

Jaroslav Páral  
Jakub Streit  
Rudolf Hlaváček  
a další

# ROBOTICKÝ MANUÁL

aneb

## Co se hodí vědět při stavbě a programování hobby robotů

[www.robotarna.cz](http://www.robotarna.cz)

[www.sokolska.cz](http://www.sokolska.cz)

[www.robotikabrnno.cz](http://www.robotikabrnno.cz)

za kolektiv autorů  
Miroslav Burda  
editor

dokumentace@robotikabrnno.cz

**verze 1.0**

2019-03-08 09:17:09

PDF:[doc.robotikabrnno.cz/RoboticsManual.pdf](http://doc.robotikabrnno.cz/RoboticsManual.pdf)

Online verze:[doc.robotikabrnno.cz](http://doc.robotikabrnno.cz)

# Obsah

<b>1 Začínáme</b>	<b>6</b>
1.1 Cíl: první robot . . . . .	6
1.2 Týmy pro stavbu robotů . . . . .	7
1.3 Postup návrhu robota . . . . .	7
1.4 Co potřebujete na začátku . . . . .	8
1.5 Přehled soutěží . . . . .	9
1.6 Stavba robota z Lego Mindstorms – doporučení . . . . .	10
<b>2 Mechanika</b>	<b>11</b>
2.1 Úvod . . . . .	11
2.2 Mechanická konstrukce robota – doporučení . . . . .	12
<b>3 Elektronika</b>	<b>13</b>
3.1 Jednoduché součástky . . . . .	13
3.1.1 Rezistor . . . . .	13
3.1.2 Kondenzátor . . . . .	14
3.1.3 Dioda . . . . .	15
3.1.4 LED . . . . .	15
3.1.5 Tranzistor . . . . .	15
3.1.6 Cívka . . . . .	16
3.2 Složitější součástky . . . . .	16
3.2.1 Driver . . . . .	16
3.2.2 Stabilizátor . . . . .	17
3.2.3 Krystal . . . . .	18

3.3	Základní veličiny v elektronice . . . . .	18
3.3.1	Proud . . . . .	18
3.3.2	Napětí . . . . .	19
3.3.3	Odpor . . . . .	19
3.3.4	Výkon . . . . .	19
3.3.5	Výpočet tepelného výkonu (ztrátového) . . . . .	19
3.3.6	Výpočet rezistoru pro LED . . . . .	20
3.4	Měření základních veličin . . . . .	21
3.5	Pájení . . . . .	23
3.5.1	Doporučení pro ty, co nikdy nepájeli . . . . .	23
3.5.2	Pájení SMD . . . . .	24
3.5.3	Pájení integrovaných obvodů . . . . .	24
3.6	Motory, serva a PWM . . . . .	25
3.6.1	Motory . . . . .	25
3.6.2	PWM . . . . .	26
3.6.3	Servo . . . . .	26
3.6.4	Řízení serva . . . . .	27
3.7	Řídící desky . . . . .	27
3.7.1	ESP 32 . . . . .	27
3.7.2	ALKS . . . . .	27
3.7.3	RBControl . . . . .	28
3.7.4	Arduino . . . . .	28
3.8	Senzory a malé desky . . . . .	29
3.8.1	Ultrazvukový senzor HC-SR04 . . . . .	29
3.8.2	Odrazový infrasenzor QRD1114 . . . . .	30
3.8.3	Bluetooth . . . . .	30
3.8.4	Převodník napěťových úrovní . . . . .	30
3.9	Napájení a baterie . . . . .	31
3.9.1	Přehled typů článků . . . . .	31
3.10	Napětí potřebná pro různé části robota a doporučené baterie . . . . .	32
3.11	Sběrnice . . . . .	33
3.11.1	USART/UART . . . . .	33

3.11.2 I2C . . . . .	33
3.12 Osciloskop . . . . .	33
3.12.1 Než začnete . . . . .	33
3.12.2 Sondy . . . . .	34
3.12.3 Zahájení měření . . . . .	35
3.12.4 Trigger . . . . .	36
3.12.5 Význam některých tlačítek – heslovitě . . . . .	36
3.12.6 Ostatní . . . . .	37
<b>4 Software</b>	<b>38</b>
4.1 Onshape . . . . .	38
4.1.1 Úvod, přihlášení, nový dokument . . . . .	38
4.1.2 Postup práce v Onshape . . . . .	40
4.1.3 Výroba skici . . . . .	41
4.1.4 Možnosti skici . . . . .	42
4.1.5 Výroba dílu – poznámky . . . . .	43
4.1.6 Výroba sestavy . . . . .	43
4.1.7 Výroba výkresů/příprava pro laser . . . . .	44
4.1.8 Slovníček pro Onshape . . . . .	44
4.2 Visual Studio Code . . . . .	45
4.2.1 Nainstalujte Visual Studio Code . . . . .	46
4.3 Cpp4Robots . . . . .	48
4.4 Lorris . . . . .	49
4.5 L <sup>A</sup> T <sub>E</sub> X . . . . .	50
4.5.1 Proč používat L <sup>A</sup> T <sub>E</sub> X . . . . .	50
4.5.2 Instalace a editory pro L <sup>A</sup> T <sub>E</sub> X . . . . .	50
4.5.3 L <sup>A</sup> T <sub>E</sub> X a SOČ . . . . .	51
4.5.4 Další inspirace k SOČ . . . . .	51
4.6 Git . . . . .	51
4.6.1 Základní pojmy . . . . .	51
4.6.2 Instalace gitu a stažení repozitáře – Linux . . . . .	53
4.6.3 Instalace gitu a stažení repozitáře – Windows . . . . .	53

4.6.4	Vytvoření commitu a nahrání změn na server pomocí terminálu – Linux . . . . .	54
4.6.5	Vytvoření commitu a nahrání změn na server pomocí TortoiseGit – Windows . . . . .	54
4.6.6	Formát Markdown (*.md) . . . . .	55
4.7	Další software . . . . .	55
4.7.1	Arduino IDE . . . . .	55
4.7.2	Atom . . . . .	55
4.7.3	Cpp4Robots . . . . .	56
4.7.4	Proficad . . . . .	56
4.7.5	Python . . . . .	57
4.7.6	Linux . . . . .	57
<b>5</b>	<b>Programování čipů v C++</b>	<b>59</b>
5.1	Základní doporučení pro programování . . . . .	59
5.1.1	title . . . . .	59
5.1.2	title . . . . .	59
5.1.3	Událostmi řízené programování . . . . .	59
5.2	Základy syntaxe v jazyce C . . . . .	59
5.2.1	Základní pojmy . . . . .	59
5.2.2	Výrazy a operátory . . . . .	61
5.2.3	Nejčastější příkazy C++ . . . . .	63
5.2.4	Funkce a procedury . . . . .	69
5.2.5	Knihovny . . . . .	70
5.3	Příkazy C++ pro čipy . . . . .	71
5.4	Příklady programů v C++ . . . . .	71
5.4.1	Blikání LED . . . . .	71
5.4.2	LED zapínaná tlačítkem . . . . .	71
5.4.3	Nejjednodušší PWM . . . . .	72
5.4.4	Infrasenzor na čáru . . . . .	73
5.4.5	Posílání dat pro Lorris po sériové lince . . . . .	74
5.4.6	Ultrazvukový senzor HC-SR04 . . . . .	75
5.4.7	Řízení serva . . . . .	76

