Simulátor robotické soutěže

Robert Jedek

Bakalářská práce
2017

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

akademický rok: 2016/2017

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Robert Jedek**
Osobní číslo:	**A14188**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informační a řídicí technologie**
Forma studia:	**kombinovaná**
Téma práce:	**Simulátor robotické soutěže**
Téma anglicky:	**A Robotics Competition Simulator**

Zásady pro vypracování:

1. **Prostudujte robotické soutěže v disciplíně "sledování čáry", tj. typická zadání tratí a pravidel.**
2. **Analyzujte nejběžnější konstrukce robotů, které se na výše uvedených soutěžích vyskytují a jejich řídicí algoritmy.**
3. **Navrhněte architekturu SW simulátoru, který umožní studentům lépe pochopit a rychleji odladit algoritmy pro řízení robotů v soutěži sledování čáry.**
4. **Vyberte technologickou platformu pro implementaci SW simulátoru.**
5. **Implementujte simulátor a otestujte jej s některými typy tratí a robotů z minulých soutěží.**

Rozsah bakalářské práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **KOLÍBAL, Zdeněk. Roboty a robotizované výrobní technologie. Brno: Vysoké učení technické v Brně – nakladatelství VUTIUM, 2016. ISBN 9788021448285.**
2. **DUCHOŇ, František. Snímače v mobilnej robotike. Bratislava: Nakladateľstvo STU, 2012, 97 s. Edícia vysokoškolských učebníc. ISBN 9788022738019.**
3. **NOVÁK, Petr. Mobilní roboty: pohony, senzory, řízení. Praha: BEN – technická literatura, 2005, 243 s. ISBN 8073001411.**
4. **SCHILDT, Herbert. Mistrovství – Java. Brno: Computer Press, 2014. Mistrovství. ISBN 9788025141458.**
5. **SCHILDT, Herbert. Java 8: výukový kurs. 2016. Přeložil Jakub GONER. Brno: Computer Press, 2016. ISBN 9788025146651.**
6. **PECINOVSKÝ, Rudolf. Java 8: úvod do objektové architektury pro mírně pokročilé. 2014. Praha: Grada Publishing, 2014. Knihovna programátora (Grada). ISBN 9788024746388.**

Vedoucí bakalářské práce: **Ing. Tomáš Dulík, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **24. února 2017**

Termín odevzdání bakalářské práce: **24. května 2017**

Ve Zlíně dne 24. února 2017

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové/bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 16.5.2017

podpis diplomanta

ABSTRAKT

Cílem této práce je vytvořit simulátor robotické soutěže v disciplíně sledování čáry. Zprvu jsou zde shrnuta základní pravidla této soutěže, včetně odlišností napříč organizátory. Dále jsou zde popsány základní součásti, ze kterých jsou roboti většinou sestaveni, včetně nejčastějších způsobů jejich řízení. Poté je popsána architektura aplikace, její uživatelské rozhraní a možnosti, které poskytuje formou nastavení a voleb. V rámci popisu implementace seznamuje uživatele s programátorským rozhraním, včetně způsobů jak ho použít při tvorbě vlastního řídícího algoritmu. Po spuštění aplikace může uživatel sledovat simulaci jízdy robota s jeho řídícím algoritmem.

Klíčová slova: Simulátor, sledování čáry, řídící algoritmus

ABSTRACT

The aim of this work was to create a simulator of the "line following" robotic competition discipline. Firstly, the basic rules of this competition, including differences between different events, are summarized. Then, the basic components from which robots are usually assembled, including the most common way of their programming, are described. In the "practical part" section of this document, the simulation application architecture and its user interface are presented. The application programming interface is also introduced, with examples how to use it when creating custom robot control algorithms.

Keywords: Simulator, line following, control algorithm

Chtěl bych poděkovat svému vedoucímu bakalářské práce Ing. Tomáši Dulíkovi, Ph.D. za odborné vedení, za pomoc a rady při zpracování této práce.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Robotické soutěže jsou stále vyhledávanější formou trávení volného času. Zábavnou formou se zde kloubí elektrotechnika a programování. Jde tedy o druh zábavy, který je v době stále se rozšiřující robotizace také naučný. Velkým přínosem je také to, že se nejedná pouze o teoretickou disciplínu, ale o disciplínu komplexní, kdy si každý student příslušného oboru, ale i kdokoli jiný, projde celým procesem výroby svého robota od návrhu až po ověření jeho správné funkčnosti například na soutěži.

Velkou předností této činnosti je bezesporu odvedená práce při jeho stavbě, což nejen že rozvíjí psychomotorické schopnosti, ale umožňuje i přímý fyzický kontakt s jednotlivými součástmi, které robota tvoří. A ve spojení s návrhem algoritmů převedených do konkrétního programovacího jazyka, které přimějí vyrobeného robota, aby dělal to, co je po něm požadováno, dělá robotické soutěže oblíbenými.

Pomocníkem při této činnosti by mohl být simulátor, pomocí něhož by mohl student pozorovat chování robota při změně jeho fyzických vlastností. Dále také návrh robota testovat na různých dráhách, nebo rychleji pozorovat, jak se projeví úprava řídícího algoritmu a odlaďovat tak případné nedostatky před tím, než algoritmus použije u robota a otestuje na konkrétní dráze.

Tato práce si bere za úkol takový simulátor navrhnout a realizovat.

I. TEORETICKÁ ČÁST

1 SOUTĚŽ A PRAVIDLA

Základním úkolem disciplíny „sledování čáry“ je sledovat černou čáru na bílém pozadí. Tato disciplína se řadí mezi ty nejrozšířenější a prakticky je téměř na každé robotické soutěži. Černá čára, kterou robot sleduje je tratí, kterou má robot za úkol projet v co nejkratším čase. Robot musí být autonomní. Musí se tedy rozhodovat sám na základě programu, který je v něm nahrán a nesmí být ovlivňován žádnými vnějšími signály.

Další pravidlo, které platí napříč všemi organizátory robotických soutěží je bezpečnostního rázu a klade důraz na to, že robot nesmí být nebezpečný svému okolí a to jak lidem, tak trati. Míra bezpečnosti nemá většinou žádné další upřesnění a o tom, zda byla tato hranice překročena, má výhradní právo rozhodovat provozovatel soutěže. Toto pravidlo se většinou odvolává na tři zákony robotiky, které zformuloval Isaac Asimov [1]. Jejich znění je následující:

1. Robot nesmí napadnout lidskou bytost.
2. Robot musí poslouchat příkazy lidí, pokud to neodporuje pravidlu číslo 1.
3. Robot musí chránit sám sebe, svoji existenci, pokud to neodporuje pravidlům číslo 1 a 2.

To ovšem nejsou veškerá pravidla, která se musí na soutěžích dodržovat, ale už se více či méně liší dle požadavků jednotlivých organizátorů soutěží [2-10][1]. Tato pravidla lze dále rozdělit do dvou základních kategorií. A to na pravidla, která jsou nějakou formou definována vždy, jen s nějakou obměnou, a dále na pravidla, která jsou vyžadována jen někdy a jsou specifická jen pro tu danou soutěž.

Do první výše uvedené kategorie patří například požadavky na konstrukci robota, kdy se kromě dvou základních (autonomnost a bezpečnost) už žádné další nevyžadují, přes omezení jejich rozměrů, počtu snímačů či motorů až po omezení na konstrukci robotů pouze z nějaké z uvedených stavebnic. Zakoupení a již sestavení roboti jsou většinou zakázáni.

Požadavky nejsou kladeny pouze na roboty, ale i na trať, a jsou opět velmi rozmanité. Mezi nejčastější patří šířka čáry, minimální vzdálenost okrajů a bariér od okraje či osy čáry

[1] Tento výčet rozhodně není kompletním seznamem pravidel a soutěží, protože jen v regionu ČR+SK+Polsko existuje přes 30 robotických soutěží.

a minimální vzdálenost mezi čárami dané trati. Někdy se na trati může vyskytnout nějaká překážka, kterou musí robot objet a zase se navázat na sledovanou čáru. Tato překážka mívá tvar kvádru, často jde o cihlu. Dále pak zda se může trať křížit, nebo se na chvíli dokonce přerušit. Někdy je definována maximální míra nerovností, jindy může dráha dokonce klesat či stoupat, mít uměle vytvořené nerovnosti a podobně. Mezi časté požadavky na soutěžní dráhu také patří, jaký maximální rádius smí zatáčka mít, zda může trať obsahovat pravoúhlé zatáčky, zda na sebe mohou zatáčky bezprostředně navazovat, nebo jestli po zatáčce musí být nějaký minimální rovný úsek a jaké délky tento úsek musí být. Tento úsek se nejčastěji vyžaduje po pravoúhlé zatáčce. Takto by se dalo pokračovat až k těm velmi specifickým požadavkům konkrétní soutěže.

Tyto tratě bývají buď vyvýšeny do úrovně výšky stolu, jindy zase leží přímo na zemi. Někteří organizátoři, kromě závodních tratí poskytují soutěžícím jejich kopie, na kterých si soutěžící mohou své roboty odzkoušet předtím, než přistoupí k samotnému závodu na soutěžní trati. Jindy zase mají soutěžící přidělen nějaký čas na seřízení svého robota na soutěžní trati těsně před jejich soutěžní jízdou.

Další kategorie pravidel se týká samotného průběhu soutěže. Zde mezi hlavní rozdíly na jednotlivých soutěžích patří, zda musí robot po celou dobu své jízdy některou svou částí neustále překrývat sledovanou čáru či ne. Pokud ano, tak je to jasné, a pokud se například v zatáčce objeví část čáry, která není pod robotem, považuje se to za porušení tohoto pravidla a reakce na toto pochybení je opět různá. Druhý přístup tkví v tom, že i když robot na chvíli opustí trať a nezastaví se (nebo se nezačne pohybovat chaoticky, ale začne se evidentně vracet na trať bez toho, aniž by z toho pro něj vyplývala nějaká výhoda, jako je například zkrácení dráhy), je toto vybočení akceptováno. O tom, kdy jde o akceptovatelné vybočení a kdy už ne, rozhoduje rozhodčí. Pokud jde o vybočení, ať už dle prvního či druhého posouzení, tak se reakce na tuto situace opět mohou lišit. Někdy je soutěžící pro tento pokus okamžitě diskvalifikován a jindy organizátoři povolují vrátit robota na místo před tímto nepovoleným vybočením. Počet těchto vrácení bývá omezen a po jeho překročení je soutěžící pro tento pokus také diskvalifikován. Za každé takové porušení se mohou soutěžícímu do celkového času projetí dráhy započítávat trestné sekundy, nebo se za dostatečné zpoždění počítá samotné vrácení se zpět. Z předešlého vyplývá, že na projetí trati bývá většinou více pokusů, přičemž se do výsledků započítává ten nejlepší, tedy nejrychlejší pokus.

Další rozdíly jsou organizačního charakteru soutěže, kdy jsou buď náhodně, nebo na základě pořadí při registraci do soutěže vytvořeny dvojice, které současně soutěží na dvou identických drahách, kdy je soutěžícímu dráha přiřazena opět nějakým losem či pravidlem. Tato organizace soutěže funguje vyřazovacím způsobem, kdy dále v soutěži postupuje vítěz, poražený pokračuje v soutěži pouze za podmínky, že je to jeho první prohra. Tímto způsobem se postupuje, až zůstanou čtyři soutěžící, mezi kterými se rozhodne o pořadí třech nejlepších. Druhý způsob určení vítěze soutěže bývá založen pouze na dosažení nejlepšího času. Pokud se soutěží na více drahách, tak se časy na jednotlivých drahách sčítají. Pokud se soutěžícímu některá z drah nepodaří dokončit, bývá penalizován nějakým předem dohodnutým časem. Často se tyto přístupy kombinují, kdy se do finále kvalifikují pouze soutěžící s nejlepšími časy, a ve finále se postupuje vyřazovacím způsobem.

2 KONSTRUKCE ROBOTŮ

Konstrukce robotů někdy úzce souvisí s pravidly soutěže, protože mohou být povoleni pouze roboti sestavení z konkrétní stavebnice. Takové robotické soutěže bývají například pro roboty ze stavebnice LEGO Mindstorms EV3 [11]. Často využívanou platformou je také Arduino [12]. Existují i stavebnice, které nejsou tzv. krabicové, ale jedná se spíše o velké množství součástí, ze kterých si může každý poskládat robota dle svých požadavků. Robota si tedy můžeme pořídit od finálního výrobku či stavebnice, nebo si ho sestavit z nabízených podvozků, pohonů, elektronických modulů a senzorů. Další skupinou konstruktérů jsou pak ti, co si své roboty sestavují z různých částí, které dříve sloužily v jiných, již nepotřebných zařízeních, tyto následně montují na své vlastní konstrukce podvozků.

2.1 Stavebnice a platformy

2.1.1 Lego

Řídícím centrem tohoto robota je kostka EV3. Tato kostka běží na OS Linux, obsahuje řadič ARM9 300MHz, Flash paměť 16MB a paměť RAM 64MB. Kostka obsahuje displej a sadu tlačítek jako rozhraní pro ovládání robota a vizualizaci jeho stavů. Také obsahuje řadu portů, které slouží pro připojení externích zařízení, jako jsou motory, senzory a také port pro připojení k počítači. Dále je na kostce reproduktor, hostitelský USB port a slot pro SD kartu. Stavebnice obsahuje dva druhy motorů. Pomalejší a silnější, nebo rychlejší, ale slabší. Sada obsahuje také senzory. Jako například barevný, který dokáže detekovat intenzitu odraženého světla, ten se dá využít pro detekci čáry. Dále pak dotykový senzor, který soutěžící často využívají pro objíždění překážky, kdy je tento senzor předsunut před robota a detekce překážky je realizována dotekem tohoto senzoru s překážkou. I když by se mohlo zdát, že jde spíše o kolizi než detekci, bývá tento způsob detekce překážky akceptován. Stavebnice ale obsahuje i infračervený senzor, který dokáže překážku detekovat bez jejich fyzického kontaktu. V závislosti na tvaru a velikosti překážky, dokáže tento senzor překážku detekovat až na 70 cm.

Součástí stavebnice je i software pomocí něhož lze robota programovat. Tento program umožňuje robota naprogramovat pomocí skládání programovacích bloků, které jsou rozděleny do několika kategorií. Jde o bloky akcí, toků, senzorů, dat a bloky pokročilých funkcí. Pro opakující se segmenty poskytuje program vytváření vlastních bloků pro

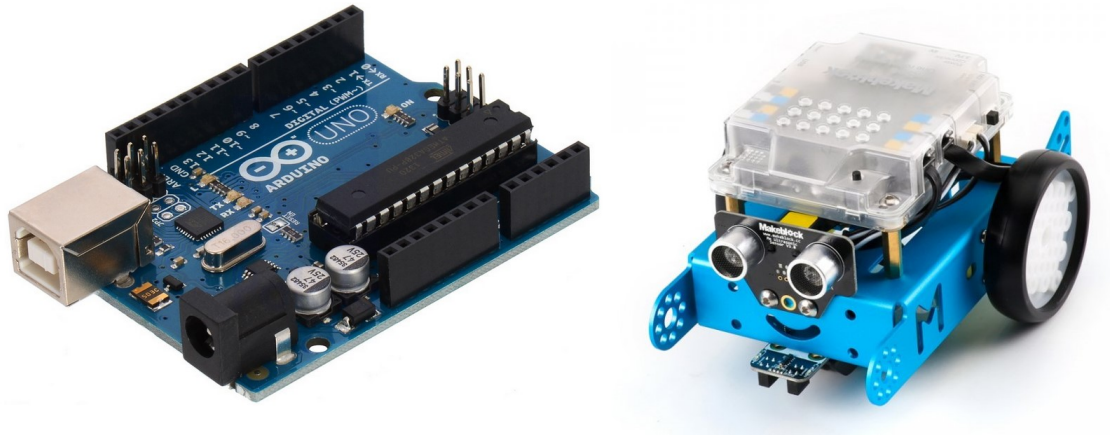
opakované použití. Programování robota je pomocí tohoto nástroje snadné a rychlé. Samozřejmostí je pak podokno konfigurace robota včetně všech nastavení.

