in ordinatne osi koordinatnega izhodišča platforme. Prav tako se ob zagonu programa žoga vedno generira v izhodišču omenjenega koordinatnega sistema. Za izračun razmerja velikosti slikovnih točk in enot v programskem orodju, smo uporabili pozicije markerjev in žoge v začetni legi. Funkcija za kalibracijo torej prejme BGRA sliko in jo preslika v HSV prostor in sivinsko sliko. Sivinsko sliko uporabimo za iskanje koordinat žoge v začetni legi in markerjev. Da ugotovimo katere koordinate pripadajo določenemu markerju, primerjamo vrednosti barvnega kanala HSV slike. Enačba prikazuje izračun razmerja med velikostjo slikovne točke in milimetra:

$$Pr = 0.1 \cdot \left(\frac{Mc_y - M2_y + M1_x - Mc_x}{2}\right)^{-1} \qquad (1)$$

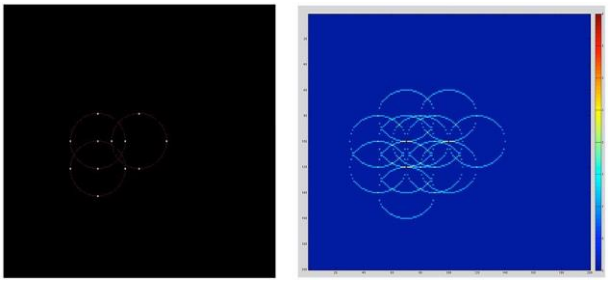
Kjer *Pr* predstavlja razmerje slikovni element/milimeter, *Mc*, *M1* in *M2* pa predstavljajo koordinate žoge v začetni poziciji in markerjev.

Potrebno je določiti še transformacijo med koordinatnim sistemom slike in platforme. Izhodišče slike je levo zgoraj; premakniti ga je potrebno v središče slike (od vrednosti koordinat najdene žoge v poljubnem položaju se odštejejo vrednosti koordinat žoge v začetni legi – na sredini platforme). Y-os slike je usmerjena v nasprotno smer kot y-os platforme, torej je potrebno vse vrednosti v vertikalni smeri pomnožiti z negativnim razmerjem med velikostjo slikovnega elementa in milimetra.

$$Bp = \left(\left(X - X_{offset}\right), -\left(Y - Y_{offset}\right)\right) \cdot P_r \qquad (2)$$

7 ZAZNAVANJE POZICIJE ŽOGICE

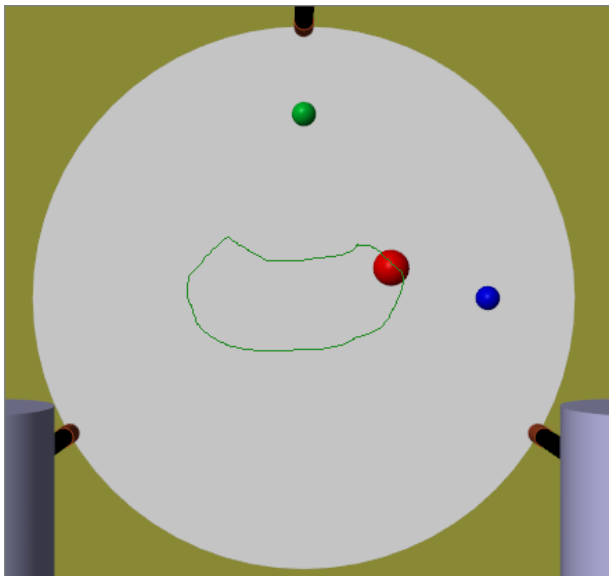
Za določanje pozicije žogice je uporabljen modificiran Hough-ov transform, ki je v osnovi algoritem iskanja linij v parametričnem 2D prostoru. Če želimo s pomočjo Hough-ovega algoritma iskati kroge, je potrebno parametrični prostor razširiti za eno dimenzijo. Krog namreč opišemo s tremi parametri, koordinatama središča x, y in polmerom kroga r. Parametrični prostor *xyr* je torej omejen z velikostjo obdelovane slike ter z največjim in najmanjšim premerom kroga ($r0 < r \leq r1$), ki ga določi uporabnik. Algoritem najprej poišče točke robov, s pomočjo katerih bodo izračunana središča iskanih krogov. V praznem akumulatorju se začnejo v prej najdenih točkah robov, risati krogi z določenim polmerom. Poiščejo se presečišča in če jih je dovolj, je krog najden (središče in polmer). Ko se pregledajo vse točke robov, se akumulator izprazni in začne iskati kroge z večjim polmerom [6].



Slika 3: Najdena središča krogov (levo) in akumulator (desno)

Pri iskanju pozicije žoge, je bila uporabljena funkcija modula OpenCV HoughCircles [8], ki kot parametre vzame vhodno 2C sivinsko sliko, ločljivost akumulatorja, najmanjšo možno razdaljo med krogoma, parametra, ki določata kakovost izvedbe iskanja robov in omejitve velikosti polmera. Ločljivost akumulatorja vpliva na popolnost najdenega kroga, torej nižja vrednost pomeni, da algoritem najde samo popolne kroge, višja pa, da so lahko krogi precej šumni. Med parametroma, ki določata kakovost izvedbe iskanja robov, je bolj pomemben drugi, saj določa prag verifikacije krogov. Višji kot je prag več krogov bo algoritem verificiral, vendar ni nujno, da bodo vsi krogi. Ker je pri iskanju pozicije žoge zelo pomembna hitrost operacije, je bilo potrebno omejitve velikosti polmera zaostriti v čim večji meri, saj to pomeni manj iteracij iskanja kroga z določenim polmerom in pridobitev na hitrosti izvajanja.

