



České vysoké učení technické v Praze

Fakulta elektrotechnická

8.4.2022

Obří slalom s Turtlebotem

Technická zpráva

Matouš Soldát, Šimon Soldát, Karolína Volfíková

Obsah

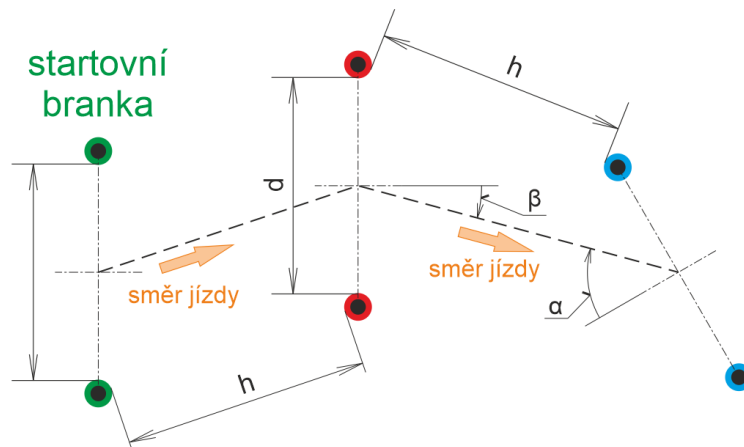
1	Úvod	2
1.1	Zadání úlohy	2
1.2	Použitý robot	3
2	Řešení	4
2.1	Segmentace	4
2.2	Orientace v prostoru	5
2.3	Pohyb na dráze	5
2.4	Poznámky k implementaci	6
2.4.1	Členění kódu	6
2.4.2	Spuštění programu	6
2.4.3	Dodatečné poznámky	6
3	Výsledky	7
3.1	Grafy, tabulky	7
4	Závěr	8

1 Úvod

1.1 Zadání úlohy

Cílem úlohy „obří slalom“ je projetí dráhy vyznačené tyčkami různých barev robotem. Dvojice stejně barevných tyček tvoří branku. Šířka jednotlivých branek je 450-600 mm a maximální vzdálenost středů dvou po sobě následujících branek je 1000 mm.

Na počátku úlohy je robot umístěn do startovní pozice. Jeho střed se nachází minimálně 300 mm a maximálně 1300 mm od startovní branky, kterou tvoří zelené tyčky, a jeho podélná osa s osou startovní branky nesvírá úhel větší než 60° . Minimální vzdálenost dvou tyček různých branek je 500 mm. Navíc směr jízdy může s osou branky svírat maximálně 30° . Na začátku je alespoň jedna z tyček startovní branky v zorném poli RGB kamery. Po průjezdu startovní brankou musí robot projet trať, na které se střídají modré a červené branky, aniž by se jich dotkl. Poslední branka není nijak definována a robot po jejím projetí může pokračovat v jízdě.



Obrázek 1: Schéma dráhy obřího slalomu

1.2 Použitý robot

Pro řešení úlohy byl použit TuttleBot 2. Jeho základnu tvoří zařízení Kobuki, které poskytuje základní funkční prvky: systém pro pohyb robotu, bumper, odometrii. Robot je ovládán přes NUC PC s frameworkem pro softwarový vývoj robotu ROS (Robot Operating System). Dále je na robotu umístěn jeden ze dvou RGBD senzorů: Orbex Astra (pro roboty s číslem 1, 2), Intel RealSense (pro roboty s číslem 3-7).



Obrázek 2: TurtleBot 2

2 Řešení

2.1 Segmentace

Pro nalezení jednotlivých tyček v obraze získaném z RGB kamery jsme zvolili segmentační metodu prahování. Nejprve jsme získali obraz z kamery pomocí funkce `get_rgb_image()`. Převodli jsme obraz z RGB do HSV barevné reprezentace a zvolili optimální prahové hodnoty. Pro každou barvu jsme experimentálně našli práh tak, že jsme RGB kamerou pořídili několik snímků každé tyčky při různém osvětlení.

Barva	H_{exp}	H_{diff}	S_{min}	V_{min}
Zelená	65	25	80	60
Modrá	100	25	230	80
Červená	2	8	150	70

Tabulka 1: Tabulka zvolených prahových hodnot, kde: H_{exp} je střední hodnota odstínu, H_{diff} je maximální povolený rozdíl měřené a střední hodnoty odstínu, S_{min} je minimální saturace a V_{min} minimální jas.

Dále jsme v obraze detekovali spojitě oblasti jako kandidáty pro tyčky pomocí OpenCV funkce `connectedComponentsWithStats()`. Odstranění nežádoucích oblastí jsme realizovali pomocí výstupů této funkce – definovali jsme minimální požadovanou plochu S a také podmínky pro poměr výšky h a šířky w :

$$S > 2500 [px], \quad (1)$$

$$r = \frac{h}{w} = 5.1, \quad (2)$$

$$r_{diff} < 3, \quad (3)$$

kde r_{diff} je rozdíl poměru měřených hodnot a definovaného poměru r .

2.2 Orientace v prostoru

Pro orientaci v prostoru jsme využili point cloud. Point cloud je sada bodů, které reprezentují daný objekt v prostoru. Každý bod je charakterizován souřadnicemi x, y, z . Point cloud získáme z funkce *get_point_cloud()* jako matici o velikosti 480 x 640 x 3.

