TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta strojní

Řídicí systém výukového modelu manipulačního vozíku

Diplomová práce

Studijní program:	N2301 Strojní inženýrství
Studijní obor:	Výrobní systémy a procesy
Autor práce:	**Bc. Marie Koutná**
Vedoucí práce:	Ing. Radek Votrubec, Ph.D. Katedra výrobních systémů a automatizace

Liberec 2021

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta strojní

Zadání diplomové práce

Řídicí systém výukového modelu manipulačního vozíku

Jméno a příjmení: **Bc. Marie Koutná**
Osobní číslo: S18000238
Studijní program: N2301 Strojní inženýrství
Studijní obor: Výrobní systémy a procesy
Zadávající katedra: Katedra výrobních systémů a automatizace
Akademický rok: **2019/2020**

Zásady pro vypracování:

1. Seznamte se s vývojovými mikroprocesorovými deskami Arduino a navrhněte vhodnou variantu desky pro řízení výukového modelu manipulačního vozíku.
2. Vypracujte recenzi možných principů řízení pohybu vozíku mezi jednotlivými pracovišti ve výrobním procesu. Jednu z variant vyberte, vybavte vozík potřebnými čidly a realizujte jeho řídicí systém.
3. Vytvořte aplikaci na mobilním telefonu k ovládání vozíku.

Rozsah grafických prací: podle potřeby
Rozsah pracovní zprávy: 50
Forma zpracování práce: tištěná/elektronická
Jazyk práce: Čeština

Seznam odborné literatury:

[1] Začínáme s Arduinem, příručka. In: Snail Instruments: www.hobbyrobot.cz [online]. 2014 [cit. 2015-01-09]. Dostupné z: http://www.snailshop.cz/literatura/1537-zaciname-s-arduinem-prirucka.html
[2] Arduino Learning, Getting Started with Arduino. In: Arduino [online]. 2014 [cit. 2015-01-09]. Dostupné z: http://arduino.cc/en/Guide/HomePage
[3] BALÁTĚ, J. Automatické řízení. 1. vyd. Praha: BEN – technická literatura, 2003, 663 s. ISBN 978-80-247-4116-1.
[4] OLEHLA, M. a S. NĚMEČEK. Automatické řízení. Vyd. 1. Liberec: Technická univerzita v Liberci, 2013. ISBN 978-807-3729-721

Vedoucí práce: Ing. Radek Votrubec, Ph.D.
Katedra výrobních systémů a automatizace

Datum zadání práce: 20. listopadu 2019
Předpokládaný termín odevzdání: 20. května 2021

L.S.

prof. Dr. Ing. Petr Lenfeld
děkan

Ing. Petr Zelený, Ph.D.
vedoucí katedry

V Liberci dne 20. listopadu 2019

Prohlášení

Prohlašuji, že svou diplomovou práci jsem vypracovala samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Jsem si vědoma toho, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědoma povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má diplomová práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědoma následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

6. června 2021

Bc. Marie Koutná

Tato práce byla podpořena Studentskou grantovou soutěží Technické univerzity v Liberci v rámci projektu Optimalizace v oblasti výrobních systémů, 3D technologií a automatizace č. SGS-2019-5011.

Vznik této práce byl podpořen projektem "Pokročilé evropské sedačky pro globální inovaci v automobilovém sektoru" (LTE12004) financovaným Ministerstvem školství, mládeže a tělovýchovy.

Poděkování

Na tomto místě chci poděkovat Ing. Radkovi Votrubcovi, Ph.D. za trpělivost při vedení mé práce. Velmi děkuji i své rodině za veškerou podporu, kterou mi setrvale poskytují. Moje vděčnost patří i širšímu okruhu – Daně, Ondrovi a i Zuzance za jejich spolehlivost.

Anotace

Diplomová práce „Řídicí systém výukového modelu manipulačního vozíku“ se zabývá oblastí automatického řízení manipulace. Je koncipována jako výukový model se snahou o co nejlepší srozumitelnost. Představuje mikroprocesorové desky Arduino, jejich druhy a způsob řízení pomocí vybrané desky Arduino Mega 2560. Vysvětluje způsoby řízení vozíků mezi pracovišti a tři vybrané principy realizuje. Zároveň představuje možný přístup k ovládání vozíku pomocí mobilní aplikace Blynk.

Klíčová slova

Arduino Mega 2560, Blynk, automatizace, PWM, AGV, IoT, manipulace, doprava

Annotation

The diploma thesis "The control system of an educational model of a handling vehicle" is focused on the automatic control of manipulation. The focus of the thesis is on its comprehensibility as an education model. The thesis involves an introduction to the Arduino microcontroller boards, their differentiation, and describes the ways of controlling the Arduino Mega 2560 board. It outlines the techniques of operating carriages between workplaces and implements three chosen methods. It showcases a possible way of controlling the handling vehicle through a mobile application with the help of the Blynk app.

Keywords

Arduino Mega 2560, Blynk, automation, PWM, AGV, IoT, manipulation, transport

Obsah

Seznam použitých zkratek a symbolů

Zkratka	Význam	Poznámka, originální znění
AC	střídavý proud	alternating current
(a)AGV	(autonomní) automaticky řízený vozík	(autonomous) automated guided vehicle
CMY(K)	cyan-magenta-yellow(-black)	azurová-purpurová-žlutá(-černá)
COM	komunikační port	communication port
ČR	Česká republika	
ČSN (EN)	česká technická norma (evropská norma)	
DC	stejnosměrný proud	direct current
DP	diplomová práce	
EEPROM	electrically erasable programmable read-only memory	elektricky vymazatelná paměť pouze pro čtení
EN	blokovací/řídicí vstup	enable
FBD	funkční blokové schéma	function block diagram
FTS	transportní systémy bez řidiče	fahrerlose Transportsysteme
GND	země	ground
HW	hardware	
CH_PD	pracovní režim čipu	chip power down
IDE	vývojové prostředí	integrated development environment
IEC	Mezinárodní elektrotechnická komise	International Electrotechnical Commission
IL	seznam instrukcí	instruction list
IN	vstup	input
IoE	internet všeho	Internet of Everything
IoP	internet lidí	Internet of People
IoS	internet služeb	Internet of Services
(I)IoT	(průmyslový) internet věcí	(Industrial) Internet of Things
I/O	vstupně-výstupní	input/output
IR	infračervené záření	infrared
LD	příčkový diagram	ladder diagram

Zkratka	Význam	Poznámka, originální znění
LED	elektroluminiscenční dioda	Light-Emitting Diode
LGV	laserem naváděný vozík	laser guided vehicle
MCU	jednočipový počítač, mikrokontrolér	microcontroller
Ni-Mh	nikl-metal hydridový akumulátor	
PC	(osobní) počítač	personal computer
PLC	programovatelný automat	programmable logic controler
PWM	pulzně šířková modulace	pulse width modulation
RFID	radiofrekvenční identifikace	radio frequency identification
RGB	červená-zelená-modrá	red-green-blue
RST	reset	
RX	příjem	receive
SFC	sekvenční funkční diagram	sequential function chart
SGV	samostatně řízený vozík	self guided vehicle
SRAM	statická paměť s adresným přístupem	static random access memory
ST	strukturovaný text	structured text
SW	software	
TX	vysílání	transmit
UART	asynchronní sériový přenos	universal asynchronous receiver-transmitter
USB	univerzální sériová sběrnice	universal derial bus
UV	ultrafialové	ultraviolet
Vin	kladné napájecí napětí	input voltage
VCC	kladné napájecí napětí	voltage common collector

Symbol	Význam	Poznámka
€	euro	
⌀	průměr	

1 Úvod

Automatické řízení je trend, který je v posledních desetiletích stěžejní ve výrobních procesech. Jednoznačná je snaha nahradit stereotypní lidskou práci stroji a lidský faktor více zaměstnat tvůrčí činností. Prostředky automatizace obecně velmi dobře fungují v oblastech, kde je malá míra proměnlivosti procesu, o něco hůře obstojí v tam, kde je nutné zapojit určitou míru kreativity a intuice. I to se ovšem s rozvojem umělé inteligence mění.

Jedno z odvětví, kde je stále do velké míry využívána lidská práce, je přeprava materiálu a výrobků. Trendem je tuto oblast stále více pokrýt automaticky řízenými vozíky. Škála metod, jak tak učinit je široká. Práce „Řídicí systém výukového modelu manipulačního vozíku“ si proto klade za cíl nejobvyklejší varianty představit nejdříve teoreticky a vybrané metody realizovat i v praxi. Bude tak učiněno pomocí vývojové desky Arduino, která je vhodná pro výukové účely jako prostší varianta napodobující průmyslové počítače.

Řídicí systém bude ovládán i za pomoci mobilní aplikace. Čtenář tak nahlédne do principů internetu věcí a zároveň bude moci získat představu i o dalším uplatnění Arduino desek – například k vlastnímu naprogramování tzv. „chytré domácnosti“.

V práci se jedná se o výukový model, hlavní obecné cíle proto jsou:

- Seznámit čtenáře se základními principy automatického řízení odborným ale zároveň dobře srozumitelným jazykem,
- vysvětlit principy automatického řízení vozíků mezi pracovišti,
- představit projekt Arduino,
- ukázat základní rozdíly mezi Arduino deskami,
- představit principy a schémata programování stavových automatů,
- popsat funkci použitého hardwaru,
- představit způsob programování v prostředí Arduino IDE i spolu s využitím mobilní aplikace Blynk.

Celá práce bude proto usilovat především o systematičnost a dobrou pochopitelnost jednotlivých kroků vývoje řídicího systému a případně o inspiraci čtenáře k vývoji svých vlastních projektů.

2 Základní pojmy

První pojem, který se přímo dotýká tématu DP, je **systém**. Balátě jej obecně definuje jako: *„…soubor prvků, mezi nimiž existují vzájemné vztahy a jako celek má určité vztahy ke svému okolí.“* [1]

Každý systém má dvě základní vlastnosti:

1. Chování – vnější vztahy k okolí, vyjadřuje jej závislost mezi vstupním a výstupním signálem,
2. strukturu – způsob vnitřního uspořádání, tedy vnitřní vztahy mezi jednotlivými prvky systému.

2.1 Řízení – definice, kategorie

*„**Řízení** je cílevědomá činnost, při níž se hodnotí a zpracovávají informace o řízeném procesu a podle nich se ovládají příslušná zařízení tak, aby se dosáhlo určitého předepsaného cíle.“ [2]*

Podle druhé, poněkud vágnější definice pod pojmem **řízení** rozumíme: *„Cílené působení na řízený objekt tak, aby se dosáhlo určitého předepsaného cíle.”* [3]

Řízení dělíme na:

1. Ruční,
2. automatické.

Zpětná vazba je *„zapojení, kdy část výstupního vektoru je propojena na vstup téhož prvku a stává se součástí jeho vstupního vektoru.“* [1] Jinými slovy, vstupní veličina je ovlivněna výstupy – výsledek předešlého kroku řízení je zohledněn v dalším řízení systému.

- Řízení bez bezprostřední zpětné vazby (kontroly), tedy řízení ruční, se nazývá **ovládání**.

Automatické řízení označuje: *“Řízení realizované samočinným technickým zařízením (regulátorem)”*. [4] Dělíme je na nižší formy (regulace) a formy vyšší (optimální řízení, adaptivní řízení, učení a umělá inteligence). Aby se jednalo o automatické řízení

v pravém slova smyslu, je vhodná komunikace řízeného objektu s řídicím systémem – zpětná vazba.

- **Regulace** zahrnuje zpětnou vazbu a korekci rozdílu mezi získanou a žádanou hodnotou výstupní veličiny.
- **Optimální řízení** znamená snahu systému dosáhnout co nejnižších energetických či časových nákladů pro dosažení požadované hodnoty.
- Při **adaptivním řízení** systém dynamicky mění svou strukturu podle měnících se parametrů řízeného objektu.
- **Učení** je druh adaptivního řízení, při kterém systém ukládá informace do paměti a později jich využívá.
- **Umělá inteligence** má schopnost samostatně analyzovat vztahy mezi předměty, objevovat souvislosti a samostatně zdokonalovat svou činnost.

Podle **technické realizace** řízení rozlišujeme řízení logické, spojité, diskrétní a fuzzy.

- **Logické řízení** využívá veličiny o dvou hodnotách, typicky zapnuto/vypnuto (otevřeno/zavřeno, 1/0, HIGH/LOW). Vztahy mezi veličinami lze snadno zapisovat pomocí výrokové logiky.
- **Spojité (analogové)** řízení má v čase nepřetržitou vazbu mezi vstupy a výstupy. Není nikdy diskrétní ani dvouhodnotové.
- **Diskrétní řízení** vztah mezi vstupy a výstupy určuje z posloupnosti impulzů, jejichž četnost je dána tzv. vzorkovací periodou (tzn. dobou mezi snímáním dvou sousedních hodnot).
- **Fuzzy** je specifický způsob řízení, kdy vstupní veličiny obdrží svou jazykovou hodnotu. Je vhodné pro těžce popsatelné systémy, kde přesně neznáme vztahy mezi vstupními a výstupními veličinami.

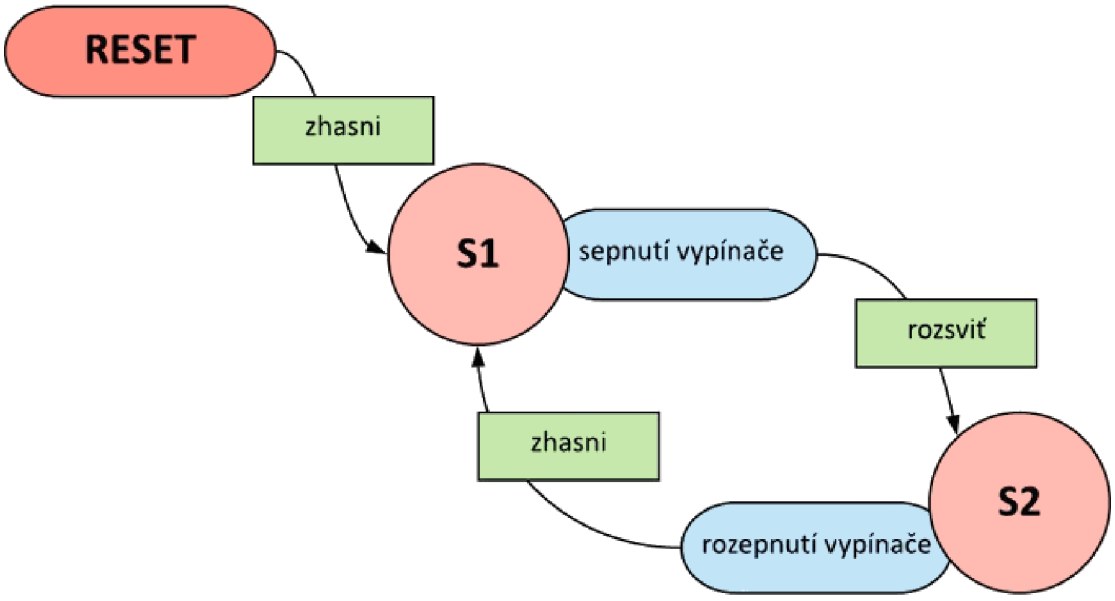
2.2 Stavový automat

Automatem rozumíme: *"Stroj nebo kontrolní mechanismus navržený tak, aby samočinně následoval posloupnost operací nebo se choval podle zakódovaného postupu."* [5] Automat tedy pracuje samočinně – automaticky.

Chování **stavového** (konečného) **automatu** lze znázornit diagramem, který se skládá ze:

- **Stavů** – okamžiků, ve kterých automat čeká na impuls pro přechod do jiného stavu. Označujeme je kruhem. Příkladem stavu je čekání na stisknutí vypínače světla.
- **Přechodů při splnění podmínky**, tedy situací mezi stavy, kdy se typicky, ale ne nutně, mění hodnoty některých výstupních proměnných. Je možný i přechod do téhož stavu. Značíme je šipkou mezi dvěma stavy. Přechodem může být situace, kdy se po stisknutí vypínače přesouváme do dalšího stavu. V následujícím stavu pak čekáme na pokyn ke zhasnutí. Podmínky jsou označeny oválem, který se dotýká kruhů náležících stavům.
- **Akcí**, které se vykonávají při přechodu mezi stavy. Jsou znázorněny obdélníky nad šipkami přechodů. Akcí může být rozsvícení žárovky, anebo naopak její zhasnutí.

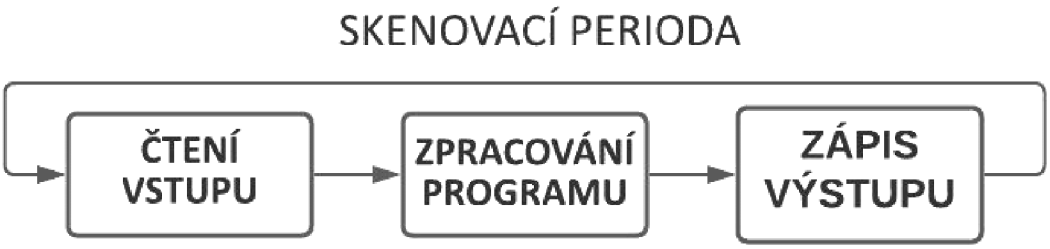
Spuštěné automaty pracují **v nekonečném cyklu** mezi **konečným počtem stavů.** Vypnutím a opětovným zapnutím proběhne reset automatu, po kterém se automat nachází v předdefinovaném výchozím stavu (např. S1, zhasnutá žárovka).



Obr. 1: Ukázka diagramu jednoduchého automatu [6]

2.2.1 Programovatelné logické automaty (PLC)

Pro programovatelné automaty se obvykle používá anglická zkratka **PLC** (programmable logic controllers). V dnešní době se jedná o v průmyslu nejpoužívanější automatizační prostředky, jejich vzestup započal již v 90. letech.



Obr. 2: Skenovací perioda PLC [7]

Program automatu probíhá v cyklech, tzv. **skenovacích periodách**. Aktualizace hodnot **vstupů** a **výstupů** probíhá právě jednou za skenovací periodu, v průběhu cyklu tedy automat pracuje s tzv. **obrazy hodnot**.

Program umožňuje řešení úloh se složitými vazbami veličin, které by jinak bylo velmi obtížné zautomatizovat.

Volba správného **programovacího jazyka** dle typu úkolu a předchozích zkušeností programátora usnadňuje a zrychluje proces tvorby programu. PLC standardně umožňují velkou variabilitu programátorských metod. Rozlišuje programovací jazyky [8]:

1. ***Grafické***

- **Příčkový diagram** (**LD**[1]) – grafická reprezentace reléové logiky, příčky jsou ve stavu zapnuto/vypnuto, v příčkách jsou elektrické součástky, funkce, funkční bloky.
- **Funkční blokové schéma** (**FBD**) – vzájemně propojené grafické bloky reprezentující logické funkce.

2. ***Textové***

- **Instruction list** (**IL**) – jazyk pokynů složený ze sekvence instrukcí.
- **Structured text** (**ST**) – vyšší programovací jazyk než IL, vychází z jazyků Pascal a C.

3. ***Sekvenční funkční diagram*** (*SFC*) – tento jazyk není uveden v normě[2], která doporučuje předchozí čtveřici jazyků, ale je často zmiňován. Skládá se z bloků

[1] Často jsou užívány anglické a německé ekvivalenty a zkratky: Příčkový diagram = ladder diagram (LD, Kontaktplan), funkční blokové schéma = function block diagram (FBD, Funktionbausteinsprache), seznam instrukcí = instruction list (IL, Anweisungsliste), strukturovaný text = structured text (ST, Strukturierter Text), sekvenční funkční diagram = sequential function chart (SFC, Ablaufsprache)

a přechodů. Bloky můžou být naprogramované ve všech pěti jazycích, reprezentují stavy a akce. Přechody zastupují posuny do dalších stavů při splnění podmínky. Program se může větvit. Tento jazyk je tedy nejpodobnější grafické reprezentaci stavového automatu (kapitola 2.2).

2.3 Pulzně šířková modulace

PWM [9] (pulse width modulation), pulzně šířková modulace je způsob ovládání analogových spotřebičů digitálním způsobem. Spotřebič v každý okamžik odebírá buď maximální hodnotu napětí, anebo nulovou.

Pro ilustraci jejího principu využijme zdroj světla, jehož intenzitu chceme ovládat. Nabízí se dva způsoby. První z nich je způsob analogový. Můžeme změnou odporu (či snížením napětí) regulovat proud, který zdrojem prochází a tím dosahovat požadovaných hodnot intenzity. Tento postup může být neekonomický a také nepřesný, protože vlivem opotřebení se hodnoty odchylují od původně požadovaných. Zařízení je také často citlivé na proudový odběr, a tak tento způsob regulace je často funkční jen na velmi omezeném intervalu.

Obr. 3: PWM – střída signálu (duty cycle) a hodnota funkce analogWrite()[3]

[2] IEC 61131, v ČR ČSN EN 61131 Programovatelné řídicí jednotky
[3] Jedná se pro funkci pro PWM signál pro Arduino, blíže popsáno v kapitole 7.1.3.

Druhý postup je **digitální** (PWM). Můžeme v krátkých intervalech světlo připojovat ke zdroji a odpojovat. V každý okamžik je zdroj buď na maximální, anebo minimální hodnotě, tedy ve stavu zapnuto/vypnuto. Při dostatečně velké frekvenci lidské oko nevnímá děj jako blikání, ale jako nižší intenzitu osvětlení. Tak dosáhneme pulzně šířkové modulace.

Jedná se tedy o periodické vysílání obdélníkových vln (pulzů), jejichž výše je rovna napětí zdroje. Poměr doby sepnutí zdroje vydělený periodou jednoho cyklu se pak nazývá **střída signálu** (anglicky duty cycle). Jde o vyjádření poměru z celkového času provozu, kdy je spotřebič připojen ke zdroji[4]. Pro snazší srozumitelnost je možné střídu signálu vyjádřit v procentech.

2.4 Průmysl 4.0

Průmysl 4.0 je termín posledního desetiletí, který nemá jasnou definici, není určen žádnou normou, a přesto se jedná o trend, který je často užíván nejen odbornou veřejností.

Kořeny pojmu lze najít v Německu, kde byl termín **Industrie 4.0** poprvé použit v roce 2011 společností Siemens ve spolupráci s německou vládou ve snaze propagovat vývoj nových technologií.

Termín je typicky spojován s či pokládán za synonymum tzv. čtvrté průmyslové revoluce. Ta navazuje na třetí průmyslovou revoluci – digitalizaci, automatizaci a robotizaci výroby. Rozšiřuje ji dále o zavedení komplexních systémů a automatizace i do domácností, využití umělé inteligence a vetší propojenost až integraci výrobních procesů. Úloha člověka v pracovním procesu se má dále vyvíjet směrem ke kreativní činnosti a vzdalovat se od činností stereotypních.

Koncepce Průmyslu 4.0 bývá často ohraničována **šesti hlavními cíli**, kterými jsou: [10]

- Interoperabilita (komunikace všech prvků výrobního systému),
- virtualizace (vše fyzické má svou virtuální kopii),
- decentralizace (autonomie subsystémů, vytvoření uzlů),
- reálný čas (informace v systémech musí korespondovat s aktuální situací),
- orientace na služby (prvky výrobního systému si navzájem poskytují „služby“),

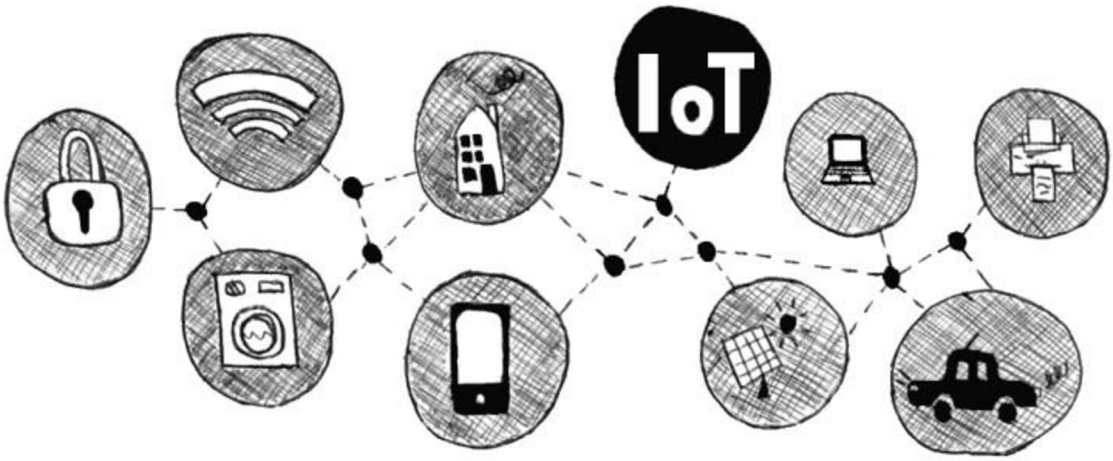
[4] Střída signálu = doba(zapnuto) / [doba(zapnuto) + doba(vypnuto)]

- modularita (snadná adaptace, rozšiřitelnost výroby).

Kvůli šíři vynaložených prostředků v rámci zefektivňování výroby i každodenní lidské činnosti je Průmysl 4.0 stále poněkud vágní pojem, jehož přesnější definici přinese až čas.

2.5 Internet věcí

S průmyslem 4.0 je často spojován pojem **internet věcí - IoT**[5]. Překvapivě se jedná o starší pojem než průmysl 4.0. Vznikl již v roce 1999, na začátku masovějšího rozvoje sítě.



Obr. 4: Internet věcí (IoT)

Jedna z **definic internetu věcí** zní takto: *„Síť fyzických zařízení, vozidel, domácích spotřebičů a dalších zařízení, která jsou vybavena elektronikou, softwarem, senzory/čidly a hlavně síťovou konektivitou. Ta umožňuje těmto zařízením se navzájem propojit a vyměňovat si data.“* [11] Podobně jako Průmysl 4.0 vede i internet věcí k větší integraci jednotlivých složek systému, které vzájemně kolaborují a jako celek fungují efektivněji. Pojem souvisí i s tzv. chytrou domácností.

S dalším rozvojem internetu i (mobilních) technologií přibývá podobných zkratek, například IoS, IoP, IoE[6] (internet služeb, lidí, všeho). Internet věcí uplatněný ve výrobě bývá někdy specifičtěji označován jako průmyslový internet věcí (IIoT[7]).

[5] Internet of Things
[6] Internet of: services, people, everything.
[7] Industrial Internet of Things

3 Principy řízení pohybu vozíků mezi pracovišti

Základní a nejstarší způsob přepravy mezi pracovišti je lidskou silou, pro neefektivitu, malou kapacitu a nízkou nosnost je v mnoha kontextech zastaralý. Velmi častým způsobem přepravy je využití paletových a vysokozdvižných vozíků, jak manuálních, tak elektrických. Výraznou výhodou vyžití lidské síly je velká variabilita tras, po kterých může být materiál převážen. Své místo má především v nevýrobních zařízeních, například v potravinářském průmyslu (např. sklad ovoce a zeleniny).

Práce se zaměřuje na vozíky řízené automaticky ve výrobním procesu.

3.1 Automaticky řízené vozíky

Automaticky řízené (naváděné) vozíky jsou nejznámější pod názvem **AGV**[8] (automated guided vehicles). Další pojmenování zahrnují laserem naváděné vozíky LGV (laserguided vehicles), samostatně řízené vozíky SGV (self guided vehicles) a aAGV (autonomně naváděné vozíky) apod. Vhodné užití (pro opakované úkony) vozíků vede ke snížení mzdových nákladů, zvýšení bezpečnosti práce, produktivity a přesnosti.

Velká míra autonomie u vozíků není obvyklá, častější je řízení po pevně dané trase (či trasách) a pro změnu tras nutné komplexní přeprogramování či změna fyzických vodítek. [12]

Rozlišujeme několik základních principů automatické navigace:

- **Pevně určená trasa**

Vozík lze naprogramovat tak, aby jezdil po pevně vytyčené trase bez interakce s okolím. Jedná se o nejjednodušší způsob ovládání.

- **Vodící systém zakomponovaný do prostor pohybu vozíku**

Vozík je naváděn **laserem** podle odrazek z reflektivní pásky, která je upevněna na okolních stěnách, tyčích a jiné infrastruktuře. Rotační laserový sensor neustále vysílá i přijímá signál. Podle rozmístění reflektivních bodů a známé mapy prostoru se vyhodnocuje poloha vozíku. Tento způsob navádění patří mezi nejpopulárnější.

[8] Často se používá i zkratka FTS (Fahrerlose Transportsysteme) – transportní systémy bez řidiče.

Principiálně podobnou alternativou může být použití **radiofrekvenčních tagů** v kombinaci s RFID čtečkou.