Obrázek 1. Stavebnice Lego [11]

2.1.2 Arduino

Základní desku tvoří mikrokontroler od firmy Atmel, převodník pro komunikaci s počítačem, krystal pro taktování, napájecí část a vyvedené piny pro připojení digitálních a analogových vstupů a výstupů. K Arduinu lze připojit řadu rozšiřujících desek, známé jako schieldy. Pro připojení k Arduinu existuje také spousta vstupních a výstupních periferií a modulů. Tento projekt, který byl původně určen pro výuku, se stal natolik oblíbený, že se nyní řadí mezi nejoblíbenější platformy v programování mikropočítačů. Díky této rozšířenosti existují všechny periferie potřebné pro tvorbu robota sledujícího čáru, a proto se také často na soutěžích můžeme s roboty sestavenými právě z Arduina setkat. Programy lze tvořit ve vlastním Arduino IDE, které obsahuje vlastní textový editor s obarvováním kódu a dalšími prvky pro tvorbu zdrojových kódů jako jsou kopírování, vkládání, nahrazení a odsazení textu. Také poskytuje výpis chybových zpráv v případě, že se nějaké vyskytují a nepodařila se kompilace programu. Dále lze vyvolat okno sériové komunikace, tzv. sériový monitor, kde si můžeme při vývoji programu například vypisovat

hodnoty z připojených periférií. K programování Arduina lze využít knihovny Wiring napsané v jazyce C++, jenž usnadňuje programování odstíněním některých detailů, které se musí řešit při programování mikropočítačů v čistém C/C++.



Obrázek 2. Arduino UNO [13], mBot v 1.1 [14]

2.1.3 Ostatní

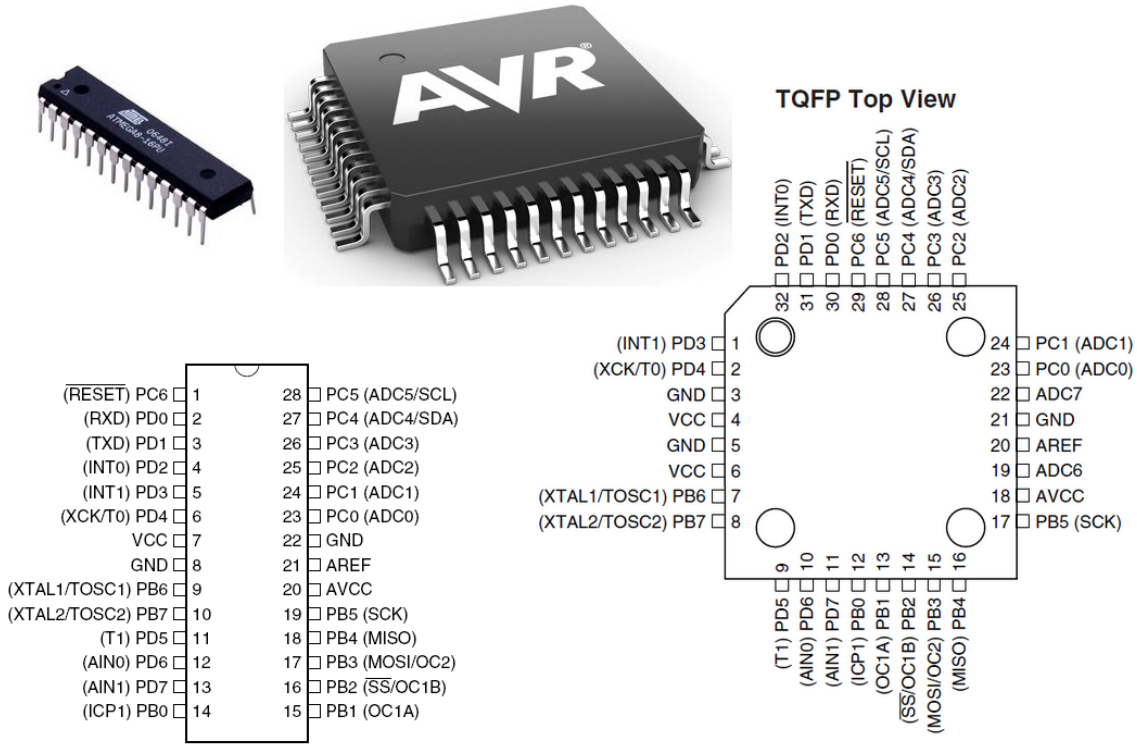
Na soutěžních tratích se můžeme setkat také s roboty, které si konstruktéři sestavili z různých dílů, které byly pro tento účel vyrobeny, nebo dříve sloužili v nějakém jiném zařízení, ale lze jich pro výrobu robota použít. Bývají to kombinace různých částí stavebnic, kterými osazují své vlastnoručně vyrobené podvozky, ale i roboti, kteří jsou originální a jsou vyrobeni jen ze samostatných součástek, bez využití jakékoli stavebnice.

2.2 Součásti robotů

2.2.1 Mikropočítač

Mikropočítač je hlavní částí celého robota, která řídí vše, co je třeba k tomu, aby se robot mohl pohybovat, jak od něj požadujeme. Přijímá signály z jednotlivých senzorů, ty zpracovává a na jejich základě řídí akční členy. Neděje se tak ale samo sebou, jde však o vykonávání sledu jednotlivých instrukcí na základě programu, který je třeba vytvořit a do mikropočítače nahrát. O zpracovávání těchto instrukcí se v mikropočítači stará procesor, který má pro tento účel svou vlastní velmi rychlou, ale malou paměť. Jedná se o tzv. registry a slouží k ukládání výsledků a mezivýsledků. Aby bylo možné do mikropočítače nějaký program nahrát, musí mít tedy i další paměť. Tato paměť bývá typu Flash, která si tato data uchovává i po odpojení napájení a je opakovatelně přepisovatelná, takže můžeme chování robota modifikovat. Dalším typem paměti, kterou mikropočítač obsahuje, je typ

RAM. Tato paměť slouží k práci s daty programu, je dočasná a po odpojení napájení se v ní data neuchovávají. Práce s touto pamětí je rychlejší než s pamětí Flash, ale pomalejší než s registry. Komunikace mezi jednotlivými částmi mikropočítače probíhá pomocí sběrnic. Tyto se pak dále dělí na adresovou, datovou a řídící. Adresová, pomocí níž se přenáší nejčastěji 16-bitová hodnota, tedy adresa zajišťuje, s kým se v rámci procesoru právě komunikuje. Datová sběrnice slouží k přenosu dat. Její šířka koresponduje s tím, kolika bitový je daný procesor. Poslední sběrnicí je sběrnice řídící, která přenáší řídící signály. Například informaci, zda jde o čtení či zápis. Z uvedeného vyplývá, že se opravdu jedná o jakýsi miniaturní počítač, který je integrován do jediného čipu. Výkon tohoto mikropočítače je však pro řízení robota dostačující [15; 16].

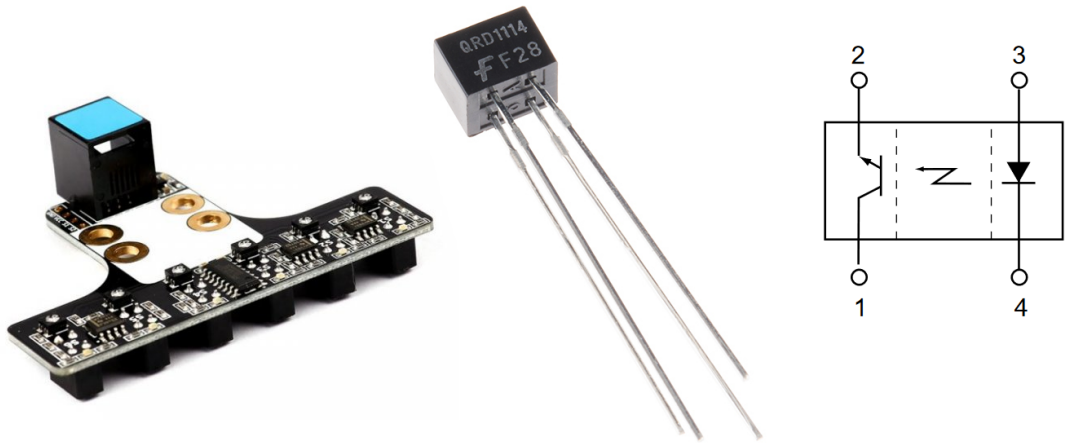


Obrázek 3. Mikropočítače [17-20]

2.2.2 Senzor čáry

Čáru lze detekovat kamerou nebo IR senzorem. Detekce kamerou se využívá u složitějších systémů, než jsou soutěžní roboti pro sledování čáry. Na robotických soutěžích se setkáváme výhradně s roboty, kteří mají detekci čáry založenou na IR senzorech. Tyto senzory jsou založeny na principu snímání odraženého světla. Jako zdroj světla slouží infračervená led dioda, která osvítí sledovaný povrch dráhy. Od tohoto povrchu se toto

záření odrazí, nebo je pohlceno. To je v závislosti na povrchu, který je pod senzorem. Bílý povrch světlo odráží, zatímco černý jej pohlcuje. Intenzita odraženého světla potom koresponduje s tím, kolik světla je odraženo a kolik je ho pohlceno. Této diodě se říká emitor. Jako detektor se pak používá buď fotodioda, fotorezistor nebo fototranzistor. Dále se pak zpracovává intenzita takto odraženého světla, jindy zase stačí detekovat, že byla překročena určitá mez. Na modulu snímače čáry je těchto senzorů více. Toho se využívá pro různé algoritmy řízení, například při drahách s křižovatkami a podobně [21].



Obrázek 4. Senzory čáry [22; 23]

Někteří roboti kromě IR senzoru k sledování čáry ještě používají senzor k detekci barvy čáry. Toho lze využít pro barevné značení úseků trati, kdy jednotlivé barvy mohou robota informovat o složitosti trati, a ten podle toho může přizpůsobit svoji rychlost [24].

Obrázek 5. RGB senzory [25-27]

2.2.3 Senzor překážky

Senzor překážky poskytuje informaci o okolí robota. Při soutěži sledování čáry bývá tato potřeba z důvodu objíždění překážek, které jsou úmyslně vloženy na trať a robot je musí objet a poté pokračovat opět jízdou po čáře. A protože robot musí být autonomní, musí vlastnit mechanizmus, jak pozorovat své okolí. Obecně existují dva druhy těchto senzorů, aktivní a pasivní. U robotů sledujících čáru se vyskytují senzory aktivní a to převážně ultrazvukové a infračervené. Občas na soutěži můžeme vidět robota, který má pro detekci překážky použit senzor taktilní.

Taktilní senzory snímají informace o dotyku, včetně různých parametrů tohoto dotyku. Existují tenzometrické systémy, které například dokážou napodobit lidský hmat. Taktní senzor, který se u soutěžního robota pro detekci překážky používá, mívá tu nejjednodušší formu. V podstatě se jedná o tlačítko, které poskytuje dvě logické úrovně k určení, zda robot je či není v doteku s překážkou. Proto je často nazýván dotekový senzor [21; 28].

Ultrazvukový senzor má oproti dotykovému tu výhodu, že poskytuje informaci o překážce dříve, než se s ní dostane do kontaktu a robot tak má možnost se této překážce vyhnout. Jedná se o aktivní senzor, kdy vysílač vyšle ultrazvukový signál a poté se měří čas, za jak dlouho se vrátí odražený signál od případné překážky [21; 28].

Obrázek 6. Senzory vzdálenosti [29; 30]

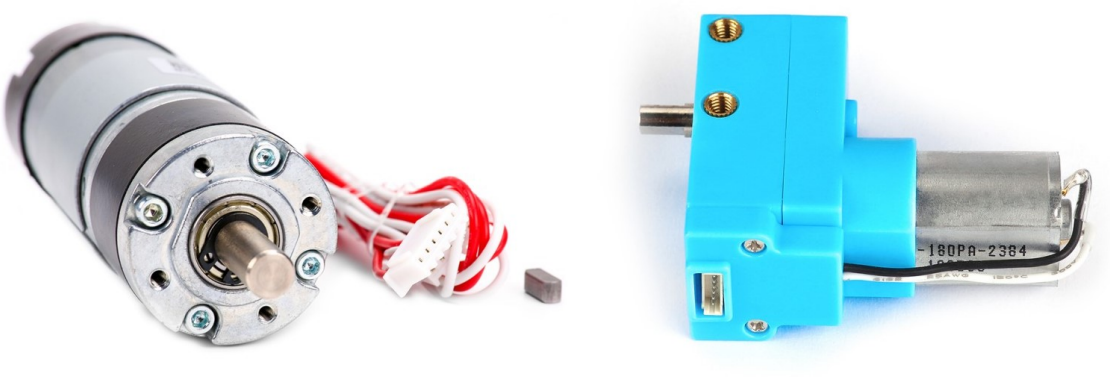
2.2.4 Enkodér

Enkodéry, hlavně ty optické, jsou v robotech užívané velmi často. Jedná se o optický převodník, který poskytuje informaci o změně polohy. Tyto senzory se podle principu činnosti dělí na inkrementální a absolutní a dále dle druhu snímaného pohybu na rotační a lineární. Tento senzor tedy může poskytovat informaci o rychlosti a směru otáčení motorů

robota. Na základě informací z enkodéru můžeme tedy odměřovat i ujetou vzdálenost [21; 28; 31].

2.2.5 Motor pohonu

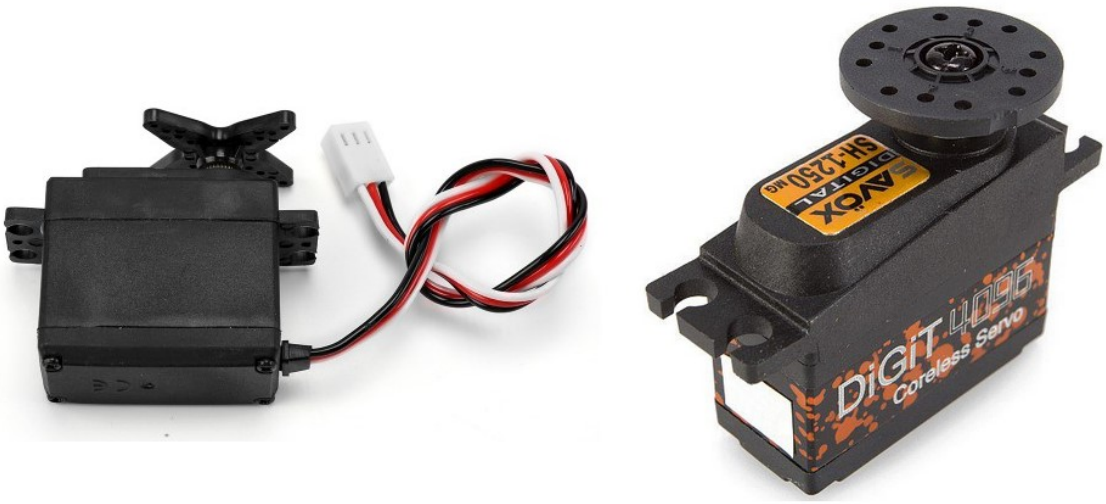
Motory soutěžních robotů jsou stejnosměrné. Protože mají vysoké otáčky a malý moment, používají se společně s převodovkou, která bývá často již součástí motoru. Některé typy motorů obsahují i enkodér. Řízení těchto motorů je snadné, provádí se změnou napájecího napětí. Nežádoucí snížení momentu při snižování napájecího napětí je kompenzováno mechanickou převodovkou. Změnu smyslu otáčení dosáhneme změnou polarity napájecího napětí. Pro řízení rychlosti a směru otáčení se často využívá pulsně šířkové modulace (PWM) a spínací můstek (H-můstek), kdy se pomocí PWM řídí rychlost otáček a pomocí H-můstku směr otáčení. K pohonu robota lze využít i krokového motoru, nebo servo pohonu, ale tyto druhy pohonu se používají jen zřídka [21; 31].