8 VNOS TRAJEKTORIJE



Slika 4: Vodenje po trajektoriji

Uporabniški vmesnik omogoča vnos željene referenčne trajektorije, po kateri naj se giblje žogica. Vnos je izveden s pomočjo računalniške miške, s katero uporabnik trajektorijo preprosto nariše neposredno na prikazno okno.

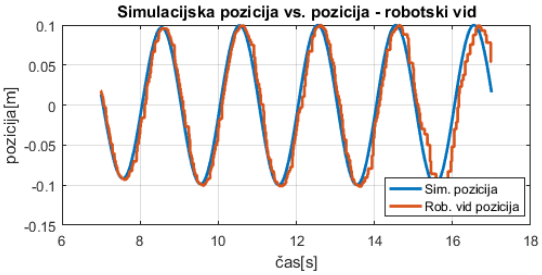
Zajemanje se prične s pritiskom gumba na miški. Med risanjem se za vsak zaznan premik miške zapiše nov vzorec v zbirko vzorcev. Po izpustu gumba na miški, se z uporabo kalibracijskih parametrov, zajete točke trajektorije v koordinatah slikovnih elementov, pretvorijo v kartezične koordinate.

Ko so vse koordinate pretvorjene, se prične prenos trajektorije:

- V začetku se pošlje krmilniku obvestilo o začetku prenosa, ki vsebuje število točk v trajektoriji. Po potrditvi, se začne prenos.
- Po končanem prenosu, se pošlje obvestilo o zaključku prenosa. Krmilnik odvrne z obvestilom o uspešnosti prenosa, ki je odvisno od tega ali se število prejetih paketov na krmilniku ujema s pričakovanim.
- Če je bil prenos trajektorije uspešen, program nadaljuje s sledenjem žogice. V nasprotnem primeru, se prenos trajektorije ponovi. Ta postopek se ponovi do trikrat, oz. dokler prenos ne uspe.

9 VREDNOTENJE ZAZNAVE ŽOGICE

Dejstvo je, da zaradi postopka kvantizacije (zajema slike) pride do nezaželenega pogreška [9]. To se zgodi, ker zajamemo okolje s končnim številom slikovnih elementov, vsa informacija med temi pa se izgubi.



Slika 5: Primerjava med pozicijo pridobljeno iz simulacije in pozicijo pridobljeno z robotskim vidom.

Iz grafa je razvidno, da je pri poziciji žogice, ki je pridobljena s pomočjo robotskega vida – prisoten kvantizacijski pogrešek in časovni zamik. Zamik je prisoten zaradi časa, ki je potreben za zaznavo, procesiranje in prenos pozicije.

10 ZAKLJUČEK

Tekom projekta smo naleteli na mnogo težav. Največje smo imeli z migracijo iz realnega robotskega sistema na simulacijo. Kljub temu, nam je na koncu uspelo doseči skoraj vse prvotno zastavljene cilje.

Ne glede na dosežene cilje, smo med razvojem naleteli na nekaj idej za izboljšave in preizkus sistema, za katere nam je zmanjkalo časa:

- Vnos 3D trajektorije s pomočjo tablice.
- Preizkus delovanja sistema z realno kamero.
- Primerjava delovanja z uporabo drugih algoritmov za iskanje krogov in primerjavo.

Kljub vsemu, smo s končnim izdelkom zadovoljni, saj smo izdelali virtualni sistem, zmožen za uporabo pri učenju naslednjih generacij.

LITERATURA

[1] Adam Hrastnik, Jaka Kovše: Vodenje žogice po trajektoriji na platformi s sistemom robotov, Vodenje robotov, maj 2020

[2] Matjaž Mihelj, Tadej Bajd, Marko Munih: Vodenje robotov, Založba FE, Ljubljana, 2011

[3] Uradna stran Phantom robotov: https://www.3dsystems.com/haptics-devices/3d-systems-phantom-premium, dostopno 5.5.2020

[4] Xiangning Liu, Lebin Cheng, Bharat Bhargava, Zhiyuan Zhao, “Experimental study of TCP and UDP protocols for future distributed databases”, Department of Computer Sciences, Purdue University, july, 1995

[5] L. Eggert, NetApp, G. Fairhurst, University of Aberdeen, G. Shepherd, Cisco Systems, RFC8085 - UDP Usage Guidelines, march 2017

[6] Reinhard Klette: Concise Computer Vision, An Introduction into Theory and Algorithms, Springer-Verlag, London 2014

[7] Python knjižnica MSS, https://python-mss.readthedocs.io/index.html, dostopno 20.4.2020

[8] Funkcija HoughCircles, https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=houghcircles#houghcircles, dostopno 4.5.2020

[9] Honghai Liu, Naoyuki Kubota, Xiangyang Zhu, Rudiger Dillmann, Dalin Zhou: 8th International conference in Intelligent Robotics and Applications Part III, 2015