Jinou variantou je umístění induktivních kabelů do podlahy. Obecně se jedná o jakékoli **dráty** umístěné **pod zemí** umožňující komunikaci s vozíkem. Tento způsob navádění je náročnější na instalaci a změnu tras, naopak ale není náchylný k rychlému opotřebení.

Další způsob je **liniový** – využívají se optické a magnetických pásky či cylindrické magnety. Jedná se o starší a méně nákladnou technologii. Navádění pomocí pásky je snadnější na implementaci a umožňuje rychlou změnu žádané dráhy. Nevýhodou je snadné poškození pásek na frekventovaných místech.

- **Autonomní (přirozené) navádění**

Jeden či více **senzorů vzdálenosti** detekuje překážky na trase vozíku. Podle referenčních objektů se orientuje a dokáže vyhodnotit umístění cíle. Tato technologie je vhodná pro chodby. Tohoto principu využívají vozíky aAGV, tedy vozíky s vyšším stupněm autonomie, které jsou schopny samostatně kalkulovat nejvýhodnější trasu.

3.1.1 Zvolené metody navigace

Pro praktickou část práce byly zvoleny tři způsoby navádění. Prvním je **přímé ruční ovládání** pomocí joysticku, které není závislé na okolním prostředí, podobně jako u dálkově ovládaných dětských autíček.

Druhým způsobem je **řízení podle čáry – černé pásky**. Tento způsob je vhodný pro výukový model díky možnosti vytvořit přenosnou „mapu“. Není nutné mít vyhrazený prostor pro pohyb vozíku, kde by bylo nutné instalovat pevné komponenty. Proto je pro účely ilustrace funkce v prostorách univerzity vhodnější, než zabudování podzemních drátů či pevně umístěných odrazek.

Třetím způsobem je jízda po **pevně vytyčené trase**. Tento způsob opět nevyžaduje složitou instalaci naváděcích objektů do okolního prostředí.

4 Arduino

Arduino je open source [9] hardwarová a softwarová platforma. Vzniklo v Itálii na Interaktivním institutu designu Ivrea. Jméno je převzato od italského krále Arduina Ivrejského, který vládl v 11. století. Prvotní úmysl byl vytvořit **snadno ovladatelný výukový nástroj** pro rychlé prototypování [10], který nevyžaduje rozsáhlé znalosti elektroniky a programování. S narůstající popularitou bylo postupně vyvinuto široké spektrum desek lišící se rozměry i technickými možnostmi, které nabízejí.

Deska se v základu skládá z **mikrokontroléru** a **konektorů**. Samostatně je deska využívána málokdy, spíše v době studijních začátků a seznamování se s jejími možnostmi. Pro reálné využití je na desky je možné napojit tzv. shieldy, což jsou specializované desky fungující ve spojení s Arduino deskou. Konektory také umožňují připojení různých senzorů, Wi-Fi modulů a mnoho dalšího. Kromě vyhodnocování vstupních dat umožňují desky i výstup, kterým může být spuštění motoru, zapnutí termostatu, rozsvícení diody apod.

K desce je **volně dostupný software** pro programování desky. Nazývá se **Arduino IDE**. Prostředí je blíže popsáno v kapitole 6.16.1.

Mezi výhody Arduina patří nízká cena, multiplatformní software, jednoduché programovací prostředí, programovací jazyk založený na jazycích C a C++ a open source software i hardware.

4.1 Významné parametry desek

Podkapitola se zaměřuje na vysvětlení základních parametrů Arduino desek s důrazem na jejich praktické dopady na řízení manipulačního vozíku. Srdcem každé Arduino desky je mikrokontrolér (jednočipový počítač, MCU). Pro běžného uživatele je nejpodstatnějším parametrem jeho paměť, která se u různých verzí čipů výrazně liší.

4.1.1 Paměť

Mikročipy mají tři druhy paměti:

[9] S volně přístupným kódem, který je možné modifikovat a redistribuovat. V případě Arduina i s popisem HW vybavení.
[10] Často je užíváno ekvivalentní anglické spojení „rapid prototyping“.

1. **Flash paměť** (programový prostor) – nevolatilní[11] paměť, kde jsou uloženy skeče Arduina. Po odpojení od zdroje zůstává nahrána poslední verze programu. Tato paměť má vždy výrazně větší kapacitu, než dva zbylé druhy paměti.
2. **SRAM**[12] (statická paměť) – volatilní paměť, do které se ukládají hodnoty proměnných a výpočtů za běhu programu.
3. **EEPROM** (elektricky vymazatelná paměť pouze pro čtení) – nevolatilní paměť vhodná pro uložení trvalejších dat. Má omezený počet zápisů.

Od projektu lze očekávat poměrně dlouhý kód programu. Při výběru desky bude tedy rozhodující **velikost flash paměti**. U SRAM paměti na Arduinu hrozí nedostatečná kapacita především při ukládání dlouhých textových řetězců do proměnných. Většina proměnných programu vozíku budou krátká čísla a dvouhodnotové proměnné (datový typ boolean), takže riziko přetečení SRAM paměti je malé. Textové řetězce je navíc často možné uložit jako konstanty do flash paměti. V případě nedostatečné kapacity SRAM je obecně možné problém řešit strukturou programu a omezením počtu proměnných – tímto způsobem může klesat přehlednost programu, ale mírný nedostatek paměti je tímto způsobem řešitelný. EEPROM je pokročilý druh paměti, pro program vozíku bude zřejmě využívána minimálně, takže její velikost ve výběru desky nehraje roli.

4.1.2 Počet a druh pinů

Piny jsou konektory Arduina, kterými můžeme ovládanému zařízení předávat hodnoty, anebo naopak získávat data od zařízení pro zpracování v programu Arduina. Arduina mohou disponovat následujícími druhy pinů:

Digitální vstupně-výstupní (I/O[13]) piny – tyto piny jsou schopné přijímat i vysílat diskrétní signál. Standardně platí následující tabulka hodnot:

Výstup:	**0 V („0“, „false“, „LOW“)**	**5 (3,3) V („1“, „true“, „HIGH“)**
Vstup 5V modely:	< 1,5 V	> 3 V
Vstup 3,3V modely:	< 1 V	> 2 V

Tab. 1: Napětí na pinech odpovídající dvouhodnotovým proměnným [13]

[11] V **nevolatilní** paměti po odpojení od napájení v ní zůstává uložen obsah, naopak paměť **volatilní** svůj obsah po odpojení od zdroje maže.

[13] I – vstupní (input); O – výstupní (output); I/O vstupně-výstupní piny

Část z digitálních pinů bývá schopna **PWM výstupu**. Je tedy možné, ale ne nutné, je využít k PWM. Většina Arduin[14] není schopna analogového výstupu v pravém slova smyslu, pulzně šířková modulace je tedy jediný způsob, jak jej simulovat. Při psaní programu je proto důležité hlídat obsazenost PWM pinů.

Analogové vstupní piny slouží ke čtení hodnot z různých senzorů. Naprostá většina Arduin nedisponuje analogovými výstupními piny. Nevyužité piny lze využít také jako vstupně-výstupní digitální piny.

V projektu lze vzhledem k vybraným metodám navigace předem počítat s obsazeností **nižších desítek pinů**.

4.1.3 Napájení

Arduina jsou napájena stejnosměrným zdrojem, všechna umožňují napájení kombinací pinů **Vin** a **GND**. Během doby, kdy je deska připojena k počítači, je možné ji napájet přes **USB** (typu mini B, micro, či B). Během přípravy programu tak většinou[15] není nutné zajišťovat externí napájecí zdroj. I mimo dobu připojení k PC je u jednodušších projektů možné Arduino napájet přes USB, například z powerbanky. Modely větších rozměrů disponují kromě USB konektoru i **napájecím konektorem.** Doporučené napájecí napětí je **7-12 V**, povolené až 6-20 V.

4.2 Srovnání relevantních modelů Arduino desek

V historii Arduina byly vyvinuty desítky desek či jejich variant. Mnohé z nich již nejsou aktivně vyráběny.

V tabulce porovnávající jednotlivé desky jsou zahrnuty běžnému uživateli nejznámější desky a také ty, které disponují vlastnostmi, které by pro účely programování řídicího systému manipulačního vozíku mohly být výhodné. Obsoletní desky nejsou zmíněny. Zaměření je cíleno přímo na desky, nikoliv na jejich rozšíření a doplňkové moduly.

Informace o ceně jsou čerpány z oficiálního webu Arduina. K dostání je méně nákladná plejáda padělků či tzv. klonů, ovšem spolehlivost jejich fungování není zaručena.

[14] Výjimkou je Arduino Due.
[15] Pokud zároveň nepotřebujeme napájet připojené součástky s většími nároky na odběr, je třeba brát v potaz, že přes USB je Arduino napájeno pouze 5 V, což způsobuje nižší napětí na výstupních pinech.

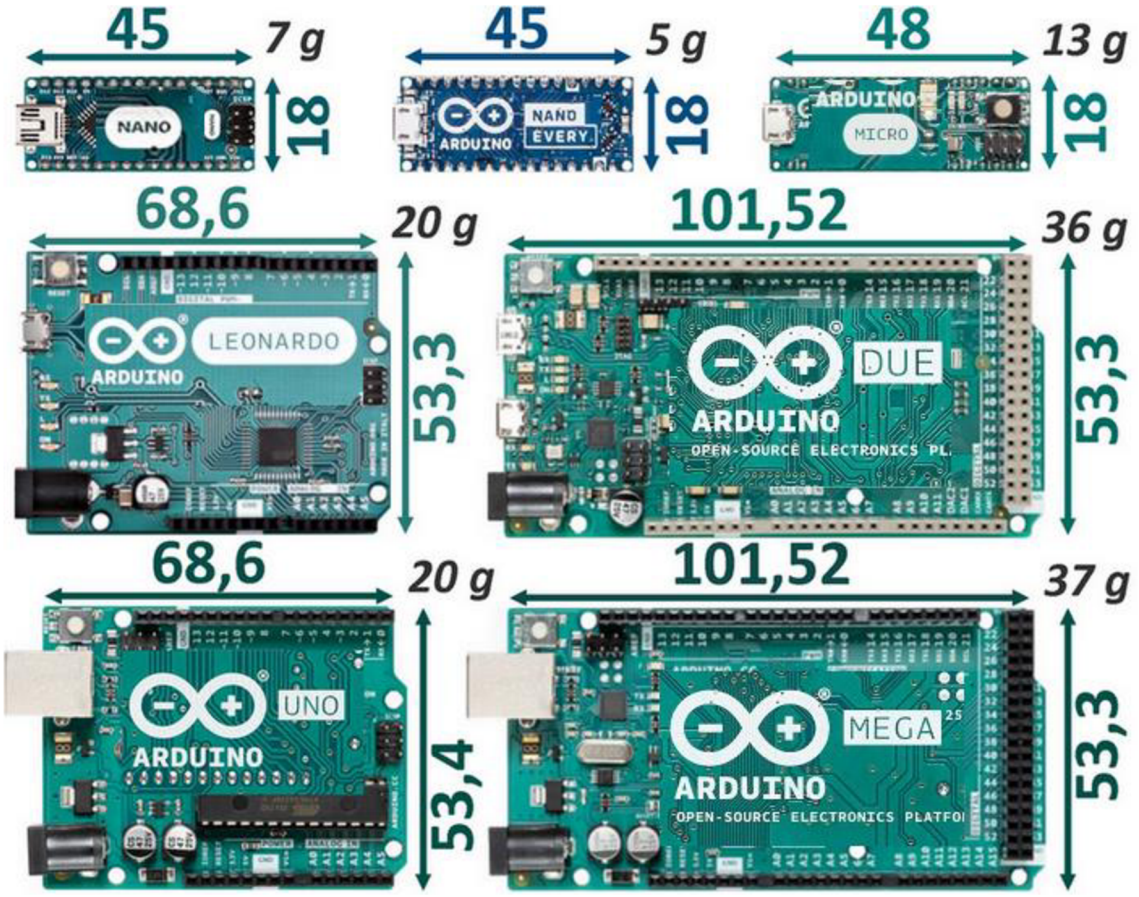
Model	Výrazný rys	Počet pinů			Mikro-kont-rolér	Flash paměť [kB]	Cena [€]
		Digitální		Analog.			
		I/O	PWM	I			
Nano	• nejmenší	22	6 z 22	8	ATmega 328	32	20
Nano Every	• nejlehčí	20	5 z 20	8	ATmega 4809	48	11
Micro	• malé • podobné Leonardu	20	7 z 20	12 z 20	ATmega 32u4	32	18
Leo-nardo	• přehlednější a větší než Micro	20	7 z 20	12 z 20	ATmega 32u4	32	18
UNO	• „základní“ deska	14	6 ze 14	6	ATmega 328P	32	20
DUE	• 3,3 V • velké množství pinů	54	12 z 54	12 (2 O)	AT91 SAM3 X8E	512	35
Mega 2560	• velké množství pinů	54	15 z 54	16	ATmega 2560	256	35

Tab. 2: Srovnání základních vlastností Arduino desek [14]

Je zřejmé, že ceny jednotlivých desek se řádově neliší. Vzhledem k tomu, že se v práci jedná o výběr desky pro jeden výukový model, nikoliv pro sériovou výrobu, nebude na cenu pro minimální úsporu dále brán zřetel. Relevantnější jsou proto další parametry, předně **paměť** a **počet** a **druh pinů**.

Arduino Nano (i **Nano Every**) je vhodné pro drobné projekty, kde je žádoucí, aby deska nezabírala mnoho místa a bylo snadné ji skrýt. Pro výukové účely nejsou rozměry hlavním parametrem, naopak větší deska může napomoci k větší přehlednosti a pochopitelnosti řešení. Také je zde, v případě většího rozsahu projektu, nebezpečí nedostatečného počtu pinů. Podobná úskalí mají i další populární desky **Micro** a **Leonardo**.

Arduino **Uno** je hojně využívané v internetových projektech a velká část kutilských návodů na serveru YouTube je právě s ním. Právě tato amatérská videa jsou pro mnohé hlavním zdrojem, jak se s Arduinem naučit pracovat. Nevýhodou ovšem je velmi nízký počet pinů, který by nemusel být dostačující. Dále deska postrádá hardwarovou sériovou komunikaci.



Obr. 5: Porovnání velikostí vybraných Arduino desek [14]

Arduino **Due** je velká a multifunkčně vybavená deska s nejvyšší kapacitou paměti, ovšem jako jediná běží na napětí 3,3 V, které není pro podobné desky typické a jako názornější se tedy jeví užití 5V desky. Výrazně se od dalších desek liší i velmi vysokou taktovací frekvencí a použitým mikrokontrolérem. Jedná se tedy o „solitéra“ na poli Arduin. Program a zapojení pro Due by mohl být náročnější na případné převedení na jiné desky.

Jako nejlepší varianta se proto jeví **Arduino Mega 2560**, které má sice poloviční paměť, než model Due a nedisponuje analogovými výstupy, naopak ale má více analogových vstupů a především PWM výstupů. Umožňuje HW sériovou komunikaci.

Pracuje se standardním napětím 5 V. Jeho technické parametry jsou blíže představeny v kapitole 5.1.

4.3 Porovnání Arduina a PLC

U Arduin se obvykle jedná o **psaní kódu** v prostředí Arduino IDE, program můžeme připodobnit k **strukturovanému textu** pro PLC. Tím, že Arduino je open-source projekt, nadšenci vytvářejí i další druhy prostředí umožňující jiné metody programování, například za použití **funkčních bloků** (Arduino Blocks[16]).

Jak u PLC, tak i u Arduin jede program ve smyčce – **skenovací perioda** je podobná **sekci loop** u Arduina. Drobný rozdíl je, že PLC tvoří obrazy hodnot vstupů a výstupů, které aktualizuje jednou za smyčku, Arduino tak činí okamžitě – tento rozdíl lze však v případě potřeby řešit správně strukturovaným programem.

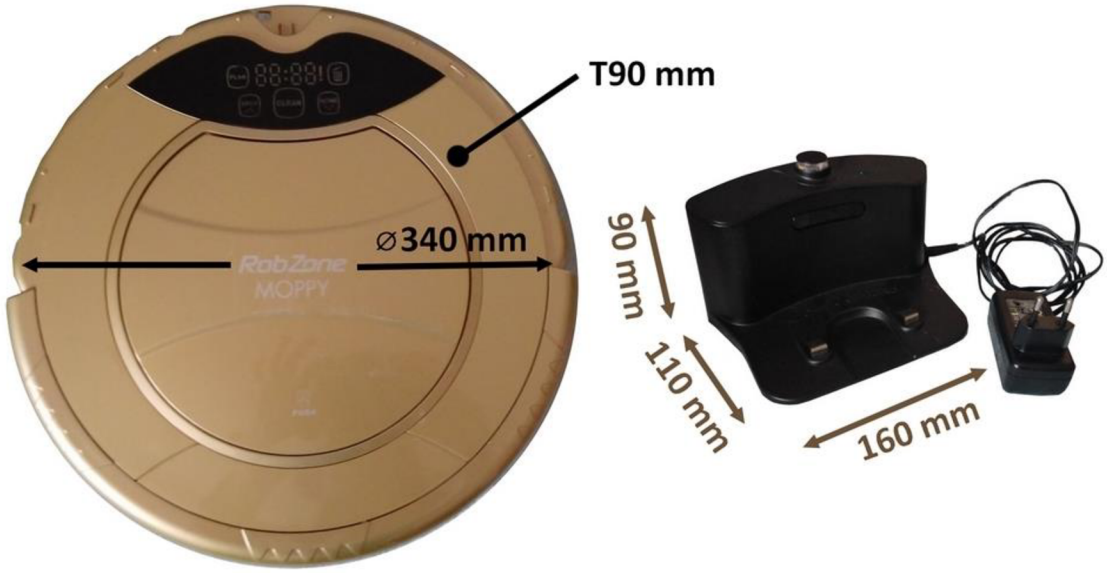
Potenciál **průmyslového využití Arduina** je limitovaný – Arduino není uzpůsobeno průmyslovým podmínkám z hlediska odolnosti mechanickému poškození, není dostatečně robustní. Má výkonová a kapacitní omezení, která je činí vhodným spíše pro domácí projekty, není nástrojem k užití ve velkovýrobě.

Přesto je zřejmé, že Arduino je velmi vhodný **nástroj pro výuku** a získání základní představy o **principech stavových automatů**. Tento výukový potenciál je především zřejmý v oblasti programování Arduina, kdy se uživatel učí správnému způsobu uvažování a tvorby kódu. Vstup do světa Arduin je nízkoprahový, a tak neodrazuje začátečníka přílišnou složitostí. Kompaktnost mikrokontrolérů a srozumitelné návody umožňují pozvolnější získání vhledu. Pro spuštění základních programů stačí Arduino zapojit do USB portu a nahrát některý z předpřipravených příkladů. Snadné připojení periferií pak uživateli dává možnost postupně a pozvolným tempem pochopit principy složitějších zapojení.

[16] Dostupné na arduinoblocks.com.

5 Použitý hardware

Výukový model automaticky řízeného vozíku je realizován na podvozku robotického vysavače Moppy firmy RobZone.



Obr. 6: RobZone Moppy – rozměry vysavače i nabíjecí základny

Vysavač má následující technické parametry: [15]

Hlavní tělo vysavače

- Rozměry: ⌀ 340 mm x 90 mm
- Napětí baterie: DC 14,4 V, 2000 mAh, Ni-Mh

Nabíjecí základna

- Rozměry: 160 x 110 x 90 mm
- Vstup: AC 100-240 V, 50-60 Hz 0,8 A
- Výstup: DC 24 V, 1 A
- Doba nabíjení: 3-4 hodiny

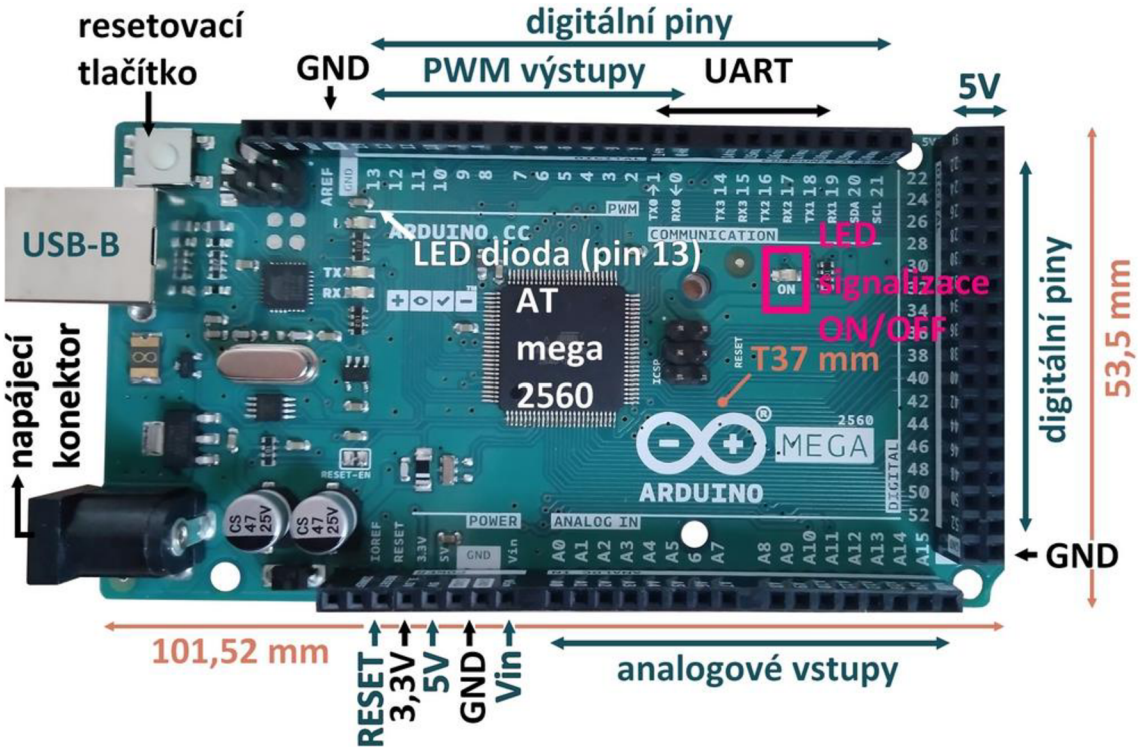
Tělo základny má tvar kruhu o průměru 340 mm a výšce 90 mm. Původní vysavač je vybaven akumulátorem o napětí 14,4 V, který bude sloužit i k napájení modelu vozíku. Zároveň je k dispozici nabíjecí základna, dobití baterie podle údajů od výrobce trvá 3-4 hodiny.

Vysavač má **dvě postranní kola** pohyblivá v jedné ose poháněná dvěma stejnosměrnými 12V motory a jedno podpůrné přední nepoháněné kolečko pohyblivé ve dvou osách.

V následujících podkapitolách budou popsány jednotlivé komponenty řídicího systému. Kompletní schéma jejich zapojení je k dispozici v příloze 5.

5.1 Arduino Mega 2560

Model Arduina zvolený pro řízení vozíku je **Arduino Mega 2560** [16] s mikrokontrolérem ATmega2560. Flash paměť je **256 kB**, SRAM 8 kB a EEPROM 4 kB. Taktovací frekvence je 16 MHz. Pracuje s napětím **5 V**.



Obr. 7: Arduino Mega 2560

Disponuje **54 I/O digitálními piny**, z toho 15 je s PWM výstupem. Dále má **16 analogových vstupů**. K 13. digitálním pinu je připojena zabudovaná LED dioda, která usnadňuje signalizaci a hledání chyb bez nutnosti zapojení externí diody.

Má čtveřici hardwarových **UART pinů** (RX, TX)[17] a dva piny pro synchronní sériovou komunikaci.

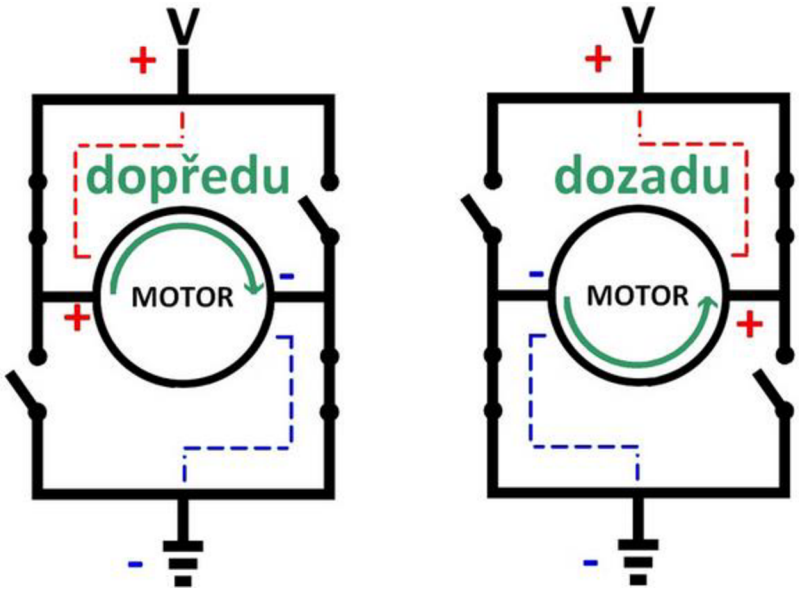
Umožňuje **reset** pomocí HW tlačítka, RESET pinu, anebo čistě softwarově v programu. Signalizuje **připojení ke zdroji** pomocí samostatné LED diody (ON).

Také má pět pinů země (GND) a dva **výstupní** napájecí piny (5 a 3,3 V), které poskytují proud o maximální velikosti 50 mA. Napájení a zároveň připojení k PC a programování Arduina je zajištěno pomocí **USB-B kabelu**. Dále je možné napájení pomocí **napájecího konektoru** a přes **Vin pin**. Doporučené napájecí napětí je **7-12 V**, povolené 6-20 V.

Rozměry desky jsou 101,52 mm x 53,5 mm x 37 mm.

5.2 Dvojitý H můstek L298N

H můstek se získal svůj název podle tvaru jeho schématu. Obecně umožňuje přepínání přívodu napětí mezi konektory motoru – lze vyměnit kladný a záporný pól napájení. Díky tomu je možné **otáčet motorem oběma směry** bez nutnosti přepojení kabelů. Princip jeho fungování je nejsnadnější ilustrovat obrázkem.



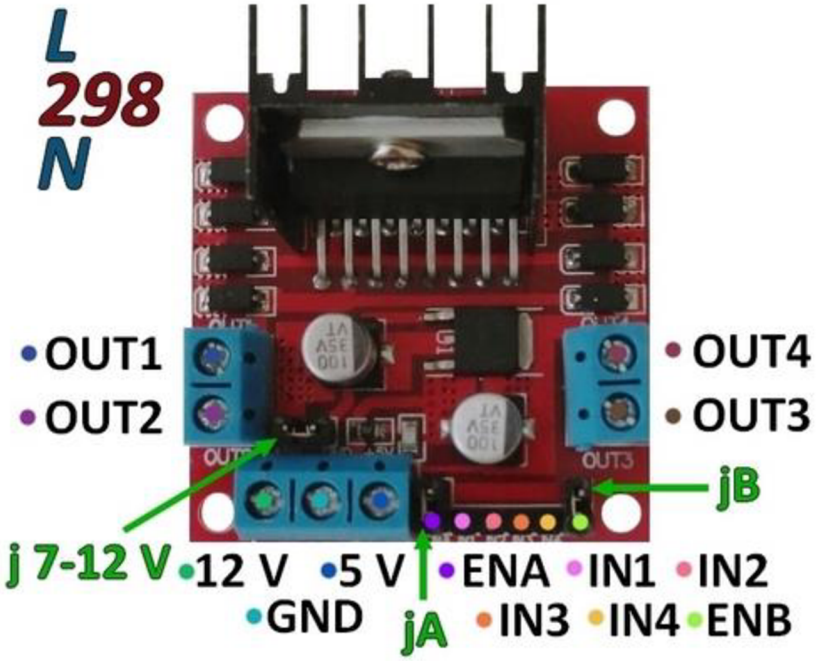
Obr. 8: Princip funkce H můstku

[17] V pořadí TX, RX je na Mega 2560: Serial0 (0,1), **Serial1 (18, 19)**, Serial2 (17,18), Serial3 (14, 15).

Ze čtyř vyobrazených spínačů jsou při pohybu motoru v jeden okamžik sepnuty maximálně dva, a to vždy jeden vedoucí od pozitivního pólu napájení a jeden od negativního ve tvaru písmene S, nebo jeho zrcadlové varianty.

Konkrétní model dvojitého **H můstku L298N** je poté ideální pro řízení Arduinem. Umožňuje ovládat jeden krokový, anebo *dva* stejnosměrné motory (proto *dvojitý* můstek). Vzhledem k užití stejnosměrných motorů bude popis ovládání krokového motoru vynechán.