5.4.8	Řízení motorů s použitím VNH2SP30 – základní . . . . .	76
5.4.9	Přerušení . . . . .	78
5.5	Desítková soustava a dvojková soustava . . . . .	78
5.5.1	Desítková soustava . . . . .	79
5.5.2	Dvojková soustava . . . . .	79
5.5.3	Převod z desítkové soustavy do dvojkové . . . . .	79
5.5.4	Převod z dvojkové soustavy do desítkové . . . . .	80
5.5.5	Bitové výrazy a práce s nimi . . . . .	81
<b>6</b>	<b>Přílohy</b>	<b>86</b>
<b>Přílohy</b>		<b>86</b>
6.1	Vývojový deník robota pro Ketchup House aneb stručná historie výroby . . . . .	86
6.2	Požadavky na počítač pro robotiky . . . . .	88
6.3	Hodnoty vybraných součástek . . . . .	88
6.4	Popis a vlastnosti desky RB3201-RBControl . . . . .	90
6.4.1	Určení a cíl . . . . .	90
6.4.2	Hlavní vlastnosti . . . . .	90
6.4.3	Další vlastnosti . . . . .	90
6.4.4	Napájení . . . . .	91
6.4.5	Expandér . . . . .	91
6.4.6	Rozložení pinů na desce RBC a jejich vlastnosti . . . . .	91
6.4.7	Programování a nápowěda . . . . .	94
6.5	Řešení některých problémů . . . . .	95
6.5.1	Chyba při uploadu programu do desky Arduino nano: .	95
<b>Rejstřík</b>		<b>96</b>

# Kapitola 1

## Začínáme

### 1.1 Cíl: první robot

Tento text je určen pro začátečníky a mírně pokročilé v oblasti stavby autonomních robotů – převážně středoškoláky v prvním ročníku, kteří se pokouší postavit svého prvního autonomního robota, nejčastěji pro nějakou soutěž. Protože byl sepsán na pracovištích Robotárna<sup>1</sup> a SPŠ Sokolská<sup>2</sup>, jsou některé části určené především členům jejich kroužků. Ale většina textu je použitelná všeobecně.

Každý robot se musí

- vyrobit (mechanická část – konstrukce)
- osadit elektronikou, motory a pod.
- naprogramovat

Přitom můžou nastat zhruba dvě situace:

- Nemám žádné znalosti a zkušenosti → doporučený postup je začít ve školním kroužku nebo samostatně stavět a programovat robota z LEGO MINDSTORMS – viz kapitola 1.6. Je to daleko nejjednodušší a nejrychlejší cesta, omezení jste cenou stavebnice a jejími možnostmi.

---

<sup>1</sup>pobočka DDM Helceletka Brno

<sup>2</sup>Střední průmyslová škola a Vyšší odborná škola Brno, Sokolská, příspěvková organizace

- Už jsem někdy něco naprogramoval, zapojil nebo postavil a chci jít dál → potom je pro vás určen tento text. Jednotlivé kapitoly se věnují různým oblastem, které je postupně potřeba zvládnout (nebo jimi pověřit jiného člena týmu) s důrazem na začátečnické problémy.

## 1.2 Týmy pro stavbu robotů

Už bylo zmíněno, že stavba robotů zahrnuje tři propojené, ale relativně nezávislé okruhy: návrh a výrobu mechanické konstrukce, návrh a zapojení elektroniky a programování. Proto je dobré roboty stavět v týmech, kde se jednotliví členové zaměřují na tyto oblasti a navzájem se doplňují. Navíc každý tým potřebuje řadu pomocných činností (nákup součástek, vyhledávání údajů na internetu a pod.). Je dobré mít proto v každém týmu ještě pomocníka, který podporuje ostatní a umožňuje jim soustředit se na jejich hlavní úkoly.

Úplně ideální potom je, když každou funkci v týmu zastávají dva lidé, takže se mohou vzájemně zastupovat. Tým by potom měl celkem osm členů – dva mechaniky, dva elektroniky, dva programátory a dva pomocníky. To se ale v praxi téměř nikdy nepodaří. Často nastane právě opačný případ, kdy tým má pouze dva nebo tři členy, kteří se o všechny činnosti musí nějak podělit. V každém případě ale platí, že je výhoda, pokud lidé v týmu znají i věci mimo jejich „hlavní obor“, tj. když například programátor zná základy elektroniky.

## 1.3 Postup návrhu robota

1. vybrat **soutěž**, které se chcete zúčastnit nebo mít jiný cíl, proč stavět robota
2. stanovit, co by měl robot splnit – přibližně
3. sepsat, co by měl robot splnit – podrobně (klidně i více stran A4), z toho vyplyně
4. zjištění, jaké senzory a pohony robot potřebuje a kde budou na robotovi umístěné
5. návrh první konstrukce robota včetně umístění senzorů a pohonů
6. zprovoznění toho všeho – **hlavní cíl tohoto textu**

Začátečníci obvykle tuto posloupnost nedodrží, začnou bodem 5 a pak staví mechaniku pro 3 a více verzí robota (někdy tak odlišných, že už se vlastně jedná o různé roboty). Až potom zjistí, že je to dost práce navíc. A taky dost času navíc, který potom před soutěží chybí.

