České vysoké učení technické v Praze
Fakulta elektrotechnická
21.4.2022

Klasický slalom s Turtlebotem

Technická zpráva

Matouš Soldát, Šimon Soldát, Karolína Volfíková

Obsah

1 Úvod

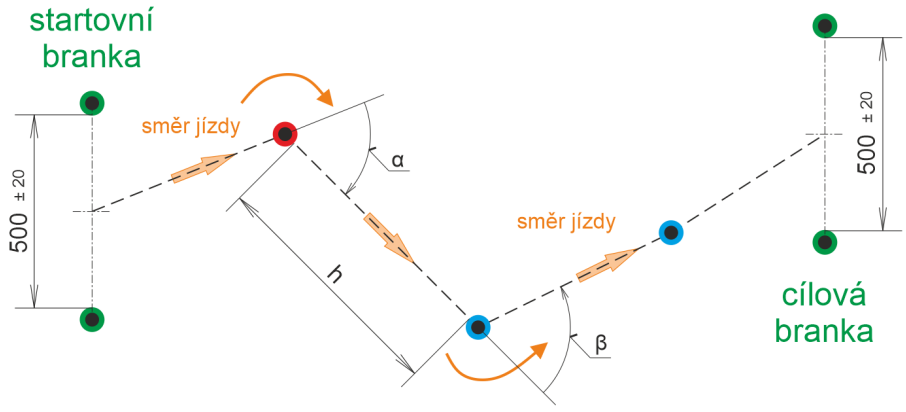
1.1 Zadání úlohy

Cílem úlohy „klasický slalom“ je projetí dráhy vyznačené tyčkami různých barev robotem.

Na počátku úlohy je robot umístěn do startovní pozice. Jeho střed se nachází minimálně 300 mm a maximálně 1300 mm od startovní branky (měřeno rovnoběžně s její osou), kterou tvoří zelené tyčky, a jeho podélná osa s osou startovní branky nesvírá úhel větší než 60°. Šířka startovní branky je 500 ± 20 mm. 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 nepravidelně střídají modré a červené branky, aniž by se jich dotkl. První tyčka bude vzdálena od startovní branky minimálně 500 mm. Směr jízdy, který je definován spojnicí středů po sobě následujících tyček se může v místě tyčky měnit maximálně o ± 60°. Minimální vzdálenost dvou tyček různých branek $h \leq$ 450 mm. Robot má za úkol objíždět červené tyčky zleva (při průjezdu zůstávají po pravé straně robotu) a modré tyčky zprava (při průjezdu zůstávají po levé straně robotu).

Konec trati je dán brankou tvořenou dvěma zelenými tyčkami, která má stejné rozměry jako branka startovní. Také platí, že úhel mezi směrem jízdy a osou strartovní nebo cílové branky může být maximálně ± 30°. Robot po projetí cílové branky může dále pokračovat v jízdě.



Obrázek 1: Schéma dráhy obřího slalomu

1.2 Použitý robot

Pro řešení úlohy byl použit TutleBot 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, nebo Intel Real Sense.

Obrázek 2: TurtleBot 2

2 Návrh řešení a úpravy

Původně jsme řešili úlohu druhé úrovně obtížnosti, „Obří slalom“, nakonec jsme ale plynule přešli na úlohu 3 – „Klasický slalom“, neboť jsme většinu funkcí, které jsou aplikovatelné v obou úlohách, již měli naimplementovanou. Část debuggingu a úprav specifikace byla tedy prováděna při řešení úlohy „Obří slalom“.

Z prvu jsme k řešení úlohy přistupovali pomocí co nejjednodušších metod, které jsme později vylepšovali a rozšiřovali, aby byl program robustnější a robot projel trať za co nejnáročnějších podmínek.

Pro nalezení tyček v obraze jsme stanovili pevné prahy, které jsme nejprve odhadovali z jednoho měření, což se později ukázalo jako nedostačující pro segmentaci obrazu v různých světelných podmínkách.

Orientace v prostoru měla být původně provedena pouze za pomoci RGB kamery a odometrie. To by ale vedlo ke složitějším výpočtům a pravděpodobně také k chybám a nepřesnostem v jízdě. Začali jsme tedy uvažovat data z hloubkové kamery.

Pohyb na dráze původně obsahoval projetí středem spojnice dvou po sobě následujících tyček, pokud mají různou barvu. Tento bod jsme později nahradili bodem x_0 popsaným v podkapitole „Objíždění branky“.