Můstek je vybaven **šesti vstupy**, na které jsou posílány informace z Arduina. Jedná se o vstupy logiky řízení IN1 a IN2 (input) a vstup EN1 (enable)[18] pro stejnosměrný motor A, resp. IN3, IN4 a EN2 pro motor B. Dvojice **IN** vstupů ovlivňuje směr rotace motorů, pokud se tedy motor točí, musí být jeden ze vstupů na hodnotě napětí odpovídající logické jedničce (v tomto případě 5 V) a druhý na logické nule.



Obr. 9: Dvojitý H můstek L298N, popis pinů

Možné kombinace hodnot vstupů a jejich důsledek na rotaci motorů nejlépe ilustruje tabulka pravdivostních hodnot. Je zřejmé, že pro to, aby se motor točil, je nutné, aby vstup **EN** byl nastaven na logické jedničce („HIGH“).

[18] Bývá označován jako **blokovací** nebo **řídicí**, případně **aktivační** vstup.

EN A (B)	IN1 (3)	IN2 (4)	Motor A (B)
0	x[19]	x	Vypnutý.
1	1	1	Zastavený.
1	1	0	Točí se vpřed.
1	0	0	Zastavený.
1	0	1	Točí se vzad.

Tab. 3: Možné stavy pinů H můstku a odpovídající výstupy [17]

Enable piny se mohou jevit jako nadbytečné a je možné jich nevyužít při ponechání zapojeného propojovacího jumperu[20] (na obrázku označené jako jA, jB). Jejich velkou výhodou je však možnost měnit úhlovou rychlost motorů pomocí PWM bez nutnosti tvorby programově složitějších smyček. Hodnotu pro PWM je v kódu pro Arduino možné nastavit na 0-255[21] (odpovídá střída signálu 0 % až 100 %).

Výstupy má modul **čtyři**, OUT1 (**+**) a OUT2 (**-**) pro ovládání motoru A, OUT3 (**+**) a OUT4 (**-**) pro ovládání motoru B.

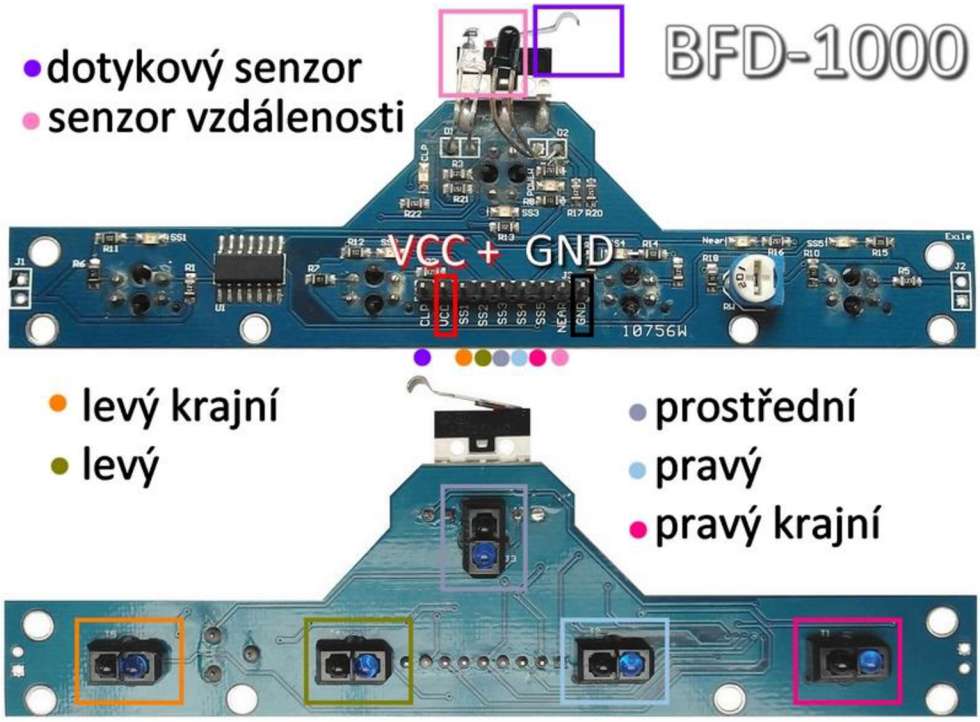
Motory ovládané přes můstek je možno **napájet napětím 5-35 V**. Logický obvod H můstku lze napájet buď **zdrojem společným i pro motory**, v tom případě je třeba ponechat jumper nad konektorem 12V na místě. Oddělené napájení je nutné při vstupním napětí pro motory vyšším, než 12 V. Při zapojeném jumperu je aktivován 5V regulátor zabudovaný v H můstku, který je určený k regulaci napájecího napětí pro logický obvod. Do konektoru 12 V je v tomto případě nutné zapojit zdroj o napětí minimálně 7,4 V při užití 6V motorů a 13,4 V při užití 12V motorů, protože v obvodu H můstku dochází k poklesu napětí o 1,4 voltů.

Pokud odpojíme jumper nad 12V konektorem, je třeba napájet **logický obvod H můstku ze zdroje odděleného** od napájení motorů (pin 5 V). Regulátor napětí zabudovaný v můstku je v tomto případě deaktivován, je tedy nutno použít přesně 5V zdroj.

[19] Písmeno **x** označuje libovolný stav, tedy 0 i 1.
[20] Česky **zkratovací propojka**, pro stručnost bude dále nazývána anglickým termínem **jumper**, který je běžně využíván i v české dokumentaci.
[21] Pomocí funkce analogWrite(), blíže popsáno v kapitole 7.1.3

5.3 Infračervený senzor sledování čáry BFD-1000

IR **senzor čáry** BFD-1000 [18] je schopen sledovat čáru na **pěti kanálech** – levý krajní, levý, prostřední, pravý a pravý krajní. Čáru je detekuje ve vzdálenosti 1-15 mm. Dále disponuje IR **senzorem překážek** (pro horizontální směr) a **spínačem** (dotykovým senzorem), které lze využít jako ochranu před nárazem. Pracuje s napájecím napětím **3,3-5,5 V**.

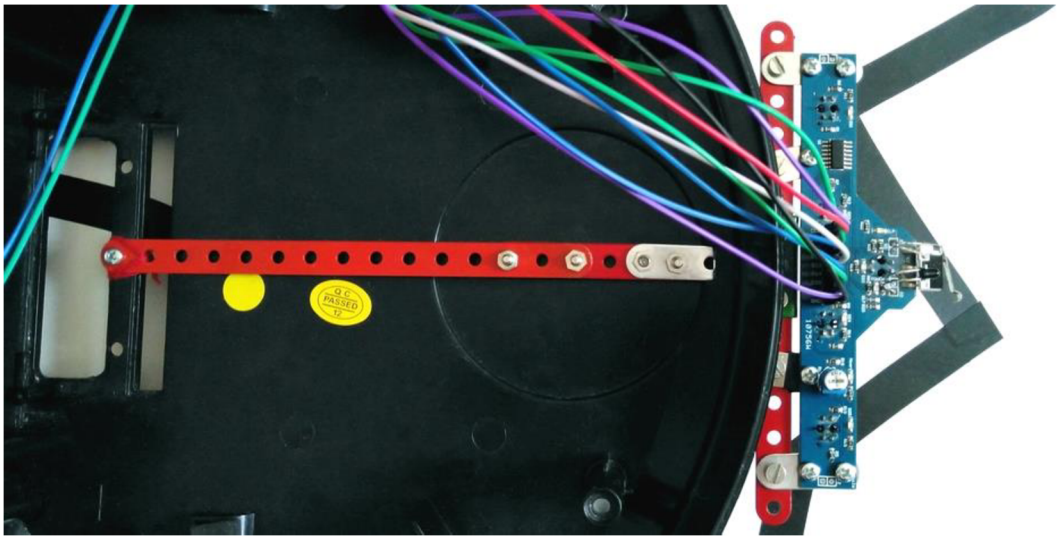


Obr. 10: IR senzor čáry BFD-1000, popis pinů

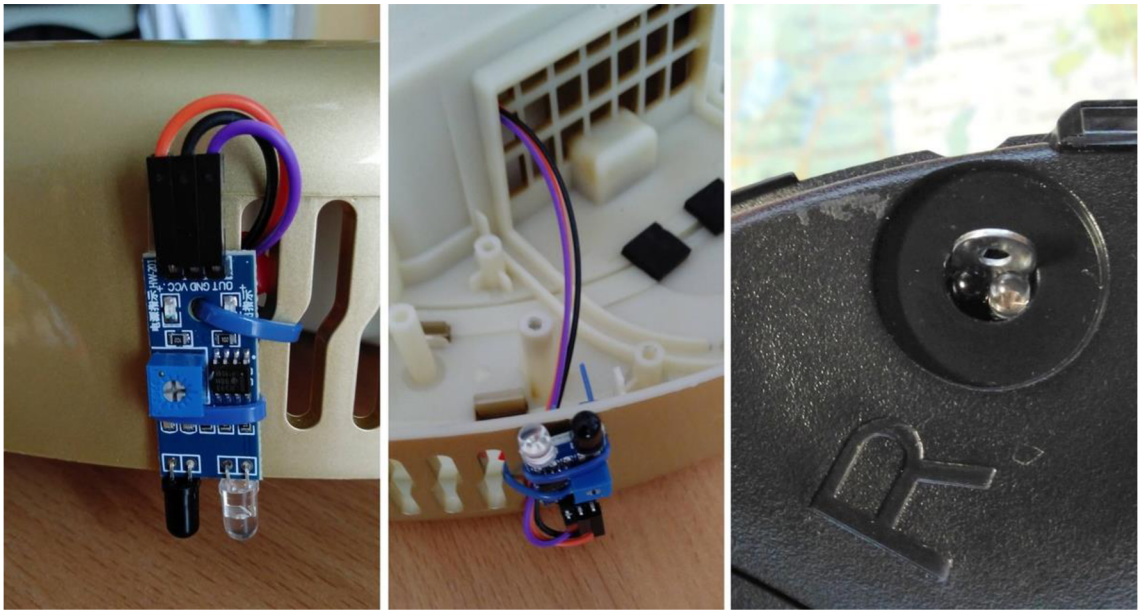
Na tělo původního vysavače byl senzor umístěn na jeho zadní stranu, proto při pohybu podle čáry vozík couvá. Na přední straně tak zůstaly dostupné původní senzory a zachován přístup k nabíjecí stanici. Zároveň se jednalo o konstrukčně nejjednodušší řešení. Upevněn byl pomocí součástek ze stavebnice Merkur.

Obr. 11: IR senzor překážek

Jednodušší komponentou senzoru čáry jsou samostatné **IR senzory překážek**. Jejich trojice byla použita pro kontrolu přiblížení k překážce z přední strany, a pro kontrolu, zda se přední či zadní část vozíku neocitla ve vzduchu.



Obr. 12: IR senzor čáry, upevnění na těle vozíku



Obr. 13: Upevnění IR senzorů výšky – zadní senzor, přední senzor

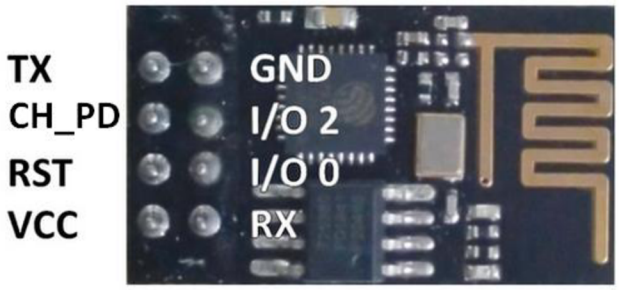
Pro kontrolu **nárazu na přední straně** byly použity původní **spínače zabudované** ve vozíku.

5.4 Wi-Fi Modul ESP8266

K ovládání modelu manipulačního vozíku bude použita mobilní aplikace, nutností je proto zajištění bezdrátové komunikace Arduino desky s aplikací přes Wi-Fi. Nabízí se i možnost ovládání přes bluetooth, avšak varianta s Wi-Fi je celkově flexibilnější.

Modul **ESP8266** od společnosti Espressif Systems je obvyklým prostředkem připojení Arduina k internetu. Z kutilské součástky se brzy rozšířila i do profesionální sféry a v dnešní době jím disponuje velká část součástí tzv. chytrých domácností. V práci použitá verze **ESP-01** disponuje dvěma vstupně-výstupními porty, pro jednoduché úkoly je tedy použitelná i samostatně. Programuje se podobným způsobem jako Arduino. Je určen pro pásmo **2,4 GHz**.

Existují i pokročilejší verze ESP modulu, například ESP-12, disponující větším množstvím pinů umožňujících softwarovou pulzně šířkovou modulaci. Na ESP modulu jsou založené i NodeMCU desky, což jsou cenově výhodné plnohodnotné desky, které svou funkcí napodobují Arduino.



Obr. 14: Rozložení pinů základního ESP-01 modulu

Pin	Význam	Připojení v režimu Wi-Fi
VCC	napájecí napětí	kladný pól napájení
GND	země	záporný pól napájení
RST	reset	(„HIGH“ pro reset)
CH_PD	úsporný režim	„HIGH“ za provozu
RX	příjem	TX Arduina
TX	vysílání	RX Arduina
I/O 2	vstupně-výstupní pin	(zapojeno při užití samotného modulu)
I/O 0	vstupně-výstupní pin	(„HIGH“ pro nahrání programu)[22]

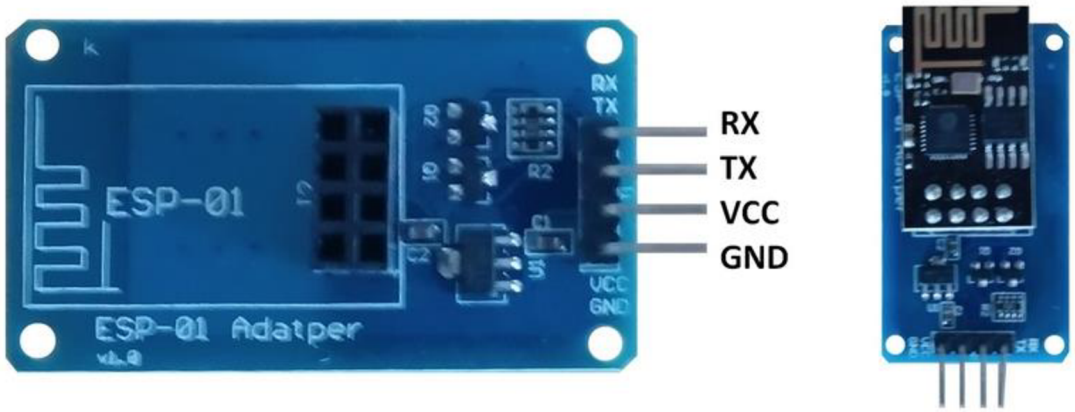
Tab. 4: Funkce pinů ESP-01 modulu

[22] Při použití ESP-01 jako samostatného mikrokontroléru lze využít i RX a TX jako I/O piny.

V provozu modulu v režimu poskytovatele Wi-Fi připojení je nutné zapojit napájení (**VCC**, **GND**), UART[23] piny (**RX**, **TX**) sériové komunikace a **CH-PD** pin[24].

Modul běží na napětí 3,3 V, zatímco Arduino na 5 V. Při užití napájecího 3,3V pinu Arduina je riziko, že nedokáže pokrýt proudový odběr modulu ve špičkách (mnohdy ale přesto funguje bezproblémově). Napájení lze řešit pomocí step-down [25] měničů (či napěťového děliče) na 3,3 V. Zároveň RX pin modulu by měl být na TX pin Arduina napojen přes napěťový dělič, aby jej nepoškodilo 5V napětí. Ze zkušenosti platí, že modul zvládne fungovat i s 5V napětím, ale oficiálně na 5 V není uzpůsoben. Tato specifika ovšem vedou k nutnosti použití velkého množství kabelů a zapojení se stává nepřehledným a je náročné hledat případné chyby či vadné kontakty.

Jednodušším a přehlednějším řešením je proto použití adaptéru se zabudovaným napěťovým regulátorem. Adaptér umožňuje zapojení modulu do nepájivého pole, stačí zapojit čtyři kabely. Napětí **zdroje** může být **4,5-5,5 V** [19]. Funkce samotných pinů (VCC, GND, RX, TX) odpovídá výše popsaným pinům ESP-01 modulu.



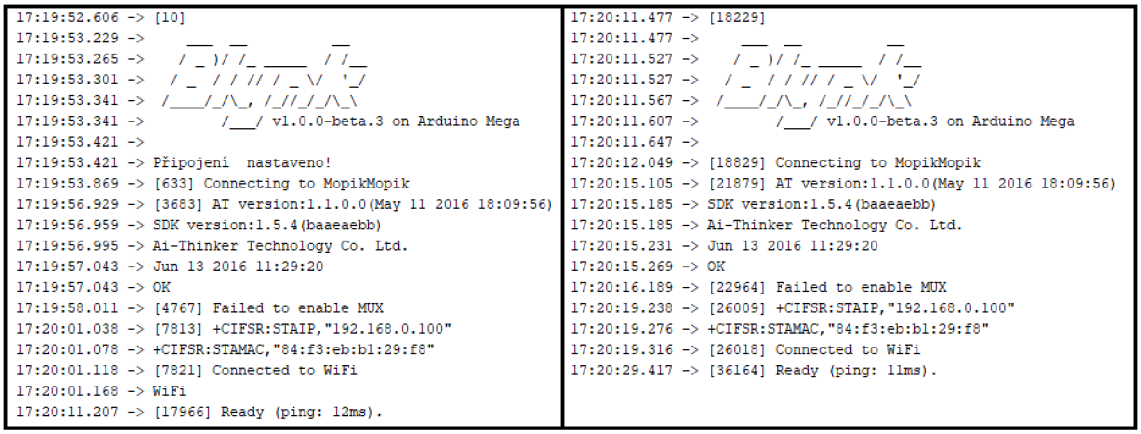
Obr. 15: Adaptér pro ESP-01 modul

5.4.1 Problémy způsobené ESP 8266 modulem

Je potřebné se zmínit, že v průběhu tvorby vozíku se jednalo o **nejproblematičtější součástku**. Ze čtyř modulů se podařilo k internetu připojit pouze s jedním z nich, aniž by byl zřejmý důvod, proč ostatní moduly nekomunikují. Hledání příčiny nefunkčnosti

[23] Univerzální asynchronní přijímač/vysílač.
[24] V hodnotě „LOW“, tzn. nezapojený CH_PD, pracuje modul v režimu nízkého proudového odběru.
[25] Stabilizátor napětí na **nižší** hodnotu, než má vstup (proto step-**down,** analogicky existují step-up měniče).

součástky velmi zpomalilo progres práce. Bylo zjištěno, že starší (dříve zakoupené) moduly s připojením k Wi-Fi a při komunikaci s Arduinem podobné problémy nevykazují. Konkrétní příčina problému je ale stále předmětem zkoumání.



Obr. 16: Sériový monitor - opakované připojování ESP modulu k Wi-Fi

Dalším úskalím je celková (ne)stabilita připojení k Wi-Fi. Modul se má tendenci poměrně často za běhu programu odpojovat. Bezprostředně po zapnutí obvodu a Arduina se modul vždy připojí nejdříve na druhý pokus, připojení vždy předchází chybová hláška: *„[23982] Failed to enable MUX.“* Následně se však zdárně připojí, o tom vždy svědčí hláška: *„[39181] Ready (ping: 12ms).“* Poté již i aplikace Blynk potvrdí kontakt s modulem slovy: *„ESP connected.“*

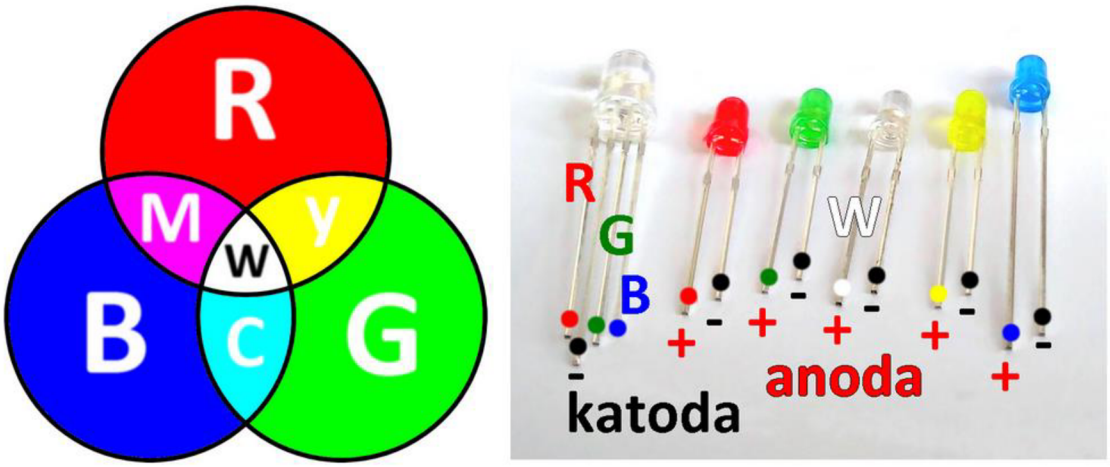
Vzhledem k časové náročnosti pátrání po chybě, která stále nebyla odhalena, by pro podobné projekty bylo vhodné zvážit užití Arduina, které již disponuje **zabudovaným Wi-Fi shieldem** a zároveň umožňuje komunikaci s aplikací Blynk. Případně využít alternativní Wi-Fi shieldy. Alternativní možností je **bluetooth připojení**[26].

5.5 LED - elektroluminiscenční diody

LED[27], česky elektroluminiscenční dioda, je dioda[28] obohacená o schopnost svítit, případně vyzařovat IR či UV záření. V základu bývá dostupná červená, žlutá (svítí spíše oranžově), zelená, modrá a bílá, jednotlivé odstíny lze i kombinovat, nejsnáze v tzv. **RGB LED**. Jedná se o kombinaci červené, zelené a modré diody zatavené

[26] Arduino Mega je možné s aplikací Blynk propojit pomocí Bluetooth modulů nRF8001, HM-10, HC-01, HC-05, HC-06 a Adafruit Bluefruit LE.
[27] Light-Emitting Diode
[28] Součástka vodivá pouze v jednom (propustném) směru, ve směru závěrném je téměř nevodivá.

v jednom pouzdře. U běžných LED je delší noha anoda (připojení na **+**), u RGB diody naopak katoda (připojení k **GND**). Napětí v propustném směru se u různých barev diod mírně liší. Základní kombinace barev lze nejlépe znázornit obrázkem.



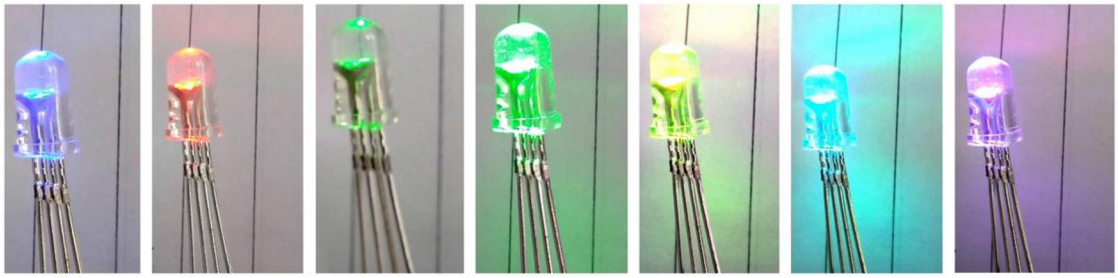
Obr. 17: Barevné kombinace LED

V práci použité virtuální diody mají signalizační funkci.

Barva	Funkce	Vysvětlení
R červená	tlačítko OFF	• Arduino napájeno • vozík vypnut - nejezdí
W bílá	tlačítko ON	• vozík softwarově zapnut • vozík přijímá pokyny k jízdě
G zelená	ESP odpojeno	• ztráta kontaktu s mobilní aplikací
B modrá	spínač TEST	• testovací režim
M purpur.	spínač ČÁRA	• režim jízdy podle černé čáry
Y žlutá	spínač TRASA	• režim jízdy po naprogramované trase
C azurová	joystick	• ovládání joystickem
O oranžová	kontrola bezpečnosti	• náraz či jiné riziko

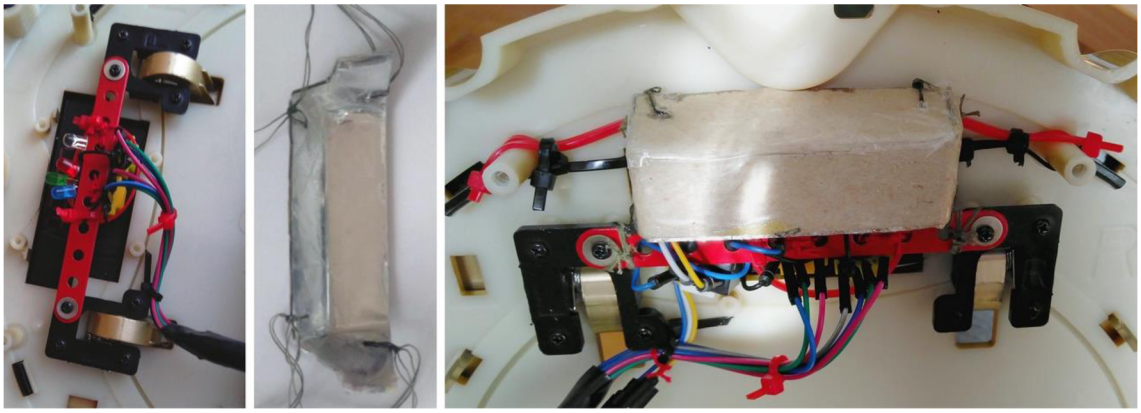
Tab. 5: Popis LED signalizace

Kromě základních barev a jejich kombinací (**RGBW**, **CMY**) lze pomocí PWM namíchat širokou paletu dalších odstínů.



Obr. 18: Různé odstíny RGB LED nastavené pomocí PWM

LED signalizace byla umístěna pod displej původního robotického vysavače. Z důvodu nedostatečné výraznosti signalizace ji následně bylo nutné ohradit vyrobenou konstrukcí se zrcadlovou fólií, která zabránila úniku záření nežádoucím směrem. K upevnění celé konstrukce byla vytvořena pomocná příčka ze stavebnice Merkur.



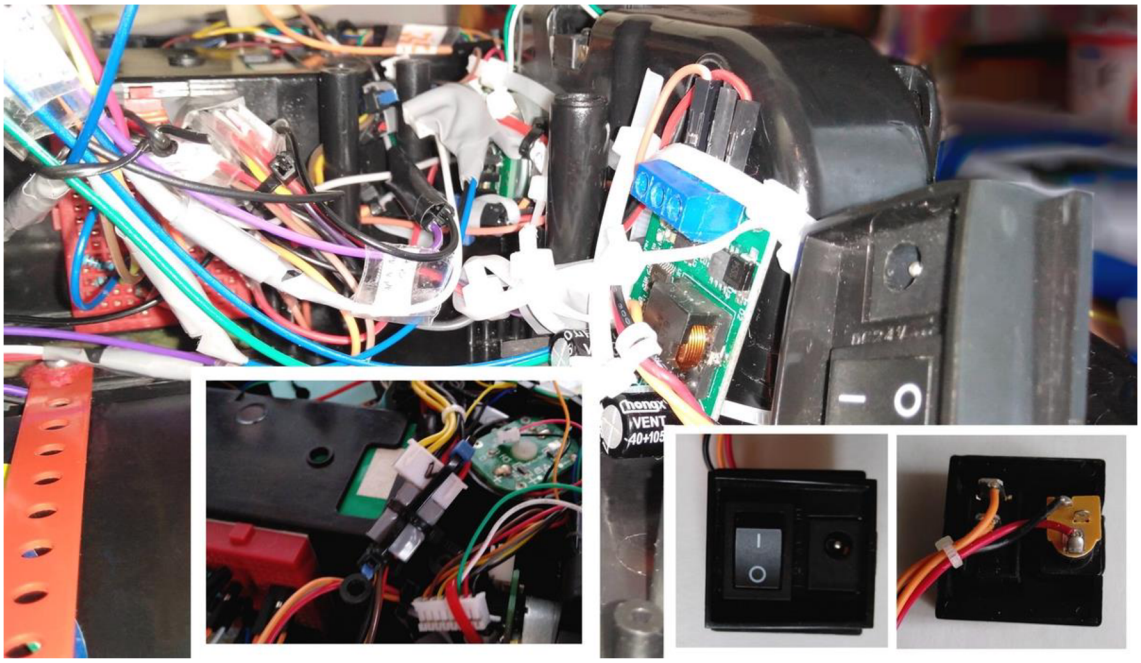
Obr. 19: LED signalizace – postup upevňování, zrcadlová konstrukce, výsledek

5.6 Napájení vozíku

Vozík je napájen z původní **14,4V** Ni-Mh **baterie**. Je možné ji napájet za pomoci originální nabíjecí základny a nabíjecího kabelu. Pro velké množství senzorů vyžadující napájení 5 V byl použit step-down měnič. Měnič převádí napětí 9-35 V na **5V výstup**.