Obrázek 7. Pohony kol [32; 33]

2.2.6 Servo

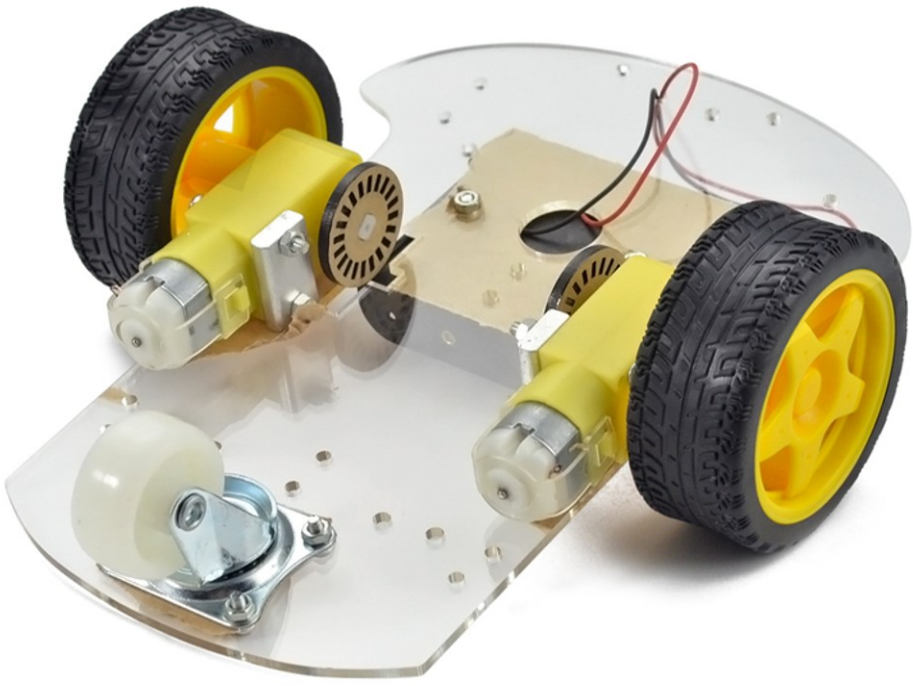
Servo pohon je polohovací jednotka natočení, která je tvořena stejnosměrným motorem, převodovkou a senzorem polohy ve formě analogového enkodéru. Spíše než pro pohon se u robotů využívá k natáčení senzoru detekujícího překážku v okolí robota. Jiný přístup konstruktérů je v použití více senzorů [21].



Obrázek 8. Serva [34; 32]

2.2.7 Podvozek

Naprostá většina podvozků u soutěžních robotů je dvojkolové konstrukce s diferenčně řízenými koly. Výhodou tohoto řešení je snadná manévrovatelnost, kdy je robot schopen otáčet se i na místě a celé řízení je založeno pouze na změně rychlosti, případně směru otáčení jednotlivých kol. Kvůli stabilitě podvozku mají tyto konstrukce podvozků ještě třetí opěrný bod tvořený buď třetím nepoháněným kolem, nebo kuličkovou rolnou [21; 31].



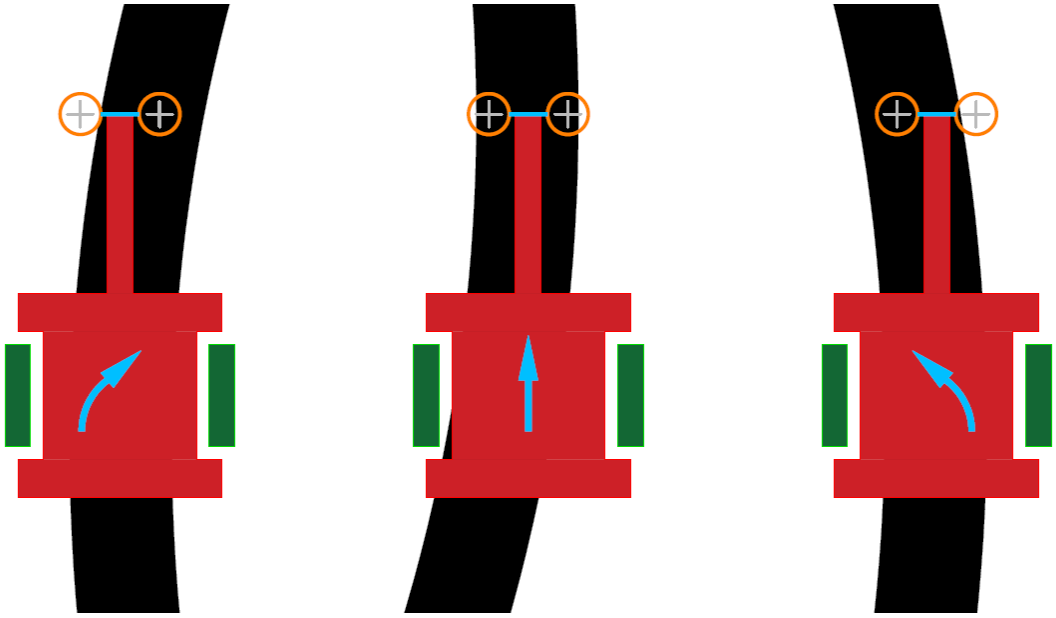
Obrázek 9. Podvozek [36]

3 ŘÍDÍCÍ ALGORITMY

Všechny algoritmy pro řízení robota sledujícího čáru mají společné to, že musí sledovat svou pozici vůči čáře. Tuto pozici musí vyhodnotit a v případě vybočení robota změnit jeho směr, aby se vrátil zpět nad čáru. Výsledkem řídícího algoritmu robota, který má konstrukci s diferenčně řízenými koly, musí být nastavení rychlosti a směru otáčení jednotlivých kol. Hlavním rozdílem řídících algoritmů je to, jak jednotliví roboti vyhodnocují svou pozici vůči čáře, nebo jakým způsobem detekují překážku. Způsob tohoto vyhodnocení má svůj dopad i na nastavení rychlosti a směru jednotlivých kol.

3.1.1 Binární řízení

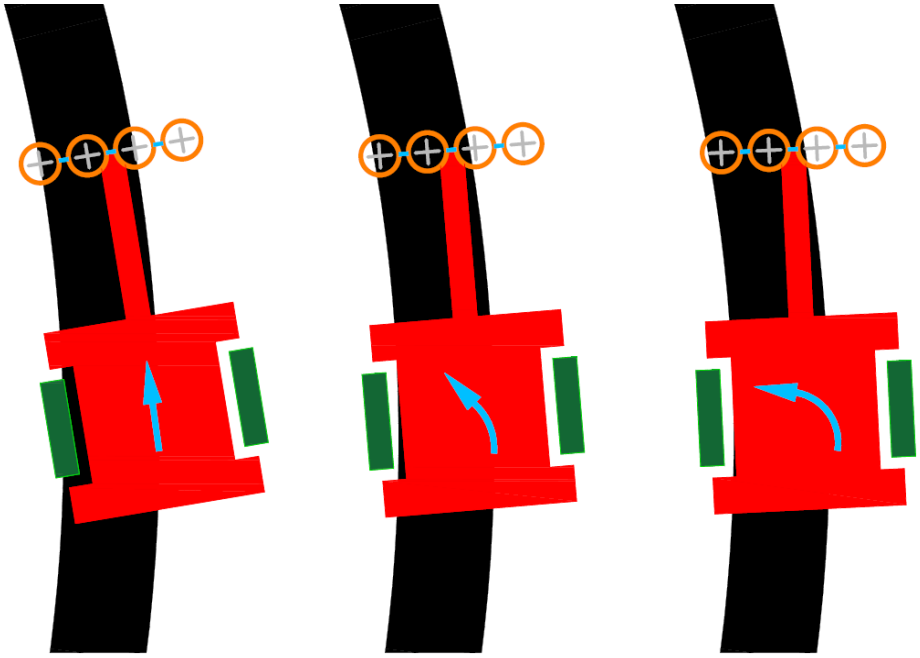
Tento způsob řízení vyhodnocuje pozici čidla pouze dvoustavově, tedy zda je čidlo nad čarou či mimo ni. Existuje nějaká hranice, dle které se toto rozhodnutí překlápí k jedné z možností. Při tomto způsobu snímání polohy je nezbytné mít k rozhodnutí pro nastavení rychlostí motorů informaci minimálně ze dvou čidel. Pomocí informací z jednoho čidla by se totiž v případě opuštění čáry nedalo zjistit, zda robot sjel z čáry směrem vpravo nebo vlevo. A to je zásadní informace pro určení dalšího směru jízdy, tedy nastavení rychlosti motorů. Při použití dvou čidel, která jsou dostatečně blízko u sebe, aby mohla být obě nad čarou, už lze rozhodnout, jestli robot sjel vpravo či vlevo. Modifikací může být uspořádání, kdy jsou od sebe vzdáleny o kousek více než je šířka čáry a za vybočení špatným směrem se bere stav, kdy čidlo vyhodnotí, že je nad čarou. S touto znalostí se již dá rozhodnout, jak nastavit rychlosti motorů, aby se robot držel v požadovaném směru. Pro vyhodnocení situace, kdy se obě čidla nachází mimo čáru, protože se robotovi nepodařilo včas vrátit, je pro toto rozhodnutí nutné ještě znát, kde byla čára detekována naposledy. Další možností je hlídání třemi čidly, kdy se dá opět rozlišovat, zda jede robot po čáře, nebo kterým směrem z ní sjíždí. Čidel bývá na senzoru zpravidla více, kdy se pomocí krajních čidel dá detekovat ostrá zatáčka. Změna rychlosti kol při tomto způsobu řízení bývá skoková, kdy se pomocí kombinace rychlostí a směrů jednotlivých kol sestaví většinou tyto možnosti: rovně, pomalu doprava, pomalu doleva, rychle doprava a rychle doleva. Jednotlivé možnosti pak odpovídají kombinacím stavů obou čidel a jejich předešlých stavů. Že se jedná o tento způsob řízení lze poznat dle „škubavé“ jízdy robota, která je pro tento přístup typická [37; 38].



Obrázek 10. Binární řízení

3.1.2 Číslicové řízení

Druhý způsob je snímání intenzity odraženého světla ze senzoru čáry. Při tomto přístupu se poloha čidla vůči čáře nevyhodnocuje dvoustavově, ale na základě intenzity odraženého světla se zjišťuje na kolik je snímaná oblast pod čidlem tmavá, tedy čarou a nakolik bílým podkladem. Cílem je, aby se robot pohyboval tak, aby bylo čidlo neustále napůl nad černým a bílým podkladem. Proto je střední hodnota ze senzoru, kdy je čidlo přesně napůl nad čarou požadovanou hodnotou pro rozdílový člen. Z rozdílu požadované a aktuální hodnoty vznikne regulační odchylka jako vstupní hodnota pro regulátor. Nejpoužívanějším typem regulátoru je PID, který vygeneruje akční zásah jako součet jednotlivých složek regulátoru. Složka P zesílí regulační odchylku, složka I opravuje odchylky minulé a D složka potlačuje odchylky budoucí. Výstup regulátoru poslouží k nastavení rychlosti obou motorů. Oproti dříve popisovanému binárnímu řízení se tento způsob chová více spojitě, ale jde o regulaci číslicovou, kdy jsou snímané hodnoty diskrétní v hodnotě i čase. Ostatní čidla po stranách opět poslouží k detekci ostrých zatáček [39; 40].



Obrázek 11. Číslicové řízení

3.2 Objíždění překážky

I když je hlavním úkolem disciplíny „sledování čáry“ udržet trajektorii robota na čáře, občas je na trať úmyslně položena překážka, a to nejčastěji v podobě cihly. Zde se k detekci používá senzoru překážky. Způsobů, jak objet překážku je mnoho, ale jedno mají společné, musí první překážku detekovat. Poté se přístupy dají rozdělit na ty, co do jisté míry používají objíždění, kterému se říká „naslepo“ a na ty, co odměřují vzdálenost od překážky po celou dobu objíždění. U způsobu „naslepo“ robot začne překážku objíždět po předem určené trajektorii, většinou obloukovitého tvaru. Při tomto objíždění buď opět detekuje překážku, a postup opakuje, anebo senzorem čáry detekuje návrat na dráhu a pokračuje v jejím sledování. U tohoto postupu se využívá kombinace rychlosti kol a času, po který jsou touto rychlostí poháněna, nebo se využívá měření uražené vzdálenosti pomocí enkodérů. Metoda nepřetržitého monitorování vzdálenosti od překážky se do jisté míry podobá sledování čáry, kdy se místo udržování kontaktu s čárou udržuje taková vzdálenost od překážky, aby ji robot bezpečně objel. Častým způsobem jsou i různé kombinace odměřování vzdálenosti od překážky s odměřováním ujeté vzdálenosti [41-43].

II. PRAKTICKÁ ČÁST

4 ARCHITEKTURA APLIKACE

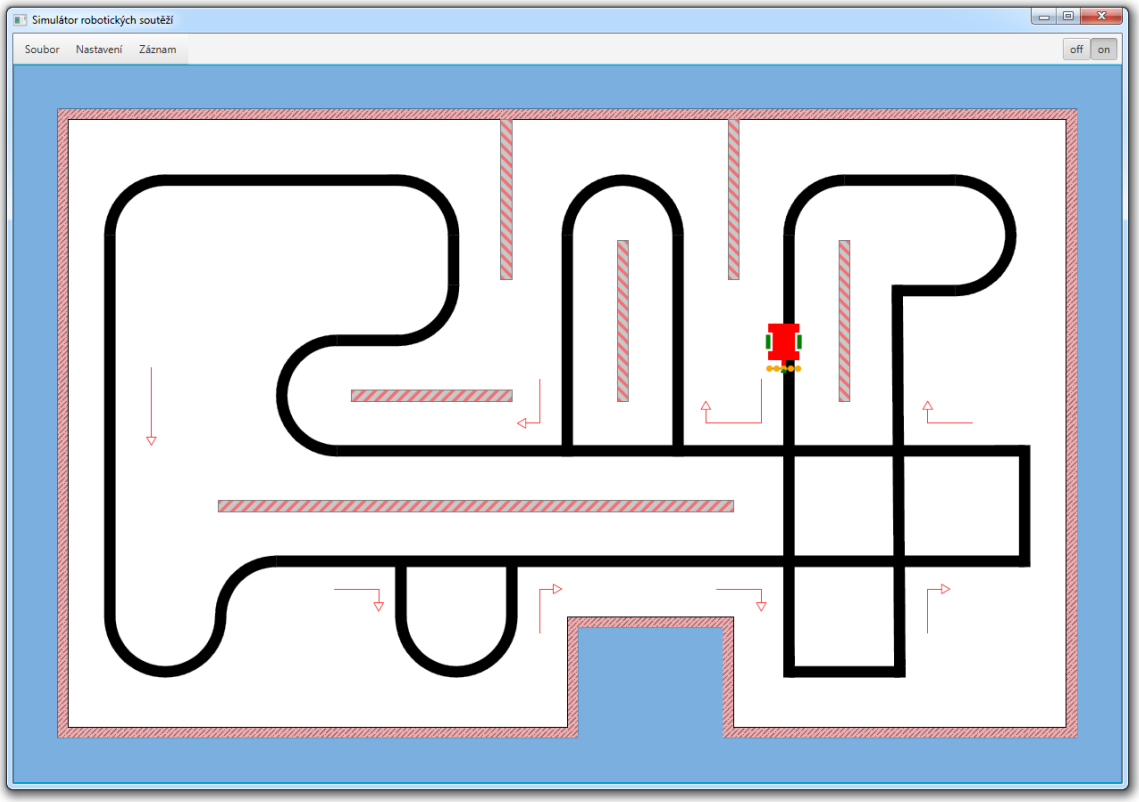
Aplikace obsahuje grafické uživatelské rozhraní s možností nahrát soutěžní trať. Na této trati se pravidelně překresluje poloha robota dle algoritmu uživatele. Aplikace dále poskytuje možnost, jak nastavit různé parametry konstrukce robota. Mezi těmito parametry jsou rozměry robota, a také nastavení související s jeho senzory a akčními členy. Dále existuje mechanismus, jak robota na soutěžní trati posunovat, aby bylo možné nastavit jeho startovní pozici. Nebo tím lze simulovat situaci, kdy je robot při soutěži přemístěn ručně a to například z důvodu opakování průjezdu určitým úsekem trati. Robot má také možnost zastavení a spuštění, což je požadavek vyplývající z pravidel. Aby mohla být soutěžní trať co největší a neubíraly jí prostoru různá nastavení, jsou tato nastavení v samostatných oknech. Jednotlivá okna jsou pro přehlednost rozdělena do menších skupin, které obsahují související položky. Hlavní okno tak kromě trati obsahuje jen nejnutnější položky, jako je menu a možnost spuštění a zastavení robota. Protože cílem aplikace je odlaďovat testované algoritmy, je zde dále možnost logovat uživatelsky definované proměnné reprezentující různé hodnoty používané v řídícím algoritmu. Tyto hodnoty si poté může uživatel prohlédnout v grafu, který se spouští v samostatném okně. Pro tyto hodnoty a graf je v aplikaci také samostatné okno pro jejich nastavení.