## 1.4 Co potřebujete na začátku

### Hardware:

- notebook nebo počítač s operačním systémem Windows 7 a novějším nebo s operačním systémem Linux (pro starší počítače např. aktuální distribuci Xubuntu, Lubuntu, Mint). Podrobnější informace o doporučených parametrech počítačů jsou v kapitole 6.2.
- **řídící desku**, pro první učení je nejlepší [ALKS](#), pro vlastní stavbu robota potom vyberete desku podle požadavků na elektroniku
- výhledově cokoli dalšího, co chcete připojit na robota (serva, ultrazvuk, senzory a motory všeho druhu ... )

### Znalosti:

- běžná práce se soubory ve vašem operačním systému (hledání, kopírování, mazání, ... )
- velmi se hodí schopnost porozumět textu psanému v jednoduché angličtině

### Taky se hodí vědět, že:

- veškeré **zelené** a **modré** odkazy a slova jsou proklikávací (s výjimkou barevného zvýraznění syntaxe<sup>3</sup> ve zdrojových textech jazyka C++)
- na začátku textu je (klikací) **obsah**
- na konci textu je rejstřík
- v žádném textu o elektronice a robotice nemůže být všechno, pokud zde něco nenajdete, použijte např. Google; také může pomoci další [literatura](#)

---

<sup>3</sup>syntaxe – způsob zápisu daného jazyka, barevný zápis je mnohem přehlednější

## 1.5 Přehled soutěží

Plán účastnit se svým robotem soutěže je nejen motivací pro začátek stavění robota, ale hlavně motivací pro jeho dokončení, protože soutěž nepočká. V současné době se soustředíme na tyto soutěžní dny: **Robotiáda** v Brně a **Robotický den** v Praze. Dále existuje **Istrobot** v Bratislavě, kam téměř nejezdíme. Hlavní důvod je, že soutěž bývá v dubnu, kdy roboti ještě nejsou hotoví.

U všech soutěží, kterých se rozhodnete zúčastnit, je potřeba důkladně nastudovat pravidla. Hlavní členové týmu by je měli znát víceméně z paměti.

Na **Robotiádě** je podmínka, že hlavní části robota (řídící systém, pohony, senzory) musí být z **Lega**.

Robotiáda probíhá obvykle na začátku února a obsahuje několik soutěží. Každá z nich je rozdělená na kategorie ZŠ a SŠ. Které soutěže se zúčastníte, je v podstatě na vás, s vyjímkou Freestyle, kterou příliš nedoporučují kvůli velmi nejistému výsledku. Pokud začnete v září, je potřeba se přípravě, stavbě a programování robota věnovat alespoň jedno odpoledne týdně.

**Robotický den** probíhá obvykle na začátku června a obsahuje také několik soutěží, které jsou rozdělené na kategorie „Roboti pouze z dané stavebnice (např. z Lega)“ a „Roboti z čehokoli“. Není zde dělení podle věku, takže ZŠ a SŠ soutěží v jedné skupině s VŠ a dospělými podle toho, jak se kdo přihlásí. Výběr vhodné soutěže je pro úspěch zásadní a je nezbytné jej konzultovat s vedoucím kroužku. Pro stavbu funkčního robota je potřeba věnovat přípravě, studiu a stavbě minimálně jedno odpoledne týdně, pokud jste pomocný člen týmu a dvě odpoledne týdně, pokud jste např. hlavní mechanik nebo hlavní programátor týmu. Čím víc, tím líp, protože **zkušenosti** ukazují jasně, že času není nikdy dost.

### Přehled soutěží Robotického dne:

**Doporučené soutěže:** Toy Cleanup Beginner (pokud se Robotického dne účastníte poprvé), Ketchup House, Bear Rescue Advanced, Line Follower,

**Težko říct:** Puck Collect, Roadside Assistance, Toy Cleanup Advanced

**NEdoporučené soutěže:** RoboCarts, Free Style, Mini Sumo

Tato (ne)doporučení vycházejí ze zkušeností z předchozích let a odhadu reálných možností středoškolských studentů.

## 1.6 Stavba robota z Lego Mindstorms – doporučení

Na Robotický den můžete stavět pouze z „čistého“ Lega, viz [pravidla](#). Na Robotiádě můžete využít do konstrukce například díly z překližky pro větší tuhost a snazší montáž. Překližkové díly se dají navrhnout například v modelovacím systému [Onshape](#) a následně je možné je vyřezat na laseru na pracovišti [Fablab](#).

Pro programování robota z Lega máte dvě možnosti:

### **obrázkový kód**

použijete nativní obrázkové programování, ke stažení [zde](#); v podstavě jde o událostmi řízené programování – viz kapitola [5.1.3](#)

### **kód v C++**

použijete prostředí, které vám umožní programovat robota v C++, například [Cpp4Robots](#) – viz kapitola [4.7.3](#), potom

Pozor, toto prostřední pracuje pouze pod Windows7 a novějšími.

# Kapitola 2

## Mechanika

### 2.1 Úvod

Roboty stavíme nejčastěji pro nějakou soutěž. Podle zadání soutěže se rozhodujete, co bude robot na hřišti dělat, jak se bude orientovat atd. Z toho potom plynou požadavky na konstrukci.

Roboty rozlišujeme podle způsobu ovládání a podle velikosti.

#### Ovládání robotů

Každý robot může být buď řízený nebo autonomní. **Autonomní** znamená, že je naprogramovaný a během soutěže nebo prezentace se pohybuje samostatně.

Roboty můžeme řídit po kabelu nebo bezdrátově, například přes [bluetooth](#).

#### Velikost robotů

Na škole a v Robotárně stavíme a programujeme hlavně roboty dvojího typu : „střední“ a „velké“. Ostrá hranice mezi nimi není, doporučení pro oba typy se dají kombinovat podle situace. Střední je robot se základnou přibližně od 10x10 cm do 25x25 cm.

## 2.2 Mechanická konstrukce robota – doporučení

Celého robota nejprve navrhneme ve vhodném CAD programu: [Onshape](#), [Solidworks](#) nebo [Fusion](#).

Pokud jako základní materiál zvolíme překližku nebo například plexisklo (kombinované s díly z lega), můžeme díly vymodelované v CADu nechat vyřezat na laseru na pracovišti [Fablab](#). To dramaticky urychluje práci. Pro konstrukci prototypů se také hodí měkčené PVC.

Při stavbě robotů se držíme osvědčeného schématu:

Podvozek středních robotů tvoří základní deska z překližky, plexiskla nebo PVC. Podvozek velkých robotů tvoří rám z hliníkových profilů typu „L“ tvaru obdélníku, osmiúhelníku nebo kruhu.

Na podvozku jsou připevněné dva motory včetně převodovek, které se koupí hotové. V žádném případě se nepokoušejte koupit pouze motor (např. protože je levnější) a vyrábět si převody sami. U velkých robotů volíme motory z akušroubováků nebo z akuvrtaček, jiné typy motorů jsou buď pomalé nebo slabé. U středních robotů stačí motory modelářské, dnes nejčastěji kupované z Číny.

Dále jsou zde dvě kola, každé připojené ke svému motoru a jedna nebo více podpěr podvozku, obvykle z kartáčku na zuby. Motory s koly lze umístit doprostřed nebo dozadu, podle toho, co má robot na hřišti dělat.

Na podvozek se umisťuje konstrukce z hliníkových tyčí, profilů nebo z merkuru, s pomocí které robot plní svoje úkoly. U velkých robotů se méně tuhé materiály neosvědčily, u středních často stačí konstrukce z překližky.