Obr. 20: Použitý stabilizátor napětí na 5 V

Jeho použití je vhodnější, než napájení senzorů z 5V pinů Arduina. U pinů Arduina by byla překročena proudová zatížitelnost pinu, což může způsobit nefunkčnost senzorů a poškození Arduina.



Obr. 21: Příprava napájení vozíku

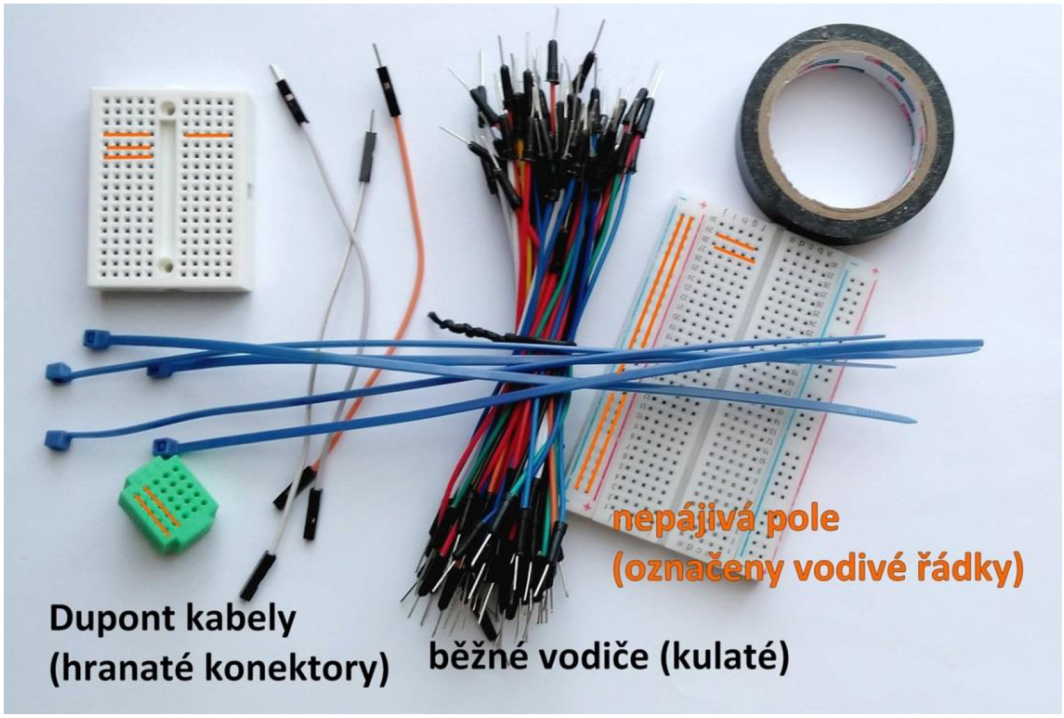
5.7 Další použité součástky

Další nutné součástky použité v projektu byly vodivé kabely, jak běžného typu, tak i typu Dupont. K propojení obvodu napájení bylo použito nepájivé pole. Také bylo použito několik rezistorů, především pro obvody s LED.

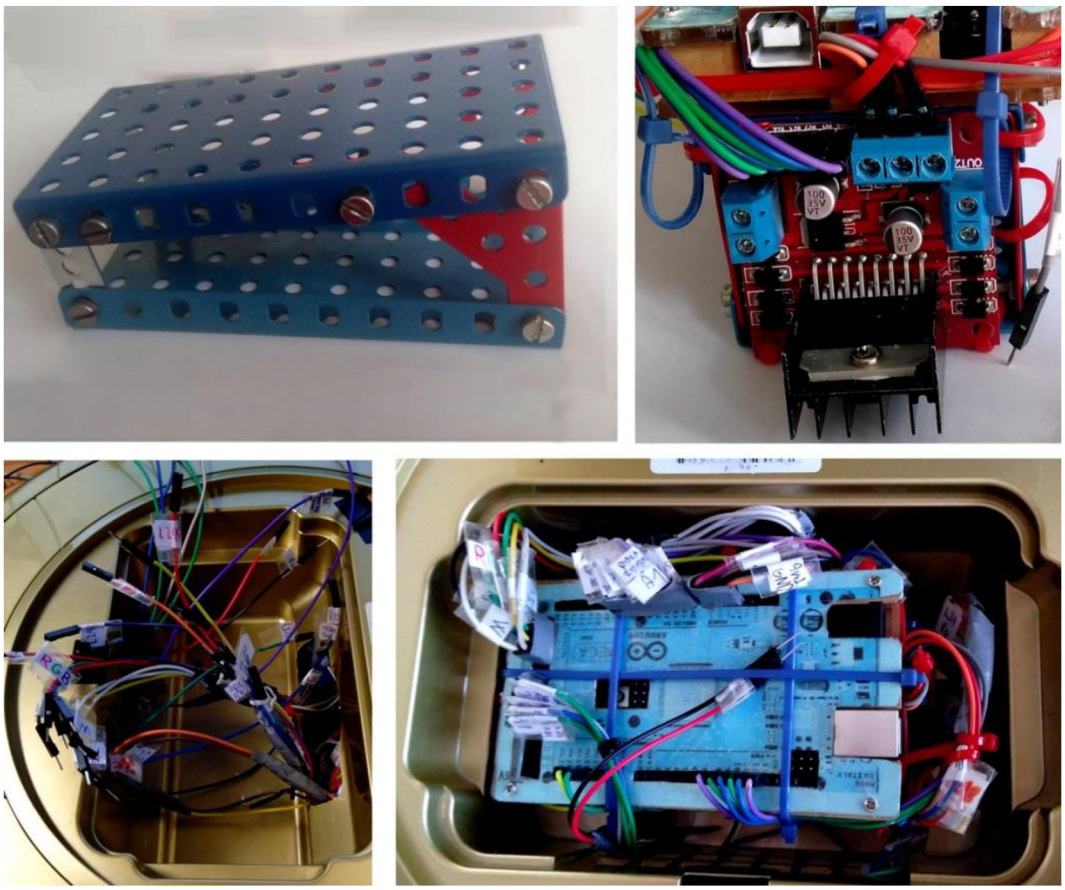
K zajištění větší bezpečnosti, organizovanosti a upevnění součástek byla použita izolační páska a plastové stahovací pásky.

Pro konstrukci podstavce pro Arduino umístěné v těle vozíku, které je umístěno v původní odpadové nádrži byla použita stavebnice Merkur. Její tvar umožňuje zapojení USB kabelu pro nahrávání programu bez nutnosti hýbat dalšími součástkami.

Součástkami za zmíněné stavebnice byly i zajištěny jednotlivé IR senzory vzdálenosti, aby byla jejich diodám poskytnuta ochrana před ohnutím. Arduino bylo umístěno do ochranného pouzdra, především proto, aby bylo zamezeno kontaktu elektroniky s kovem na konstrukci.



Obr. 22: Nepájivá pole, kabely a další pomůcky



Obr. 23: Proces umístění Arduina do nádržky

6 Použitý software

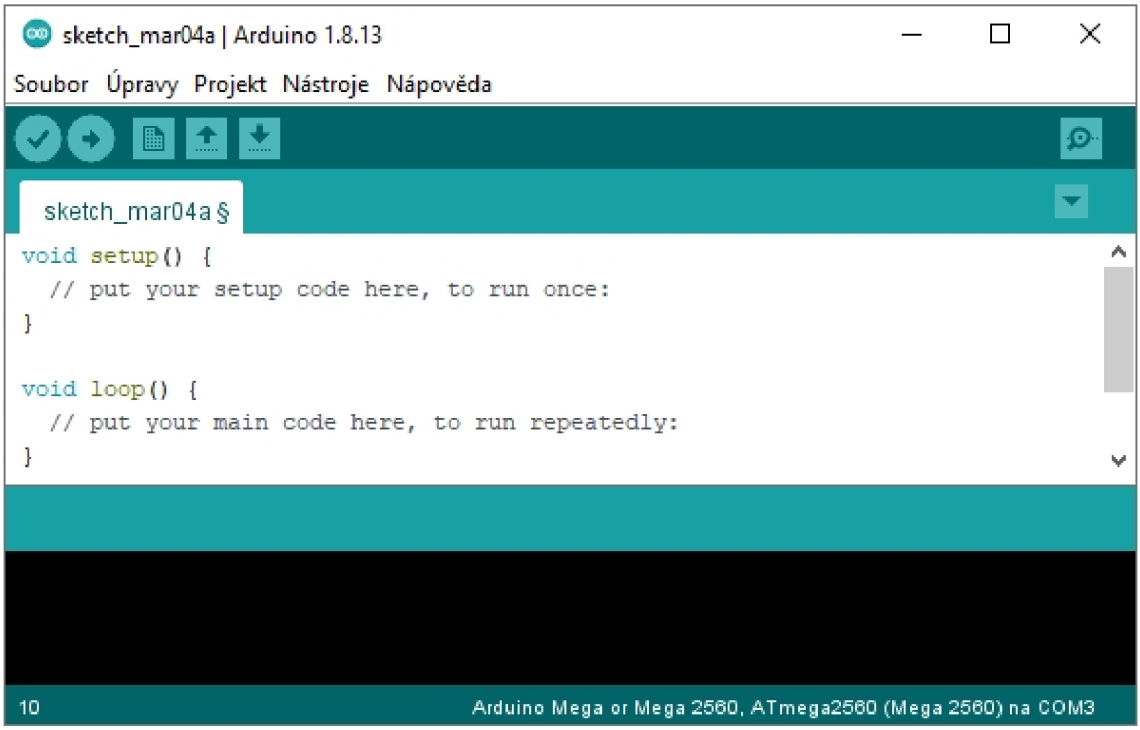
6.1 Vývojové prostředí Arduino IDE

Arduino má vlastní textový editor pojmenovaný Arduino IDE[29], ve kterém se píšou programy, takzvané **skeče**. Ukládají se s koncovkou ".ino". Programy se píší v jazyce nazvaném **Wiring** (je založený na jazycích C a C++).

Na **začátku programu** se deklarují globální proměnné s jejich datovými typy.

V části **setup** definujeme funkci jednotlivých pinů a vstupních a výstupních periferií, Tato část je načtena jedenkrát po spuštění Arduina.

V následující sekci **loop** se v nekonečné smyčce vykonává program samotný. Pro přehlednost je vhodné v loopu volat funkce definované mimo smyčku, aby kód v samotné smyčce byl co nejsrozumitelnější.



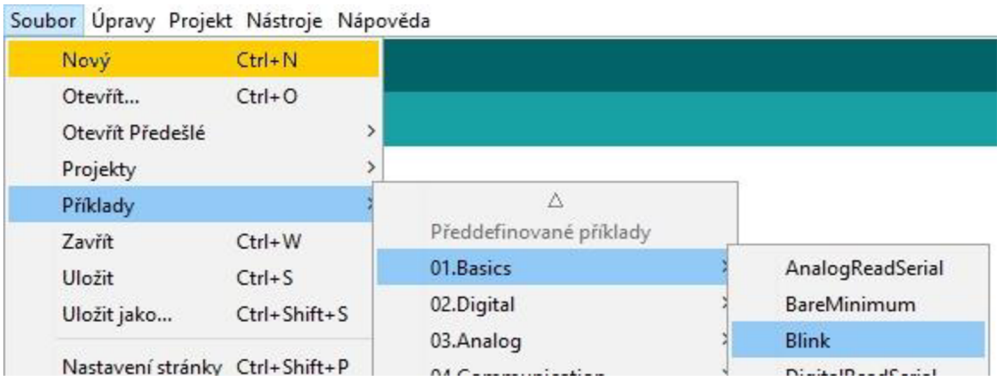
Obr. 24: Vývojové prostředí Arduino IDE.

Prostředí umožňuje napojení (nejen) na všechny aktuální typy Arduino desek i načítání knihoven periferií.

[29] IDE je anglická zkratka pro vývojové prostředí (integrated development environment), Arduino IDE je dostupné z oficiálních stránek Arduina: arduino.cc/en/software.

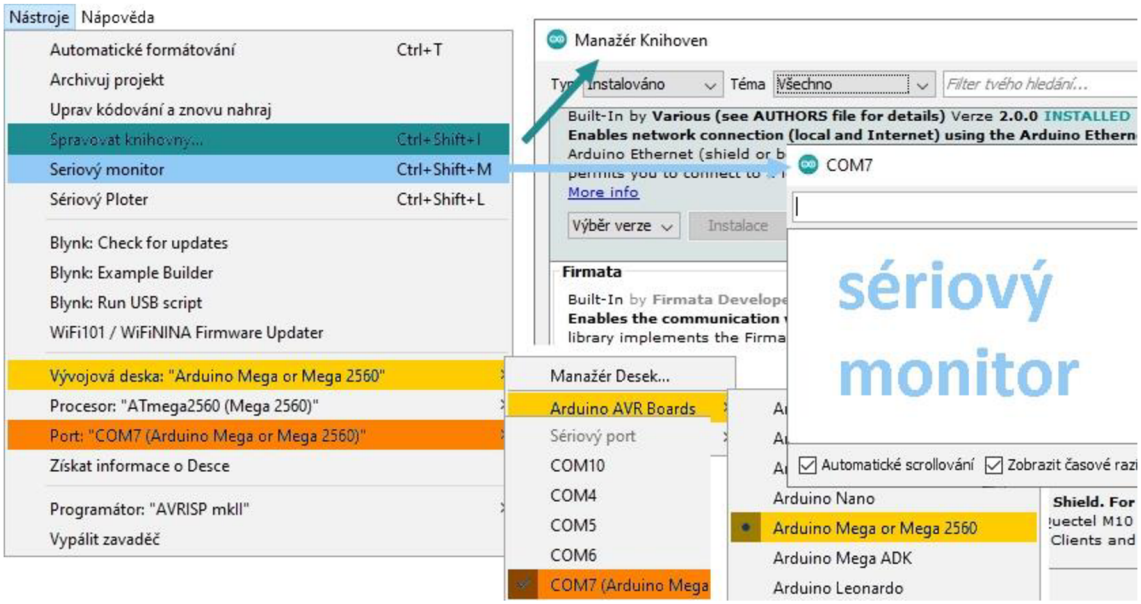
6.1.1 Významné části prostředí Arduino IDE

Tvorba skeče probíhá přes kartu Soubor → Nový (Ctrl+N). Pro prvotní seznámení s jazykem si uživatel může vybrat z desítek předpřipravených příkladů, na kterých jsou ilustrovány základní funkce Arduina. Uložení souboru probíhá standardně, každý program se automaticky ukládá do samostatné složky, která nese název skeče. Skeče mají koncovku „.ino“.



Obr. 25: Karta „Soubor“ – nový skeč, příklady

Zásadní pro nastavení správné funkce je karta „Nástroje“. V části „Spravovat knihovny“ (Ctrl+Shift+I) je možné **instalovat a aktualizovat knihovny** potřebné k ovládání periferií. Dále se v ní nastavuje **typ desky** a **COM port**[30], ke kterému je připojena.

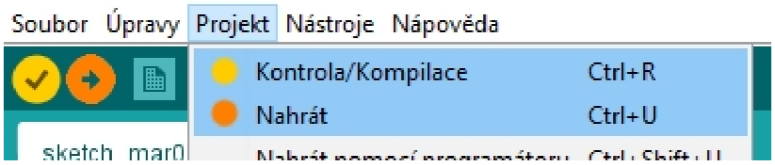


Obr. 26: Panel „Nástroje“ – knihovny, sériový monitor, volba desky a COM portu

[30] Tedy správné číslo portu, ke kterému je připojen USB kabel.

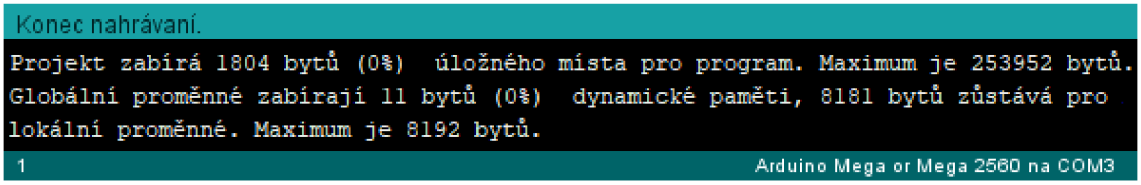
Panel obsahuje také **monitor sériové linky** (Ctrl+Shift+M), na který je možné za běhu programu promítat text a zjednodušuje kontrolu správnosti vstupů a výstupů. Je nutné poznamenat, že každé nové otevření sériového monitoru resetuje program nahraný na Arduinu.

Hotový projekt lze **ověřit** (Ctrl+R), anebo přímo **nahrát** (Ctrl+U) do desky. Ověření se spustí symbolem zaškrtnutí, nahrávání symbolem šipky, případně je obé možné přes kartu „Projekt“



Obr. 27: Karta „Projekt“ – kontrola/kompilace, nahrávání

Vespod programu je možné ve stavovém řádku vidět progres nahrávání, v černém okně kontrolovat případné chybové hlášky a na spodní liště prověřit správnost zvoleného COM portu a desky.

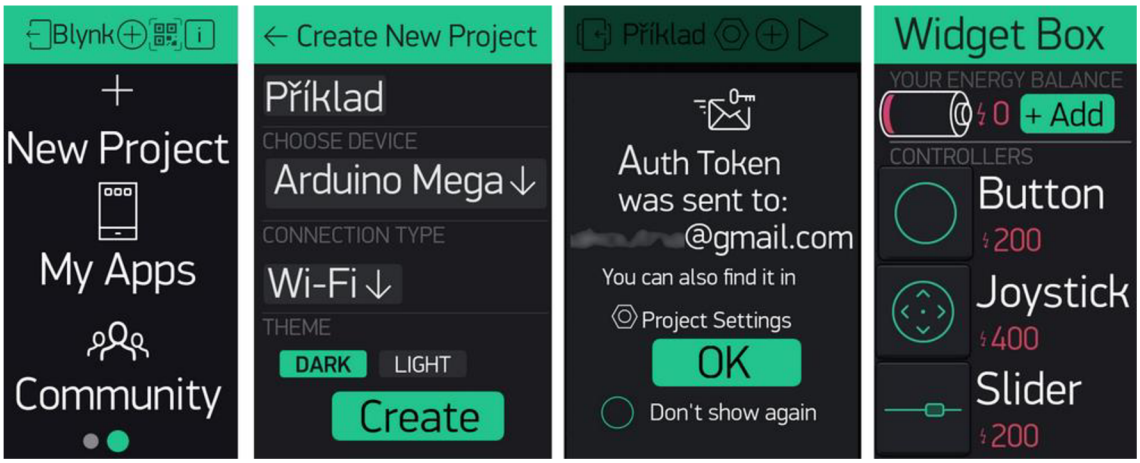


Obr. 28: Spodní lišty v prostředí Arduino IDE

6.2 Aplikace Blynk

Blynk je aplikace umožňující **tvořit vlastní aplikace k ovládání mikrokontrolérů** především vybraných Arduin. Je k dispozici pro Android i pro iOS. Dále také samostatně funguje s ESP8266 modulem, NodeMCU či Raspberry Pi a řadou dalších. **Komunikace** mezi aplikací a Arduinem je možná přes USB či ethernetový kabel, přes bluetooth i **Wi-Fi** (ESP8266).

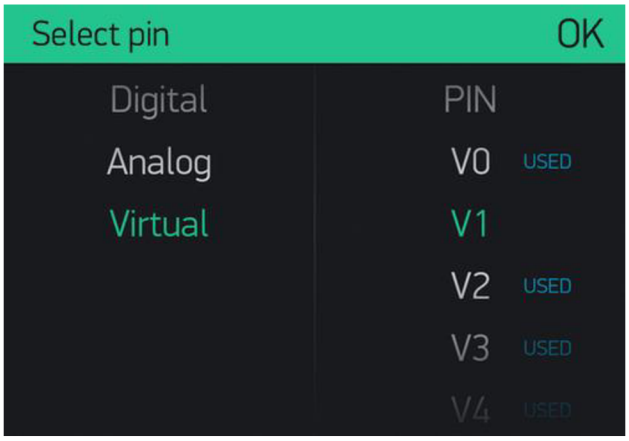
Aplikace je do určité míry bezplatná, každý má po stažení určitý kredit (2 000 „energie“), který se utrácí použitím různých widgetů, kterých Blynk nabízí velkou šíři. Po vyčerpání základního kreditu je pro rozsáhlejší projekty možné energii zakoupit.



Obr. 29: Blynk – tvorba nového projektu pro Arduino Mega s ESP8266 modulem, autorizační token a nabídka widgetů

Widgetů je celá řada – od tlačítek, přes joysticky, displeje po signalizaci – využití aplikace je univerzální.

Při tvorbě nového projektu je zásadní tzv. **autorizační token** což je heslo, které slouží k propojení aplikace s deskou a je třeba je vždy uvést v kódu nahraného do Arduina.



Obr. 30: Volba pinu v aplikaci Blynk – virtuální piny

Každý z widgetů je možné propojit s **digitálními** i **analogovými piny** Arduina, případně vytvořit tzv. **virtuální piny**. V případě použití digitálních a analogových pinů není třeba tvořit žádný kód (krom kódu propojujícího Arduino s Blynkem, který se liší podle formy spojení). Ovládání např. spínačů, diod, čtení dat z jednoduchých senzorů je proto snadné a pohodlné i bez programátorských dovedností.

Složitější na použití jsou **piny virtuální**. Jejich velkou výhodou je, že předané hodnoty můžeme snadno zahrnout do kódu, různých podmínek, cyklů… Je možná **současná oboustranná komunikace** – například můžeme synchronizovat stav pinu v aplikaci s pinem fyzickým a současně v tomtéž programu podle hodnoty na fyzickém pinu ovlivnit hodnotu na pinu virtuálním v Blynku.

Tvůrci aplikace poskytují na svém webu příklady (Blynk Example Browser) a dokumenty (Docs[31]). V příkladech je **základní tvar kódu pro propojení** Arduina a aplikace Blynk i ukázky **čtení hodnot jednotlivých widgetů**. V dokumentech je **podrobný návod** k užití aplikace i vysvětlení funkcí, které se specificky používají k programování s Blynkem. Dále existuje webová stránka s nápovědou (Blynk Help Centre) a také diskuzní fórum pro uživatele (Community).

Obr. 31: Web s příklady pro Blynk – volba desky a způsobu spojení [19]

[31] Příklady: examples.blynk.cc, dokumenty: docs.blynk.cc, nápověda: help.blynk.cc, diskuzní fórum: community.blynk.cc.

7 Struktura programu Arduina a jeho specifika

Jak již bylo zmíněno, jazyk programování Arduina Wiring vychází z jazyků C a C++. Pro účely této práce se předpokládá, že čtenář má již základní znalost jazyka C, především:

- Základní syntaxi,
- datové typy,
- logické výrazy,
- cykly if, for a while,
- tvorbu a volání funkcí,
- tvorbu struktur,
- tvorbu poznámek,
- lokální a globální proměnné,
- použití knihoven.

Blíže se tedy následující odstavce zaměřují pouze na funkce specificky spojené s ovládáním Arduina i jiných hardwarových částí a také aplikace Blynk.

7.1 Ovládání Arduina

Jak již bylo popsáno, každý kód nahrávaný do Arduina musí obsahovat sekci **setup** (výchozí nastavení po zapnutí) a **loop** (nekonečně se opakující smyčka). Za třetí část kódu můžeme považovat vše, co se nachází **mimo tyto sekce**, tedy především deklaraci proměnných a definici uživatelských funkcí. Následující kapitola usiluje o přiblížení **struktury kódu pro Arduino** a následně i popis specifik **propojení s aplikací Blynk.** Popsané principy nelze považovat za jediný možný způsob programování Arduina, jazyk Wiring umožňuje velkou flexibilitu. Uvedené příklady proto budou vycházet především z kódu napsaného v praktické části práce.

7.1.1 Část před sekcí setup

Na místo v programu před funkcí setup patří direktivy (především `#include` – zahrnutí knihoven a `#define`), definice struktur a deklarace **globálních proměnných.**

```
#include <BlynkSimpleShieldEsp8266.h>  // zahrnutí knihovny
#define EspSerial Serial1              // direktiva define
```

```
struct podminka{   // tvorba struktury pro hodnoty z pinů Arduina
    int pin;
    char pinA;
    bool hodnota;  };
    podminka levaleva, leva, prostredni, prava, pravaprava; // vstupy
      senzoru čáry
int enB;   int in3;   int in4; // deklarace pinů motoru B
bool SI;  bool SII;  bool SIII; bool SIV; bool SV; // dvouhodnotové
  proměnné pro stavy automatu
```

7.1.2 Sekce setup (nastavení výchozích hodnot)

V sekci setup dochází k inicializaci proměnných a nastavení pinů Arduina.

Pro debugging je důležitý **sériový monitor**, který je třeba inicializovat.

```
Serial.begin(9600); // inicializace sériového monitoru
```

Dochází k **přiřazení hodnot proměnným**, pokud nenastalo již při jejich deklaraci před funkcí setup.

```
  // Stavy automatů pro 1. úroveň (zapni/vypni) - tlačítka ON/OFF
SI = 1; // po „resetu“ (setup) se začíná ve stavu SI
SII = 0; SIII = 0; SIV = 0; SV = 0;
```

Dále je nastavena funkce užívaných pinů.

Nastavení **vstupních (input)** pinů:

```
narazvzad.pinA = A1;         // zadní náraz měřen na pinu A1
priblizenivzad.pinA = A2;    // přiblížení k překážce vzadu na pinu A2
pinMode(narazvzad.pinA, INPUT); // nastavení pinů na vstup
pinMode(priblizenivzad.pinA, INPUT);
```

Vstupní piny můžou být nastaveny na `INPUT_PULLUP`, v tom případě je pin napojen na interní pull-up rezistor. Slouží k eliminaci rušivého signálu. Hodnota vstupu je při jeho použití negovaná.

```
pinMode(narazvpravo.pinA, INPUT_PULLUP);
```

Třetí možností jsou piny **výstupní** – tedy piny, na které jsou z Arduina vysílány pokyny.

```
pinMode(enB, OUTPUT);  // výstupní piny pro ovládání motoru
pinMode(in3, OUTPUT);
pinMode(in4, OUTPUT);
```

V části setup je také možné nastavit hodnoty na některých pinech, které je potřeba nastavit bezprostředně po spuštění Arduina. Způsob nastavování hodnot je popsán v příští podkapitole.

7.1.3 Sekce loop (smyčka)

V sekci loop se ve smyčce opakuje hlavní program automatu. Je žádoucí ji pro přehlednost ponechat co nejstručnější a funkce v ní volané definovat dále v programu.

Pro čtení hodnot ze **vstupních pinů** slouží funkce `digitalRead()`, z pinů analogových je možné vyčíst hodnotu pomocí funkce `analogRead()`. Funkce vrací načtenou hodnotu ze zvoleného pinu.

```
digitalRead(10); // načtení digitální hodnoty (0, 1) z pinu 10
analogRead(A5);  // načtení analogové hodnoty (0-1023) z pinu A5
  // Většina Arduin umožňuje i následující operaci:
digitalRead(A5); // načtení digitální hodnoty (0, 1) z pinu A5
```

Analogová hodnota napětí je převedena na číslo 0-1023 vzhledem k referenčnímu napětí - u Arduina Mega 2560 se ve výchozím stavu jedná o 5 V.

Na výstupní piny u většiny Arduin lze posílat pouze digitální signál pomocí funkce `digitalWrite()`, některé piny ovšem umožňují PWM.