4.1 Hlavní okno aplikace

Jedná se o vstupní bod aplikace, hlavní okno se zobrazí po spuštění aplikace. Pomocí menu je zde přístup k ostatním volbám, pomocí nichž lze aplikaci ovládat a zobrazovat ostatní podpůrná okna. Jsou zde také tlačítka pro spuštění a zastavení robota, protože jde o funkci, která by měla být vždy rychle přístupná.

4.1.1 Plátno s dráhou

Tento oddíl hlavního okna zabírá téměř veškerý jeho prostor. Jde o obrázkový soubor rastrové grafiky, který si uživatel může nahrát vlastní a testovat tak svůj algoritmus na jakékoliv dráze. Aplikace již obsahuje tři dráhy, z nichž jedna je nahrána při spuštění aplikace. Načíst jiný soubor s dráhou je možné z menu volbou „Soubor » Načíst dráhu“, nebo pomocí klávesové zkratky „Ctrl + D“.



Obrázek 12. Hlavní okno aplikace

4.1.2 Plátno s robotem

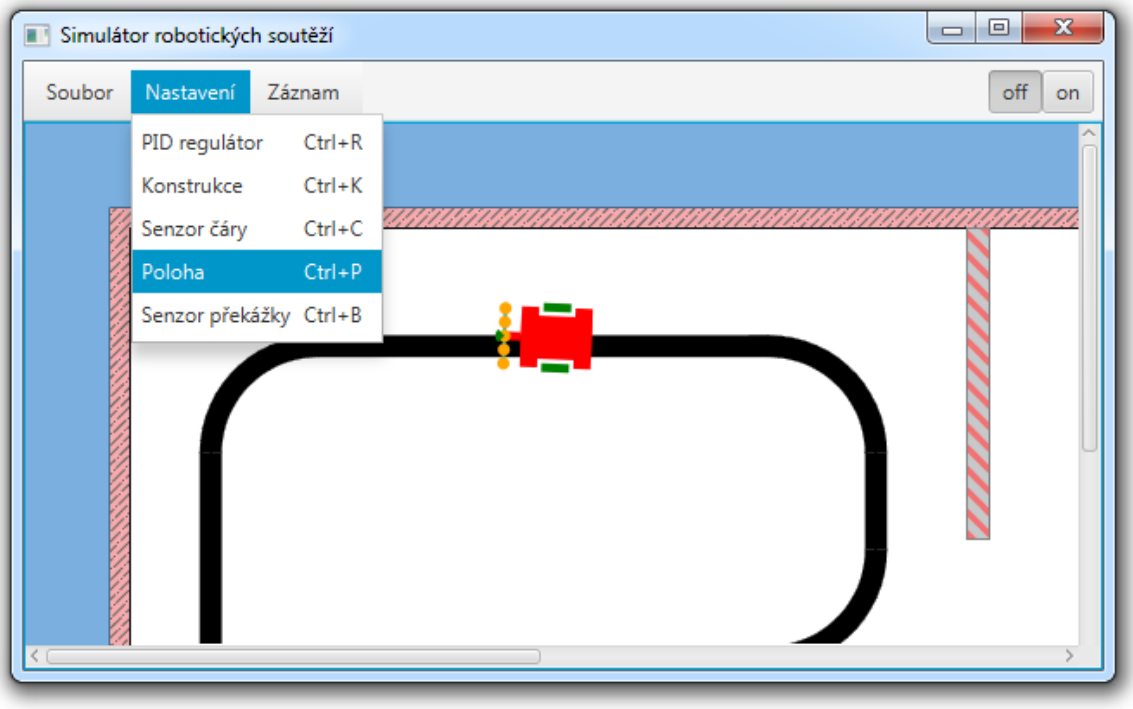
Plátno s robotem je pravoúhlá oblast o stejné velikosti jakou má obrázek s dráhou. Při změně souboru s dráhou se dle potřeby automaticky mění i velikost plátna s robotem. Toto plátno je transparentní a slouží k překreslování polohy robota, která se mění na základě uživatelova řídícího algoritmu.

4.1.3 Menu

Menu je rozděleno do tří hlavních sekcí (Soubor, Nastavení, Záznam). Menu „Soubor“ obsahuje položky pro načtení souboru s dráhou, spuštění robota, zastavení robota a ukončení aplikace. Menu „Nastavení“ obsahuje položky sloužící k zobrazení oken s nastaveními. Jedná se o okna s nastavením PID regulátoru, konstrukce, senzoru čáry, senzoru překážky a nastavení polohy. Menu “Záznam“ obsahuje položky pro pořízení záznamu uživatelských hodnot, jejich zobrazení pomocí grafu a nastavení související s těmito hodnotami a jejich zobrazením. Veškeré položky menu jsou přístupná i pomocí klávesových zkratek, které shrnuje tabulka 1.

Tabulka 1. Seznam klávesových zkratek

Menu	Položka	Zkratka
Soubor	Načíst dráhu	Ctrl + D
	Spustit robota	Ctrl + I
	Zastavit robota	Ctrl + O
	Zavřít aplikaci	Ctrl + E
Nastavení	PID regulátor	Ctrl + R
	Konstrukce	Ctrl + K
	Senzor čáry	Ctrl + C
	Poloha	Ctrl + P
	Senzor překážky	Ctrl + B
Záznam	Graf záznamu	Ctrl + G
	Nový záznam	Ctrl + Z
	Nastavení	Ctrl + N



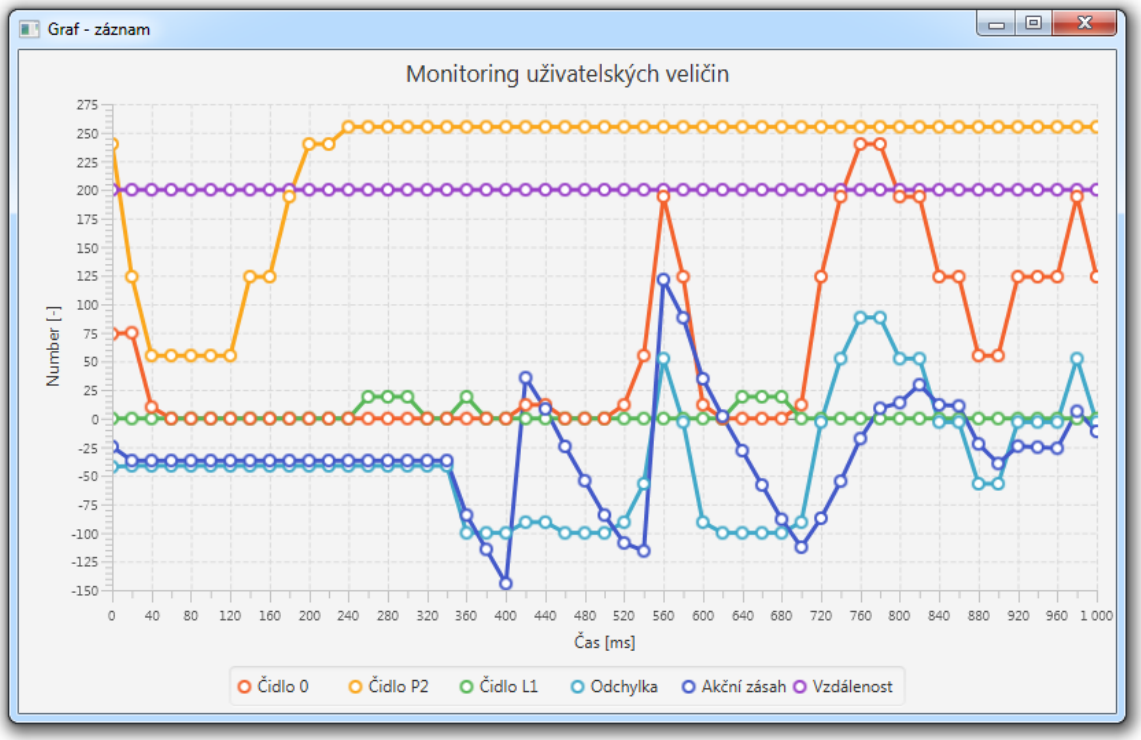
Obrázek 13. Menu aplikace

4.2 Okna grafu

Tato okna slouží k zobrazení uživatelských dat a jejich nastavení. Jsou přístupná z menu „Záznam“.

4.2.1 Graf záznamu

Jedná se o samostatné okno aplikace, které slouží k zobrazení grafu s uživatelsky definovanými hodnotami položek, které zadal, že se mají monitorovat. Uživatel zde tedy může sledovat průběh hodnot, se kterými pracuje ve svém řídícím algoritmu a zpětně analyzovat, jestli je jejich průběh shodný s jeho záměrem. Muže se například jednat o položky jako je intenzita odraženého světla z jednotlivých čidel senzoru čáry, nebo vzdálenost od překážky ze senzoru překážky. Do stejného grafu pak může vykreslovat i hodnoty související s řízením robota, jako je odchylka a akční zásah, a sledovat tak, jak pracuje jeho řídící algoritmus.

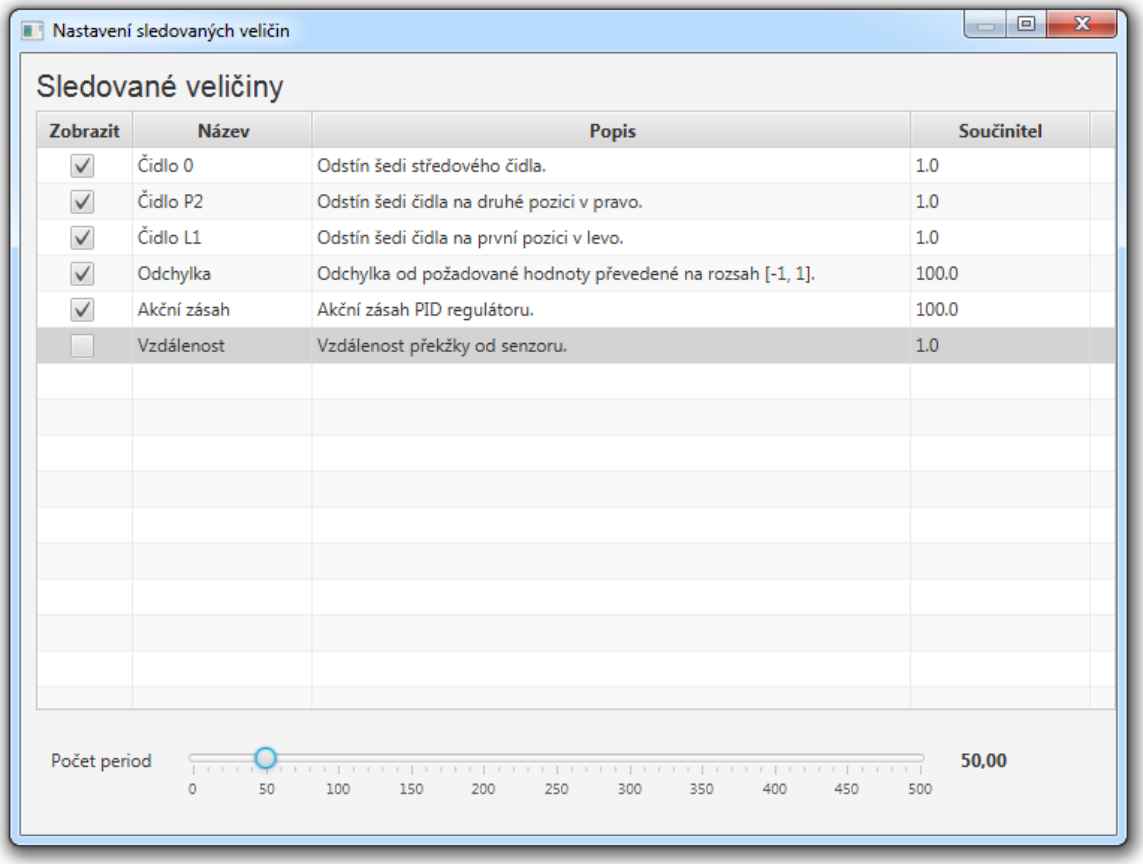


Obrázek 14. Okno grafu s uživatelsky definovanými položkami

4.2.2 Nastavení grafu

V tomto okně je zobrazena tabulka, která obsahuje seznam položek, jenž uživatel zadal, protože je chce zaznamenávat. Tyto položky jsou definovány ve zdrojovém kódu pomocí anotací. První sloupec tabulky (Zobrazit) umožňuje vybrat ze zaznamenávaných položek

ty, které se budou vykreslovat v grafu. Položka, která je označena pomocí zaškrtávacího políčka, bude v grafu zobrazena. Druhý a třetí sloupec (Název, Popis) nesou název zaznamenávané veličiny s upřesňujícím popisem. Jejich obsah určuje uživatel v anotaci sledované veličiny. Poslední sloupec tabulky (Součinitel) je násobící koeficient zaznamenávané hodnoty, aby si mohl uživatel upravit její zobrazení v grafu. Pod tabulkou se ještě nachází posuvník, pomocí něhož se nastavuje počet měřených period. Tento počet odpovídá počtu zobrazovaných měření v grafu. Změna nastavení se zobrazí až v dalším pořízeném záznamu.



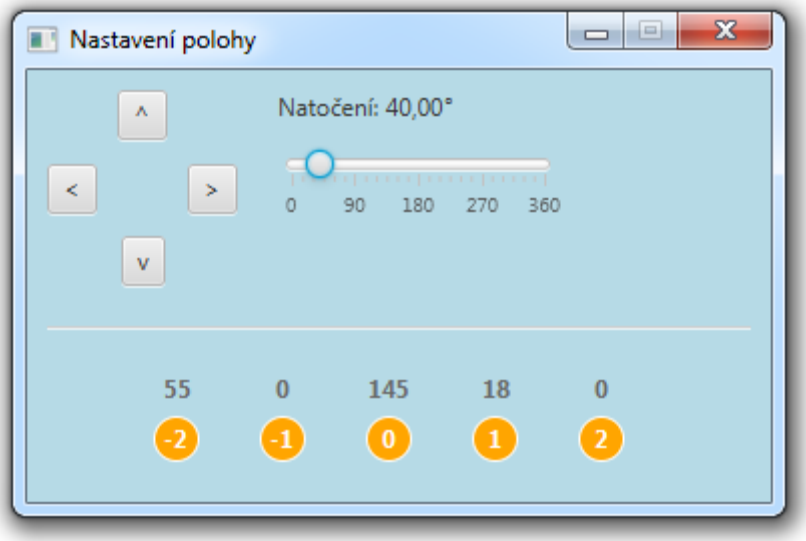
Obrázek 15. Nastavení sledovaných veličin a grafu záznamu

4.3 Okna nastavení

Jedná se o okna přístupná z menu „Nastavení“. Tato okna slouží k nastavení vlastností robota, jeho senzorů a polohy. Je zde také okno PID regulátoru, který lze využít pro řízení. Veškeré změny prováděné v těchto nastaveních jsou okamžitě promítnuty do těch částí, se kterými toto nastavení souvisí.

4.3.1 Poloha

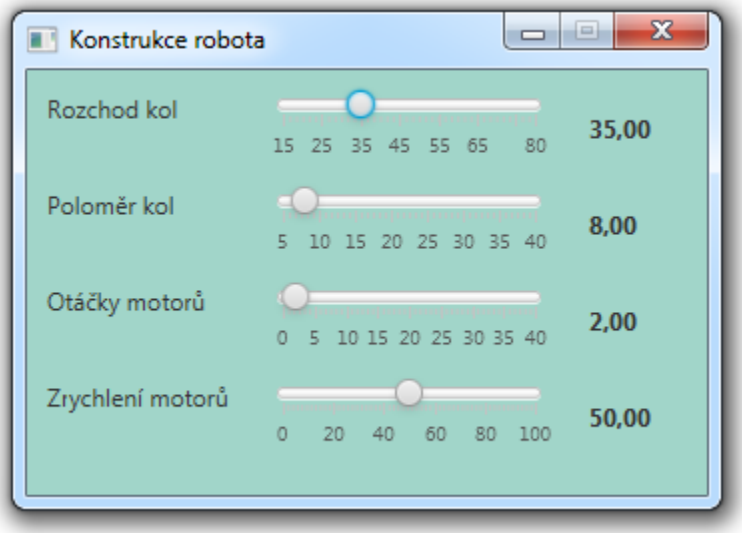
Toto okno slouží k nastavení polohy robota na dráze. Tlačítka s šipkami posunují robota ve směru těchto šipek a posuvníkem se nastavuje úhel natočení. Ve stejném okně pod čarou jsou zobrazena čidla senzoru čáry v počtu, který je dán z nastavení senzoru čáry. Uvnitř vyobrazeného čidla je uvedeno číslo, které koresponduje s tím, jak se na něj uživatel odvolává z kódu, když požaduje jeho snímek. Nad těmito čidly je uvedeno další číslo, které reprezentuje aktuální odstín šedi daného čidla. Spojení nastavení pozice s aktuálními hodnotami senzoru čáry v jednom okně má simulovat reálné pokládání robota na startovní pozici při soutěži. Toto okno není možné zobrazit, pokud je robot zapnutý. V tomto případě je volba zobrazení tohoto okna zašedlá a pro aktivaci této volby je nutné robota zastavit. Za jízdy lze robota přemístit pouze pomocí stisknutí levého tlačítka myši na pozici, kde se má robot přemístit. Je ale nutné počítat s tím, že se v tomto případě zachová jeho natočení.