Vektor třetí dimenze si uložíme jako \mathbf{z} a v matici jej nahradíme vektorem jedniček – tím převedeme matici do homogenních souřadnic. Novou matici označíme jako $M1_{CAM}$. Následně využijeme rovnici:

$$\mathbf{X} = \lambda \cdot \mathbf{K}^{-1} \cdot M1_{CAM}, \quad (4)$$

kde \mathbf{K}^{-1} je matice hloubkové kamery se souřadnicemi se středem v kameře získaná funkcí *get_depth_K()* a λ je empiricky zjištěná hodnota. Pro její získání jsme provedli osm měření vzdáleností tyček od sebe s hodnotou $\lambda = 1$. Poté jsme na výsledky aplikovali metodu nejmenších čtverců a dostali hodnotu $\lambda = 429$.

Z matice \mathbf{X} opět odstraníme vektor třetí dimenze a nahradíme jej vektorem \mathbf{z} . Výsledná matice M_{CAM} je matice souřadnic se středem v robotu.

Při pohybu jsme využívali funkci *reset_odometry()* a *get_odometry()*. *Reset_odometry()* nastaví počátek souřadnic na aktuální pozici robotu. *Get_odometry()* vrací relativní vzdálenost uraženou od posledního volání *reset_odometry()*.

2.3 Pohyb na dráze

Po spuštění robotu se aktivuje hledání zelené branky. Robot se otáčí, dokud nedetekuje dvě zelené tyčky. Poté změří jejich relativní pozici vůči němu a najede na osu této branky do vzdálenosti půl robotu + 5 cm před ní. Brankou projíždí a zastaví se až v momentě, kdy projede celé tělo robotu.

Následně proběhne hledání největší tj. nejbližší tyčky. Robot se otočí doleva o $30^\circ + 20^\circ$, pak doprava o 60° , a ukládá si oblast s největší plochou včetně její barvy. Díky tomu zjistíme, zda dráha začíná červenou nebo modrou brankou. Poté robot zopakuje tento proces otáčení, během kterého najde alespoň dvě tyčky, vybere dvě největší a porovná jejich výšky. Musí platit:

$$h_{diff} = \frac{a}{b} < 1.05 \quad (5)$$

$$a > b, \quad (6)$$

kde a je měřená výška a b šířka tyčky. Poté se robot vycentruje na rozpoznanou branku a opakuje se proces popsany pro branku startovní.

2.4 Poznámky k implementaci

2.4.1 Členění kódu

Kód je strukturován do několika hlavních souborů, které jsou v této sekci stručně popsány.

Soubor *main.py* obsahuje hlavní smyčku programu, ve které se pravidelně volá metoda *Driver.drive()*, pomocí které je řidiči umožněno ovládat robota. (Třída *Driver* je diskutována níže.) Dále je zde implementována inicializace hardwaru i softwaru.

Soubor *turtle.py* obsahuje třídu *Turtle*, ve které jsou implementovány wrappery na funkce robota. Pomocí těchto funkcí může řidič jednoduše získávat data ze senzorů robota a robota ovládat.

Soubor *driver.py* obsahuje veškerou logiku ovládání robota. Je zde třída *Driver*, která ovládá robota pomocí tzv. *aktivit*. Aktivita - třídy rozšiřující abstraktní třídu *Activity* - popisují jednotlivé akce, které robot vykonává, a mohou se skládat z dalších aktivit. (Příkladem takové aktivity může být třeba aktivita "jet na souřadnice", která se skládá z dalších pod-aktivit "otočit se" a "jet dopředu".) Díky tomuto objektovému přístupu je kód přehlednější, modulární a k aktivitám můžeme přistupovat jako k funkcím se vstupními a výstupními hodnotami, přestože jsou to často komplexní akce, jejichž vykonání trvá mnoho cyklů hlavního programu.

V souboru *camera.py* jsou implementovány statické funkce pro zpracování obrazu a dat ze senzorů.

Soubor *CONST.py* obsahuje konstanty úlohy, vlastností robota, naměřené a empiricky zjištěné konstanty pro segmentaci obrazu i konstanty hlavního programu. (Konstanty týkající se logiky ovládání robota jsou v souboru *driver.py*.)

2.4.2 Spuštění programu

Pro spuštění programu stačí spustit soubor *main.py*. (Například příkazem *python3 main.py*.)

2.4.3 Dodatečné poznámky

Během implementace a testování jsme narazili na nečekané chování robota v případě, že je příliš často volána metoda *Turtlebot.cmd_velocity(linear, angular)*. Robot se při častém volání této metody choval nekonzistentně a pohyboval se často neplynule.

Proto je v našem kódu nastavování rychlosti robota rozděleno do dvou metod; *Turtle.keep_speed()*, která interně volá nativní metodu robota *Turtlebot.cmd_velocity(linear, angular)* beze změny rychlosti, a *Turtle.set_speed(linear, angular)*, která přenastaví hodnoty rychlosti robota uvnitř třídy *Turtle* bez volání nativní metody *Turtle.set_speed(linear, angular)*. Využití těchto dvou

metod je na programátorovi řidiče, doporučujeme ale volat metodu *Turtle.keep_speed()* pouze jednou v každém volání *Driver.drive()* a ve zbylé logice řidiče a aktivit používat pouze metodu *Turtle.set_speed(linear, angular)*.

3 Výsledky

3.1 Grafy, tabulky

– měření času –

4 Závěr

:)