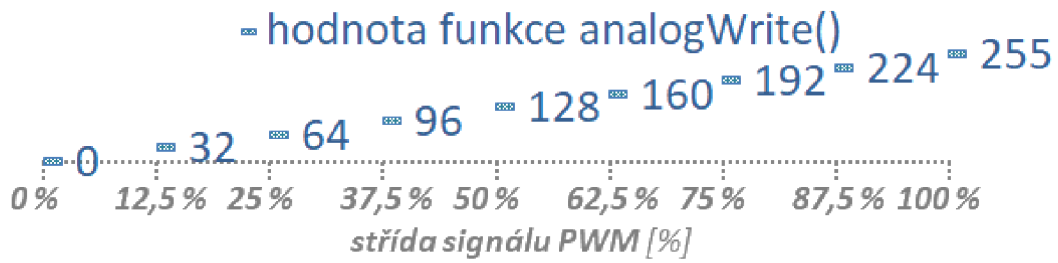
```
digitalWrite(9, 1);    // na digitální pin 9 je poslána hodnota 1
digitalWrite(9, HIGH);
digitalWrite(9, true); // význam všech tří zápisů je stejný
  // Piny s analogovým vstupem lze použít i jako digitální výstup:
digitalWrite(A4, 0);      // na analogový pin A4 je poslána hodnota 0
digitalWrite(A4, LOW);
digitalWrite(A4, false);  // význam všech tří zápisů je stejný
```

Pro PWM slouží funkce `analogWrite()`, která může nabývat hodnot 0-255. Lze ji ovšem použít jen na PWM pinech Arduina.

```
analogWrite(enB, LOW);  // střída signálu 0 % na pinu enB
```

```
analogWrite(enB, HIGH); // střída signálu 100 %
analogWrite(enB, 255);  // střída signálu 100 %
analogWrite(enB, 128);  // střída signálu 50 %
```

Přiřazení hodnoty funkce `analogWrite()` ke střídě signálu znázorňuje následující graf:



Obr. 32: Srovnání hodnoty funkce analogWrite() a střídy signálu PWM

Pro převedení jednoho rozsahu hodnot na druhý slouží funkce `map()`.

```
int hodnota = analogRead(A3); // načtení hodnoty z pinu A3 do
  proměnné hodnota
hodnota = map (hodnota, 0, 1023, 0, 255); // hodnota načtená
  v rozsahu 0-1023 bude převedena na hodnotu z rozsahu 0-255
```

Existují dva způsoby **odpočtu času**. Pro jednodušší programy je určena funkce `delay()`. Její nevýhodou je, že po jejím zavolání Arduino zůstane v předchozím stavu a do uplynutí určené doby není schopno dalších akcí. Je vhodné ji užívat pouze v programech, kde v době odpočtu není nutné vykonávat další akce a načítat či posílat hodnoty. Její použití současně s PWM výstupem může být problematické.

```
delay(1000); // zamrznutí Arduina na 1 s (1000 ms)
```

Druhým způsobem je použití funkce `millis()`, která udává **čas** v milisekundách **uběhlý od spuštění Arduina**, pomocí jednoduché matematické nerovnice je tak možné vytvořit časovač. Je důležité, aby všechny proměnné, do kterých je uložen časový údaj, byly datového typu `unsigned long`. [32]

Použití funkce `delay()` i `millis()` je ilustrováno v následujících dvou příkladech. Oba kódy vedou ke shodnému výsledku, v Ardinu zabudovaná LED u 13. pinu

[32] Tento datový typ může uložit 4 byty informací. Nejvyšší dosažitelná časová hodnota pomocí funkce `millis()` je tedy 4 294 967 295 ms, což odpovídá 49 dnům a 17 hodinám. Poté funkce znovu počítá od nuly.

5 sekund svítí a pět sekund je zhasnutá. Příklad dobře ilustruje jedinou výhodu užití funkce `delay()` – kód je stručnější a přehlednější.

```
void setup() { // vstupní hodnoty
pinMode(LED_BUILTIN, OUTPUT); // dioda na 13. pinu nastavena na
výstup
digitalWrite(LED_BUILTIN, LOW); } // na začátku je zhasnuto

void loop() { // začátek smyčky
  delay(5000); // po pěti sekundách
    digitalWrite(LED_BUILTIN, HIGH);  // dioda se rozsvítí
  delay(5000); // po pěti sekundách
    digitalWrite(LED_BUILTIN, LOW); } // dioda zhasne, konec loop
```

```
unsigned long cas = millis(); // uložení aktuálního času
bool rozsvit = 1; // na začátku chceme rozsvítit

void setup() { // vstupní hodnoty
pinMode(LED_BUILTIN, OUTPUT); // dioda na 13. pinu nastavena na
výstup
digitalWrite(LED_BUILTIN, LOW); } // na začátku je zhasnuto

void loop() { // začátek smyčky
if(rozsvit){  // chceme rozsvítit
  if (millis() - cas > 5000){ // po pěti sekundách
  digitalWrite(LED_BUILTIN, HIGH); // dioda se rozsvítí
  cas = millis(); // uložení aktuálního času
  rozsvit = 0;  }}
if(!rozsvit){ // chceme zhasnout
  if (millis() - 5000 > cas){ // po pěti sekundách (jiný zápis téhož)
  digitalWrite(LED_BUILTIN, LOW); // dioda zhasne
  cas = millis(); // uložení aktuálního času
  rozsvit = 1;  }}} // konec loop (smyčky)
```

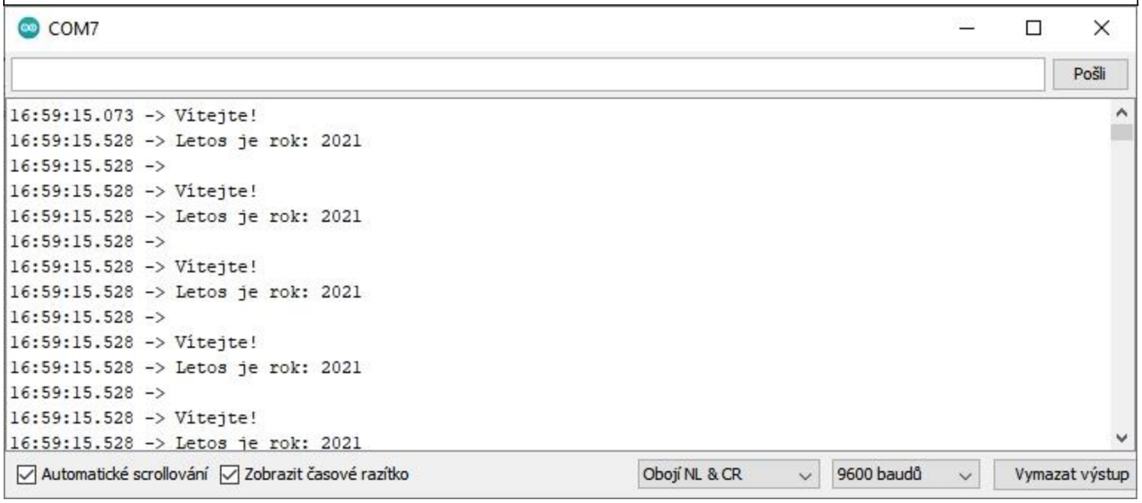
Se **sériovým monitorem** inicializovaným v části setup se dále pracuje ve smyčce pomocí několika příkazů pro tisk do sériového monitoru. Jedná se o nástroj, který je nápomocný při zobrazování hodnot z pinů a hledání chyb v kódu. Následující krátký program ilustruje základní příkazy spojené se sériovým monitorem.

```
int rok = 2021; // letošní rok uložen do proměnné

void setup() { // začátek setup
```

```
  Serial.begin(9600); } // inicializace sériového monitoru

void loop() { // začátek smyčky
  Serial.println("Vítejte!");      // vytištění textu, příští zpráva
    bude na novém řádku (ln = line)
  Serial.print("Letos je rok: "); // následující text bude na stejném
    řádku (print bez ln)
  Serial.println(rok); // vytištění hodnoty proměnné
  Serial.println(); }  // „enter"
```



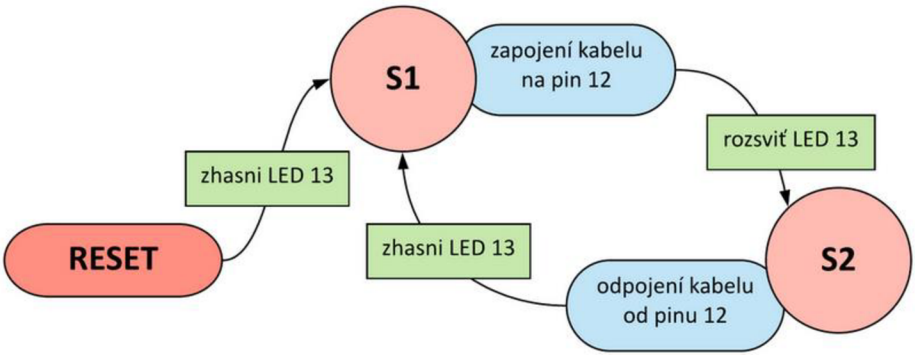
Obr. 33: Sériový monitor, základní příklad

7.1.4 Část po sekci loop

Po sekci loop je zvykem definovat funkce v ní volané. Definice uživatelských funkcí probíhá standardním způsobem. Samozřejmostí je možnost v jejich rámci plně využívat funkce pro práci s piny Arduina i ostatní výše představené funkce.

7.2 Specifika programování stavového automatu pro Arduino

Pro ilustraci toho, jak se v jazyce Wiring programuje stavový automat použijeme modifikaci příkladu **automatu na rozsvícení a zhasínání světla** z kap. 2.2.



Obr. 34: Stavový automat – jednoduchý praktický příklad programu [6]

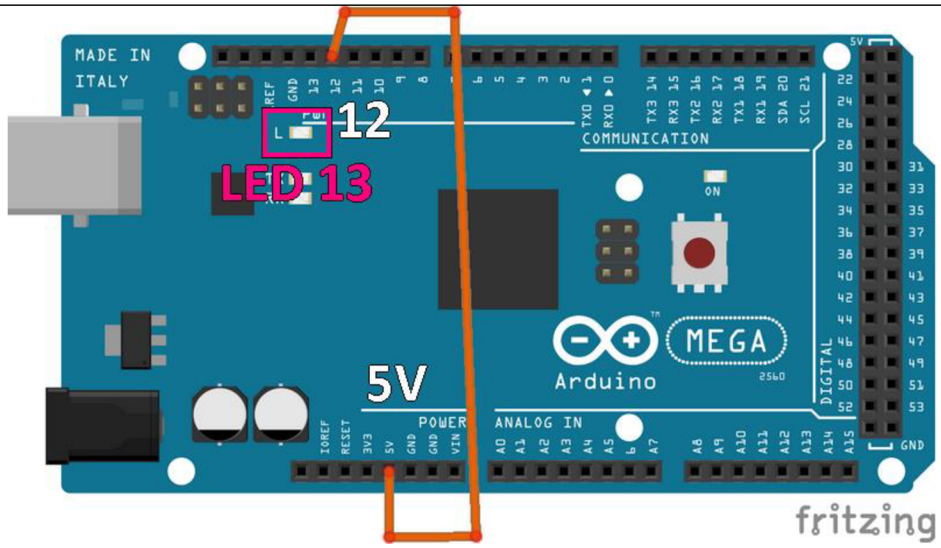
Budeme **rozsvěcovat a zhasínat LED** zabudovanou v Arduinu. Kromě desky a USB kabelu vystačíme s jedním kabelem. Kabel bude připojen k 5V výstupnímu pinu a budeme jej fyzicky připojovat k a odpojovat od 12. pinu. Tím budeme simulovat **jednoduchý spínač** – v zapojeném stavu bude na 12. pinu napětí 5 V (1, true, HIGH), v odpojeném 0 V (0, false, LOW). Program bude vypadat následujícím způsobem:

```
bool S1; bool S2; // Deklarace dvouhodnotových proměnných stavů
automatu.

void setup(){ // Začátek sekce setup.
  pinMode(12, INPUT); // Pin 12 (kabel) nastaven na vstup.
  pinMode(LED_BUILTIN, OUTPUT); // LED na 13. pinu nastavena na
výstup.
  S1 = HIGH;     S2 = LOW; // Po resetu nastane S1, nesmí tedy být S2.
  digitalWrite(LED_BUILTIN, LOW); // Na začátku je zhasnuto. = AKCE
  digitalWrite(12, LOW); // Na začátku je kabel odpojen.
} // Konec setup.

void loop(){ // Začátek nekonečné smyčky.
if (S1) { // Ve stavu 1:
  if (digitalRead(12)){ // Když je kabel zapojen: = PODMÍNKA
    digitalWrite(LED_BUILTIN, HIGH); // Rozsviť diodu! = AKCE
    delay(50); // Zamrzni na 50 ms (zpomalení smyčky).
    S1 = LOW;
    S2 = HIGH;}} // Přesun do S2. = PŘESUN DO DALŠÍHO STAVU

if (S2){ // Ve stavu 2:
  if (!digitalRead(12)){ // Když je kabel odpojen: = PODMÍNKA
    digitalWrite(LED_BUILTIN, LOW); // Zhasni diodu! = AKCE
    delay(50); // Zamrzni na 50 ms (zpomalení smyčky).
    S1 = HIGH;
    S2 = LOW;}} // Přesun do S1. = PŘESUN DO DALŠÍHO STAVU
} // Konec loop.
```



Obr. 35: Schéma zapojení příkladu stavového automatu

7.3 Aplikace Blynk

Konkrétní aplikace pro zpracovávaný projekt je dostupná po naskenování QR kódu.

Obr. 36: Řídicí Blynk aplikace „Mopík“ a její QR kód

Vozík je nastaven na připojení k Wi-Fi se jménem „**MopikMopik**“ a heslem „**Moppy2021**“.

7.3.1 Propojení aplikace Blynk s Arduinem.

Zásadní pro propojení s Arduinem je tzv. **autorizační token**, který je nutné uvést v kódu. Pokud v je v aplikaci nastavena možnost spojení různými způsoby (např. Wi-Fi, USB kabel, bluetooth), je nutné pro každé z těchto zařízení vygenerovat samostatný token. Konkrétně pro vozík, **Wi-Fi** připojení pomocí **ESP** modulu, vypadá část kódu nutná ke spojení následovně:

V sekci **před** funkcí `setup()`:

```
#define BLYNK_PRINT Serial
#include <ESP8266_Lib.h>
#include <BlynkSimpleShieldEsp8266.h>

  // Údaje WiFi + spojení s aplikací Blynk
char auth[] = "o_T4omHT_8aTfQhJbNVNP9_lejhFS3oz";   // autorizační
  token z aplikace Blynk
```

```
char ssid[] = "MopikMopik";
char pass[] = "Moppy2021";
char server[] = "blynk-cloud.com";
int port = 8080;

#define EspSerial Serial1 // TX, RX 1 (18, 19), 2 (17, 18),
  3 (14, 15) HW UART na Arduino Mega 2560
#define ESP8266_BAUD 115200 // baud rate našeho ESP8266-01
ESP8266 wifi(&EspSerial);
unsigned long cas_pripojeni;
```

Před sekcí setup není nutné uvádět server a port (označeno mentolovým pozadím). Naopak nezbytné jsou **autorizační token**, **název Wi-Fi** sítě a její **heslo**. Proměnná `cas_pripojeni` slouží dále k nastavení opakovaných pokusů při neúspěšném prvním připojení, lze ji vynechat. Všechny ostatní části kódu je nutné zahrnout.

V rámci funkce **`setup()`**:

```
Serial.begin(9600); // spuštění sériového monitoru
  // Nastavení připojení k Wi-Fi pomocí ESP modulu
EspSerial.begin(ESP8266_BAUD);
  delay(10);
  Blynk.config(wifi, auth, server, port);
  Serial.println("Připojení  nastaveno!");
  Blynk.connectWiFi(ssid, pass);
  Serial.println("WiFi");
  Blynk.connect();  // pokusí se zahájit komunikaci
  cas_pripojeni = millis() + 60000;  // při neúspěšném pokusu se ESP
po minutě znovu zkouší připojit (60 000 ms)
  Blynk.begin(auth, wifi, ssid, pass);
```

Zbytné části kódu jsou opět označeny mentolovou. Rozšíření o více funkcí slouží ke specifičtějšímu nastavení připojení a informování uživatele o stavu připojení pomocí sériového monitoru. Také připravuje proměnnou času pro funkci `spojeniBlynk()` volanou v sekci **`loop()`**.

Také je nutné z oficiálních stránek aplikace Blynk stáhnout dvojici knihoven a zahrnout je v kódu, jedná se o `ESP8266_Lib.h` a `BlynkSimpleShieldEsp8266.h`.

```
void spojeniBlynk(){
  if (Blynk.connected()){ // Kontrola připojení k Wi-Fi - fce.
      Blynk.connected = 1, pokud je spojení funkční
    Blynk.run();  } // Blynk jede.
  else if (millis() > cas_pripojeni){ // za čas připojení se
      nepodařilo připojit k Wi-Fi.
    Blynk.connectWiFi(ssid, pass); // pokusí se znovu připojit
    Blynk.connect(); // pokusí se zahájit komunikaci
    cas_pripojeni = millis() + 60000;  }} // nastaví časovač na 60 s
```

Tato funkce není pro spojení s Blynkem nutná. Její výhodou však je, že pokud byl pokus o připojení k Wi-Fi neúspěšný, po minutě (či jinak uživatelsky definovaném časovém úseku) se opětovně pokouší připojit.

7.3.2 Načítání a aktualizace hodnot virtuálních pinů

V případě použití virtuálních pinů je nutné pro jejich korektní fungování znát několik **funkcí** a **maker.**

Základem je makro **`BLYNK_WRITE`**. Je určené pouze pro virtuální piny, s digitálními a analogovými piny nefunguje.

```
BLYNK_WRITE(V10){ // OFF
 off.nacti = param.asInt();  } // při změně stavu tlačítka OFF
  v aplikaci (virtuální pin V10) se jeho hodnota načte do proměnné
  off.nacti
```

`BLYNK_WRITE` není možné vložit do funkce či podmínky. Ve funkcích lze pracovat pouze s načtenou proměnnou, Případně podmínku použít přímo v rámci makra.

```
BLYNK_WRITE(V1) { // ČÁRA
 if(SIII){cara.nacti = param.asInt(); }} // podmínka if v rámci
  BLYNK_WRITE
```

Je důležité poznamenat, že přiřazená hodnota proměnné v rámci `BLYNK_WRITE` se aktualizuje pouze v případě **manuální změny stavu**, například když je spínač ručně sepnut v rámci mobilní aplikace. Pokud je změny stavu docíleno softwarově pomocí funkce `Blynk.virtualWrite()` proměnná v rámci `BLYNK_WRITE` svou hodnotu nezmění.

```
Blynk.virtualWrite(10, 1);  // tlačítko OFF (pin V10) je sepnuto (1),
  tedy je mu přiřazena hodnota 1 (HIGH, true)
```

Pro změnu hodnoty proměnné při **programové změně hodnoty** na pinu je nutné použít funkci **`Blynk.syncVirtual()`**.

```
Blynk.syncVirtual(10); // pin V10, OFF tlačítko - zaktualizuje se
  hodnota off.nacti
```

Ucelený pohled na práci s aplikací Blynk, widgety a Arduinem nabízí dříve zmíněné příklady, dokumenty, nápověda a diskuzní fórum pro uživatele.[33]

[33] Příklady: examples.blynk.cc, dokumenty: docs.blynk.cc, nápověda: help.blynk.cc, diskuzní fórum: community.blynk.cc.

Problematickým bodem při práci s virtuálními piny je rychlost odesílání dat. Pokud je jich v programu vysíláno velké množství (konkrétně více jak deset příkazů `Blynk.virtualWrite()` za sekundu), Blynk v rámci omezení datového toku odpojí ESP modul.

Existuje ještě jeden způsob zápisu, který se v praxi projevil rychlejšími reackcemi systému i bez užití funkce **`Blynk.syncVirtual()`**. U pinů, které jsou používaný opakovaně, lze tedy doporučit i tento zápis:

```
BLYNK_READ  (V10){
   Blynk.virtualWrite(V10, off.vystup);
    } // OFF
```

Datový tok lze také omezit použitím tzv. speciálního BlynkTimeru, v praxi v projektu se však neosvědčil, došlo ke zpomalení reakcí bez benefitu méně častého odpojování ESP modulu.

8 Automaty a funkce použité v programu

Celý finální program manipulačního vozíku funguje jako jeden rozsáhlý automat. V automatu bylo nutné zohlednit velké množství vstupů. Konkrétně se jedná o:

- HW zapínání a vypínání systému,
- SW zapínání a vypínání automatu,
- sledování čáry,
- jízdu po předem naplánované trase,
- ovládání pomocí joysticku
- nastavení rychlosti vozíku,
- možnost zapnout testovací režim,
- LED signalizaci stavu.

V podkapitolách budou předvedeny úryvky kódu pro exekuci funkcí automatu. Kompletní kód vozíku je k dispozici v příloze 2.

8.1 Popis úrovní automatu

Je zřejmé, že takový automat bude obsahovat desítky stavů a pro přehlednost a srozumitelnost je třeba rozdělit jeho schéma do podkategorií.

Program bude rozdělen do tří úrovní. Jedná se však stále o jeden automat a jednotlivé úrovně jsou jen vnořeny pod stavy vyšší úrovně.

První úroveň řeší stisk tlačítka *ON/OFF*. Zabývá se tedy zapínáním a vypínáním automatu pro další ovládání.

Druhá úroveň řeší *ovládací mód*. Vybírá se v ní, zda vozík pojede podle čáry, po předem určené trase, anebo bude ovládán joystickem.

Souběžně *kontroluje bezpečnost jízdy*, tedy zda nedošlo k nárazu, anebo vozíku nehrozí nebezpečí, které by mohlo vést k poškození. Umožňuje zapnutí *testovacího režimu*, kdy jsou kontroly bezpečnosti vypnuty.

Třetí úroveň je rozdělena na tři rovnocenné části, zabývá se konkrétně ovládáním motorů pro:

1. Jízdu podle čáry,

2. jízdu po předem určené trase,
3. ovládání joystickem.

Celé schéma programu je tedy rozděleno na **šest základních částí (1-2-3)**. Třetí úroveň již není řešena jako stavový automat, protože je nutné neustále ověřovat všechny podmínky – automat by tedy nesnížil výkonovou zátěž. Jednotlivé stavy by nesly jen symbolickou funkci a program nijak nezpřehledňovaly.

8.2 První úroveň – zapínání a vypínání systému

Na první úrovni se řídí zapnutí a vypnutí celého řídicího systému pomocí tlačítek ON a OFF v aplikaci Blynk. Po sepnutí ON tlačítka se také neustále ověřuje, zda nedošlo k odpojení od Wi-Fi. Pokud k němu došlo, vozík se zastaví.

Je nadřazena dalším úrovním – když je automat ve stavu OFF, vozík nereaguje na spuštění sledování čáry, trasy, ani na ovládání joystickem. Pokud je během jízdy automat vypnut (tlačítko OFF je sepnuté), zastavuje se automaticky jízda vozíku.

Podrobněji první úroveň programu popisuje její schéma:

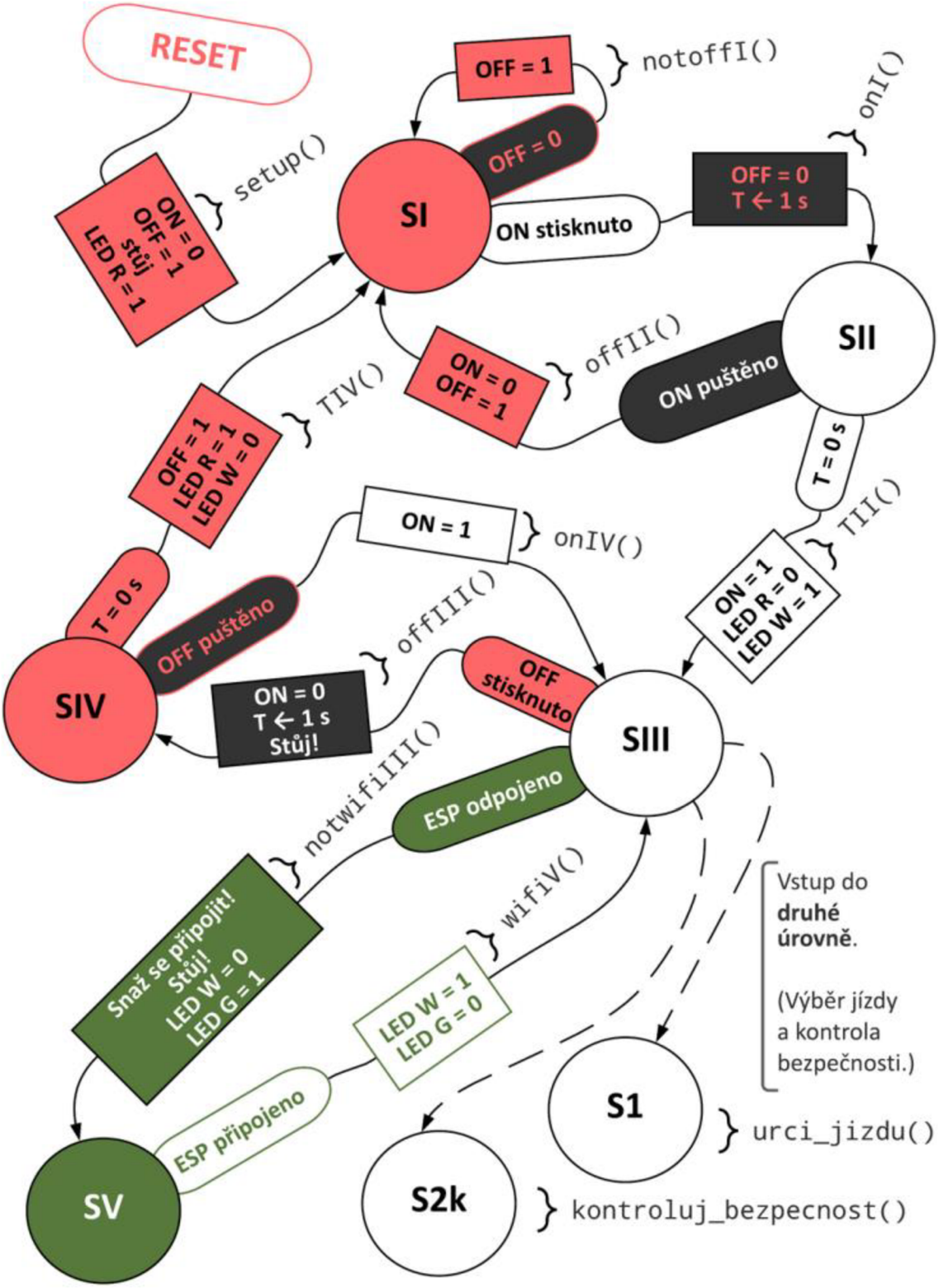
Po **RESET**u svítí červená LED, tlačítko OFF je sepnuté. Vozík je zastaven.

Ve stavu **SI** probíhá kontrola stisku tlačítka ON. Po jeho stisku se spouští přesun do SII, odpočet jedné sekundy a vypne se tlačítko OFF. Pokud je v SI ručně rozpojeno tlačítko OFF, opětovně se sepne.

Ve stavu **SII** pokračuje odpočet jedné sekundy. Jestli mezitím uživatel pustí tlačítko ON, následuje návrat do stavu SI a opětovné rozsvícení tlačítka OFF. Když tlačítko ON je sepnuté alespoň jednu sekundu, následuje jeho trvalé softwarové sepnutí, zhasnutí červené a rozsvícení bílé LED a přesun do stavu SIII.

Stav **SIII** je vstupní brána do **druhé úrovně programu**, která je popsána v nadcházejících podkapitolách. Dále v něm probíhá kontrola stisku tlačítka OFF a také ztráty spojení s Blynkem.

Při stisku OFF tlačítka se spouští odpočet jedné sekundy a dojde k přesunu do stavu SIV. Je nutné podotknou, že sekundové intervaly odpočtů jsou vzhledem k latenci v komunikaci s aplikací Blynk spíše orientační – reálně je čas stisku delší.