Obrázek 16. Nastavení polohy robota

4.3.2 Konstrukce

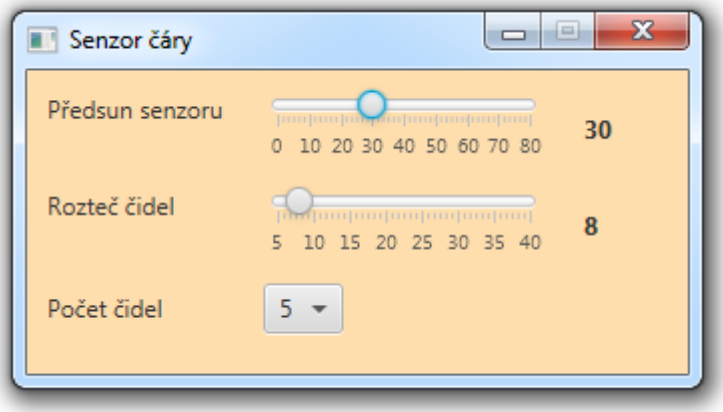
Zde se definují základní rozměry robota, které mají vliv na jízdní vlastnosti robota, jimiž jsou rozchod kol, což je rozměr mezi jejich středy a jejich poloměr. Dále se zde nastavují otáčky a zrychlení motorů, které tato kola pohánějí. Po označení příslušného posuvníku lze pro jemnější nastavení využít směrových šipek na klávesnici.



Obrázek 17. Nastavení vlastností robota a motorů

4.3.3 Senzor čáry

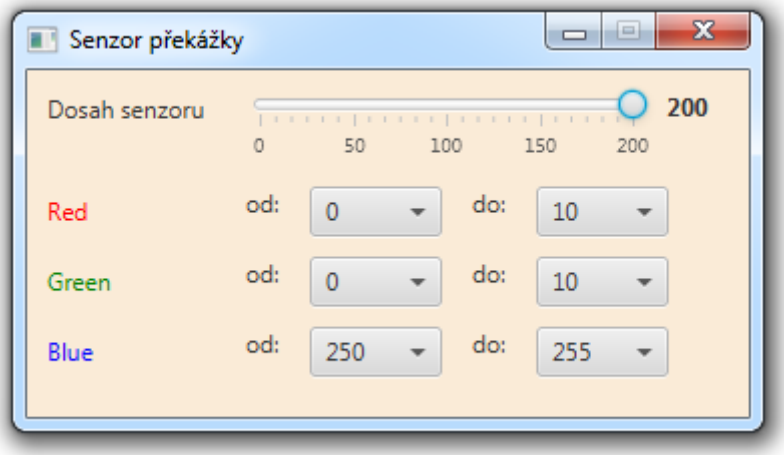
U senzoru čáry je možné nastavovat jeho předsun, což je vzdálenost středu tohoto senzoru od středu mezi koly. Dále pak rozteč mezi čidly, kde se jedná o vzdálenost mezi dvěma sousedícími čidly. Tato rozteč je společná všem takovým dvojicím. Posledním nastavením je počet čidel na senzoru, a to v rozmezí od jednoho po sedm čidel. Protože se změny v aplikaci projevují okamžitě, je nutné u změny počtu čidel pamatovat na to, že pokud se v řídícím algoritmu uživatel odkazuje na neexistující pozici čidla, tak se aplikace zastaví. A to až do doby dokud nebude opět nastaven takový počet čidel, že mezi nimi budou všechna, na která se uživatel z aplikace odvolává. Pokud k takové situaci dojde, upozorní na to aplikace zprávou „Pozice čidla mimo rozsah“ v hlavním okně vedle ovládacích tlačítek robota.



Obrázek 18. Nastavení senzoru čáry

4.3.4 Senzor překážky

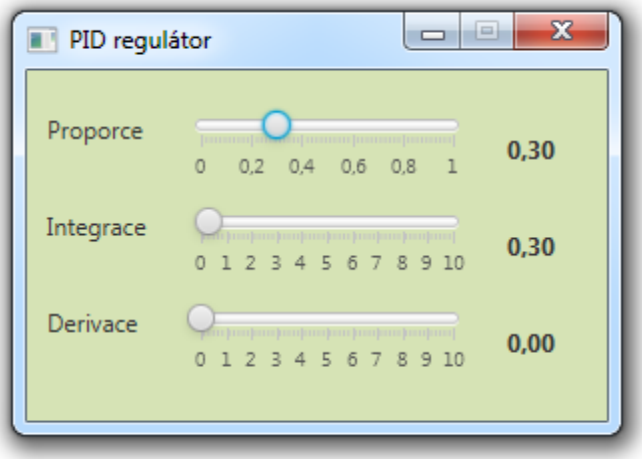
Senzory překážky mají určitý dosah, který se dá nastavit pomocí posuvníku v horní části okna. Pod ním se pak nachází nastavení, kde se pomocí rozsahů jednotlivých složek barevného modelu RGB definuje, co bude pro tento senzor považováno za překážku. Nastaví se tedy pro každou složku barevného modelu rozmezí, ve kterém se tato složka má nacházet. Pokud bude potom na trati nějaký předmět, který bude splňovat uvedená rozmezí ve všech třech složkách, bude považován za překážku.



Obrázek 19. Nastavení senzoru překážky

4.3.5 PID regulátor

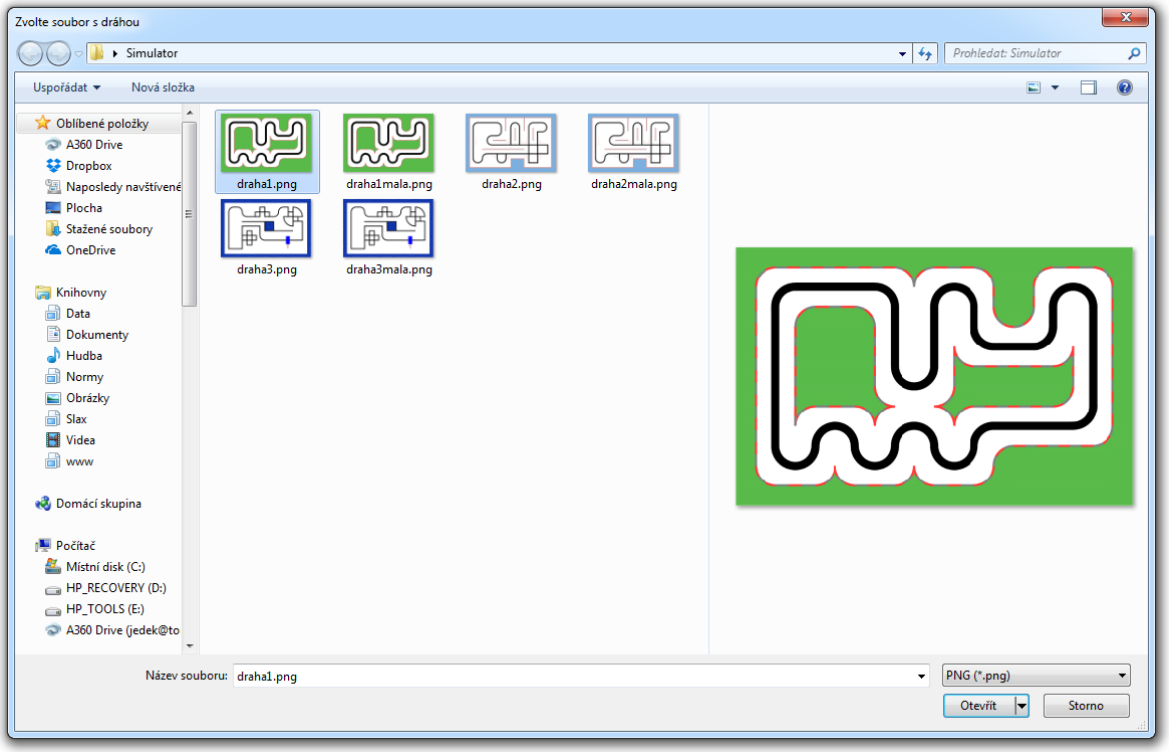
Protože jsou PID regulátory pro řízení robotů hojně využívány, poskytuje aplikace jeden předpřipravený. K nastavení jeho parametrů obsahuje aplikace samostatné okno, kde se nastavují jeho jednotlivé složky.



Obrázek 20. Nastavení PID regulátoru

4.4 Okno načtení dráhy

Toto okno, které je přístupné z menu „Soubor“ slouží pro načtení soutěžní dráhy. Výchozím adresářem pro zobrazení tohoto dialogového okna je ten, ve kterém se nachází spustitelný jar soubor aplikace. Je zde nastaven filtr pro načítání pouze grafických formátů PNG, BMP a JPG.



Obrázek 21. Dialogové okno pro nahrání souboru s dráhou

5 IMPLEMENTACE

Aplikace má dva možné způsoby použití. První způsob, který běžnému uživateli (jenž nemusí být programátorem), umožňuje měnit pomocí implementovaného GUI nastavení parametrů robota a sledovat simulaci toho, jak se různé parametry projevují na funkci vzorové implementace algoritmu sledování čáry. Druhou možností je použití uživatelem - programátorem, který může implementovat vlastní algoritmus sledování čáry dle příkladu třídy *Simulator* a k jeho ověření může využít simulátoru, přičemž v případě potřeby je zde možnost simulaci i ladit (krokovat) pomocí debuggeru v libovolném vývojovém prostředí pro jazyk Java.

5.1 Výběr platformy

K tvorbě simulátoru byla vybrána platforma Java ve své aktuální verzi Java SE 8. Sada JDK 8 obsahuje technologii JavaFX, která je novou technologií platformy pro tvorbu GUI (grafické uživatelské rozhraní) aplikací a je považována za nástupce Swingu pro tvorbu GUI aplikací v platformě Java. JavaFX obsahuje knihovny, které značně vylepšují a usnadňují tvorbu GUI aplikací. Mezi něž patří například vazba mezi vlastnostmi jednotlivých komponent grafického rozhraní, protože jsou veškeré vlastnosti komponent pozorovatelné. Vazby mohou být jednosměrné i obousměrné, dají se libovolně přidávat a mazat, například podmíněně. Pro tvorbu těchto podmíněných vazeb opět existuje mechanismus, který je již součástí JavaFX. Tato technologie dále obsahuje pozorovatelné kolekce, což je rozšířením kolekcí v Javě, kdy je možné sledovat změny v těchto kolekcích a dokonce rozlišovat o jaký druh změny se jedná. Velkým přínosem je také rozšířená podpora pro práci s více vlákny, která je uzpůsobená pro práci s JavaFX GUI aplikacemi. Pomocí balíčku *javafx.scene.canvas*, který nabízí plochu pro kreslení, bude snazší překreslování aktuální pozice robota. V neposlední řadě se při tvorbě aplikace ocení knihovna pro práci s grafy, která značně ulehčí jejich tvorbu. Je tedy patrné, že bude možné celou aplikaci simulátoru napsat pomocí JavaFx bez použití dalších rozšiřujících knihoven, protože tato technologie již obsahuje vše potřebné. Použití JavaFX nijak nebrání použití jakékoliv funkčnosti platformy Java SE 8. Java je tedy velmi dobrou volbou pro tvorbu aplikace, a to i z toho důvodu, že jde o jazyk velmi rozšířený a dokonce co se syntaxe týče, tak je velice podobný zbytku dominujících programovacích jazyků. Proto by neměla být tato platforma překážkou pro tvorbu řídícího algoritmu tvořeného uživateli aplikace.

5.2 Popis implementace

Třídy tvořící aplikaci jsou rozděleny do tří skupin nazývaných „Balíčky“, které je dělí do logických celků obsahující třídy, které spolu nějak souvisí. Tímto způsobem se jednotlivé třídy dělí na jmenné prostory, které zabraňují kolizi názvů tříd. Třída je pak jednoznačně identifikována nejen svým názvem, ale i svým jmenným prostorem. Tento přístup je obzvláště přínosný při využívání knihoven třetích stran, kdy je kolize názvů tříd nejvíce pravděpodobná.

5.2.1 Balíček gui

Tento balíček obsahuje dvě třídy, které mají za úkol definovat grafické uživatelské prostředí. Jsou zde definována okna popsaná v kapitole Architektura aplikace.

Třída SledovaniCary

Každá JavaFX aplikace musí mít své jeviště (Stage) a to zase musí mít minimálně jednu scénu (Scene). Jedná se o objekty vytvářené z tříd, které jsou součástí JavaFX. Jeviště je kontejner nejvyšší úrovně, který obsahuje scénu, která se skládá z komponent tvořící GUI aplikace. Objekt jeviště je vytvořen platformou a předán metodě *start* třídy *Application* jako argument. Třída *Application* je tedy vstupním bodem každé JavaFX aplikace. Její metoda *start* je abstraktní, a proto musí být přepsána vlastní implementací. V jejím těle je vytvořen objekt scény, který v argumentu přijímá „layout“ obsahující jednotlivé vizuální komponenty. Dále se zde nastavují vlastnosti jeviště a události s ním spojené.

Třída GUI

Druhá třída tohoto balíčku s názvem *GUI* je rozvržením, které se předává při vytváření objektu scény v argumentu konstruktoru. Tato třída je poměrně rozsáhlá, protože definuje vzhled celé aplikace včetně všech událostí a vazeb jednotlivých komponent. Dále také zpracovává výsledky práce vláken pracujících na pozadí, které mají vliv na překreslování grafického uživatelského rozhraní.

5.2.2 Balíček robot

Balíček seskupuje třídy, které souvisí s robotem, nebo jeho částmi a třídy, které robot nebo jeho části využívají ke své činnosti.

Třída Cidlo

Třída reprezentuje čidlo, které se nachází na obou druzích senzorů, kterými robot disponuje. Jedná se o senzor čáry a senzor překážky. Čidlo při požadavku na pořízení snímku pořídí v okolí souřadnic, které dostane předané v argumentu, hodnoty složek barevného modelu RGB. Okolím je myšlena matice pixelů okolo bodu definovaného předanými souřadnicemi. Tato matice je definovaná dvojrozměrným polem, kde jeho rozměry reprezentují velikost tohoto okolí, indexy jsou souřadnicemi tohoto okolí a hodnoty definují důležitost tohoto pixelu, se kterou přispívá do váženého průměru pixelů této matice. Pomocí tohoto pole jde tedy ovlivňovat snímané okolí čidla, čímž jde tedy simulovat například jeho výšku nad dráhou. Tato vlastnost není ovlivnitelná z uživatelského rozhraní a musí se nastavovat v kódu pomocí rozměrů a hodnot pole *vahaPixelu*. Algoritmus snímání této matice obsažený ve třídě je implementován tak, aby na tyto změny reagoval. Pokud je pole definováno tímto způsobem:

```
private int[][] vahaPixelu = { { 5, 10, 10, 10, 5 },
                               { 10, 20, 30, 20, 10 },
                               { 10, 30, 35, 30, 10 },
                               { 10, 20, 30, 20, 10 },
                               { 5, 10, 10, 10, 5 }, };
```

Lze snímanou matici graficky zobrazit, jak ukazuje tabulka 2.