3 Řešení

3.1 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()*. Převedli jsme obraz z RGB do HSV barevné reprezentace a zvolili optimální prahové hodnoty. Pro každou barvu jsme experimentálně nalezli 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	40
Modrá	100	25	150	40
Červená	2	8	150	40

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 spojité 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:

$$S > 3000\ [px], \tag{1}$$

v případě startovní branky a

$$S > 800[px], \tag{2}$$

jinde,

$$r = \frac{h}{w} = 5.1, \tag{3}$$

$$r_{diff} < 3, \tag{4}$$

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

3.2 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 získáváme z funkce *get_point_cloud()* jako matici o velikosti 480 x 640 x 3.

Při řešení úlohy „Obří slalom“ jsme zjistili, že robot vždy projížděl brankou vpravo od jejího středu. Tento posun byl dál umístěním hloubkové kamery, které nebylo přesně ve středu robotu. V implementaci jsme tedy použili posun o tuto konstantu, 3 cm.

Souřadnice tyčky jsme pak získali podle výsledků segmentace – vybrali jsme souřadnice všech bodů, které přísluší dané tyčce a spočítali jsme medián v každé souřadnici. Tyto souřadnice poté transformujeme do souřadné soustavy vztažené ke startovní pozici robotu.

Dále jsme pro orientaci využívali funkcí *reset_odometry()* a *get_odometry()*. *Reset_odometry()* nastaví počátek souřadnic na aktuální pozici robotu, což se provede ihned po spuštění robotu. *Get_odometry()* vrací relativní vzdálenost uraženou od posledního volání *reset_odometry()*.

3.3 Pohyb na dráze

3.3.1 Hledání startovní branky

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 polovina těla robotu.

3.3.2 Hledání nejbližší tyčky

Následně robot hledá nejbližší tyčku. Tato úloha se rozděluje na 2 případy:

- Robot hledá první branku po projetí startem: Otočí se o úhel 30° na obě strany.
- Robot hledá branku ve zbytku trati: Otočí se o 60° doprava v případě, že poslední branka, kterou projel, byla červená, nebo se otočí o 60° doleva, pokud poslední branka, kterou projel, byla modrá.

Určení nejbližší z tyček pak probíhá pomocí její první a druhé souřadnice (vertikální souřadnici neuvažujeme) – počítá se eukleidovská vzdálenost robotu od tyčky. Robot si zapamatuje její barvu

i souřadnice. Pokud jde o zelenou tyčku, spustí se stejný proces jako při projíždění startovní brankou. Pokud ne, spustí se objíždění branky.

3.3.3 Objíždění branky

Robot objíždí branku podle obrázku číslo 3. Pomocí souřadnic detekované tyčky se vypočítají body x_0, x_1 a x_2. Ozn. o_1 a o_2 jako spojnice aktuální detekované a předchozí tyčky. Pro velikosti vektorů **a** a **b** platí:

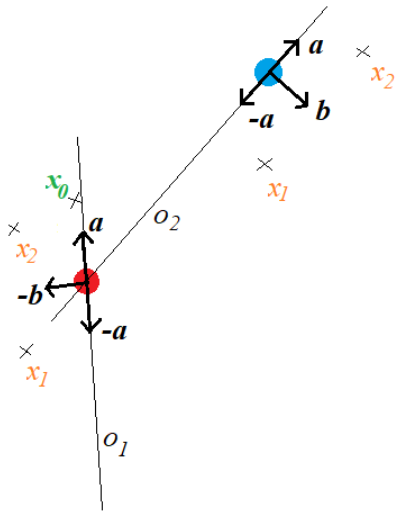
$$a = r + t + 3, \tag{5}$$

$$b = r + t + 5, \tag{6}$$

kde r je poloměr robotu, t je poloměr tyčky a konstantay 3, 5 [cm] jsou zvolené rezervy.

Vektory **a** a **b** jsou ortogonální, vektor **b** je kolmý na osu, a oba vektory se přepočítají po projetí branky, tj. v bodě x_2.

Po detekci nové tyčky se vypočítá se nová osa, o_2, a bod x_0 je pak ve vzdálenosti |**a**| od tyčky, u které se robot nachází, ve směru vektoru **a** a ve vzdálenosti |**b**| této tyčky ve směru vektoru **b**, pokud je její barva modrá, nebo ve směru vektoru **-b**, pokud je její barva červená.



Obrázek 3: Schéma objíždění branky

Bod x_1 je vzdálenosti |**a**| od detekované tyčky ve směru vektoru **-a** a vzdálenosti |**b**| ve směru vektoru **b**, pokud její barva je modrá, nebo ve směru vektoru **-b**, pokud je její barva červená.