Obr. 37: Schéma první úrovně programu [6]

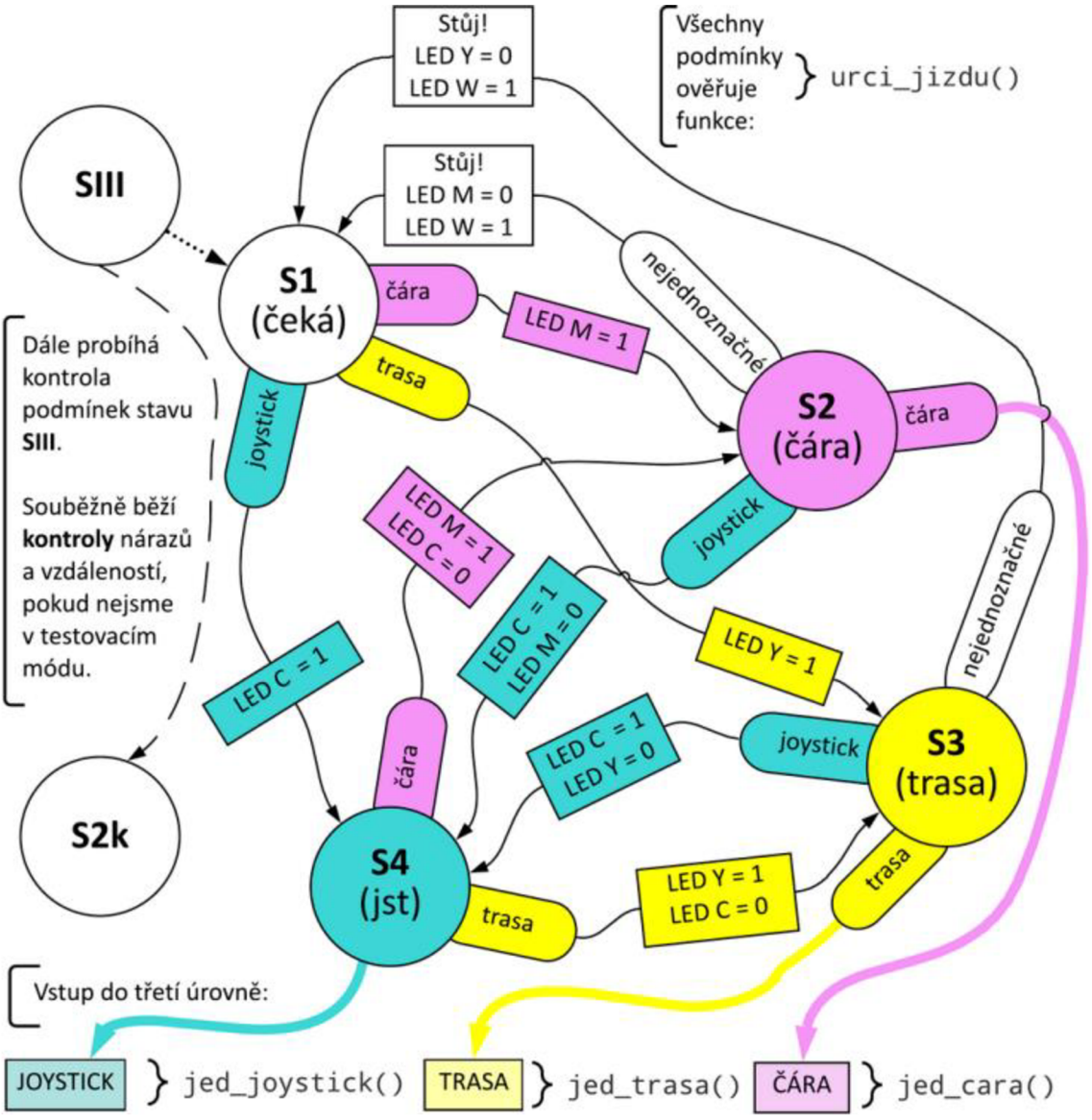
V případě, že dojde ke ztrátě spojení s Blynkem, zhasne bílá LED, rozsvítí se zelená a následuje přesun do stavu **SV**. Po opětovném navázání spojení se automat navrací do stavu **SIII** a je možné obnovit pohyb vozíku.

Ve stavu **SIV** pokračuje odpočet jedné sekundy stisku tlačítka OFF. Po jedné sekundě následuje návrat do výchozího stavu SI, kdy svítí tlačítko OFF a červená LED. Pokud došlo k puštění tlačítka v průběhu odpočtu jedné sekundy, navrací se automat do SIII.

Odpočet doby stisku tlačítek ON a OFF byl zaveden především kvůli zamezení rychlých akcí z důvodu jejich nechtěného stisku.

8.3 Druhá úroveň – volba způsobu řízení vozíku

V první větvi druhé úrovně je volena metoda, podle které se vozík pohybuje.



Obr. 38: Schéma druhé úrovně programu – volba způsobu jízdy [6]

Obecně platí, že aby vozík jel, musí být nastaven pouze na **jeden způsob jízdy**. Jinými slovy, současně sepnutý spínač ČÁRA i TRASA vede k zastavení vozíku a čekání

na vyjasnění pokynu. Pokud dojde k pohybu joysticku současně se sepnutým spínačem ČÁRA nebo TRASA, spínač je automaticky vypnut.

V případě joysticku, jehož poloha se po puštění automaticky navrací na počáteční souřadnic[34], počítá vozík s ovládáním joystickem, pokud mu bezprostředně předcházelo řízení pomocí joysticku.

Ze stavu **S2** vychází třetí úroveň – jízda podle *čáry*, ze stavu **S3** třetí úroveň - jízda po vybrané *trase* a z **S4** třetí úroveň – řízení *joystickem*. I v těchto stavech pokračuje kontrola bezpečnosti a podmínek stavu SIII (stlačení tlačítka OFF či ztráta spojení).

8.4 Třetí úroveň – pohyb, řízení motorů

Veškerý pohyb vozíku závisí na dvou zabudovaných stejnosměrných motorech. Pohyb motorů je ovládán sedmi základními funkcemi:

Funkce	Pravý motor A	Levý motor B	Pohyb vozíku
`stuj()`	Zastaven.	Zastaven.	Stojí.
`jedvpred()`	Točí se vpřed.	Točí se vpřed.	Jede rovně vpřed.
`jedvzad()`	Točí se vzad.	Točí se vzad.	Couvá rovně vzad.
`jedvpravo()`	Zastaven.	Točí se vpřed.	Jede dopředu doprava.
`jedvlevo()`	Točí se vpřed.	Zastaven.	Jede dopředu doleva.
`jedvpravovzad()`	Zastaven.	Točí se vzad.	Couvá doprava.
`jedvlevovzad()`	Točí se vzad.	Zastaven.	Couvá doleva.

Tab. 6: Funkce ovládající pohyb motorů – dopad na jízdu vozíku

Při testování vozíku bylo zjištěno, že rychlost otáčení pravého motoru je výrazně pomalejší, než rychlost levého, v důsledku vozík nezamýšleně zatáčel doprava. Empiricky byla zavedením korekčního koeficientu rychlost pravého motoru zvýšena o 19 %, čímž se rozdíl minimalizoval. Při couvání je naopak nutné zvýšit rychlost levého motoru.

Jízda vozíku je řízena funkcí `jed()`, která posílá na piny Arduina proměnné nastavené ve funkcích ovládajících směr jízdy.

```
// Funkce řídící jízdu vozíku
void jed(){  cas_jizdy = millis(); // časový odpočet
    digitalWrite(in1.pin, in1.hodnota); // směr jízdy - motor A
```

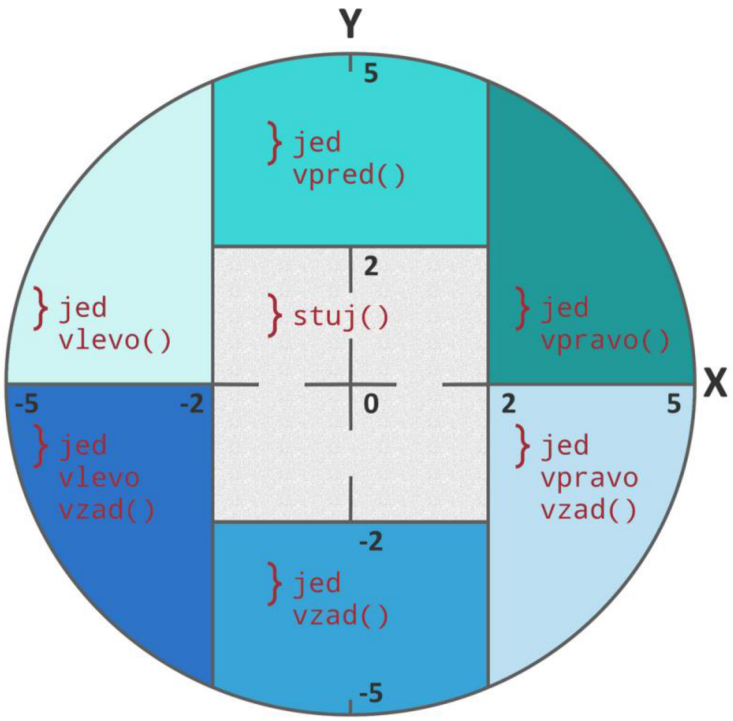
[34] Do nulové polohy [0; 0], logická hodnota je tedy 0 a joystick není „sepnut“.

```
    digitalWrite(in2.pin, in2.hodnota);
    digitalWrite(in3.pin, in3.hodnota); // směr jízdy - motor B
    digitalWrite(in4.pin, in4.hodnota);
  if(S2 || S4){ // při jízdě podle joysticku nebo po čáře
    analogWrite(enA.pin, enA.rychlost); // pojede určenou rychlostí
    analogWrite(enB.pin, enB.rychlost); }
// Jízda po trase
  else{  while(millis() - cas_jizdy < interval_jizdy){
    analogWrite(enA.pin, enA.rychlost);
    analogWrite(enB.pin, enB.rychlost); }}}
```

8.4.1 Ovládání joystickem

Funkce volané joystickem jsou určené jeho polohou v ose x a y. Hodnoty nižší, než 40 % kladného rozsahu (poloměru) joysticku nejsou považovány za významné a vozík stojí.

Funkce `urci_smer_jst()` se řídí následujícím grafem:

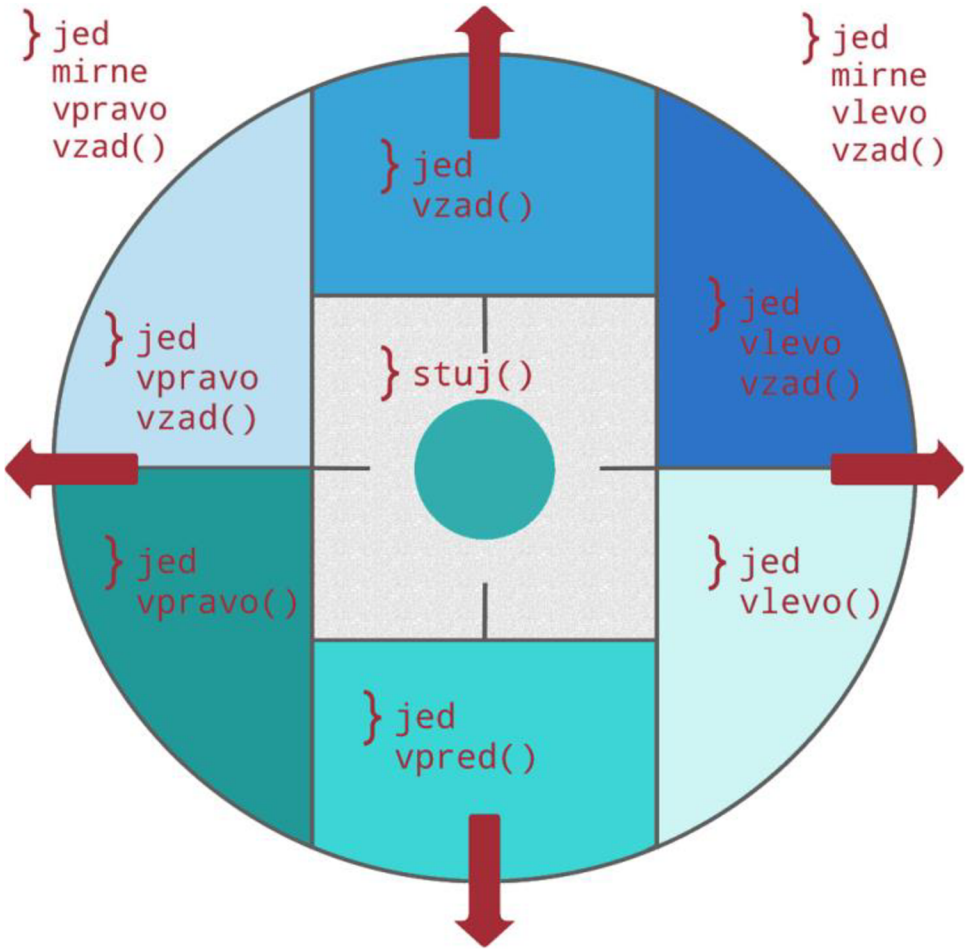


Obr. 39: Graf – závislost směru jízdy na poloze joysticku [6]

Rychlost jízdy je nastavována odděleně posuvkou a není řízena vychýlením osy y joysticku. Widget joysticku, který aplikace Blynk nabízí, je rozměrově malý a řízení rychlosti pomocí něj není dostatečně komfortní a přesné. Jedná se o nejproblematičtější část aplikace. Ovládání joystickem je náročnější na datový přenos, v důsledkem je častá zráta spojení s hardwarem.

8.4.2 Sledování čáry

Podle čáry vozík couvá, použité funkce je možné si pro lepší orientaci představit jako graf joysticku otočený o 180 °.



Obr. 40: Otočené funkce jízdy při couvání

Funkce pro pohyb byly pro účely jízdy podle čáry rozšířeny o dvojici funkcí pro mírné zatáčení při couvání. Rychlost jednoho vůči druhému motoru jsou vůči sobě v poměru 7/10.

Funkce	Pravý motor A	Levý motor B	Pohyb vozíku
`jedmirnevlevovzad()`	Točí se vzad.	0,7 rychlosti A	Couvá doleva.
`jedmirnevpravovzad()`	0,7 rychlosti B	Točí se vzad.	Couvá doprava.

Tab. 7: Funkce pro mírné zatočení vozíku

Rozhodovací algoritmus, podle kterého vozík volí směr jízdy, nejlépe ilustruje tabulka.

IR SENZOR ČÁRY

pořadí podmínky	leva leva	leva	pros tredni	prava	prava prava	Jede:
0.	1	1	1	1	1	*stojí (vypnutý senzor)*
1.		0	1	0		rovně
2.		0	0	1		mírně vpravo
3.	0	0	0	0	1	vpravo, vpravo
4.		1	0	0		mírně vlevo
5.	1	0	0	0	0	vlevo, vlevo
6.		1	0	1		rovně
7.		0	1	1		mírně vpravo
8.		1	1	0		mírně vlevo
9.		1	1	1		rovně
10.	0		0	0		rovně
11.	jinak					stojí

1	je na černé čáře
0	není na čáře
	nebráno v potaz

Tab. 8: Algoritmus jízdy podle čáry

Lze říci, že algoritmus spočívá v tom, že vozík zatáčí na tu stranu, na které zaznamenal černou pásku. Ve speciálních případech buď stojí, anebo jede rovně.

Rychlost vozíku musí neustále být poměrně vysoká, při střídě signálu PWM nižší než zhruba 100/255 má vozík velké problémy s rozjezdem. Toto omezení má dopad i na přesnost navigace – nelze počítat se schopností vozíku pružně reagovat na ostré změny směru jízdy.

8.4.3 Jízda po trase

Pro ilustraci trasy byly zvoleny tři příkladové okruhy, a to tvary:

- Srdce,

- kruhu,
- čtverce.

Uvedené trasové úseky jsou ukázkou toho, jakými příkazy lze docílit, aby vozík opsal základní tvary, ze kterých je případně možné skládat delší úseky. Každé zavolání funkce přesto způsobí vytvoření mírně odlišného tvaru, vozík je citlivý na nerovnosti i materiál povrchu a kolečka často prokluzují. Slouží proto k vytvoření základní představy o tvorbě jednotlivých tvarů. Regulací rychlosti je přirozeně regulována i velikost útvarů.

Obr. 41: Zvýrazněné tvary opsané čelem vozíku – srdce, kruh, čtverec

8.4.4 Řízení rychlosti vozíku

V plném výkonu má vozík potenciál jezdit poměrně rychle. S ohledem na omezení rychlosti komunikace a reakce na pokyny z aplikace Blynk se však jedná o rychlost příliš vysokou. Rychlost proto bylo nutné softwarově omezit. Regulace rychlosti je realizování v aplikaci Blynk na stupnici **1-5**. Výchozí hodnota je nastavena střední – číslo **3**. Velikost rychlosti je rozdílná pro každý mód jízdy, nejrychlejší pro joystick, nejpomalejší pro jízdu podle čáry. Ve funkci řídící rychlost se zároveň nastavuje interval jízdy funkce `jed()`, tedy jak dlouho se bude točit kolo zvoleným směrem v případě jízdy po předem dané trase.

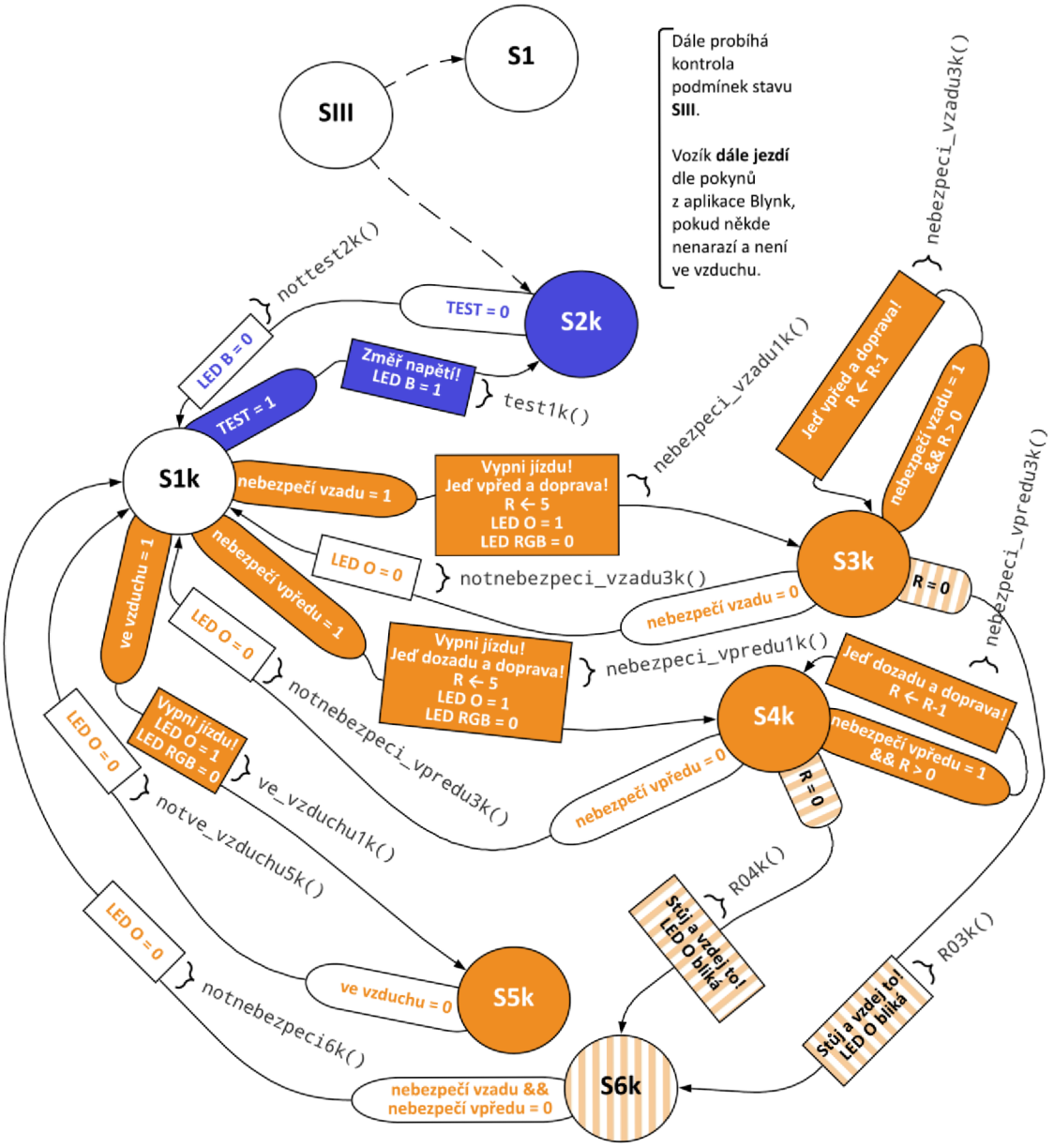
```
void urci_rychlost(){
    Blynk.syncVirtual(5); // aktuální nastavená rychlost -
synchronizace hodnoty v Blynku
  if(S4){ // když jedeme podle joysticku
    rychlost_motor = 140 + 15 *(rychlost.analogove - 1); }
  else if(S2){ // když jedeme podle čáry
    rychlost_motor = 110 + 5 * (rychlost.analogove - 1); }
  else if(S3){ // když jedeme  po trase
    rychlost_motor = 140 + 10 *(rychlost.analogove - 1);
    interval_jizdy = 420; }}  // konec urči rychlost
```

8.5 Druhá úroveň – návrh systému kontroly bezpečnosti

Kontrola bezpečnosti, tedy nárazů, rizik nárazů a zvednutí vozíku do vzduchu, může probíhat souběžně s možností volit druh jízdu. Kontroly bezpečnosti je možné vypnout spínačem TEST, který zapíná testovací režim a změří napětí na zdroji.

Při nárazu, nebo jeho nebezpečí, vozík zastaví a v několika cyklech se snaží vzdálit překážce. Pokud se nezadaří, ve stavu S6k čeká na vnější pomoc.

Pokud se vozík ocitne ve vzduchu, zastaví se, dokud není opětovně položen na zem.



Obr. 42: Schéma druhé úrovně programu – kontrola bezpečnosti [6]

9 Závěr

Předložená diplomová práce se zabývala návrhem řídicího systému výukového modelu manipulačního vozíku. Systém byl řízen pomocí desky Arduino Mega 2560 a mobilní aplikace Blynk určené pro komunikaci (nejen) s Arduino deskami.

Byl vytvořen program a s ním propojená mobilní aplikace, která řídí softwarové zapnutí a vypnutí vozíku. Byly realizovány tři způsoby navigace vozíku – podle černé pásky, po předem určených trasách a pomocí joysticku.

Jako náročné se ukázalo využití původních senzorů vozíku, protože výrobce odmítl dodat původní dokumentaci. Přesto se podařilo napojit na správné kontakty původních senzorů nárazu.

Zvolená deska Arduino Mega 2560 se osvědčila, protože u finálního projektu bylo dle předpokladu využito velké množství pinů. Její větší rozměry umožňují lepší orientaci v množství kabelů. Paměť desky nechává prostor pro další potenciální rozšiřování programu modelu. Projekt zabírá přibližně desetinu flash paměti a zhruba třetinu SRAM. I z tohoto hlediska se volba desky osvědčila – u většiny ostatních srovnávaných desek by se projekt pohyboval na hranici paměťových možností.

Problematických bylo v průběhu práce několik bodů. Především šlo o problémy s funkčností ESP 8266 modulu, který není dostatečně spolehlivý v připojování k Wi-Fi. Dalším bodem je rychlost komunikace vozíku s aplikací Blynk, která se projevuje především v pomalých reakcích na řízení joystickem, často i zaseknutím v určitém stavu jízdy a ztrátou spojení. Vzhledem k množství uživatelů aplikace má rychlost datového přenosu své limity. Řešením, které ovšem přesahuje rozsah práce, by mohla být tvorba autonomní aplikace. Nápomocná v omezení latence by mohla být i instalace lokálního Blynk serveru v místě využívání výukového modelu, které by mohlo komunikaci zefektivnit. Tyto potíže bohužel vedly k nespolehlivé funkčnosti kódu jako celku, ačkoliv jeho jednotlivé segmenty fungovaly bezproblémově.

Dalším omezením byl limitovaný rozsah rychlosti, ve kterém jsou motory vozíku schopné funkce.

Kromě řešení problémů v komunikaci se jeví několik dalších možných cest rozšíření práce. Jízdu vozíku po předem dané trase by bylo možné obohatit o kontrolu lokace

vozíku, jako nejpříhodnější se jeví využití RFID čtečky a RFID tagů zabudovaných na trase. Dále by bylo možné zdokonalit algoritmus jízdy podle čáry a rozšířit jej o hledání čáry.

Původní vozík měl zabudovaný reproduktor, proto je možné prozkoumat možnosti zvukové signalizace, především za užití PCM Arduino knihoven, která umožňuje přehrávat zvuk bez dalších přídatných modulů.

Také by, vzhledem k rychlosti komunikace s aplikací Blynk, bylo vhodné hledat další způsoby, jak omezit rozsah vysílaných a přijímaných dat bez dopadu na funkci vozíku.

Další prostor pro rozvoj vidím v možnosti vytvoření anglické verze aplikace, případně i kódu.

Seznam použité literatury

[1] BALÁTĚ, Jaroslav. *Automatické řízení: programování pro začátečníky*. Praha: BEN - technická literatura, 2003. Průvodce (Grada). ISBN 80-730-0020-2.

[2] OLEHLA, Miroslav, Slavomír NĚMEČEK a Ivan ŠVARC. *Automatické řízení*. Liberec: Technická univerzita v Liberci, 2013. ISBN 978-807-3729-721.

[3] *Základy automatického řízení*: teorie [online]. [cit. 2020-04-13]. Dostupné z: https://adoc.tips/zaklady-automatick-ho-rzen60cea5ec952c399041c5c6df24aaf90571246.html

[4] ŠVARC, Ivan. *Automatizace: automatické řízení*. Brno: Akademické nakladatelství CERM, 2002. ISBN 80-214-2087-1.

[5] Automaton: *Definition of Automaton by Merriam-Webster*. Dictionary by Merriam-Webster [online]. Merriam-Webster, 2020 [cit. 2020-04-13]. Dostupné z: https://www.merriam-webster.com/dictionary/automaton

[6] Online Diagram Software & Visual Solution: *Lucidchart* [online]. Lucid Software, 2020 [cit. 2020-04-13]. Dostupné z: https://www.lucidchart.com/

[7] *ELUC: PLC* [online]. Olomoucký kraj [cit. 2021-01-16]. Dostupné z: https://eluc.kr-olomoucky.cz/verejne/lekce/965

[8] Programovací jazyky pro PLC. *Výukový portál COPTel* [online]. 2019, 2019(3905), 2 [cit. 2021-03-09]. Dostupné z: https://coptel.cz/mod/page/view.php?id=6737

[9] Co se skrývá pod výrazy Industry 4.0 / Průmysl 4.0 ? *Automatizace.HW.cz* [online]. Praha 4: HW server, 2021 [cit. 2021-02-22]. Dostupné z: https://automatizace.hw.cz/mimochodem/co-je-se-skryva-pod-vyrazy-industry-40-prumysl-40.html

[10] Jsme připraveni na Průmysl 4.0. *Elektrotechnický zkušební ústav, s.p* [online]. Praha 8: Elektrotechnický zkušební ústav, s.p., 2021 [cit. 2021-02-22].

Dostupné z: https://ezu.cz/aktuality/prumysl-4-0-2/

[11] Co to je IoT? *IoT Port* [online]. Praha 6: České Radiokomunikace a.s, 2021 [cit. 2021-02-24]. Dostupné z: https://www.iotport.cz/iot-novinky/ostatni-clanky-o-iot/co-to-je-iot

[12] Continental Increases Efficiency with Mobile, Autonomous Transport Robots. Continental [online]. Hanover: *Continental*, 2020 [cit. 2021-01-17]. Dostupné z: https://www.continental.com/en/press/press-releases/autonomous-transport-robots-242650.

[13] Constants. *Arduino* [online]. Arduino, 2021 [cit. 2021-02-26]. Dostupné z: https://www.arduino.cc/reference/en/language/variables/constants/constants/

[14] Arduino Boards & Modules. *Arduino Official store* [online]. 2021 [cit. 2021-01-13]. Dostupné z: https://store.arduino.cc/arduino-genuino/boards-modules

[15] *RobZone MOPPY - Návod k použití*. Praha: RobZone INT

[16] ARDUINO MEGA 2560 REV3. *Arduino Official Store* [online]. Arduino, 2021 [cit. 2021-03-01]. Dostupné z: https://store.arduino.cc/arduino-mega-2560-rev3

[17] Controlling DC Motors with the L298N Dual H-Bridge and an Arduino. *Tutorials & Projects [online]*. Toronto: DroneBot Workshop, 2021 [cit. 2021-02-10]. Dostupné z: https://dronebotworkshop.com/dc-motors-l298n-h-bridge/#1

[18] 5-kanálový Infračervený senzor sledování čáry BFD-1000. *LASKARDUINO.cz* [online]. Rychnov nad Kněžnou: laskarduino.cz, 2021 [cit. 2021-5-1]. Dostupné z: https://www.laskarduino.cz/5-kanalovy-infracerveny-senzor-sledovani-cary-bfd-1000/

[19] ESP8266 Adaptér pro ESP-01 Modul 3,3 V 5V. *Dratek.cz* [online]. Termesivy 41, 58001 Havlíčkův Brod: ECLIPSERA s.r.o, 2020 [cit. 2021-2-22]. Dostupné z: https://dratek.cz/arduino/1400-esp8266-adapter-pro-esp-01-modul-3-3-v-5v.html

[20] B*lynk Example Browser* [online]. New York: Blynk, 2020 [cit. 2021-03-05].

Dostupné z: https://examples.blynk.cc/

[21] Arduino - Introduction. *Arduino - Home* [online]. 2020 [cit. 2020-04-13]. Dostupné z: https://www.arduino.cc/en/Guide/Introduction.

[22] Arduino - Environment. *Arduino - Home* [online]. 2020 [cit. 2020-04-13]. Dostupné z: https://www.arduino.cc/en/Guide/Environment.

[23] Automatic Guided Vehicle (AGV, LGV, SGV, AGC etc). *Automated Manufacturing Packaging and Logistics* [online]. Robotic Automation P/L, 2020 [cit. 2020-04-13]. Dostupné z: http://www.roboticautomation.com.au/agvs.