Tabulka 2. Váha pixelů čidla

5	10	10	10	5
10	20	30	20	10
10	30	35	30	10
10	20	30	20	10
5	10	10	10	5

Návratovou hodnotou metody vracející snímek senzoru je objekt třídy „SnimekSenzoru“.

K tomu aby mohlo čidlo tyto snímky pořizovat, potřebuje instanci třídy *PixelReader*, která je svázána s aktuálně načtenou dráhou. Tuto instanci včetně připojené závislosti na dráze získává v argumentu konstruktoru.

Třída Enkoder

Tato třída slouží k odměřování úhlové dráhy kola poháněnjícího robota. Uživatel si může ve svém řídícím algoritmu vyžádat instanci enkodéru od objektu robota. Pak stačí zavolat metodu *boolean odmer(double uhlovaDraha)* pro nastartování odměřovaní vzdálenosti z argumentu. Opakovaným voláním se pak hlídá, zda již bylo docíleno odměřované vzdálenosti. Odměřování končí ve chvíli, kdy metoda vrátí hodnotu *true*, což značí, že vzdálenosti z argumentu již bylo dosaženo. Probíhající odměřování lze vynulovat voláním metody *reset*. O načítání ujeté vzdálenosti se stará třída *Controller* z balíčku *simulator*.

Třída Motor

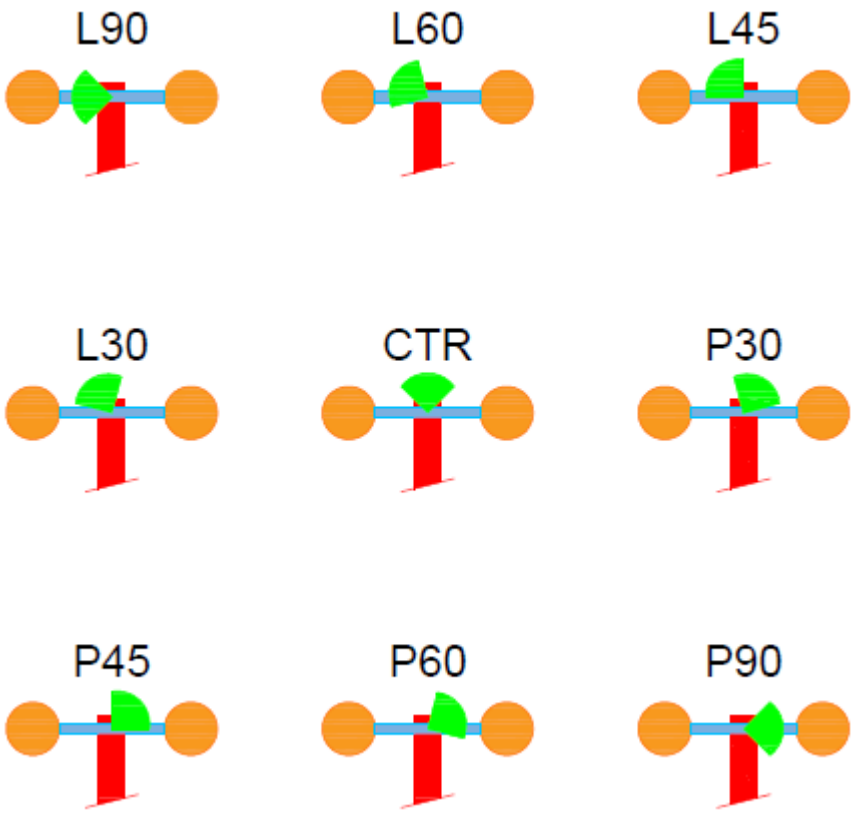
Jedná se o reprezentaci motoru obsahující vlastnosti, které jsou svázány s uživatelským rozhraním z důvodu nastavení jejich hodnot. Dále udržuje informace o aktuální a požadované rychlosti. Požadovaná rychlost v sobě nese informaci o velikosti, ale i směru otáčení indikovaného znaménkem. Tato hodnota je vypočítána z požadavku definovaného uživatelem pomocí dvou parametrů metody *nastavRychlost(double velikost, boolean smer)*. První argument je požadovaná rychlost v procentech a druhý indikuje směr otáčení. Na tuto požadovanou rychlost se dotazuje objekt třídy *Controller*, který rozhoduje o splnitelnosti tohoto požadavku a nastavuje skutečnou rychlost motoru.

Výčet NatoceniSenzoru

Senzor překážky je z důvodu možnosti měřit vzdálenost překážky ve více směrech umístěn na servu. Výčtový typ je takový datový typ, který může nabývat pouze hodnot definovaných tímto výčtem. Výčet *NatoceniSenzoru* definuje směry natočení serva, na kterém je umístěn senzor překážky. Tento výčet tedy reprezentuje toto servo, které má přesně definované polohy natočení. Výčet obsahuje následující hodnoty:

```
L90 (Math.PI / 2),
L60 (Math.PI / 3),
L45 (Math.PI / 4),
L30 (Math.PI / 6),
CTR (0),
P30 (11 * Math.PI / 6),
P45 (7 * Math.PI / 4),
P60 (5 * Math.PI / 3),
P90 (3 *Math.PI / 2);
```

Tyto hodnoty reprezentují natočení z obrázku 22.



Obrázek 22. Možnosti natočení serva se senzorem překážky

Třída Prekazka

Objekt vytvořený z této třídy reprezentuje překážku, definovanou barevným modelem RGB. Tento objekt ke své činnosti využívá senzor překážky, který si z něj získává aktuální nastavení překážky. K synchronizaci s uživatelským rozhraním je zde využito *properties* a *binding API* z technologie JavaFX.

Třída RGB

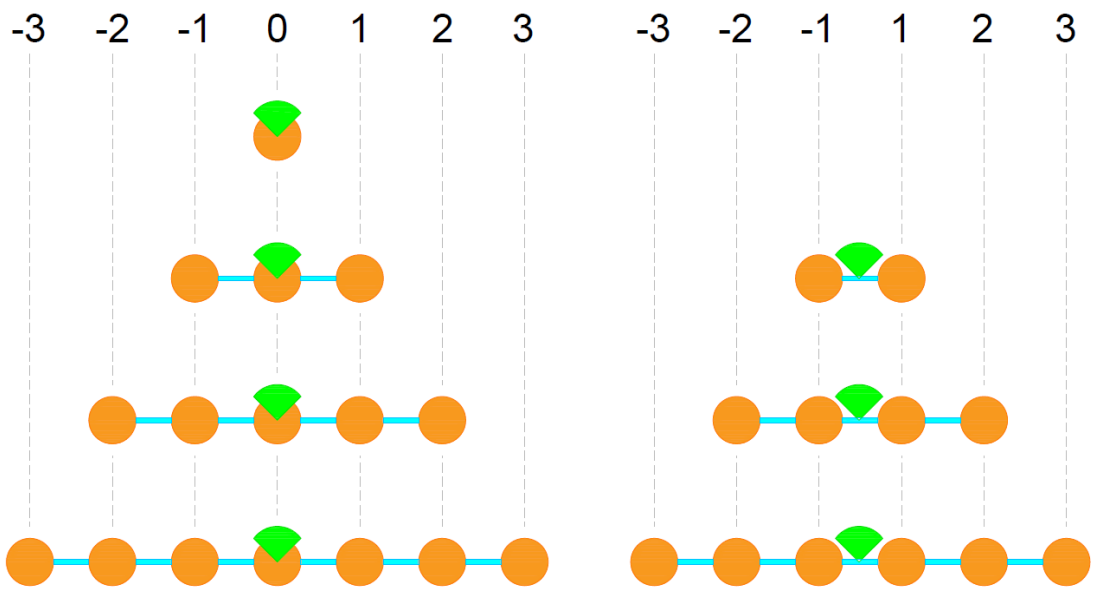
Tato třída je přepravkou jednotlivých složek barevného modelu RGB.

Třída Robot

Zde se jedná o třídu, která v aplikaci reprezentuje robota. Soustředí v sobě tedy jednotlivé součásti, které robota tvoří. Jde o senzor čáry, senzor překážky, levý a pravý motor, jejich enkodéry a informace týkající se rozměrů, jako je rozchod kol a jejich poloměr. Dále v sobě udržuje aktuální pozici a úhel natočení. Dle způsobu použití těchto vlastností robota jsou tyto informace buď zapouzdřeny pomocí členských proměnných, nebo je u nich využito *properties* a *binding API*. Třída dále implementuje dvě veřejné metody, které jsou určeny pro komunikaci s uživatelským řídícím algoritmem. První z nich je:

```
/**
 * Vrátí snímek senzoru (složky RGB a odstín šedi)
 *
 * @param poziceCidla - liché:(-2 -1 0 1 2), sudé:(-2 -1 1 2)
 */
public SnimekSenzoru vratSnimekSenzoruCary(int poziceCidla) {
      return senzorCary.vratSnimekCidla(poziceCidla, getPozX(), getPozY(),
                                        getUhel());
}
```

Již z názvu a návratové hodnoty vyplývá, že tato metoda vrací objekt snímku senzoru, který byl již zmíněn v souvislosti s třídou *Cidlo*, protože právě v této třídě se tento objekt vytváří. Jde tedy o delegování z robota na senzor a ze senzoru na čidlo, které dle souřadnic a definice citlivosti čidla sestaví objekt snímku a ten vrátí pomocí návratové hodnoty do místa volání této metody z řídícího algoritmu. Jako argument se této metodě předává pozice čidla, která je dána následujícím schématem:



Obrázek 23. Označení čidel senzoru čáry

Druhou je metoda, která vrací vzdálenost překážky.

```
/**
 * Vrátí aktuální vzdálenost od překážky v závislosti na dosahu a natočení
 * senzoru.
 *
 * @return vzdálenost z rozsahu nebo (-1)
 */
public double vratVzdalenostPrekazky() {
      return senzorPrekazky.vratVzdalenostPrekazky(getPozX(), getPozY(),
                                                   getUhel());
}
```

Tato metoda vrací kladnou hodnotu od 0.0 do hodnoty dosahu senzoru, pokud se nachází překážka ve směru jeho natočení a dosahu. V opačném případě vrací hodnotu -1. O samotné zjišťování vzdálenosti se starají třídy *SenzorPrekazky* a *Cidlo*.

Třída SenzorCary

Vlastnosti této třídy jsou nastavitelné z uživatelského rozhraní. Jedná se o předsun senzoru, počet čidel a jejich rozteč. Jsou zde implementovány ještě další dvě metody. První s názvem *vratSnimekCidla*, která je veřejná a je to ta metoda, jež využívá třída *Robot* ve své metodě *vratSnimekSenzoruCary,* a která je volána řídícím algoritmem pro získání snímku senzoru na pozici z argumentu. Aby bylo možné tento snímek získat, je třeba znát přesné souřadnice tohoto čidla. Ty se získají tak, že objekt robota přidá k informaci o pozici čidla na senzoru ještě informace o souřadnicích referenčního bodu robota a o jeho natočení. Tyto informace jsou pak společně v podobě argumentů předány metodě *vratSnimekCidla* z objektu senzoru čáry, který díky svým vlastnostem, jako jsou předsun, počet čidel a jejich rozteč a informacím předaných v argumentu, dokáže vypočítat souřadnice čidla na jakékoli jeho pozici. A pokud jsou známy souřadnice, tak už nic nebrání tomu, aby objekt čidla pořídil snímek na těchto souřadnicích způsobem, který je popsán u třídy *Cidlo*. Druhá privátní metoda *vratSouradniceCidla* je pomocnou metodou, jejíž úlohou je zjistit souřadnice, na kterých se bude zmiňovaný snímek pořizovat.

Třída SenzorPrekazky

Tento senzor má jako svou vlastnost opět předsun, je totiž umístěn na servu, jenž je nad senzorem překážky, který má tento předsun nastavitelný. Další vlastností, která je nastavitelná z uživatelského rozhraní, je dosah tohoto senzoru. Natočení senzoru je vlastností této třídy, jež se nastavuje z řídícího algoritmu pomocí metody *setNatoceníSenzoru*.

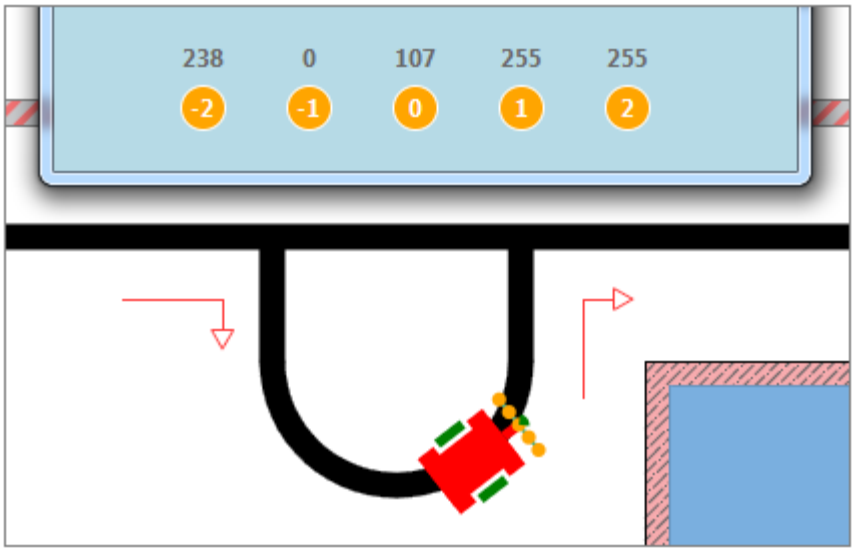
```
/** Nastaví natočení senzoru */
public final void setNatoceniSenzoru(NatoceniSenzoru value) {
      natoceniSenzoru.set(value);
}
```

Třída ke své činnosti ještě potřebuje objekt čidla a překážky. Objekt překážky je jí předán v konstruktoru, v jehož těle si vytváří objekt čidla. K tomu, aby byl senzor schopen zjistit, zda se v jeho dosahu nachází překážka, potřebuje informace o souřadnicích referenčního bodu robota a jeho natočení. Proto je z řídícího algoritmu detekce překážky opět zjišťována přes objekt robota, který tyto informace přidá a předá je metodě objektu senzoru překážky.

Jedná se o metodu *vratVzdalenostPrekazky*. Tato metoda pracuje tak, že ve smyčce postupně pořizuje snímky dráhy ve vzdálenosti nula od umístění senzoru po vzdálenost danou dosahem senzoru. Souřadnice, na kterých se pořizuje snímek, dodává metoda *vratPoziciMereniCidla*, které se kromě informací o souřadnicích a natočení robota ještě předává aktuálně vyšetřovaná vzdálenost od senzoru. Tato metoda pozici vypočítá na základě předaných parametrů a informace o natočení senzoru, toto natočení je vlastností objektu senzoru. Po získání snímku, o který se opět postará objekt čidla, se u jeho jednotlivých složek barevného modelu RGB zjistí, zda jsou v rozsahu definovaném pro překážku. Pokud ano, je tento cyklus ukončen návratovou hodnotou metody, která vrací aktuálně vyšetřovanou vzdálenost, jež vyhověla podmínce definice překážky. Pokud podmínce nevyhoví, pokračuje se další iterací, dokud některý ze snímků podmínce nevyhoví, anebo dokud není aktuálně vyšetřovaná vzdálenost menší nebo rovna dosahu senzoru. Pokud v daném rozsahu žádný ze snímků podmínce nevyhoví, tak metoda vrátí hodnotu -1, která indikuje, že v daném směru a dosahu senzoru nebyla překážka detekována.

Třída SnimekSenzoru

Jde o třídu, jež reprezentuje snímek senzoru, který pořizuje třída *Cidlo* pro potřeby senzoru čáry a senzoru překážky. Třídě jsou v konstruktoru předány složky barevného modelu RGB, které objekt čidla získal pomocí instance třídy *PixelReader*. S využitím Binding API technologie JavaFX je zde vytvářena vazba pro výpočet odstínu šedi, který lze považovat za intenzitu odraženého světla senzoru čáry v rozmezí 0 až 255.



Obrázek 24. Snímek senzoru čáry

5.2.3 Balíček simulátor

Třída Arena

Objekt této třídy se stará o pravidelné překreslování polohy robota. Třída je potomkem třídy *ScheduledService<V>* z balíčku *javafx.concurrent*. Díky tomu je možné používat objekt arény jako plánovanou službu opakující se v pravidelných intervalech. Třídě je v konstruktoru předán objekt robota a plátna arény, protože tyto objekty potřebuje k překreslení polohy.