# Kapitola 3

## Elektronika

### Datasheet

**Datasheet** je dokument, ve kterém jsou detailně popsány vlastnosti a možnosti dané elektronické součástky. Každá součástka má svůj datasheet.

Datasheet pro každou součástku je možné najít na webu, například na stránce [www.datasheetcatalog.com](http://www.datasheetcatalog.com) nebo na stránkách výrobce/prodejce součástky. Všechny datasheety jsou anglicky.

### Nepájivé kontaktní pole

**Nepájivé kontaktní pole** slouží pro první rychlé zapojení součástek do vytvářeného obvodu. Je to deska s maticí otvorů, které jsou vždy po pěti propojené. Do těchto otvorů zasunujete vývody součástek a podle potřeby je spojujete pomocí drátků. Díky tomu nemusíte pracovat s páječkou, a zároveň je zapojený obvod snadno rozebiratelný, takže můžete zapojení sestavovat, testovat, upravovvat a opět rozmontovávat. Po stranách jsou navíc dlouhé lišty otvorů určené pro přivedení napájení.

### 3.1 Jednoduché součástky

#### 3.1.1 Rezistor

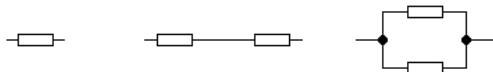
**Rezistor** nebo také odpor<sup>1</sup> je součástka, která klade elektrickému proudu

---

<sup>1</sup>Správně se součástce říká rezistor, odpor je její vlastnost. Ale mnoho lidí používá pro

určitý odpor neboli ho omezuje. Toho se používá jako ochrana před zničením čipu nebo jeho části. Odpor se značí  $R$ . Jednotkou odporu je 1 Ohm, značka  $\Omega$ .

Dva rezistory (nebo jiné součástky) mohou být zapojeny buď **sériově** (tj. za sebou) nebo **paralelně** (tj. vedle sebe), viz obrázek 3.1.



Obrázek 3.1: vlevo: značka rezistoru, uprostřed: rezistory zapojené sériově, vpravo: rezistory zapojené paralelně

Speciální roli v čipu mají interní<sup>2</sup> tzv. **pull-up a pull-down** rezistory.

### 3.1.2 Kondenzátor

**Kondenzátor** je součástka, která uchovává elektrický náboj. Jeho hlavní vlastností je kapacita. Jednotkou kapacity je Farad, značka F. V praxi se používají násobky jako mikrofarad ( $\mu\text{F}$ ), nanofarad (nF) a pikofarad<sup>3</sup> (pF). Kondenzátory se nabíjí a vybíjí různě rychle a mají různou kapacitu. Keramické kondenzátory mají nejmenší kapacitu (pF, nF) a jejich nabítí a vybití je nejrychlejší, tantalové mírají kapacitu okolo pár  $\mu\text{F}$  a jejich nabítí a vybití je pomalejší a nejpomalejší jsou elektrolytické s kapacitou stovek až tisíců  $\mu\text{F}$ . U tantalových a elektrolytických kondenzátorů musíme dát pozor na polaritu, tj. kam připojujeme + a kam -. Další důležitý údaj je maximální hodnota napětí, kterou kondenzátor snese.

Kondenzátory dokážou eliminovat napěťové špičky, které by jinak znemožnily provoz řídící desky. Proto je připojujeme paralelně ke zdrojům napěťových špiček (motory, serva).

---

označení součástky slovo odpor.

<sup>2</sup>vnitřní, zabudované

<sup>3</sup>Mikrofarad je milionina faradu, nanofarad je tisíckrát menší a pikofarad milionkrát menší než mikrofarad.

### 3.1.3 Dioda

**Dioda** je součástka, která usměrňuje elektrický proud. To znamená, že pokud ji zapojíme do elektrického obvodu, tak zajistí, že proud bude téct pouze jedním směrem. Proto budeme diodu používat jako ochranu proti tzv. **přepólování** – chybnému zapojení baterie nebo součástky do obvodu, které obvykle vede ke zničení součástky. U samotné diody také záleží na polaritě, tj. při jejím zapojení musíme dávat pozor, kde má kladný pól a kde záporný. Na diodě vzniká úbytek napětí, se kterým musíme počítat při návrhu obvodu. Tak například pokud připojíme diodu s úbytkem napětí 0,6 V připojíme 12 V, tak za diodou bude napětí 11,4 V.

Ze začátku nám bude stačit, pokud budeme používat diody [1N4148](#) a [1N4007](#).

### 3.1.4 LED

**LED**<sup>4</sup> je součástka, která není primárně určená k usměrnění proudu, ale k signalizaci, zda obvodem protéká proud. K LED se vždy musí připojit vhodný rezistor.

### 3.1.5 Tranzistor

**Tranzistor** je součástka, která umožňuje pomocí malých proudů z čipu řídit větší proudy, například do reproduktoru nebo motorku.

Tranzistor má tři nožičky: **báze**, **kolektor** a **emitor**. Tranzistorů existuje mnoho typů: **bipolární**, **JFET**, **MOSFET** a další. Bipolární tranzistory existují ve dvou provedeních PNP a NPN<sup>5</sup>. Tranzistory mají prakticky dvě použití: mohou pracovat jako spínač (vypínač) nebo jako zesilovač. Budeme se zabývat jednodušším použitím, tj. jako spínače. Budeme používat tranzistory NPN. Pokud bude přes bázi do emitoru téct omezený (malý) proud, tranzistor se otevře a přes kolektor do emitoru bude téct velký proud. Tak nám stačil malý proud k řízení velkého proudu. A toho budeme využívat.

Ze začátku nám bude stačit používat tranzistory [BCC337](#), [BCC547](#) a [BD911](#).

---

<sup>4</sup>Light emitting diode – světlo vysílající dioda

<sup>5</sup>Pro lepší zapamatování značky: eN-Pé-eN - šipka VEN

### 3.1.6 Cívka

**Cívka** neboli **tlumivka** je součástka, jejíž hlavní vlastností je indukčnost, jednotka henry, značka H. V praxi se používají milihenry (mH) a mikrohenry ( $\mu$ H).

## 3.2 Složitější součástky

### Mikroprocesor, mikrokontrolér

**Mikroprocesor, mikrokontrolér, čip** znamenají totéž – integrovaný obvod, který se snažíme naprogramovat, aby řídil robota nebo jeho část.

**Pin** je vývod (nožička) čipu. Jednoduché čipy (např. ATTiny) mají osm pinů, složitější čipy mají 32, 40, nebo také 100 pinů.

Pin může být nastavený jako vstupní nebo jako výstupní.