[24] Introduction to Pulse Width Modulation (PWM). *Software Expert Witness Services | Barr Group* [online]. USA: Barr Group, 2021 [cit. 2021-01-16]. Dostupné z: https://barrgroup.com/Embedded-Systems/How-To/PWM-Pulse-Width-Modulation

[25] ČERNÝ, Michal. *Začínáme s Arduinem: 46 řešených příkladů a 123 námětů pro vlastní pokusy*. Hobbyrobot, 2014.

[26] *Možnosti nasazení prvků konceptu průmyslu 4.0 v rámci měrového střediska velkosériové výroby*. Praha, 2019. Diplomová práce. České vysoké učení technické v Praze.

Seznam obrázků

Seznam tabulek

Seznam příloh

Příloha 1 – Drobné tipy pro program

Změna barvy v aplikaci Blynk

Prostředí mobilní aplikace Blynk nabízí poměrně úzkou škálu barev pro své widgety. Pomocí funkce `Blynk.setProperty()` je však možné v rámci kódu nastavit barvu libovolného widgetů pomocí jejího hexadecimálního (šestnáctkového) kódu[35]. Změna barvy je trvalá a zůstává v paměti aplikace, po nahrání je tedy možné tuto část kódu vymazat. V rámci aplikace vozíku byly nastaveny například tyto hodnoty:

```
Blynk.setProperty(V1, "color", "#F693F6");  // spínač ČÁRA
Blynk.setProperty(V2, "color", "#FFFF00");  // spínač TRASA
Blynk.setProperty(V3, "color", "#32ACAC");  // joystick
Blynk.setProperty(V5, "color", "#175e5e");  // RYCHLOST
```

[35] Jedná se o hodnoty v pořadí RGB uvedené v šestnáctkové soustavě (FF = 255).