Třída Controller

Tato třída využívá stejně jako třída Arena *Javafx Concurrency Framework*. Také dědí od třídy *ScheduledService<V>*, ale tentokrát využívá i návratovou hodnotu, kterou určuje generický parametr *<V>*. Jedná se tedy o plánovanou službu, která pravidelně vykonává tělo metody *call* třídy *Task*, kterou služba zapouzdřuje. Třída při vytváření přijímá v konstruktoru instanci robota, regulátoru, a také periodu s jakou je služba spouštěna. Instanci robota a regulátoru obratem použije k vytvoření instance třídy *Simulator*, která ve své metodě run vykonává řídící algoritmus. V konstruktoru se ještě vytváří instance třídy *ControllerTaskData*, která je návratovou hodnotou úkolu této služby. Třídy Simulator a *ControllerTaskData* jsou také z tohoto balíčku a jsou podrobněji popsány dále v této kapitole. V metodě *call* se jako první provede volání metody *run* třídy *Simulator*, která jak již bylo zmíněno, provádí řídící algoritmus, jehož výsledkem by měl být požadavek na nastavení rychlosti a směru otáčení motorů robota. Tento požadavek (rychlost motorů) je součástí instancí pravého a levého motoru, které v sobě obsahuje instance robota. Toto bylo ovšem úkolem simulátoru. Třída *Controller* má za úkol tuto metodu pouze na začátku zavolat, a poté zpracovat její výsledky. V první řadě musí na základě nastaveného požadavku rychlosti motorů zjistit dosažitelnost tohoto požadavku a nastavit aktuální rychlost motorů. V tuto chvíli má objekt třídy *Controller* všechny informace k tomu, aby vypočítal úhlovou dráhu jednotlivých motorů, kterou předá příslušným enkodérům. Provede přepočet úhlové dráhy motorů na obvodovou dráhu kol a vypočítá nové souřadnice referenčního bodu robota a jeho úhel natočení. Dále zjistí aktuální hodnoty sledovaných fieldů (členských proměnných) třídy *Simulator*. Tyto hodnoty společně s hodnotami o nové poloze robota uloží do objektu třídy *ControllerTaskData*, který je následně vrácen jako návratová hodnota. Tato služba je z JavaFX aplikačního vlákna starajícího se o překreslování uživatelského rozhraní pozorovatelná. Díky tomu lze nastavit

událost, která bude vyvolána v okamžiku, kdy služba dokončí naplánovaný úkol a předá návratovou hodnotu, která je v tomto případě objekt třídy *ControllerTaskData*.

Třída ControllerTaskData

Tato třída je přepravkou pro přenos dat do JavaFX aplikačního vlákna, kde jsou její data využita k překreslování uživatelského rozhraní. Je využívána službou třídy *Controller*.

Rozhraní IRegulator

Deklarace rozhraní pro regulátory, které ho implementují.

Rozhraní ISimulator

Toto rozhraní ukládá povinnost třídě s řídícím algoritmem implementovat metodu *run*. Třída Controller, která vytváří instanci třídy *Simulátor,* obsahuje mechanizmus jak zjistit, která třída je jako simulátor použita. Stačí, aby implementovala rozhraní *ISimulator*. Toto zjišťování je provedeno metodou *getSimulatorClass* následovně:

```
/**
 * Zjišťuje název třídy simulátoru pro čtení anotace @SledovanaHodnota
 *
 * Umožňuje jiný název pro třídy simulátoru, tato třída musí implementovat
 * rozhraní ISimulator
 */
public Class<? extends ISimulator> getSimulatorClass() {
	return simulator.getClass();
}
```

Díky této funkčnosti může mít třída, která slouží jako simulátor jakýkoliv název. To uživateli umožňuje mít v balíčku *simulator* více implementací simulátoru a rychle je zaměňovat pouze pomocí názvu třídy, případně i úpravou parametrů konstruktoru, při inicializaci v konstruktoru třídy *Controller*.

```
/** vytvoří instanci controlleru */
public Controller(Robot robot, IRegulator pid, int perioda) {
	this.robot = robot;

	// ================================================================
	// === Zde odkomentujte nebo inicializujte požadovaný simulátor ===
	// ================================================================

	/** Kompletní případ s objížděním překážky */
	simulator = new Simulator(robot, pid);

	/** Binární řízení bez paměti */
	//simulator = new SimulatorBinarneBezPameti(robot);

	/** Binární řízení s pamětí */
	//simulator = new SimulatorBinarneSPameti(robot);
```

```
        /** Číslicové řízení */
        //simulator = new SimulatorCislicove(robot, pid);

        /** Číslicové řízení s detekcí pravoúhlé zatáčky */
        //simulator = new SimulatorCislicoveVpravo(robot, pid);

        this.perioda = perioda;
        this.data = new ControllerTaskData();
}
```

Třída PID

Jedná se o PID regulátor, který může uživatel aplikace využít ve svém řídícím algoritmu.

Třída PolohaService

Tato třída je také potomkem třídy *ScheduledService<V>,* jde tedy o plánovanou službu. Tato služba je podmíněně spouštěna při zobrazení okna s nastavením polohy a na pozadí pořizuje snímky jednotlivých čidel senzoru čáry a ty předává do JavaFX aplikačního vlákna pro aktualizaci zobrazovaných hodnot u těchto čidel.

Třída PolohaServiceTaskData

Další třída, která je přepravkou pro přenos dat do JavaFX aplikačního vlákna. Tuto třídu jako návratovou hodnotu používá služba třídy *PolohaService*.

Třída Pozice

Třída reprezentující souřadnice referenčního bodu a úhlu natočení robota, která je součástí návratové hodnoty služby třídy *Controller* prostřednictvím třídy *ControllerTaskData*.

Třída Simulator

Třída *Simulator*, přesněji třída implementující rozhraní *ISimulator* je ta, která slouží k vytvoření uživatelského kódu s řídícím algoritmem. Řídící algoritmus je nutné vložit do těla metody *run*. V těle této metody by měl uživatel na základě hodnot získaných pomocí senzorů a zvoleného algoritmu řízení nastavit požadavek na rychlost pro každý z motorů. Třída dále obsahuje mechanizmus, jak označit proměnnou, jejíž hodnotu chce sledovat. Průběh této proměnné je poté možné sledovat v grafu. Označuje se pomocí anotace *@SledovanaHodnota* následujícím způsobem:

```
@SledovanaHodnota(nazev = "Čidlo 0", popis = "Odstín šedi středového čidla.")
private int sSenzor;
```

Parametr *nazev* slouží pro uložení názvu sledované hodnoty, který se bude zobrazovat v legendě grafu a tabulce nastavení sledovaných veličin. Hodnota z parametru *popis* je pak viditelná pouze v tabulce nastavení a slouží pro upřesňující informace sledované hodnoty.

Anotace SledovanaHodnota

Jedná se o anotaci k označení proměnných, které mají být sledovány pomocí reflexe. Samotná anotace je označena standartní anotací Javy a to proto, aby byla její životnost zachována až do běhu aplikace a byla viditelná virtuálním strojem Javy.

Třída SledovanaVelicina

Jedná se o třídu, která v sobě uchovává informace spojené s proměnnými, které jsou označeny anotací *@SledovanaHodnota*. Jedná se o členy anotace *nazev* a *popis*, dále pak o *properties pozadavekNaZobrazeni* a *pomer* sloužící k propojení s tabulkou nastavení. Třída také obsahuje kolekci uložených hodnot sledované veličiny pro zobrazení v grafu.

5.3 Ukázkové implementace řídících algoritmů

Součástí zdrojových kódů je i pět ukázkových implementací řídících algoritmů, z nichž každá je v samostatné třídě. Všechny tyto třídy implementují rozhraní *ISimulator*. Díky této technice, popisované u rozhraní *ISimulator*, lze ve třídě *Controller* jednoduše zaměňovat implementace těchto tříd s řídícím algoritmem. Je ovšem nutné, aby měly různé názvy, protože jsou umístěny ve stejném balíčku.

5.3.1 Binární řízení bez paměti

Tato ukázka demonstruje nejjednodušší formu řídícího algoritmu, která je nutná k tomu, aby byl robot schopný projet vyznačenou dráhu. Jednoduchost tohoto algoritmu sebou nese určitá omezení na konstrukci robota, jeho rychlost, ale i na složitost trati. Celé řízení je ukázáno v následujícím kódu, který je obsažen ve třídě *SimulatorBinarneBezPameti*.

```
// nastavení úrovně čidla L1 (první vlevo), 0 => bílá, 1 => černá
leveCidlo = (robot.vratSnimekSenzoruCary(-1).vratOdstinSedi()) > 10 ? 0 : 1;

// nastavení úrovně čidla P1 (první vpravo), 0 => bílá, 1 => černá
praveCidlo = (robot.vratSnimekSenzoruCary(1).vratOdstinSedi()) > 10 ? 0 : 1;

// 0 => bílá; 1 => černá
if(leveCidlo == 1 && praveCidlo == 1){
	levyMotor.nastavRychlost(100, true);
	pravyMotor.nastavRychlost(100, true);

} else if (leveCidlo == 0 && praveCidlo == 1){
	levyMotor.nastavRychlost(100, true);
```

```
        pravyMotor.nastavRychlost(50, true);

} else if (leveCidlo == 1 && praveCidlo == 0){
        levyMotor.nastavRychlost(50, true);
        pravyMotor.nastavRychlost(100, true);
}
```

Předpokladem je použití dvou čidel senzoru čáry, které jsou od sebe v takové vzdálenosti, že se obě nacházejí nad sledovanou čárou. V takovém případě se motorům předává požadavek na plný chod vpřed. Pokud robot detekuje vybočení jedním z čidel (vrací hodnotu 0), tak se motoru, který je na stejné straně jako toto čidlo, nechá požadavek na plnou dopřednou rychlost a druhému se sníží rychlost na polovinu.

5.3.2 Binární řízení s pamětí

Tento způsob řízení rozšiřuje předchozí způsob tak, že si navíc pamatuje předchozí stavy čidel senzoru čáry. Rozšiřuje tak řízení o možnost rozhodovat se o provedeném zásahu nejen na základě aktuálního stavu čidel, ale k rozhodování může použít i stavy předešlé. V ukázce níže je tato možnost využívána v situaci, kdy sledovanou čáru opustí obě čidla. Pak robot začne prudce zatáčet ve směru, kde byla čára detekována naposledy. Algoritmus je součástí třídy *SimulatorBinarneSPameti.*

```
// paměť předchozích hodnot
leveCidloEx = leveCidlo;
praveCidloEx = praveCidlo;

// nastavení úrovně čidla L1 (první vlevo), 0 => bílá, 1 => černá
leveCidlo = (robot.vratSnimekSenzoruCary(-1).vratOdstinSedi()) > 10 ? 0 : 1;

// nastavení úrovně čidla P1 (první vpravo), 0 => bílá, 1 => černá
praveCidlo = (robot.vratSnimekSenzoruCary(1).vratOdstinSedi()) > 10 ? 0 : 1;

// 0 => bílá; 1 => černá
if(leveCidlo == 1 && praveCidlo == 1){
        levyMotor.nastavRychlost(100, true);
        pravyMotor.nastavRychlost(100, true);

} else if (leveCidlo == 0 && praveCidlo == 1){
        levyMotor.nastavRychlost(100, true);
        pravyMotor.nastavRychlost(50, true);

} else if (leveCidlo == 1 && praveCidlo == 0){
        levyMotor.nastavRychlost(50, true);
        pravyMotor.nastavRychlost(100, true);

        // Zde je práce s minulými hodnotami (pamětí)
} else if (leveCidlo == 0 && praveCidlo == 0){

        if (leveCidloEx == 0 && praveCidloEx == 1){
                levyMotor.nastavRychlost(100, true);
                pravyMotor.nastavRychlost(20, false);
```

```
        } else if(leveCidloEx == 1 && praveCidloEx == 0){
                levyMotor.nastavRychlost(20, false);
                pravyMotor.nastavRychlost(100, true);
        }
}
```

V algoritmu přibyly dvě proměnné, ve kterých jsou ukládány předchozí hodnoty úrovní čidel. S těmi se poté pracuje, pokud je zjištěno, že je robot oběma čidly mimo čáru. Zde algoritmus rozhoduje na základě toho, které z čidel detekovalo čáru naposledy. Předchozí stavy lze samozřejmě využívat i v ostatních situacích.

5.3.3 Číslicové řízení

V tomto algoritmu se využívá senzoru čáry způsobem, kdy se ze snímku čidla využije jeho naměřená intenzita odraženého infračerveného světla. Tuto intenzitu v rozsahu 0 až 255 vrací senzor čáry pomocí objektu, který reprezentuje snímek senzoru. Požadovaným cílem je udržet robota tímto čidlem na hraně sledované čáry, tedy někde kolem poloviny rozsahu odstínu šedi z objektu snímku senzoru. K tomuto účelu algoritmus využívá předpřipraveného PID regulátoru. Celý obsah tohoto řídícího algoritmu je ve třídě *SimulatorCislicove*.

```
// získá odstín šedi z čidla P1 (první vpravo)
cidlo = robot.vratSnimekSenzoruCary(1).vratOdstinSedi();

// vypočte odchylku z vráceného odstínu
// implementace rozdílového členu e = w - y
// meze: pro odstín 0 odchylka -1; pro odstín 255 odchylka 1
if (cidlo < 128) {
        odchylka = cidlo * (1.0 / 128) - 1;
} else {
        odchylka = (cidlo - 255) * (1.0 / 128) + 1;
}

// jednoduchý PID regulátor, kterému se předá odchylka
// a on vrátí akční veličinu
// Jedná se o třídu PID, tu můžete upravovat,
// nebo si napsat svoji vlastní
u = regulator.vratAkcniVelicinu(odchylka);

// procentuální nastavení výkonu motorů
if (u > 0) {
        if (u > 1) {
                levyMotor.nastavRychlost(10, false);
        } else {
                levyMotor.nastavRychlost((1 - u) * 100.0, true);
        }
        pravyMotor.nastavRychlost(1 * 100.0, true);
} else {
        levyMotor.nastavRychlost(1 * 100.0, true);
        if (u < -1) {
```

```
                pravyMotor.nastavRychlost(10, false);
        } else {
                pravyMotor.nastavRychlost((1 + u) * 100.0, true);
        }
}
```

Zde je vidět, že jsou rychlosti motorů nastavovány v závislosti na velikosti akčního zásahu vypočítaného regulátorem. Toto nastavení je mnohem plynulejší, což se projevuje i na plynulosti zatáčení robota. Pokud nastane situace, kdy regulátor vrací příliš velký akční zásah, což značí zatáčku, jejíž projetí se nedaří algoritmem obsloužit, je pomocí zpětného chodu příslušného kola nastaveno rychlé odbočení směrem k hraně čáry. I zde je ale možné místo napevno nastavené hodnoty zpětného chodu využít jejího plynulého nárůstu. Tato změna je naznačena v kódu níže.

```
if (u > 0) {
        if (u > 1) {
                levyMotor.nastavRychlost(10, false);
                levyMotor.nastavRychlost((1 - u) * 100.0, false);
        } else {
                levyMotor.nastavRychlost((1 - u) * 100.0, true);
        }
        pravyMotor.nastavRychlost(1 * 100.0, true);
} else {
        levyMotor.nastavRychlost(1 * 100.0, true);
        if (u < -1) {
                pravyMotor.nastavRychlost(10, false);
                pravyMotor.nastavRychlost((1 + u) * 100.0, false);
        } else {
                pravyMotor.nastavRychlost((1 + u) * 100.0, true);
        }
}
```

5.3.4 Číslicové řízení s detekcí pravoúhlé zatáčky

Často se při soutěžích stává, že robot musí projet pravoúhlou zatáčkou, kdy jde většinou o křižovatku, na které má robot zatočit vpravo. Pro tento případ, je vhodné upravit některý z předešlých algoritmů a doplnit ho o rutinu, která bude tyto situace obsluhovat. Ukázkový algoritmus z třídy *SimulatorCislicoveVpravo* využívá ten z předchozí ukázky, tedy číslicové řízení. Pro obsluhu této situace řídícím algoritmem je využito dalšího čidla senzoru čáry. Zatáčku detekuje pouze na pravé straně, pomocí čidla, které musí být v dostatečné vzdálenosti od sledované čáry, aby nedocházelo k falešné detekci zatáčky. Detekce pravé strany je minimum, které je třeba provést pro cílené zatočení vpravo, pokud senzor projíždí křižovatkou. Napřed je tedy nutné získat snímek tohoto čidla.

```
// získá odstín šedi z čidla P2 (druhé vpravo)
p2Senzor = robot.vratSnimekSenzoruCary(2).vratOdstinSedi();
```

Poté je nutné hlídat stav najetí tohoto čidla nad čáru, která svírá se směrem jízdy robota pravý úhel.

```
if (p2Senzor < 180) {
	levyMotor.nastavRychlost(100, true);
	pravyMotor.nastavRychlost(100, false);
	odboceni = 90;
}
```

Kdy se po detekci této situace okamžitě nastaví požadavky motorům pro rychlé zatáčení vpravo. A také se nastaví pomocné proměnné *odboceni* hodnota 90, která značí, že je detekována zatáčka vpravo. Tato proměnná slouží jako parametr přepínači (příkaz switch), který rozhoduje, která obslužná rutina se bude vykonávat. Je to z toho důvodu, že zatáčení je v ukázkovém algoritmu provedeno s pomocí enkodéru a nelze ho provést v jednom kroku. Tím zároveň dojde k odstavení algoritmu sledování čáry, který by v této situaci prováděl opačný zásah, stáčel by jízdu na levou stranu. Po odměření požadované dráhy, která zaručí požadované odbočení se proměnná *odboceni* nastaví na hodnotu 0, jež značí algoritmus sledování čáry. Kód s obsluhou zatáčení je následující:

```
// zatočení vpravo
case 90:
	boolean enk = robot.getLevyEnkoder().odmer(2);
	if (!enk) {
		levyMotor.nastavRychlost(100, true);
		pravyMotor.nastavRychlost(0, true);
	} else {
		odboceni = 0;
		levyMotor.nastavRychlost(50.0, true);
		pravyMotor.nastavRychlost(50.0, true);
	}
	break;

default:
	break;
```

Pokud se pro příkaz *case* s hodnotou 0 přiřadí kód s algoritmem sledující čáru, vznikne řídící systém, který se přepíná v závislosti na hodnotě čidla detekující pravoúhlou zatáčku na pravé straně ve směru jízdy.