Bod x_2 je ve vzdálenosti |**a**| od detekované tyčky ve směru vektoru **a** a vzdálenosti |**b**| ve směru vektoru **b**, pokud její barva je modrá, nebo ve směru vektoru **-b**, pokud je její barva červená.

Robot projíždí branku postupně po bodech x_0, x_1, x_2. Poté se přepočítá osa a robot pokračuje novými body x_0, x_1, x_2. Pokud následující branka má stejnou barvu jako branka předchozí, robot vynechá bod x_1. V případě projíždění první branky po startovní brance, robot vynechá bod x_0.

Po ukončení objíždění branky se vždy znovu aktivuje hledání nejbližší tyčky.

3.4 Poznámky k implementaci

3.4.1 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 robot ovládat. (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 robotu. Pomocí těchto funkcí může řidič jednoduše získávat data ze senzorů robotu a ovládat jej.

Soubor *driver.py* obsahuje veškerou logiku řízení robotu. Je zde třída *Driver*, která ovládá robot pomocí tzv. *aktivit*. Aktivity – třídy rozšiřující abstraktní třídu *Activity* – popisují jednotlivé akce, které robot vykonává, a mohou se skládat z dalších aktivit. (Příkladem takové aktivity může být například 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 robotu, naměřené a empiricky zjištěné konstanty pro segmentaci obrazu i konstanty hlavního programu. Konstanty týkající se logiky ovládání robotu jsou v souboru *driver.py*.

3.4.2 Spuštění programu

Pro spuštění programu spustíme soubor *main.py* (Například přikazem *python3 main.py*.) a poté stiskneme tlačítko „B0“ na základně robotu.

3.4.3 Dodatečné poznámky

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

Proto je v našem kódu nastavování rychlosti robotu rozděleno do dvou metod: *Turtle.keep_speed()*, která interně volá nativní metodu *Turtlebot.cmd_velocity(linear, angular)* beze změny rychlosti, a *Turtle.set_speed(linear, angular)*, která přenastaví hodnoty rychlosti robotu 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)*.

4 Závěr

Funkčnost našeho řešení jsme ověřili na několika drahách, otestovali jsme projíždění v limitních – jak světelných, tak vzdálenostních – podmínkách. Naše řešení je robustní vůči světelným podmínkám. Hlavním nedostatkem implementace je chybějící transformace souřadnic mezi RGB a hloubkovou kamerou. Pokud nastane situace, že se tyčky překrývají v zorném poli robotu, může robot změřit souřadnice jedné tyčky a barvu druhé.

Naše implementace by se dala rozšířit o automatický výpočet jasu obrazu z RGB kamery, pomocí čehož by se automaticky přepočítaly používané konstanty barev. Dále by se daly přepočítat souřadnice získané z RGB a hloubkové kamery pomocí funkce *get_depth_K()* a rovnice:

$$\mathbf{X} = \lambda \cdot \mathbf{K^{-1}} \cdot \mathbf{M_{CAM}}. \tag{7}$$

4.1 Návrh modifikace úlohy, poznámky

4.1.1 Porovnání úloh 2 a 3

Vzhledem k tomu, že jsme si vyzkoušeli splnit jak úlohu 2 „Obří slalom“, tak úlohu 3 „Klasický slalom“, máme možnost úlohy dobře porovnat. K řešení úlohy 3, která by měla být „obtížnější“, jsme využili – po drobných úpravách – většinu funkcí, které jsme napsali pro úlohu 2. Přibylo především vymyšlení logiky objíždění tyčky. Naopak jsme se nemuseli věnovat hledání druhé tyčky branky do páru, které bylo navíc v limitních případech problematické, protože druhá tyčka branky nemusela být v zorném poli kamery. Po celkovém shrnutí je podle našeho názoru úloha 3 mírně jednodušší než úloha 2.

4.1.2 Nedostatek tyček v laboratoři

V laboratoři jsme měli k dispozici velký počet robotů, ale malý počet tyček. Vzhledem k tomu, že v laboratoři chtělo často testovat více týmů než dva najednou, stávalo se, že tyčky na další týmy nevyzbyly. Abychom mohli ladit náš program i v době, kdy bylo v laboratoři více týmů, vyrobili jsme si tyčky svoje (se stejnými rozměry jako poskytnuté).

4.1.3 Návrh na modifikaci

Jako modifikaci úlohy pro příští rok navrhujeme například úkol posouvat robotem barevné krabice na značky odpovídající barvou dané krabici.