Příloha 2 – Kompletní zdrojový kód

```
#define BLYNK_PRINT Serial
#include <ESP8266_Lib.h>
#include <BlynkSimpleShieldEsp8266.h>

// Údaje WiFi + spojení s aplikací Blynk
char auth[] = "o_T4omHT_8aTfQhJbNVNP9_lejhFS3oz";   // autorizační
       token z aplikace Blynkcas
char ssid[] = "MopikMopik";
char pass[] = "Moppy2021";
char server[] = "blynk-cloud.com";
int port = 8080;

WidgetTerminal terminal(V11);
#define EspSerial Serial1 // TX, RX 1 (18, 19), 2 (17, 18), 3 (14,
15) HW UART na Arduino Mega 2560
#define ESP8266_BAUD 115200 // baud rate našeho ESP8266-01
ESP8266 wifi(&EspSerial);
unsigned long cas_pripojeni;
unsigned long interval_jizdy;

// Piny Arduina
 struct podminka{  // hodnoty z pinů Arduina = podmínky pro přechod
do dalších stavů
    int pin;
    char pinA;
    bool hodnota;  };
    podminka levaleva, leva, prostredni, prava, pravaprava; // vstupy
senzoru čáry
    podminka narazvzad, priblizenivzad, vyskavzad, narazvpravo,
narazvlevo, priblizenivpred, vyskavpred, napeti, sklapnuti; //
kontrola bezpečnosti (vzdálenost a náraz)
    podminka RGBrLED, RGBgLED, RGBbLED, rLED, gLED, oLED, wLED, bLED;
// LED diody pod displayem vozíku
    bool faze;

// Podmínky automatů - virtuální piny Blynku
  struct podminkaBlynk{  // hodnoty z widgetů aplikace Blynk =
podmínky pro přechod do dalších stavů
    bool vstup;
    bool nacti;
    bool vystup;
    int analogove;
    int jizda; };
    podminkaBlynk  on, off, cara, trasa, trasa_slider, joystickx,
joysticky, rychlost, test, LEDvirtual21, LEDvirtual22, LEDvirtual23,
displej;
    int test_cyklus;

// Motory
  // Motor A (pravý)
  struct motor{
    int pin;
    int rychlost;
    bool hodnota;
    bool puvodni; };
    motor in1, in2, in3, in4, enA, enB;

  int korekceA = 1.19; // motorky se točí jinou rychlostí, je nutná
```

```
korekce
  int rychlost_motor;
  unsigned long cas_jizdy;

// 1. úroveň (zapni/vypni)
  // Stavy automatů
  bool SI;  bool SII;  bool SIII; bool SIV; bool SV; // z SIII
vychází další podúrovně

  unsigned long cas; // měření času při držení tlačítek ON/OFF
  unsigned long T;  // proměnná pro odpočet času (timer)

// 2. úroveň přepínání módů čára/trasa/joystick a kontrola
bezpečnosti
  // Přepínání módů
  bool S1;  bool S2;  bool S3;  bool S4;

  // Kontrola bezpečnosti
  bool S1k; bool S2k; bool S3k; bool S4k; bool S5k; bool S6k;
  int R;  // proměnná pro odpočet cyklů
  unsigned long cas_oLED;

  bool nebezpeci_vzadu;
  bool nebezpeci_vpredu;
  bool ve_vzduchu;

// 3. úroveň
  // Vstupy joysticku
  int koef;  // koeficient - procento vychýlení, které považujeme za
         významné
  int rozsah; // poloměr kruhu pohybu

void setup(){ // setup - nastavení počátečních hodnot
  Serial.begin(9600); // spuštění sériového monitoru
// Nastavení připojení k Wi-Fi pomocí ESP modulu
  EspSerial.begin(ESP8266_BAUD);
  delay(10);
  Blynk.config(wifi, auth, server, port);
  Serial.println("Připojení  nastaveno!");
  Blynk.connectWiFi(ssid, pass);
  Serial.println("WiFi");
  Blynk.connect();
  cas_pripojeni = millis() + 60000;  // při neúspěšném pokusu se ESP
po minutě znovu zkouší připojit (60 000 ms)
  Blynk.begin(auth, wifi, ssid, pass);

  terminal.clear();
  terminal.println(" Blynk verze " BLYNK_VERSION);
  terminal.flush();

// Stavy automatů a hodnoty podmínek
// 1. úroveň (zapni/vypni)
  // Stavy on/off
  SI = 1;
  SII = 0; SIII = 0; SIV = 0; SV = 0;

  // Výchozí hodnoty na virtuálních pinech Blynk on/off
  Blynk.virtualWrite(0, 0);   // tlačítko ON
  Blynk.virtualWrite(10, 1);  // tlačítko OFF

// 2. úroveň
```

```
  // Stavy - přepínání módů čára/trasa/joystick
  S1 = 0; // je rovno SIII, nastane po zapnutí virtuálního zapínacího
tlačítka
  S2 = 0;  S3 = 0;  S4 = 0;

  // Výchozí hodnoty na virtuálních pinech Blynk  čára/trasa/joystick
  Blynk.virtualWrite(1, 0);   // spínač ČÁRA
  Blynk.virtualWrite(2, 0);   // spínač TRASA
  Blynk.virtualWrite(6, 0);   // tlačítko trasa - výchozí je NIC
  Blynk.virtualWrite(3, 0);   // osa x JOYSTICK
  Blynk.virtualWrite(4, 0);   // osa y JOYSTICK
  Blynk.virtualWrite(5, 3);   // tlačítko RYCHLOST

  // Stavy - kontrola bezpečnosti
  S1k = 0; S2k = 0; S3k = 0; S4k = 0; S5k = 0; S6k = 0;
  R = 0;  // proměnná pro odpočet cyklů

  // Kontrola bezpečnosti
  Blynk.virtualWrite(7, 1);   // spínač TEST - zpočátku je testovací
režim zapnutý, aby nedocházelo k nechtěné kotnrole nárazů
  test_cyklus = 0;
  test.vstup = 1;
  test.vystup = 1;
  Blynk.setProperty(V11, "color", "#4848DA");  // terminál

  narazvzad.pinA = A1;
  priblizenivzad.pinA = A3;
  vyskavzad.pinA = A2;
  pinMode(narazvzad.pinA, INPUT);
  pinMode(priblizenivzad.pinA, INPUT);
  pinMode(vyskavzad.pinA, INPUT);

  narazvpravo.pinA = A4;
  narazvlevo.pinA = A5;
  priblizenivpred.pinA = A6;
  vyskavpred.pinA = A7;
  pinMode(narazvpravo.pinA, INPUT_PULLUP);
  pinMode(narazvlevo.pinA, INPUT_PULLUP);
  pinMode(priblizenivpred.pinA, INPUT);
  pinMode(vyskavpred.pinA, INPUT);

  sklapnuti.pinA = A0;
  pinMode(sklapnuti.pinA, INPUT);

//3. úroveň
// Jízda podle čáry
  levaleva.pin =  31;
  leva.pin =  33;
  prostredni.pin = 35;
  prava.pin =  37;
  pravaprava.pin = 39;

  pinMode(levaleva.pin, INPUT);
  pinMode(leva.pin, INPUT);
  pinMode(prostredni.pin, INPUT);
  pinMode(prava.pin, INPUT);
  pinMode(pravaprava.pin, INPUT);

// Jízda podle joysticku
rozsah = 5; // rozsah os joysticku
koef = 0.4 * rozsah; // číslo, od kterého považujeme vychýlení
```

```
joysticku za významné

// Nastavení motorů
   // Motor A (pravý)
    enA.pin = 7; in1.pin = 6; in2.pin = 5;
    korekceA = 1.19; // motorky se točí jinou rychlostí, je nutná
korekce
    // Motor B (levý)
    enB.pin = 8; in3.pin = 10; in4.pin = 9;
  // Motor A (pravý) - nastavení pinů na výstupní
    pinMode(enA.pin, OUTPUT);
    pinMode(in1.pin, OUTPUT);
    pinMode(in2.pin, OUTPUT);
  // Motor B (levý) - nastavení pinů na výstupní
    pinMode(enB.pin, OUTPUT);
    pinMode(in3.pin, OUTPUT);
    pinMode(in4.pin, OUTPUT);
  // Na začátku jsou motory vypnuté (LOW na enable pinech)
          a nastavené na pohyb vpřed
  // Motor A (pravý)
    enA.rychlost = LOW;
    in1.hodnota = LOW;  in1.puvodni = in1.hodnota;
    in2.hodnota = LOW;  in2.puvodni = in2.hodnota;
    digitalWrite(enA.pin, enA.hodnota);
    digitalWrite(in1.pin, in1.hodnota);
    digitalWrite(in2.pin, in2.hodnota);
  // Motor B (levý)
    enB.rychlost = LOW;
    in3.hodnota = LOW;  in3.puvodni = in3.hodnota;
    in4.hodnota = LOW;  in4.puvodni = in4.hodnota;
    digitalWrite(enB.pin, enB.hodnota);
    digitalWrite(in3.pin, in3.hodnota);
    digitalWrite(in4.pin, in3.hodnota);

  interval_jizdy = 500;

// Fyzické LED diody pod displayem vozíku
  RGBrLED.pin = 53;  RGBgLED.pin = 51;  RGBbLED.pin = 49; // RGB LED
  rLED.pin = 52;  // červená LED
  gLED.pin = 50;  // zelená LED
  oLED.pin = 48;  // oranžová (žlutá) LED
  wLED.pin = 47;  // bílá LED
  bLED.pin = 46;  // modrá LED

// Nastavení pinů fyzických LED na výstup
  pinMode(RGBrLED.pin, OUTPUT); pinMode(RGBgLED.pin, OUTPUT);
pinMode(RGBbLED.pin, OUTPUT);
  pinMode(rLED.pin, OUTPUT);
  pinMode(gLED.pin, OUTPUT);
  pinMode(oLED.pin, OUTPUT);
  pinMode(wLED.pin, OUTPUT);
  pinMode(bLED.pin, OUTPUT);

  RGBrLED.hodnota = 1; RGBgLED.hodnota = 0; RGBbLED.hodnota = 0;
       // na počátku všechyn LED krom R vypnuty
  faze = 0;
  gLED.hodnota = 0;
  oLED.hodnota = 0;
  wLED.hodnota = 0;
  bLED.hodnota = 0;
  rLED.hodnota = 1;  digitalWrite(rLED.pin, rLED.hodnota);
```

```
          // zapnutí fyzické červené LED

// Virtuální LED v Blynku kopírující fyzické LED - narozdíl od
fyzických LED stačí zapnout a pak jen přepínat barvu
  Blynk.setProperty(V21, "color", "#FF0000");  // barva virtuální LED
nastavena na R - červenou
  Blynk.virtualWrite(V21, 255);   // virtuální dioda bude svítit max.
intenzitou
  Blynk.setProperty(V21, "label", " SI ");  // název virtuální LED
nastaven na SI

  Blynk.setProperty(V22, "color", "#000000");  // barva virtuální LED
nastavena na K - černou
  Blynk.virtualWrite(V22, 255);   // virtuální dioda bude svítit max.
intenzitou
  Blynk.setProperty(V22, "label", " ");  // název virtuální LED
nastaven na nic

  Blynk.setProperty(V23, "color", "#000000");  // barva virtuální LED
nastavena na K - černou
  Blynk.virtualWrite(V23, 255);   // virtuální dioda bude svítit max.
intenzitou
  Blynk.setProperty(V23, "label", " ");  // název virtuální LED
nastaven na nic

} // konec setup

void loop(){    // začátek smyčky (loop), která se za běhu Arduina
opakuje
 urovenI();
} // konec smyčky (loop)

// Funkce 1. úrovně automatu
void urovenI(){
 spojeniBlynk(); // kontrola Wi-Fi připojení a komunikace s Blynkem
  obrazvstupuI();
    if (SI){
      onI();
      notoffI();  }
    if (SII){
     notonII();
     TII(); }
    if (SIII){
        urovendva();  // spouští se druhá úroveň
     offIII();
     notwifiIII();  }
    if (SIV){
     onIV();
     TIV(); }
    if (SV){
     wifiV(); }
  obrazvystupuI();  }

  //SI - vypnuto, čeká na podržení tlačítka ON
    void onI(){
       if(on.vstup){  Serial.println(" SII (odpočet tlačítka ON)
         "); // po stisku ON tlačítka přejdeme do druhého stavu
        terminal.println(" SII (odpočet tlačítka ON) ");
        Blynk.setProperty(V21, "label", "SII");  // název virtuální
             LED
        off.vystup = 0; // vypneme OFF tlačítko
```

```
        cas = millis(); // aktuální čas při stisku tlačítka
        T = 1000; // odpočet 1000 ms
        SI = 0; SII = 1; }}  // přesun do stavu SII

    void notoffI(){
      if(!off.vstup){// manuální vynutí OFF tlačítka
        off.vystup = 1;  }} // při manuálním vypnutí OFF ve stavu SI
             se tlačítko znovu sepne

  //SII - zapínání, kontroluje, zda tlačítko ON držíme alespoň 1 s
    void notonII(){
      if(!on.vstup){  Serial.print(" SI ");  // puštění tlačítka ON v
            časovém limitu 1s
        terminal.println(" SI ");
        Blynk.setProperty(V21, "label", "SI  ");  // název virtuální
            LED
        on.vystup = 0; // vypneme ON tlačítko
        off.vystup = 1;  // zapneme OFF tlačítko
        SII = 0; SI = 1; }} // návrat do stavu S1

    void TII(){
       if(on.vstup && !off.vstup){ // kontrola stisknutí  tlačítka ON
         if(abs((millis()-cas)) > T){ // stisk tlačítka delší než
               1000 ms - není přesné kvůli komunikaci s Blynkem
           Serial.println(" SIII, S1, S1k ");
           terminal.println(" SIII, S1, S1k ");
           Blynk.setProperty(V21, "label", "SIII");  // název
               virtuální LED
           Blynk.setProperty(V22, "color", "#FFFFFF");  // barva
               virtuální LED nastavena na W - bílou
           Blynk.setProperty(V22, "label", "S1");  // název virtuální
               LED nastaven na S1
           Blynk.setProperty(V23, "label", "S1k");  // název
               virtuální LED nastaven na S1k
           on.vystup = 1;  // tlačítko ON zůstane sepnuté
           Blynk.virtualWrite(V0, on.vystup);
           off.vystup = 0; // tlačítko OFF zůstane vypnuté
           Blynk.setProperty(V21, "color", "#FFFFFF");  // barva
               virtuální LED nastavena na W - bílou
           rLED.hodnota = 0;  wLED.hodnota = 1;  // vypnutí R a
               zapnutí W fyzické LED
           RGBrLED.hodnota = 0; RGBgLED.hodnota = 0;
               RGBbLED.hodnota = 0;
           SII = 0; SIII = 1; S1 = 1; S2k = 1;  }}}  // přechod do
               stavu SIII, vstup do druhé úrovně

  //SIII - ON zapnuto, vstup do dalších pdoúrovní, čeká na pokyn
    k vypnutí (stlačení OFF, kontrola připojení)
   void offIII(){
       if(off.vstup){ Serial.println(" SIV ");
        terminal.println(" SIV ");
        Blynk.setProperty(V21, "label", "SIV");  // název virtuální
            LED
        cas = millis(); // aktuální čas při stisku tlačítka
        on.vystup = 0; // vypneme ON tlačítko
        off.vystup = 1;  // zapneme OFF tlačítko
        T = 1000; // odpočet 1 s (1000 ms)
        stuj(); // zastav jízdu z druhé úrovně
        Blynk.setProperty(V22, "color", "#000000");  // barva
            virtuální LED nastavena na K - černou
        Blynk.setProperty(V22, "label", " ");  // název virtuální LED
```

```
            nastaven na nic
        Blynk.setProperty(V23, "label", " ");  // název virtuální LED
            nastaven na nic
        oLED.hodnota = 0;
        SIII = 0; // opuštění stavu SIII
        S1 = 0; S2 = 0; S3 = 0; S4 = 0; // vypnutí stavů z druhé
            úrovně - volba jízdy
        S1k = 0; S1k = 0; S2k = 0; S3k = 0; S4k = 0; S5k = 0;
        S6k = 0;  // vypnutí stavů z druhé úrovně - kontrola
            bezpečnosti
        SIV = 1;  }}  // vypnutí stavů z druhé úrovně - kontrola
            bezpečnosti, přechod do stavu SIV

  void notwifiIII(){
     if (Blynk.connected() == 0){  // pokud se přeruší Wi-Fi
            spojení, vše zastaví
        Serial.println(" SV ");
        stuj();
        wLED.hodnota = 0; gLED.hodnota = 1;  // vypnutí W a zapnutí G
            fyzické LED
        RGBrLED.hodnota = 0; RGBgLED.hodnota = 1; RGBbLED.hodnota =
            0;
        oLED.hodnota = 0;
        SIII = 0;
        S1 = 0; S2 = 0; S3 = 0; S4 = 0; // vypnutí stavů z druhé
             úrovně - volba jízdy
        S1k = 0; S1k = 0; S2k = 0; S3k = 0; S4k = 0; S5k = 0; S6k =
            0;  // vypnutí stavů z druhé úrovně - kontrola
         bezpečnosti
        SV = 1;
        Serial.println("Chyba připojení k Wi-Fi");
        Blynk.connectWiFi(ssid, pass); // pokusí se znovu připojit
        Blynk.connect();  }} // pokusí se zahájit komunikaci


//SIV - vypínání, kontroluje, zda tlačítko OFF držíme alespoň 1 s
   void onIV(){
      if(!off.vstup && on.vstup){  Serial.println(" SIII, S1,
            S1k ");
        terminal.println(" SIII, S1, S1k ");
        Blynk.setProperty(V21, "label", "SII");  // název virtuální
            LED
        Blynk.setProperty(V22, "label", "S1");  // název virtuální
            LED nastaven na S1
        Blynk.setProperty(V23, "label", "S1k");  // název virtuální
            LED nastaven na S1k
        on.vystup = 1;  // sepne se tlačítko ON
        SIV = 0; SIII = 1; S1 = 1; S2k = 1;  }} // přesun do stavu
            SIII, druhá úroveň

   void TIV(){
      if(off.vstup){
       if(abs((millis()-cas)) > T){  Serial.println(" SI ");  //
            stisk tlačítka delší než 1 s (1000 ms)
         terminal.println(" SI ");
         Blynk.setProperty(V21, "label", "SI");  // název virtuální
            LED
         Blynk.setProperty(V21, "color", "#FF0000");  // barva
            virtuální LED nastavena na R - červenou
         Blynk.virtualWrite(V10, off.vystup);
         cara.vystup = 0; trasa.vystup = 0; test.vystup = 1;
```

```
test_cyklus = 0;
          wLED.hodnota = 0; rLED.hodnota = 1;  // vypnutí W a zapnutí
               R fyzické LED
          rLED.hodnota = 1; RGBgLED.hodnota = 0; RGBbLED.hodnota = 0;
          SIV = 0; SI = 1; }}}

  //SV - kontrola opětovného spojení po jeho ztrátě, návrat do SIII
  void wifiV(){
    spojeniBlynk(); // pokusí se znovu připojit
    if (Blynk.connected() == 1){  // pokud se spojení obnoví
      Serial.println(" SIII, Wi-Fi opětovně připojena. ");
      terminal.println(" SIII, Wi-Fi opětovně připojena. ");
      Blynk.setProperty(V21, "color", "#FFFFFF");  // barva virtuální
          LED nastavena na W - bílou
      gLED.hodnota = 0; wLED.hodnota = 1;  // vypnutí G a zapnutí W
          fyzické LED
      rLED.hodnota = 0; RGBgLED.hodnota = 0; RGBbLED.hodnota =
          0;
      SIII = 1; SV = 0; }}

  // Další funkce první úrovně
    void obrazvstupuI(){
     on.vstup = on.nacti; // ON tlačítko
     off.vstup = off.nacti;  } // OFF tlačítko, konec obrazu vstupů

    void obrazvystupuI(){
     digitalWrite(RGBrLED.pin, RGBrLED.hodnota);
digitalWrite(RGBgLED.pin, RGBgLED.hodnota); digitalWrite(RGBbLED.pin,
RGBbLED.hodnota); // RGB LED
     digitalWrite(rLED.pin, rLED.hodnota);  // červená LED
     digitalWrite(wLED.pin, wLED.hodnota);  // bílá LED
     digitalWrite(gLED.pin, gLED.hodnota);  // zelená LED
     digitalWrite(oLED.pin, oLED.hodnota);  // oranžová LED
     digitalWrite(bLED.pin, wLED.hodnota);  // modrá LED
          } // konec obrazu výstupů

  // Získávání hodnot z aplikace Blynk pro první úroveň
    BLYNK_WRITE (V0) { on.nacti = param.asInt();          } // ON
    BLYNK_READ  (V0) { Blynk.virtualWrite(V0, on.vystup); } // ON
    BLYNK_WRITE (V10){ off.nacti = param.asInt();         }
        // OFF
    BLYNK_READ  (V10){ Blynk.virtualWrite(V10, off.vystup);} // OFF

// Schéma 2. úrovně
void urovendva(){ // Funkce (akce při splnění podmínky) 2. úrovně
 urci_jizdu();
 // kontroluj_bezpecnost();
 } // konec 2. úrovně (smyčky)

void urci_jizdu(){  // funkce pokrývající pohyb mezi stavy S1 až S4
 obrazvstupu2();
 if(S1){  stuj();  // čeká na pokyn k jízdě, pojistka zastavení
   cara1();
   trasa1();
   joystick1(); }
 if(S2){  // jede podle čáry
   cara2();
   joystick2();
   nejednoznacne2();  }
 if(S3){  // jede po trase
   trasa3();
```

```
   joystick3();
   nejednoznacne3(); }
 if(S4){  // jede podle joysticku
   joystick4();
   cara4();
   trasa4();  }} // konec urci_jizdu

 // S1 - čeká na pokyn k jízdě
 void cara1(){
  if(cara.vstup && !trasa.vstup && !joystickx.vstup
     && !joysticky.vstup){
    Serial.println(" S2 (čára): ");
    terminal.println(" S2 (čára): ");
    Blynk.setProperty(V22, "label", "ČÁR");
    Blynk.setProperty(V22, "color", "#FF00FF");  // barva virtuální
         LED nastavena na M - purpurová
    S1 = 0; S2 = 1; }} // jede podle čáry

 void trasa1(){
   if(trasa.vstup && !cara.vstup && !joystickx.vstup
     && !joysticky.vstup){
    Serial.println(" S3 (trasa): ");
   terminal.println(" S3 (trasa): ");
     Blynk.setProperty(V22, "label", "TRS");
     Blynk.setProperty(V22, "color", "#FFFF00");  // barva virtuální
         LED nastavena na Y - žlutá
     S1 = 0; S3 = 1; }}  // jede po trase

 void joystick1(){
   if((joystickx.vstup || joysticky.vstup) && !trasa.vstup
    && !cara.vstup){
    Serial.println(" S4 (joystick): ");
    terminal.println(" S4 (joystick): ");
     Blynk.setProperty(V22, "label", "JST");
     Blynk.setProperty(V22, "color", "#00FFFF");  // barva virtuální
         LED nastavena na C - azurová
     S1 = 0; S4 = 1;  }} // jede podle joysticku

 // S2 - jede podle čáry
 void cara2(){
  if(cara.vstup && !trasa.vstup && !joystickx.vstup &&
!joysticky.vstup){
    jed_cara(); }}

 void joystick2(){
   if(joystickx.vstup || joysticky.vstup){
    Serial.println(" S4 (joystick): ");
    terminal.println(" S4 (joystick): ");
     Blynk.setProperty(V22, "label", "JST");
     Blynk.setProperty(V22, "color", "#00FFFF");  // barva virtuální
          LED nastavena na C - azurová
     S2 = 0; S4 = 1;  }} // jede podle joysticku

 void nejednoznacne2(){
   if (!cara.vstup || trasa.vstup && cara.vstup){
    Serial.println(" S1 (čeká na pokyn): ");
    terminal.println(" S1 (čeká na pokyn): ");
     Blynk.setProperty(V22, "label", "S1");
     Blynk.setProperty(V22, "color", "#FFFFFF");  // barva virtuální
         LED nastavena na W - bílou
     cara.vystup = 0; trasa.vystup = 0; // vypnutí spínačů ČÁRA
```

```
      a TRASA
     Blynk.virtualWrite(1, cara.vystup);
     Blynk.virtualWrite(2, trasa.vystup);
     stuj();
     S2 = 0; S1 = 1;  }} // zastavení a návrat do stavu S1

// S3 - jede po trase
 void trasa3(){
  if(trasa.vstup && !cara.vstup && !joystickx.vstup
      && !joysticky.vstup){
    jed_trasa();  }}

 void joystick3(){
  if(joystickx.vstup || joysticky.vstup){
    Serial.println(" S4 (joystick): ");
    terminal.println(" S4 (joystick): ");
    Blynk.setProperty(V22, "label", "JST");
    Blynk.setProperty(V22, "color", "#00FFFF");  // barva virtuální ¨
        LED nastavena na C - azurová
    S2 = 0; S4 = 1;  }} // jede podle joysticku

 void nejednoznacne3(){
   if (!trasa.vstup || trasa.vstup && cara.vstup){
    Serial.println(" S1 (čeká na pokyn): ");
    terminal.println(" S1 (čeká na pokyn): ");
     Blynk.setProperty(V22, "label", "S1");
     Blynk.setProperty(V22, "color", "#FFFFFF");  // barva virtuální
         LED nastavena na W - bílou
     cara.vystup = 0; trasa.vystup = 0; // vypnutí spínačů ČÁRA
         a TRASA
     Blynk.virtualWrite(1, cara.vystup);
     Blynk.virtualWrite(2, trasa.vystup);
     stuj();
     S3 = 0; S1 = 1;  }} // zastavení a návrat do stavu S1

// S4
  void joystick4(){
   if(!trasa.vstup && !cara.vstup){
      jed_joystick(); }}

 void cara4(){
  if(cara.vstup && !trasa.vstup && !joystickx.vstup
    && !joysticky.vstup){
    Serial.println(" S2 (čára): ");
    terminal.println(" S2 (čára): ");
    Blynk.setProperty(V22, "label", "ČÁR");
    Blynk.setProperty(V22, "color", "#FF00FF");  // barva virtuální
      LED nastavena na M - purpurová
    S4 = 0; S2 = 1; }}

 void trasa4(){
  if(trasa.vstup && !cara.vstup && !joystickx.vstup &&
!joysticky.vstup){
    Serial.println(" S3 (trasa): ");
    terminal.println(" S3 (trasa): ");
    Blynk.setProperty(V22, "label", "TRS");
    Blynk.setProperty(V22, "color", "#FFFF00");  // barva virtuální
         LED nastavena na Y - žlutá
    S4 = 0; S3 = 1;  }}
```

```
// Obraz vstupů pro učení jízdy
void obrazvstupu2(){
//  Blynk.syncVirtual(1); Blynk.syncVirtual(2);
 // Blynk.syncVirtual(3);
  cara.vstup = cara.nacti;
  trasa.vstup = trasa.nacti;
  joystickx.vstup = joystickx.nacti;
  joysticky.vstup = joysticky.nacti;
  if(S4){
  joystickx.jizda = joystickx.analogove;
  joysticky.jizda = joysticky.analogove; }}

// Získávání hodnot z aplikace Blynk pro druhou úroveň - volbu způsbu
        jízdy
    BLYNK_WRITE(V1) { // Vyčítání ČÁRA
      if(SIII){cara.nacti = param.asInt(); }
      else    {cara.nacti = 0;  }}
    BLYNK_READ (V1) { Blynk.virtualWrite(V1, cara.vystup); } // ČÁRA
    BLYNK_WRITE(V2) { // Vyčítání TRASA
      if(SIII){trasa.nacti = param.asInt(); }
      else    {trasa.nacti = 0;  }}
    BLYNK_READ (V2) { Blynk.virtualWrite(V2, trasa.vystup); }
       // TRASA
    BLYNK_WRITE(V6) { // Vyčítání TRASA
      if(SIII){trasa_slider.analogove = param.asInt();  }
      else    {trasa_slider.analogove = 0;  }}
    BLYNK_WRITE(V3) { // Vyčítání JOYSTICK - osa x
      if(SIII){ joystickx.nacti = param[0].asInt();
      joysticky.nacti = param[1].asInt();
      if(S4){joystickx.analogove = param[0].asInt();
      joysticky.analogove = param[1].asInt(); }}}

void kontroluj_bezpecnost(){
  obrazvstupu2k();
  if (S1k){ // testovací režim
    test1k();
    nebezpeci_vzadu1k();
    nebezpeci_vpredu1k();
    ve_vzduchu1k(); }
  if (S2k){ // výchozí stav kontroly bezpečnosti
    nottest2k(); }
  if (S3k){ // nebezpečí vzadu
    notnebezpeci_vzadu3k();
    nebezpeci_vzadu3k();
    R03k(); }
  if (S4k){  // nebezpečí vepředu
    notnebezpeci_vpredu4k();
    nebezpeci_vpredu4k();
    R04k(); }
  if (S5k){  // vozík zvednut do vzduchu
    notve_vzduchu5k();  }
  if (S6k){  // vozík někde ohrožen a neúspěšně se snažil vymanit
    not_nebezpeci6k();  }
    }  // konec kontroluj_bezpecnost

// Akce 2. úrovně - kontroly bezpečnosti
void test1k(){
  if (test_cyklus == 1 || test.vstup){
    Serial.println(" S2k, testovací režim! ");
    terminal.println(" S2k, testovací režim! ");
  Blynk.setProperty(V23, "color", "#0000FF");  // barva virtuální LED
```

```
        nastavena na B - modrou
  float napeti_test = 0;
  napeti.pinA = A8;
  pinMode(napeti.pinA, INPUT);
  analogReference(INTERNAL2V56);  // referenční napětí pro měření
         je 2,56 V
    for (int i = 0; i < 5; i++){
    napeti_test = napeti_test + analogRead(napeti.pinA);  // měření
         pěti hodnot napětí }
    Serial.print(" Test napětí - pět hodnot: ");
    Serial.print(napeti_test);}
    napeti_test = napeti_test/5;  // průměr měření napětí
    Serial.print(" Test napětí - průměr: ");
    Serial.print(napeti_test);
    napeti_test = map(napeti_test, 0.00, 1000.00, 0.00, 14.40);
          // převedení napětí
    Serial.print(" Napětí baterie je: ");
    Serial.println(napeti_test);
  analogReference(DEFAULT);
  bLED.hodnota = 1; // modrá LED svítí
  test_cyklus++;
  S1k = 0;  S2k = 1;  }}

void nebezpeci_vzadu1k(){
  if (test_cyklus != 1 && nebezpeci_vzadu){
    Serial.println(" S3k, AU vzadu! ");
    terminal.println(" S3k, AU vzadu! ");
  stuj();
  cara.vystup = 0; trasa.vystup = 0;
  S2 = 0; S3 = 0; S4 = 0; S1 = 1; // vypnutí jízdy
  Blynk.setProperty(V23, "color", "#EF8D22");  // barva virtuální
       LED nastavena na oranžovou
  oLED.hodnota = 1; // rozsvícení oranžové (žluté) LED
  R = 5;  // odpočet pěti cyklů
  jedvpred();  jedvpravo();  stuj();
  S1k = 0;  S3k = 1;  }}

void nebezpeci_vpredu1k(){
  if (test_cyklus != 1 && nebezpeci_vpredu){
    Serial.println(" S4k, AU vepředu! ");
    terminal.print(" S4k, AU vepředu! ");
  stuj();
  cara.vystup = 0; trasa.vystup = 0;
  S2 = 0; S3 = 0; S4 = 0; S1 = 1; // vypnutí jízdy
  Blynk.setProperty(V23, "color", "#EF8D22");  // barva virtuální
       LED nastavena na oranžovou
  oLED.hodnota = 1; // rozsvícení oranžové (žluté) LED
  R = 5;  // odpočet pěti cyklů
  jedvzad();  jedvpravovzad();  stuj();
  S1k = 0;  S4k = 1;  }}

void ve_vzduchu1k(){
  if (test_cyklus != 1 && ve_vzduchu){
    Serial.println(" S5k, letím! ");
    terminal.print(" S5k, letím! ");
  stuj();
  cara.vystup = 0; trasa.vystup = 0;
  S2 = 0; S3 = 0; S4 = 0; S1 = 1; // vypnutí jízdy
  Blynk.setProperty(V23, "color", "#EF8D22");  // barva virtuální
       LED nastavena na oranžovou
  oLED.hodnota = 1; // rozsvícení oranžové (žluté) LED
```

```
  S1k = 0;  S5k = 1;  }}

void nottest2k(){
  if (test_cyklus != 1 && !test.vstup){
    Serial.println(" S1k, konec testovacího režimu! ");
    terminal.println(" S1k, konec testovacího režimu! ");
  oLED.hodnota = 0; // zhasnutí oranžové (žluté) LED
  S1k = 1;  S2k = 0;  }}

void notnebezpeci_vzadu3k(){
  if (!nebezpeci_vzadu){
    Serial.println(" S1k, nebezpečí vzadu pominulo! ");
    terminal.println(" S1k, nebezpečí vzadu pominulo! ");
  stuj();
  Blynk.setProperty(V8, "color", "#000000");  // barva virtuální
       LED nastavena na K
  oLED.hodnota = 0; // zhasnutí oranžové (žluté) LED
  S3k = 0;  S1k = 1;  }}

void nebezpeci_vzadu3k(){
  if (nebezpeci_vzadu && R > 0){
    Serial.print(" S3k, AU vzadu (znovu)! R = ");
    terminal.print(" S3k, AU vzadu (znovu)! R = ");
  Serial.println(R); terminal.println(R);
  R--;  // odpočet pěti cyklů
  jedvpred();  jedvpred(); jedvpred();
  jedvpravo(); jedvpravo(); stuj();  }}

void R03k(){
   if(R == 0){
    Serial.println(" S6k, čekám na vnější pomoc! ");
    terminal.println(" S6k, čekám na vnější pomoc! ");
   stuj();
   cara.vystup = 0; trasa.vystup = 0;
   S3k = 0; S6k = 1;
   cas_oLED = millis();
   oLED_blikej(2000); }}

void notnebezpeci_vpredu4k(){
   if (!nebezpeci_vpredu){
    Serial.println(" S1k, nebezpečí vepředu pominulo! ");
    terminal.print(" S1k, nebezpečí vepředu pominulo! ");
   stuj();
   Blynk.setProperty(V23, "color", "#000000");  // barva virtuální
          LED nastavena na K
   oLED.hodnota = 0; // zhasnutí oranžové (žluté) LED
   S4k = 0;  S1k = 1;  }}

void nebezpeci_vpredu4k(){
  if (nebezpeci_vpredu && R > 0){
    Serial.print(" S4k, AU vepředu (znovu)! R = ");
    terminal.print(" S4k, AU vepředu (znovu)! R = ");
  Serial.println(R); terminal.println(R);
  R--;  // odpočet pěti cyklů
  jedvzad();  jedvzad();  jedvzad();
  jedvpravovzad();  jedvpravovzad();  stuj(); }}

void R04k(){
   if(R == 0){
    Serial.print(" S6k, čekám na vnější pomoc! ");
    terminal.print(" S6k, čekám na vnější pomoc! ");
```

```
    stuj();
    cara.vystup = 0; trasa.vystup = 0;
    S4k = 0; S6k = 1;
    cas_oLED = millis();
    oLED_blikej(2000); }}

void notve_vzduchu5k(){
    if (!ve_vzduchu){
     Serial.println(" S1k, vozík je na zemi! ");
     terminal.print(" S1k, vozík je na zemi! ");
    stuj();
    Blynk.setProperty(V23, "color", "#000000");  // barva virtuální
       LED nastavena na K
    oLED.hodnota = 0; // zhasnutí oranžové (žluté) LED
    S5k = 0;  S1k = 1;  }}

void not_nebezpeci6k(){
  if (!nebezpeci_vzadu && !nebezpeci_vpredu){
     Serial.println(" S1k, vozík opět připraven k jízdě! ");
     terminal.print(" S1k, vozík opět připraven k jízdě! ");
    stuj();
    Blynk.setProperty(V23, "color", "#000000");  // barva virtuální
        LED nastavena na bílou
    oLED.hodnota = 0; // zhasnutí oranžové (žluté) LED
    S6k = 0;  S1k = 1;  }}

void oLED_blikej(unsigned long interval){ // funkce pro blikání
       oranžové LED
  bool zhasni = 1;
  if(S6k){
     if(abs((millis()-cas_oLED)) > interval  && zhasni){
     Blynk.setProperty(V23, "color", "#FFFFFF");  // barva virtuální
         LED nastavena na bílou
     oLED.hodnota = 0; // zhasnutí oranžové (žluté) LED
     zhasni = 0;
     cas_oLED = millis(); }
     if(abs((millis()-cas_oLED)) > interval && !zhasni){
     Blynk.setProperty(V23, "color", "#EF8D22");  // barva virtuální
         LED nastavena na oranžovou
     oLED.hodnota = 1; // rozsvícení oranžové (žluté) LED
     zhasni = 1;
     cas_oLED = millis(); }}}

// Načítání a posílání hodnot kontrolní úrovně
void obrazvstupu2k(){
  if(!test.vstup || test_cyklus > 2){
  narazvzad.hodnota =
digitalRead(narazvzad.pinA);  priblizenivzad.hodnota =
digitalRead(priblizenivzad.pinA);  vyskavzad.hodnota =
digitalRead(vyskavzad.pinA);
  nebezpeci_vzadu = narazvzad.hodnota || priblizenivzad.hodnota ||
vyskavzad.hodnota; // hrozí nebo nastal náraz vepřed
  narazvpravo.hodnota =
!digitalRead(narazvpravo.pinA);  narazvlevo.hodnota =
!digitalRead(narazvlevo.pinA);
  priblizenivpred.hodnota =
!digitalRead(priblizenivpred.pinA);  vyskavpred.hodnota =
digitalRead(vyskavpred.pinA);
  nebezpeci_vpredu = narazvpravo.hodnota || narazvlevo.hodnota ||
priblizenivpred.hodnota || vyskavpred.hodnota;  // hrozí nebo nastal
náraz vzadu
```

```
  sklapnuti.hodnota = digitalRead(sklapnuti.pinA);
  ve_vzduchu = sklapnuti.hodnota; }

    if(nebezpeci_vzadu){
     Serial.print(" Náraz vzad: ");
     terminal.print(" Náraz vzad: ");  }
    else if(nebezpeci_vpredu){
     Serial.print(" Náraz vpravo: ");
     terminal.print(" Náraz vpravo: ");  }
    else if(ve_vzduchu){
     Serial.print(" Ve vzduchu: ");
     terminal.print(" Ve vzduchu: ");  }} // konec obrazů vstupů
kontrolní úrovně

BLYNK_WRITE(V7) {// Vyčítání TESTOVACÍ MÓD
    test.nacti = param.asInt();  }
BLYNK_READ (V7) {if(S1k){ Blynk.virtualWrite(V7, test.vystup);  }}

//3. úroveň - samotné funkce určijící směr pohybu
  // Jízda podle čáry
  void jed_cara(){

    // Načtení hodnot ze senzoru
     levaleva.hodnota = !digitalRead(levaleva.pin);
     leva.hodnota = !digitalRead(leva.pin);
     prostredni.hodnota = !digitalRead(prostredni.pin);
     prava.hodnota = !digitalRead(prava.pin);
     pravaprava.hodnota =
!digitalRead(pravaprava.pin);

// Vypnutý senzor (vše je "HIGH")
/* 0 */   if(levaleva.hodnota  && leva.hodnota && prostredni.hodnota
&& prava.hodnota && pravaprava.hodnota){
   // všechny (senzor není zapnut, režim úprav kódu)
   terminal.println( " Čára, podmínka 0! " );
   stuj();  } // jede rovně
// 1 senzor
/* 1 */  else if((!leva.hodnota) && prostredni.hodnota &&
!prava.hodnota){ // jen prostřední
   terminal.println( " Čára, podmínka 1! " );
   jedvzad();  } // jede rovně
/* 2 */ else if(!leva.hodnota && (!prostredni.hodnota) &&
prava.hodnota){ // jen  pravá (prostřední)
   terminal.println( " Čára, podmínka 2! " );
   jedmirnevlevovzad(); } // jede mírně vpravo
/* 3 */  else if(!levaleva.hodnota  && !leva.hodnota &&
!prostredni.hodnota && !prava.hodnota && pravaprava.hodnota){
   // jen pravápravá (krajní)
   terminal.println( " Čára, podmínka 3! " );
   jedvlevovzad();  jedvlevovzad(); ;} // jede vpravo
/* 4 */  else if(leva.hodnota && !prostredni.hodnota &&
!prava.hodnota){ // jen levá (prostřední)
   terminal.println( " Čára, podmínka 4! " );
   jedmirnevpravovzad();}  // jede vlevo
/* 5 */  else if(levaleva.hodnota  && !leva.hodnota &&
!prostredni.hodnota && !prava.hodnota && !pravaprava.hodnota){
   // jen leválevá (krajní)
   terminal.println( " Čára, podmínka 5! " );
   jedvpravovzad(); jedvpravovzad(); } // jede doleva
// 2 senzory
/* 6 */  else if(leva.hodnota && !prostredni.hodnota &&
```

```
prava.hodnota){  // levá a pravá (a uprostřed nic)
   terminal.println( " Čára, podmínka 6! " );
   jedvlevovzad(); }  // jede vpravo
/* 7 */  else if(!leva.hodnota && prostredni.hodnota
    && prava.hodnota){ // prostřední a pravá (prostřední)
   terminal.println( " Čára, podmínka 7! " );
   jedmirnevlevovzad(); }  // jede mírně vpravo
/* 8 */  else if(leva.hodnota && prostredni.hodnota
     && !prava.hodnota){ // prostřední a levá (prostřední)
   terminal.println( " Čára, podmínka 8! " );
   jedmirnevpravovzad();} // jede mírně vlevo
// 3 senzory
/* 9 */  else if(leva.hodnota && prostredni.hodnota
    && prava.hodnota){ // levá a prostřední a pravá
   terminal.println( " Čára, podmínka 9! " );
   jedvzad();  }  // jede rovně
/* 10 */  else if((!levaleva.hodnota || !leva.hodnota)
    && !prostredni.hodnota && (!prava.hodnota ||
!pravaprava.hodnota)){
   terminal.println( " Čára, podmínka 10! " );
   jedvzad();  }  // jede rovně
/* 11 */  else  {
  terminal.println( " Čára, podmínka 11! " );
  stuj(); } // stojí
} // konec jízdy podle čáry

// Jízda po předem dané trase
  void jed_trasa(){
  Blynk.syncVirtual(2);
    if (trasa_slider.analogove == 1){
       jed_trasa_srdce(); }
    else if (trasa_slider.analogove == 2){
      jed_trasa_kruh(); }
    else if (trasa_slider.analogove == 3){
      jed_trasa_ctverec();  }
    else {Serial.println(" Čekám na pokyn o trase! ");
    terminal.println(" Čekám na pokyn o trase! ");  }}
        // konec  určení tvaru trasy

 // Funkce popisující, jak má vozík je pro dané tvary
    void jed_trasa_srdce(){ Serial.println(" Jedu podle tvaru
            srdce. ");
              terminal.println(" Jedu podle tvaru srdce. ");
      for(int i = 0; i < 10; i++){  jedvpred(); }
        jedvlevo(); jedvpred();
      for(int j = 0; j < 6; j ++){
        jedvlevo(); }
      for(int k = 0; k < 2; k++){
        jedvpravo(); }
      for(int l = 0; l < 3; l++){
        jedvpred(); }
      for(int m = 0; m < 6; m++){
        jedvlevo(); }
      for(int n = 0; n < 5; n++){
        jedvpred(); }
      stuj();
      trasa.vystup = 0;
      Blynk.virtualWrite(6, 0); Blynk.syncVirtual(6);
         // nastavení posuvky s volbou trasy na nulu
      Blynk.setProperty(V8, "color", "#FFFFFF");  }
        // barva virtuální LED nastavena na bílou
```

```
    void jed_trasa_kruh(){  Serial.println(" Jedu kolem tvaru
         kruhu. ");
    terminal.println(" Jedu kolem tvaru kruhu. ");
      for(int i = 0; i < 34; i++){
          jedvpred();  jedvpravo(); }
      stuj();
      trasa.vystup = 0;
      Blynk.virtualWrite(6, 0); Blynk.syncVirtual(6); // nastavení
posuvky s volbou trasy na nulu
      Blynk.setProperty(V8, "color", "#FFFFFF");  }  // barva
virtuální LED nastavena na bílou

    void jed_trasa_ctverec(){  Serial.println(" Jedu podle tvaru
čtverce. ");
    terminal.println(" Jedu podle tvaru čtverce. ");
      for(int i = 0; i < 4; i++){
        for(int j = 0; j < 8; j++){
          jedvpred(); }
        for(int k = 0; k < 3; k++){
          jedvpravo(); }}
      stuj();
      trasa.vystup = 0;
      Blynk.virtualWrite(6, 0); Blynk.syncVirtual(6);
         // nastavení posuvky s volbou trasy na nulu
      Blynk.setProperty(V8, "color", "#FFFFFF");  }
        // barva virtuální LED nastavena na bílou

// Ovládání joystickem
void jed_joystick(){ // Spouští se a začíná stav Sj1
  if(S4){
  urci_smer_jst(); // ovládání jízdy joysticku
  Serial.print(" Hodnota joysticku, osa x: ");
  Serial.print(joystickx.jizda);
  Serial.print(" osa y: ");
  Serial.print(joysticky.jizda);
  Serial.print(" Rychlost PWM  ");
  Serial.print(rychlost_motor); }
  }  // konec jedjoystick

void urci_smer_jst(){
  if (-koef < joystickx.jizda && joystickx.jizda < koef
      && -koef < joysticky.jizda && joysticky.jizda < koef){
     stuj();    } // stojí
  if (-koef < joystickx.jizda && joystickx.jizda < koef
     && joysticky.jizda >= koef){
     jedvpred();    } // jede vpřed (rovně)
  if (koef <= joystickx.jizda && joysticky.jizda >= 0){
     jedvpravo();    } // jede (vpřed) vpravo
  if (-koef >= joystickx.jizda && joysticky.jizda >= 0){
     jedvlevo();    } // jede (vpřed) vlevo
  if (-koef < joystickx.jizda && joystickx.jizda < koef
     && joysticky.jizda <= -koef){
     jedvzad();    } // jede vzad (couvá rovně)
  if (koef <= joystickx.jizda && joysticky.jizda < 0){
     jedvpravovzad();    } // jede (vzad) vpravo (couvá vpravo)
  if (-koef >= joystickx.jizda && joysticky.jizda < 0){
     jedvlevovzad();    } // jede (vzad) vlevo (couvá vlevo)
  else{stuj();    }} // stojí, kdyby nastala nějaká chyba;
       konec urcismer_jst
```

```
// Funkce ovládající motorky vozíku - směr jízdy (stojí, rovně vpřed,
rovně vzad, dopředu vpravo/vlevo, dozadu vpravo/vlevo)
void stuj(){ Serial.println(" Stojím! ");
 terminal.println(" Stojím! ");
  // Motor A (pravý)
  enA.rychlost = LOW;
  in1.hodnota = LOW;  in2.hodnota = LOW;
  // Motor B (levý)
  enB.rychlost = LOW;
  in3.hodnota = LOW;  in4.hodnota = LOW;
    jed();  }

void jedvpred(){
  if(S3 || S4){  Serial.println(" Jedu vpřed! ");
  terminal.println(" Jedu vpřed! "); }  // při jízdě po trase a podle
      jst
  if(S2){  Serial.println(" Jedu vzad! ");
  terminal.println(" Jedu vzad! "); } // při jízdě podle čáry
  urci_rychlost();
  // Motor A (pravý)
  enA.rychlost = rychlost_motor*korekceA;
  in1.hodnota = HIGH;  in2.hodnota = LOW;
  // Motor B (levý)
  enB.rychlost = rychlost_motor;
  in3.hodnota = HIGH;  in4.hodnota = LOW;
    jed();  }

void jedvzad(){
  if(S3 || S4){  Serial.println(" Jedu vzad! ");
  terminal.println(" Jedu vzad! "); }  // při jízdě po trase
       a podle jst
  if(S2){  Serial.println(" Jedu vpřed ");
  terminal.println(" Jedu vpřed "); } // při jízdě podle čáry
       (otočení směrů)
  urci_rychlost();
  // Motor A (pravý)
  enA.rychlost = rychlost_motor*korekceA;
  in1.hodnota = LOW;  in2.hodnota = HIGH;
  // Motor B (levý)
  enB.rychlost = rychlost_motor;
  in3.hodnota = LOW;  in4.hodnota = HIGH;
    jed();  }

void jedvpravo(){
  if(S3 || S4){  Serial.println(" Jedu vpravo! ");
   terminal.println(" Jedu vpravo! "); }  // při jízdě po trase
        a podle jst
  if(S2){  Serial.println(" Jedu vlevo vzad (couvám doleva)! ");
  terminal.println(" Jedu vlevo vzad (couvám doleva)! "); }
   // při jízdě podle čáry (otočení směrů)
  urci_rychlost();
  // Motor A (pravý)
  enA.rychlost = LOW;
  in1.hodnota = HIGH;  in2.hodnota = LOW;
  // Motor B (levý)
  enB.rychlost = rychlost_motor;
  in3.hodnota = HIGH;  in4.hodnota = LOW;
    jed();  }

void jedvlevo(){
  if(S3 || S4){  Serial.println(" Jedu vlevo! ");
```

```
  terminal.println(" Jedu vlevo! "); }  // při jízdě po trase
       a podle jst
  if(S2){  Serial.println(" Jedu vpravo vzad (couvám doprava)! ");
  terminal.println(" Jedu vpravo vzad (couvám doprava)! "); }
      // při jízdě podle čáry (otočení směrů)
  urci_rychlost();
  // Motor A (pravý)
  enA.rychlost = rychlost_motor*korekceA;
  in1.hodnota = HIGH;  in2.hodnota = LOW;
  // Motor B (levý)
  enB.rychlost = LOW;
  in3.hodnota = HIGH;  in4.hodnota = LOW;
    jed();  }

void jedvpravovzad(){
  if(S3 || S4){  Serial.println(" Jedu vpravo vzad (couvám
  doprava)! ");
  terminal.println(" Jedu vpravo vzad (couvám doprava)! "); }
     // při jízdě po trase a podle jst
  if(S2){  Serial.println(" Jedu vlevo ");
  terminal.println(" Jedu vlevo "); } // při jízdě podle čáry
     (otočení směrů)
  urci_rychlost();
  // Motor A (pravý)
  enA.rychlost = LOW;
  in1.hodnota = LOW;  in2.hodnota = HIGH;
  // Motor B (levý)
  enB.rychlost = rychlost_motor*korekceA;
  in3.hodnota = LOW;  in4.hodnota = HIGH;
    jed();  }

void jedmirnevpravovzad(){
  if(S3 || S4){  Serial.println(" Jedu vpravo vzad
   (couvám doprava)! ");
   terminal.println(" Jedu vpravo vzad (couvám doprava)! "); }
      // při jízdě po trase a podle jst
  if(S2){  Serial.println(" Jedu vlevo ");
  terminal.println(" Jedu vlevo "); } // při jízdě podle čáry
     (otočení směrů)
  urci_rychlost();
  // Motor A (pravý)
  enA.rychlost = rychlost_motor*0.7;
  in1.hodnota = LOW;  in2.hodnota = HIGH;
  // Motor B (levý)
  enB.rychlost = rychlost_motor*korekceA;
  in3.hodnota = LOW;  in4.hodnota = HIGH;
    jed();  }

void jedvlevovzad(){
  if(S3 || S4){  Serial.println(" Jedu vlevo vzad
     (couvám doleva)! ");
  terminal.println(" Jedu vlevo vzad (couvám doleva)! "); }  // při
      jízdě po trase a podle jst
  if(S2){  Serial.println(" Jedu vpravo ");
  terminal.println(" Jedu vpravo "); } // při jízdě podle čáry
     (otočení směrů)
  urci_rychlost();
  // Motor A (pravý)
  enA.rychlost = rychlost_motor;
  in1.hodnota = LOW;  in2.hodnota = HIGH;
  // Motor B (levý)
```

```
  enB.rychlost = LOW;
  in3.hodnota = LOW;  in4.hodnota = HIGH;
    jed();  }

void jedmirnevlevovzad(){
  if(S3 || S4){  Serial.println(" Jedu vlevo vzad
  (couvám doleva)! ");
   terminal.println(" Jedu vlevo vzad (couvám doleva)! "); }
   // při jízdě po trase a podle jst
  if(S2){  Serial.println(" Jedu vpravo "); terminal.println(" Jedu
   vpravo "); } // při jízdě podle čáry (otočení směrů)
  urci_rychlost();
  // Motor A (pravý)
  enA.rychlost = rychlost_motor;
  in1.hodnota = LOW;  in2.hodnota = HIGH;
  // Motor B (levý)
  enB.rychlost = rychlost_motor*0.7*korekceA;
  in3.hodnota = LOW;  in4.hodnota = HIGH;
    jed();  }

// Funkce řídící jízdu vozíku
void jed(){  cas_jizdy = millis(); // časový odpočet
    digitalWrite(in1.pin, in1.hodnota); // směr jízdy - motor A
    digitalWrite(in2.pin, in2.hodnota);
    digitalWrite(in3.pin, in3.hodnota); // směr jízdy - motor B
    digitalWrite(in4.pin, in4.hodnota);
  if(S2 || S4){ // při jízdě podle joysticku nebo po čáře
    analogWrite(enA.pin, enA.rychlost); // pojede námi určenou
       rychlostí
    analogWrite(enB.pin, enB.rychlost); }
 // Jízda po trase
  else{while(millis() - cas_jizdy < interval_jizdy){
    analogWrite(enA.pin, enA.rychlost); // pojede námi určenou
       rychlostí
    analogWrite(enB.pin, enB.rychlost); }}}  // konec fce pro jízdu

void urci_rychlost(){
    Blynk.syncVirtual(5); // aktuální nastavená rychlost -
synchronizace hodnoty v Blynku
  if(S4){ // když jedeme podle joysticku
    rychlost_motor = 140 + 15 *(rychlost.analogove - 1);  }
  else if(S2){ // když jedeme podle čáry
    rychlost_motor = 110 +  5 * (rychlost.analogove - 1); }
  else if(S3){ // když jedeme  po trase
    rychlost_motor = 140 +  10 * (rychlost.analogove - 1);
    interval_jizdy = 420; }}  // konec urči rychlost

BLYNK_WRITE(V5) {// Vyčítání RYCHLOST
  if(SIII){  rychlost.analogove = param.asInt();  }}

// Kontrola spojení s Blynkem
void spojeniBlynk(){
  if (Blynk.connected()){ // Kontrola připojení k Wi-Fi.
    Blynk.run();    // Blynk jede.
    timer.run();  } // timer jede
  else if (millis() > cas_pripojeni){ // za čas připojení se
nepodařilo připojit k Wi-Fi.
    stuj(); // zastaví motory
    Blynk.connectWiFi(ssid, pass); // pokusí se znovu připojit
    Blynk.connect(); // pokusí se zahájit komunikaci
    cas_pripojeni = millis() + 60000;  }} // nastaví časovač na 60
```

Příloha 3 – Informace na víku vozíku

Příloha 4 – Obsah přiloženého CD

- **text diplomové práce**

  diplomova_prace_2021_Marie_Koutna.docx

  diplomova_prace_2021_Marie_Koutna.pdf
- **zdrojový kód programu**

  Mopik_hlavni_program.ino

  Mopik_hlavni_program.html
- **schémata automatů v plném rozlišení**

  uroven_1.png

  uroven_2.png

  uroven_2k.png
- **schéma HW zapojení**

  schema.fzz

  schema.png

Příloha 5 – Schéma zapojení