Pokud je pin nastavený jako **vstupní**, umí určit, zda na něm je napětí odpovídající logické jedničce (5 V nebo 3,3 V podle typu čipu) nebo logické nule (0 V). U některých pinů lze i přečíst, jaké je na něm analogové napětí (např. v rozsahu 0 - 1023  $\Rightarrow$  0 V - 3,3 V).

Pokud je pin nastavený jako **výstupní**, umí se nastavit na logickou jedničku nebo logickou nulu.

### 3.2.1 Driver

Čip nemůže řídit například motor přímo, protože jedním pinem může protékat obvykle maximálně 40 mA. Většina motorů potřebuje mnohem větší proud. Proto se používají součástky zvané **drivery**, které podle pokynů z čipu řídí proud z baterií do motorů a servomotorů. Jsou to speciální integrované obvody pro řízení motorů, které jsou složeny z mnoha **tranzistorů** a dalších prvků.

Níže jsou některé drivery uvedeny.

je driver postavený na čipu HR8833 a použitý na desce **RBCControl**. Je ideální pro řízení pohonů středních robotů napájených dvěma Li-On bateriemi. Každý motor může být poháněn 1,5 A na 10 V.

## VNH2SP30

Driver **vnh2sp30** umí řídit pomocí PWM motor až do napětí 16 V, trvalého proudu 14 A a špičkového proudu 30 A. Pro takto velké proudy potřebuje účinné chlazení, např. chladič s ventilátorem. Je ideální pro řízení pohonů velkých robotů včetně motorů z akuvrtaček. Driver umí pomocí PWM jízdu vpřed, vzad, brzdění (motory jsou ve zkratu, když je na ně přiváděna logická 1 z PWM) a signál stop. Příklad programu pro použití tohoto driveru je v kapitole [5.4.7](#). Tyto drivery jsou na Robotárně dostupné na deskách [Arduino VNH2SP30 Monster Moto Shield](#). Relativně dobrá knihovna pro ně je [zde](#).

## RoboClaw

Driver [RoboClaw](#) je deska, která umožňuje řídit dva motory o odběru 15 A. Pro tyto motory má také na sobě enkodéry (A/B signál). Dále má na sobě 5 V 3 A spínaný zdroj. [Stránky výrobce](#). Nevýhodou je vyšší cena.

## Odrive

Samostatnou kapitolou jsou drivery pro střídavé motory. Střídavé motory jsou malé, lehké, výkonné. [Odrive](#) umožňuje je napájet z běžných baterií. Protože ale driver i motory něco stojí a zatím je nikdo nekoupil a nerozběhal, nemáme s jejich provozem zkušenosti.

### 3.2.2 Stabilizátor

Většinu čipů je potřeba napájet přesně 5 V nebo 3,3 V. Jak toho docílit z baterií, na kterých je například 9 V nebo 12 V, zkrátka více než 5 V? Navíc napětí na bateriích kolísá podle toho, jak moc proudu zrovna odebírají motory. Pro napájení čipů je proto nutné použít stabilizátor.

**Stabilizátor** je součástka, která z kolísavého vyššího napětí vyrobí přesné napětí nižší. Přitom nějaké napětí také sama spotřebuje. Nejčastěji se používají stabilizátory řady 78XX, kde XX značí, na kolik voltů součástka stabilizuje, například 7805 stabilizuje na 5 V.

Aby stabilizátor mohl pracovat správně, je potřeba, aby napětí, které přivedeme na jeho vstup, bylo obvykle aspoň o 2 V vyšší než které potřebujeme,

tj. pokud budu chtít stabilizovat napětí na 5 V, musím stabilizátor napájet aspoň 7 V. Přesné hodnoty pro každý stabilizátor jsou v jeho datasheetu. Stabilizátor má tři piny (vstup, zem, výstup). Zapojí se takto: kladný pól baterie (+) se napojí na vstup, záporný pól na zem (-). Mikrokontrolér se zapojí pinem VCC na výstup stabilizátoru a GND se zapojí na zem stabilizátoru. Tímto máme připojený mikrokontrolér na napájení.

Pokud máme stabilizátor před sebou tak, abychom přečetli jeho označení, např. L7805, potom první pin zleva je vstup, druhý je zem a třetí, tj. úplně vpravo je výstup. Na vstup připojíme 7 V až 12 V, prostřední pin uzemníme, a poslední pin vyvedeme na VCC mikrokontroléru. Dále je potřeba věnovat pozornost zapojení kondenzátorů. Mezi vstup a zem připojíme podle datasheetu kondenzátor s kapacitou 330 nF. Mezi výstup a zem kondenzátor 100 nF.

### 3.2.3 Krystal

Pokud budeme potřebovat provozovat některé procesory na vyšší frekvenci, použijeme **krystal**. Například původní frekvence mikrokontroléru ATMega16 je nastavena na 1 MHz. S pomocí krystalu ji můžeme zvýšit až na 16 MHz. Krystal zapojíme takto: Jeden pin krystalu (je jedno, který) připojíme na pin XTAL1 a druhý na XTAL2. Dále na pin XTAL1 připojíme jednu nožičku kondenzátoru a druhou na digitální zem (11-GND). To samé u pinu XTAL2. Hodnotu kondenzátoru můžeme volit od 12 pF do 22 pF.

## 3.3 Základní veličiny v elektronice

### 3.3.1 Proud

Pokud použijeme vodní model, tak (elektrický) proud je množství vody, které proteče vodičem za jednu sekundu.<sup>6</sup> Značka:  $I$ , jednotka: 1 A = 1 ampér.

---

<sup>6</sup>Ve skutečnosti je to protečení množství elektronů nebo jiných nosičů el. nábojů s celkovým nábojem 1 Coulomb za sekundu.

### 3.3.2 Napětí

Elektrické napětí měříme vždy mezi dvěma body. Můžeme si ho představit jako rozdíl výšek dvou vodních hladin. Z výše položeného jezera (kladné napětí, +) teče voda (el. proud) do níže položeného (zem, nulový potenciál, -). Značka:  $U$ , jednotka: 1 V = 1 volt.

### 3.3.3 Odpor

Mezi napětím  $U$  a elektrickým proudem  $I$  platí vztah:

$$U = R \cdot I,$$

Tento vztah je znám pod názvem **Ohmův zákon**.

Konstanta  $R$  se nazývá elektrický odpor, měříme ho ohmec, značka  $\Omega$ .

Z Ohmova zákona můžeme vyjádřit proud  $I$ :

$$I = U/R.$$

Ze vzorce je vidět, že pokud použijeme rezistor s větším odporem při stejném napětí, tak se protékající proud zmenší.

### 3.3.4 Výkon

Výkon je definován jako součin napětí a proudu:

$$P = U \cdot I.$$

Značka:  $P$ , jednotka: 1 W = 1 watt.

### 3.3.5 Výpočet tepelného výkonu (ztrátového)

Na součástkách, na kterých je úbytek napětí a kterými protéká proud, vznikají tepelné ztráty.