5.3.5 Kompletní případ s objížděním překážky

Tato ukázková implementace, která je obsahem třídy *Simulator,* je rozšířením předešlého algoritmu z třídy *SimulatorCislicoveVpravo.* Je zde přidána ukázka kódu, který je jedním z možných způsobů, jak objet překážku umístěnou na dráze. Je zde využito přepínače již vytvořeného z důvodu odbočování vpravo, kde jsou přidány další příkazy *case*, které reprezentují jednotlivé fáze objíždění. Překážku je nutné první detekovat, proto je přidána

nová proměnná, do které je ukládána aktuálně naměřená vzdálenost senzoru překážky. Pokud má být hodnota této proměnné zaznamenávána, musí obsahovat anotaci *@SledovanaHodnota*.

```
@SledovanaHodnota(nazev = "Vzdálenost", popis = "Vzdálenost překážky.")
private double vzdalenostPrekazky;
```

Ukládání aktuální hodnoty je poté prováděno následovně:

```
// senzor vzdálenosti
vzdalenostPrekazky = robot.vratVzdalenostPrekazky();
```

Výchozím směrem natočení senzoru překážky je 0 stupňů, což odpovídá hodnotě *NatoceniSenzoru.CTR* z výčtu *NatoceniSenzoru*. Změna tohoto natočení se provádí pomocí metody *setNatoceniSenzoru*. Například natočení o 45 stupňů vpravo se provede takto:

```
robot.getSenzorPrekazky().setNatoceniSenzoru(NatoceniSenzoru.P45);
```

Objíždění je provedeno v několika krocích pomocí kombinace odměřování vzdálenosti od překážky a měření ujeté vzdálenosti pomocí enkodérů. Předávání řízení mezi jednotlivými kroky algoritmu se provádí pomocí proměnné *objizdeni*, která je parametrem příkazu *switch*. Použití senzoru překážky při objíždění v jednom kroku například hledá její konec.

```
// hledání konce překážky
case 7:
	if (robot.vratVzdalenostPrekazky() > -1) {
		levyMotor.nastavRychlost(50.0, true);
		pravyMotor.nastavRychlost(50.0, true);

	} else {
		levyMotor.nastavRychlost(100.0, true);
		pravyMotor.nastavRychlost(100.0, true);
		objizdeni = 8;
	}
	break;
```

Způsobů, jak objíždět překážku je mnoho. Tato implementace řídícího algoritmu hlavně ukazuje, jak pracovat se senzorem překážky.

ZÁVĚR

Výsledkem této práce je okenní aplikace pro simulaci robotické soutěže v disciplíně sledování čáry. Pro potřeby návrhu aplikace bylo zprvu nutné shromáždit informace k pravidlům soutěže. Dále pak také získat informace o konstrukci soutěžních robotů, zejména o součástech, které roboty tvoří, aby je bylo možné zahrnout do simulátoru. Protože žádná oficiální pravidla soutěže neexistují, kromě faktu vyplývajícího z podstaty disciplíny, která tkví ve sledování čáry, a to výhradně autonomně, jde tedy spíše o soupis nejčastěji se vyskytujících požadavků napříč organizátory těchto soutěží. Stejné je to s konstrukcí soutěžních robotů, proto se práce zabývá těmi nejpoužívanějšími součástmi, které jsou konstruktéry nejčastěji aplikovány.

Z informací takto získaných a požadavků, které jsou na aplikaci kladeny, bylo navrženo uživatelské prostředí simulátoru, jenž má uživateli přívětivou formou umožňovat nastavení základních parametrů robota a možnost nahrát obrázek se soutěžní dráhou, na které bude testovat svůj řídící algoritmus. Aplikace také poskytuje možnost sledování uživatelsky definovaných veličin, které uživatel používá ve svém řídícím algoritmu a může tak později analyzovat jejich průběh. K tomuto účelu je do uživatelského rozhraní přidána možnost sledovat tyto záznamy pomocí grafu. Součástí práce je popis tohoto rozhraní, který má být návodem, jak s aplikací pracovat a co vše aplikace nabízí prostřednictvím uživatelského rozhraní.

Nakonec se práce zabývá samotnou implementací simulátoru. V této části jsou popsány jednotlivé třídy tvořící aplikaci. Tento popis má za úkol seznámit uživatele s funkcí, kterou daná třída v aplikaci plní. V popisu jsou vyzvednuty části kódu, které jsou určeny k použití v řídícím algoritmu. Kromě popisu implementace je tato část práce i návodem, který se tentokrát zaměřuje na to, jaké nástroje rozhraní nabízí k tvorbě řídícího algoritmu. Součástí zdrojových kódů jsou i ukázkové implementace řídících algoritmů včetně jejich popisu.

SEZNAM POUŽITÉ LITERATURY

[1] Asimovovy zákony robotiky. *Vesmír / Přírodovědecký časopis* [online]. [cit. 2017-05-11]. Dostupné z: http://casopis.vesmir.cz/clanek/asimovovy-zakony-robotiky

[2] Základní společná pravidla. *Robotiáda* [online]. [cit. 2017-05-11]. Dostupné z: http://www.robotiada.cz/www/data/user/2017_Robotiada/Zakladni_spolecna_pravidla.pdf

[3] Pravidla robotického sledovače. *SPŠ* [online]. [cit. 2017-05-14]. Dostupné z: http://www.spsjedovnice.cz/files/roboti/sledovacCZ.pdf

[4] Pravidlá kategórie stopár. *Robotika.sk* [online]. [cit. 2017-05-14]. Dostupné z: http://www.robotika.sk/contest/2016/index.php?page=rules&type=follower

[5] Čára (Line follower). *Robotiáda* [online]. [cit. 2017-05-11]. Dostupné z: http://www.robotiada.cz/www/data/user/2017_Robotiada/pravidla_cara.pdf

[6] Line Follower – Sledovač čáry. *Robotický den 2017* [online]. [cit. 2017-05-11]. Dostupné z: http://roboticday.org/2017/rules/2017-Line_Follower-CZv1.pdf

[7] Line following competition rules. *Robotiada* [online]. [cit. 2017-05-13]. Dostupné z: http://robotiada.lt/en/wp-content/uploads/2017/02/Line-following-competition-rules.pdf

[8] Line Following Rules. *RoboGames* [online]. [cit. 2017-05-11]. Dostupné z: http://robogames.net/rules/line-following.php

[9] Line Following Challenge. *NASA - Robotics Alliance Project* [online]. [cit. 2017-05-11]. Dostupné z: https://robotics.nasa.gov/students/follow.php

[10] General FAQ. *Robotický den 2017* [online]. [cit. 2017-05-11]. Dostupné z: http://roboticday.org/2017/rules/2017-FAQ.php

[11] Mindstorms. *LEGO* [online]. [cit. 2017-05-12]. Dostupné z: https://www.lego.com/cs-cz/mindstorms/downloads

[12] SELECKÝ, Matúš. *Arduino: uživatelská příručka*. Brno: Computer Press, 2016. ISBN 978-80-251-4840-2.

[13] Arduino Uno R3. In: *Pololu Robotics and Electronics* [online]. [cit. 2017-05-12]. Dostupné z: https://www.pololu.com/product/2191

[14] MBot v 1.1 - Blue (Bluetooth Version). In: *Makeblock* [online]. [cit. 2017-05-12]. Dostupné z: http://www.makeblock.com/image/cache/catalog/90053/90053-1-500x500.JPG

[15] MATOUŠEK, David. *Práce s mikrokontroléry ATMEL AT89C2051: [měření, řízení a regulace pomocí několika jednoduchých přípravků]*. Praha: BEN - technická literatura, 2006. μC. ISBN 80-730-0174-8.

[16] Programování mikropočítačů. *Univerzita Tomáše Bati ve Zlíně - Fakulta aplikované informatiky* [online]. [cit. 2017-05-13]. Dostupné z: http://www.utb.cz/fai/veda-a-vyzkum/programovani-mikropocitacu

[17] AVR. In: *ABI HAIDAR Motors* [online]. [cit. 2017-05-14]. Dostupné z: http://i0.wp.com/abihaidarmotors.com/wp-content/uploads/2016/07/avr_programming1.jpg?fit=480%2C301

[18] ATMega 8. In: *Tokopedia* [online]. [cit. 2017-05-14]. Dostupné z: https://ecs7.tokopedia.net/img/cache/300/product-1/2015/5/10/329033/329033_c378dcc6-f693-11e4-b258-4b6487772fba.jpg

[19] Atmega8_dip. In: *ElectroPark* [online]. [cit. 2017-05-14]. Dostupné z: http://electropark.pl/img/cms/mikrokontrolery/atmega8_dip.jpg

[20] Atmega8_smd. In: *Mobilerobots.pl* [online]. [cit. 2017-05-14]. Dostupné z: https://img.rcgroups.com/http://www.mobilerobots.pl/web_images/atmega8_smd.jpg?h=Rb55rnNO5Li8OO1xqsIzuQ

[21] NOVÁK, Petr. *Mobilní roboty: pohony, senzory, řízení*. Praha: BEN - technická literatura, 2005, 243 s. ISBN 8073001411.

[22] Me Line Follower Array. In: *Makeblock* [online]. [cit. 2017-05-12]. Dostupné z: http://www.makeblock.com/me-line-follower-array

[23] QRD1114. In: *SparkFun* [online]. [cit. 2017-05-14]. Dostupné z: https://cdn.sparkfun.com//assets/parts/1/8/8/00246a-01.jpg

[24] PocketBot 2 - deníček vývojáře. *Ondřej Staněk - ostan* [online]. [cit. 2017-05-11]. Dostupné z: http://ostan.cz/PocketBot2/development_diary/index.html#robotChallenge2012

[25] Tcs3200-pinout. In: *TheoryCIRCUIT* [online]. [cit. 2017-05-14]. Dostupné z: http://www.theorycircuit.com/wp-content/uploads/2016/12/tcs3200-pinout.png

[26] MFG_439. In: *Digi-Key* [online]. [cit. 2017-05-14]. Dostupné z: http://media.digikey.com/Photos/Adafruit%20Industries%20LLC/MFG_439.jpg

[27] 9694-1-MAIN-490. In: *Mattonito* [online]. [cit. 2017-05-14]. Dostupné z: https://cdn.mattonito.com/img/products/full/9694-1-MAIN-490.jpg

[28] DUCHOŇ, František. *Snímače v mobilnej robotike*. Bratislava: Nakladateľstvo STU, 2012, 97 s. Edícia vysokoškolských učebníc. ISBN 9788022738019.

[29] Ultrazvukový senzor. In: *EDUXE* [online]. [cit. 2017-05-14]. Dostupné z: http://www.eduxe.sk/resize/domain/eduxe/files/les/45504.jpg?w=640&h=480

[30] Me Ultrasonic Sensor. In: *Makeblock* [online]. [cit. 2017-05-14]. Dostupné z: http://www.makeblock.com/image/cache/catalog/x/977/8__24089-500x500.jpg

[31] KOLÍBAL, Zdeněk. *Roboty a robotizované výrobní technologie*. Brno: Vysoké učení technické v Brně - nakladatelství VUTIUM, 2016. ISBN 9788021448285.

[32] 36mm Encoder DC Motor. In: *Makeblock* [online]. [cit. 2017-05-14]. Dostupné z: http://www.makeblock.com/image/cache/catalog/81335/81335_05-500x500.jpg

[33] 180 Optical Encoder Motor. In: *Makeblock* [online]. [cit. 2017-05-14]. Dostupné z: http://www.makeblock.com/image/cache/catalog/81340/81340-3-500x500.jpg

[34] MG995 Standard Servo. In: *Makeblock* [online]. [cit. 2017-05-14]. Dostupné z: http://www.makeblock.com/image/cache/catalog/n/234/5__24740-500x500.jpg

[35] SH-1250MG Digitální servo. In: *DigiModely* [online]. [cit. 2017-05-14]. Dostupné z: http://www.digimodely.cz/32354-788008-thickbox/savox-sh-1250mg-digitalni-servo.jpg

[36] 2WD Podvozek pro Inteligentní Auto. In: *Robotstore.cz* [online]. [cit. 2017-05-14]. Dostupné z: http://robotstore.cz/obchod/arduino/2wd-podvozek-pro-inteligentni-auto-arduino-robot/#

[37] All About Line Followers. *Nishant Nath | My experiences with Tech"crazy"onology* [online]. [cit. 2017-05-11]. Dostupné z: https://nishantnath.wordpress.com/2012/09/26/all-about-line-followers/

[38] ZACPAL, Michal. *Malý robot pro sledování čáry* [online]. Fakulta elektrotechniky a komunikačních technologií, 2013 [cit. 2017-05-11]. Dostupné z: https://www.vutbr.cz/studium/zaverecne-prace?zp_id=66190

[39] A PID Controller For Lego Mindstorms Robots. *Jim Sluka's Page* [online]. [cit. 2017-05-14]. Dostupné z: http://www.inpharmix.com/jps/PID_Controller_For_Lego_Mindstorms_Robots.html

[40] S - Bot – robot pro sledování barevné čáry. *České vysoké učení technické v Praze* [online]. [cit. 2017-05-11]. Dostupné z: http://www1.fs.cvut.cz/stretech/2015/sbornik_2015/0290.pdf

[41] Robot BOB. *Solarskit* [online]. [cit. 2017-05-11]. Dostupné z: http://www.solarskit.wz.cz/bob.html

[42] Bender – robot sledující čáru. *Archiv vítězných prací SOČ* [online]. [cit. 2017-05-14]. Dostupné z: https://socv2.nidv.cz/archiv32/getWork/hash/80ad46e4-11a5-11df-ae82-001e6886262a

[43] Kubrt v 1.2. *ROBOTIKA.SK* [online]. [cit. 2017-05-14]. Dostupné z: http://www.robotika.sk/contest/2002/RobotKubrt.html

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

GUI Graphical User Interface - *grafické uživatelské rozhraní*

API Application Programming Interface - *rozhraní pro programování aplikací*

RGB Red Green Blue – *červená zelená modrá*

IR Infrared - *infračervený*

PWM Pulse Width Modulation - *pulzně šířková modulace*

PID Proporcionální Integrační Derivační

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: CD

Obsah přiloženého CD:

- Bakalářská práce v elektronické podobě
- Zdrojový kód programu simulátoru sledování čáry
- Spustitelná aplikace