**Příklad 1:** Na rezistoru je úbytek napětí 3,6 V a protéká jím 240 mA. Jaký je tepelný ztrátový výkon?

$$P = U \cdot I = 3,6 \text{ V} \cdot 0,24 \text{ A} = 0,864 \text{ W} = 864 \text{ mW}$$

Ztráty na rezistoru činí 864 mW, a proto musíme volit rezistor, který zvládne větší výkon, tj. minimálně 1 W.

**Příklad 2:** Diodou 1N4148 bude procházet 100 mA, při tomto proudu bude úbytek napětí na diodě 1 V. Nezničíme diodu 1N4148?

$$P = U \cdot I = 1 \text{ V} \cdot 0,1 \text{ A} = 0,1 \text{ W} = 100 \text{ mW}$$

Diodu nezničíme, protože je vyráběná na celkový výkon 500 mW.

**Příklad 3:** Na stabilizátor L7805 přivádí 12 V, stabilizátor mi vytváří 5 V stabilizovaného napětí a odebírám z něho 250 mA. Jaký tepelný výkon bude potřeba uchladit?

$$P = U \cdot I = (12 \text{ V} - 5 \text{ V}) \cdot 0,25 \text{ A} = 1,75 \text{ W}$$

Tepelné ztráty na stabilizátoru budou 1,75 W<sup>7</sup>.

### 3.3.6 Výpočet rezistoru pro LED

LED je součástka, která není primárně určená k usměrnění proudu, ale k signalizaci. Může svítit světlem bílým, modrým, zeleným, červeným, ultrafialovým či infračerveným<sup>8</sup>.

Pokud připojím diodu správně na napětí, tj. tak aby mohl procházet proud a ono přesto nic, tak jsem diodu spálil, protože jí tekl moc velký proud. A proto musíme vždy k diodě připojit do série rezistor, který omezí proud protékající přes LED. A to podle vzorce:

$$R = \frac{U}{I} = \frac{\text{Napeti\_zdroje} - \text{Ubytek\_napeti}}{\text{Pozadovany\_proud}}$$

**Příklad 1:** Vypočítejte odpor rezistoru, který zapojíme do série k LED. Připojujeme k ní napětí 5 V a provozní proud je 20 mA a maximální proud, při kterém dojde ke zničení diody je 40 mA. Úbytek na diodě je 1,2 V.

$$R = \frac{U}{I} = \frac{5 \text{ V} - 1,2 \text{ V}}{0,02 \text{ A}} = 190 \Omega$$

---

<sup>7</sup>Ve vzorci bylo použito 12 V – 5 V, je to protože na stabilizátoru ubyde 7 V, abychom se dostali na požadovaných 5 V.

<sup>8</sup>Toho využijeme jako senzoru pro robota.

Použijeme rezistor s odporem  $190 \Omega$  nebo nejbližší vyšší odpor.

**Příklad 2:** Máme sériově spojeny tři LED, s úbytky napětí  $0,6 \text{ V}$ ,  $0,8 \text{ V}$  a  $1,2 \text{ V}$ . Připojíme je k zdroji o napětí  $12 \text{ V}$ . Rezistor o jak velkém odporu musíme použít, jestliže má diodami protékat  $20 \text{ mA}$ ?

$$R = \frac{U}{I} = \frac{12 \text{ V} - 0,6 \text{ V} - 0,8 \text{ V} - 1,2 \text{ V}}{0,02 \text{ A}} = 470 \Omega$$

Musíme použít rezistor o odporu  $470 \Omega$ .

Měli bychom ještě spočítat tepelný výkon rezistoru. Můžeme použít úbytek napětí na rezistoru vynásobený procházejícím proudem:

$$P = U \cdot I = (12 \text{ V} - 0,6 \text{ V} - 0,8 \text{ V} - 1,2 \text{ V}) \cdot 0,02 \text{ A} = 9,4 \text{ V} \cdot 0,02 \text{ A} = 0,188 \text{ W} = 188 \text{ mW}$$

Stačí  $250 \text{ mW}$  rezistor. Nebo si můžu vypočítat úbytek napětí z  $U = R \cdot I$ , to protože znám odpor a protékající proud.

$$P = U \cdot I = R \cdot I \cdot I = R \cdot I^2 = 470 \Omega \cdot (0,02 \text{ A})^2 = 0,188 \text{ W} = 188 \text{ mW}$$

Vyšel nám stejný výsledek, použijeme tedy  $250 \text{ mW}$  rezistor o odporu  $470 \Omega$ .

## 3.4 Měření základních veličin

Napětí, proud a odpor měříme pomocí **multimetru**.

### Zdírky

**zdírka COM:** Záporný pól neboli zem (GND), používá se vždy, připojuje se do něj vždy **černý** kabel. Ostatní zdírky jsou kladné póly a používá se z nich ta, která je potřeba. Připojuje se do nich vždy **červený** kabel.

**zdírka V/Ω:** pro měření napětí a odporu

**zdírka A:** pro měření proudu

**zdírka 20 A** pro měření velkých proudů (bez pojistky!) nebudeme používat

## Měření odporu

Multimetr přepněte na rozsahy nahoře, skupina  $\Omega$ .

Po přepnutí na  $\Omega$  se vlevo zobrazí 1. To znamená, že měřená hodnota je mimo nastavený měřící rozsah. V našem případě je to odpor vzduchu mezi měřícími hroty.

Změřte odpor **rezistorů** a) držených v ruce, b) umístěných v **nepájivém kontaktním poli** nebo položených na lavici. Pokud se naměřené hodnoty liší, pokuste se vysvětlit proč.

Při zapojování součástek do nepájivého kontaktního pole dejte pozor, abyste je nezkratovali.

Měřit začněte od největšího očekávaného rozsahu a postupujte dolů.

Pozor, pokud budete měřit odpor rezistorů zapojených se zdrojem v obvodu, může se stát, že naměříte vnitřní odpor zdroje.

Lidské tělo má taky konečný odpor – pokuste se ho změřit.

## Měření napětí

DC je zkratka pro *direct current* – stejnosměrný proud. AC je podobně zkratka pro *alternate current* – střídavý proud. Na multimetru se používají zkratky DCV pro stejnosměrné napětí a DCA pro stejnosměrný proud. Dále zkratky pro střídavé napětí ACV a střídavý proud ACA. Protože baterie poskytuje vždy stejnosměrné napětí, budeme měřit stejnosměrné napětí a stejnosměrný proud.

Nastavte vhodný rozsah multimetru a změřte napětí na baterii.

Dále změřte napětí na rezistoru.

**Pozor**, pokud měříte **napětí**, připojte multimeter **paralelně** (multimetr je mimo měřený obvod a má nastavený obrovský vnitřní odpor).

Pokud měříte **proud** (viz dále): připojte multimeter **sériově** (multimetr je zapojený do obvodu tak, aby proud tekly přes něj).

Zapojte sériově dva rezistory  $10\text{k}\Omega$  a  $20\text{k}\Omega$  a baterii a změřte napětí na každém rezistoru a na baterii, výsledky zapište. Mimořádně, právě jste sestavili tzv. **napěťový dělič** – zapojení, které se používá poměrně často.

Pokuste se získaný výsledek zobecnit a odvodit z Ohmova zákona.

## Měření proudu

**Pozor !** Přepojte červený měřící kabel do zdířky A.

Změřte proud tekoucí zapojeným obvodem.

Odpovídá naměřená hodnota očekávání? Ověřte podle Ohmova zákona.

**Pozor !** Pokud se baterie silně zahřívá, je zkratovaná a musí se okamžitě vypojit z obvodu.

Měřáky na konci měření vypínejte – šetří se tím podstatně baterie.

## 3.5 Pájení

### 3.5.1 Doporučení pro ty, co nikdy nepájeli

1. **Páječka** je to zařízení, se kterým se pájí. **Pájka** je ten materiál, kterým se snažíme vodivě spojit součástky. Často se pájce říká **cín**, i když je to slitina více kovů.
2. Teplota hrotu páječky pro běžné pájení je asi 285 stupňů. Pokud máte součástku s větším odvodem tepla (např. stabilizátory a tranzistory v pouzdře TO220) můžete zvednou teplotu na 310 stupňů, případně použít mohutnější hrot (větší tepelná kapacita). Zvýšit teplotu pájky můžete i v případě pájení na části DPS, kde je např. rozlitá zem (= velký odvod tepla). Pak je občas potřeba zvýšit teplotu někdy až k 350 stupňům.
3. Pokud pájka nechce přilnout k nožičkám součástek nebo desce plošného spoje (DPS), pomůže pájecí kapalina ([Tekuté tavidlo TAVIDLO R](#) - hůř se z DPS umývá) nebo pájecí želé ([Gelové tavidlo FUTURE REWORK JELLY](#) - je výrazně dražší, ale funguje mnohem líp, jde lépe nanášet i umývat a vydrží vám velmi dlouho - lze koupit i ve více lidech a rozdělit se). Pájecí kapalina i pájecí želé se musí po dokončení pájení z DPS umýt. Obojí se umývá lihem a nebo izopropanolem (izopropyl alkohol - [Čistící přípravek IZOPROPANOL 400 ml ve spreji](#), [Čistící přípravek IZOPROPANOL 1000 ml v plechovce](#)).
4. **Kalafuna** se používala u trafopáječek za stejným účelem jako pájecí kapalina. Trafopájky ovšem měly pájecí špičky o teplotě přibližně 220 stupňů. Na hrotech o teplotě přes 300 stupňů se velmi rychle vypaří,

proto se pro použití s nimi nehodí. V nouzových případech lze ale použít i kalafunu.

5. S páječkou se obvykle prodávají kulaté tenké pájecí hroty, které jsou pro pájení běžných součástek naprosto nevhodné, protože nedokážou přenést dost tepla. Tenký hrot se použije hlavně tam, kde se silnější hrot nevejde. Pro běžnou práci je vhodný hrot tvaru dláta o šířce asi 2 mm. Takový hrot také velmi dobře prohřívá pájecí plošky pro SMD součástky (viz další kapitola). Platí, že hrot by měl být silnější, než je pájený předmět, aby dokázal přenést přiměřeně rychle dostatek tepla. Potřebné hroty se dají koupit samostatně v prodejnách elektro součástek a dají se na páječce vyměnit.
6. Pokud se vám při pájení chvějí ruce, opřete si je o zápěstí. Hodně taky dělá cvik. Zároveň je vhodně mít DPS dobře položenou (aby se vám při pájení nepohybovala) a nebo upevněnou ve **třetí ruce**.

### 3.5.2 Pájení SMD

Součástky SMD (surface mount device) nemají nožičky, ale pájí se přímo k ploškám na desku. Postup je podobný jako nožičkových součástek, liší se v tom, že součástka nedrží za nožičky v DPS, takže je potřeba je přidržovat pinzetou na správné pozici na DPS (**nikdy ne prstem - mohli byste se spálit**). Druhou rukou se trochu pájky součástka přichytí k desce. Potom se pořádně zapájí druhá ploška a poté se opraví zapájení první plošky.

### 3.5.3 Pájení integrovaných obvodů

Postup:

1. Naneste pájecí želé nebo pájecí kapalinu na štěteček a potřete kontakty, na které chcete pájet.
2. Přiložte integrovaný obvod (IO) a zkontrolujte jeho orientaci na desce. Každý IO má na sobě zobáček nebo kolečko, které označuje pin č.1. Čip také může být zkosený na straně pinu č.1.
3. Naberte na páječku malinko cínu, držte hrot páječky na jednom krajinm pinu a pinzetou doladěte polohu čipu. Jako první odložte páječku, cín za cca sekundu vychladne a potom pusťte čip.

4. Naberte na páječku více cínu a projedte celou řadu pinů. Díky pájecí kapalině cín přilne přesně tam, kam má.

Další zajímavé informace k pájení: <https://technika.tasemnice.eu/trac/wiki/SMDkecy>

## 3.6 Motory, serva a PWM

### 3.6.1 Motory

Pokud není přímo napsáno něco jiného, znamená motor všude v tomto textu elektromotor na stejnosměrný proud.

Motor připojujeme k řídící desce přes tzv. [drivery](#) na ty piny desky, které umožňují vysílat [PWM](#) signál.

#### Rotační enkodéry

Abychom věděli, co se s motorem (a tím i s robotem – pokud mu zrovna ne-prokluzují kola) v průběhu času děje, používáme součástku, která se jmenuje ([rotační](#) enkodér a zjišťuje, jak se motor otáčí. Mít motory, na kterých nebo ve kterých je enkodér rovnou zabudovaný je při stavbě robotů velká výhoda. Tzv. pomalé enkodéry, např. [KY-040](#) se dají také používat například při zjištění otočení volantu nebo páčky při řízení robota a pod.

#### Řízení motorů pomocí tranzistorů

Malé motory stačí k desce připojit pomocí tranzistorů.

Pokud chceme řídit motor z čipu stylem start – stop, postačí přes odpor např.  $1\text{ k}\Omega$  spojit výstupní pin čipu s bází tranzistoru a na emitor a kolektor připojit baterii a motorek zapojený do série.

Dále je potřeba bázi tranzistoru propojit pomocí např.  $10\text{ k}\Omega$  se zemí. Jinak totiž při vypnutém signálu z čipu báze „visí v luftě“ a chová se jako anténa – indukují se na ní různé signály a většinou se díky tomu motor samovolně slabě otáčí.

Nakonec je potřeba mít společnou zem pro čip i pro motor – pokud to není splněno, obvykle motor nejede. Pokud chceme řídit motor programově pomocí (viz kapitola [3.6.2](#)), musíme navíc mezi emitor a kolektor tranzistoru

vložit (obyčejnou) diodu pólovanou závěrně vůči baterii. Při vypnutí tranzistoru vznikají totiž na cívkách motorku napěťové špičky, které deformují tvar PWM signálu. Další proudy se v motoru indukují, když se motor po vypnutí proudu setrvačností otáčí dál.

### 3.6.2 PWM

**PWM** (Pulse Width Modulation) neboli pulzně šířková modulace je učený název pro tzv. obdélníkový signál – z pinu vychází hodnoty napětí, které zakreslené do grafu mají tvar **obdélníku**. Proč zrovna obdélník? Protože na tranzistorech i driverech jsou při řízení motoru nejmenší ztráty, když jsou zcela otevřené (přenáší maximum napětí nebo proudu) nebo zcela zavřené (nepřenáší nic). Nejmenší ztráty znamenají také nejsnazší možné chlazení. Proto se snažíme stavům mezi oběma krajinami mezemi vyhnout a zkrátit je na minimum.

### 3.6.3 Servo

(**Modelářské**) **servo** je krabička, která obsahuje motorek s převodovkou do pomala a řídící elektroniku, která se stará o jeho správné natočení. Obvykle se umí otočit v úhlu 180 stupňů s velkou přesností. Jeho klíčová vlastnost je, že polohu, do které se otočil, se snaží udržet.

Na desce **ALKS** jsou zapojeny vývody pro pět serv, která lze z desky přímo napájet a řídit. Protože servo (jako každý motor) potřebuje hodně proudu, lze přímo z desky napájet pouze nejmenší serva každé zvlášť. Pro větší serva je na desce připraven mini-USB konektor, do kterého je možné připojit samostatné napájení pro serva.

Pro připojení serv na desku ALKS platí, že GND (zem, mínu) je na okraji desky, 5 V je uprostřed a datový pin je nejblíže čipu.

Deska **RBCControl** umí po osazení spínanými zdroji napájet a ovládat 4 serva nebo 8 mikroserv, která pracují současně. Maximálně je možné připojit až 32 serv nebo mikroserv.

Běžné servo odebírá při provozu 1 A i více, podle velikosti, a to i když se nehýbe. Spotřebuje tento proud na to, aby se udrželo v zadané poloze. **Mikroservo** je malý typ serva, kterému stačí proud cca 0,5 A.

**Pozor!** Servo nesnáší přepólování napětí, když se přepóluje, tak shoří (když se přepóluje signál, tak to tolik nevadí).

### 3.6.4 Řízení serva

Jak se řídí pohyb serva? Pro tento účel je ideální právě generování PWM signálu.

Servo se řídí logickým signálem (jedničkou) po dobu od 1 ms do 2 ms (často i od 0,5 ms do 2,5 ms), a celková perioda je 20 ms. Podle toho, jak dlouho signál trvá, tak se servo natočí. Tj. pokud budeme chtít servo maximálně natočit na jednu stranu, nastavíme pin, který slouží jako řídící signál pro servo, na logickou jedničku po dobu 1 ms a pak 19 ms logickou nulu a pak zase logickou jedničku, logickou nulu, atd...

Pokud budeme chtít servo posunout do druhé krajní polohy, necháme logickou jedničku po dobu 2 ms a logickou nulu po dobu 18 ms. Pokud budeme chtít střední polohu, tak jedničku nastavíme na 1,5 ms a nulu na 18,5 ms. Jestliže budeme potřebovat jiný úhel natočení, nastavíme logickou jedničku na odpovídající dobu.

Jednoduchý program pro řízení serva je v kapitole [5.4.6](#).

## 3.7 Řídící desky

### 3.7.1 ESP 32

Deska **ESP32-DevKitC** je vývojová deska osazená čipem ESP-WROOM-32, který má řadu výborných [vlastností](#).

Deska se napájí z USB (5 V) a je na ní napěťový převodník na 3,3 V. Přitom USB může dodávat oficiálně 100 mA, v reálu ale běžně dodává 500 mA až 1 A. USB porty jsou také vcelku odolné proti zkratu.

### 3.7.2 ALKS

Deska ALKS<sup>9</sup> byla navržena přímo na Robotárně Brno právě proto, že hotová deska vás hlavně ze začátku zbavuje nutnosti vědět, co si můžete dovolit kam připojit a jestli to bude fungovat. Na webu má ALKS vlastní [wiki stránky](#). Zde najdete zapojení desky ALKS, její pinout<sup>10</sup> a spoustu dalších informací.

---

<sup>9</sup>Arduion Learning Kit Starter

<sup>10</sup>přehled zapojení pinů

Na ALKS se dají nasadit desky ESP 32, Arduino uno a Arduino nano, které jí také poskytují napájení.

**POZOR !** Při připojování čehokoliv dalšího k této nebo jiné desce si nechte před zapojením napájení všechno zkontrolovat. Hlavně, pokud připojovaná součástka spotřebuje víc proudu (serva a motory) nebo pokud vyžaduje vyšší napětí.

### ALKS a ESP 32

Pro nasazení desky [ESP 32](#) na ALKS byla napsaná knihovna *LearningKit*. Aby fungovala, musí být do souboru *platformio.ini* dopsán řádek `lib_deps = 1745` (bez mezery na začátku řádku) a do záhlaví souboru *main.cpp* doplňte `include "LearningKit.h"`. – viz obrázek [4.1](#) vpravo.

### ALKS a Arduino nano

Pro nasazení desky [Arduino nano](#) na ALKS byla napsaná knihovna *LearningKit\_nano.h*. Aby fungovala, musí být v záhlaví souboru *main.cpp* doplněno `include "LearningKit_nano.h"`. Protože knihovna *LearningKit\_nano.h* využívá knihovnu *LearningKit.h*, musí být také do souboru *platformio.ini* dopsán řádek `lib_deps = 1745` (bez mezery na začátku řádku) – viz obrázek [4.1](#) vpravo.

### 3.7.3 RBControl

#### Určení a cíl

RB3201 - RBControl (RBC) je univerzální deska pro stavbu hobby robotů, vyvinutá na Robotárně Brno. Jde v podstatě o shield k desce ESP32 dev kit, který má dva hlavní cíle: rozšířit počet pinů desky ESP32 a umožnit snadné připojení velkého množství různých periférií, především robotických.

Podrobnější popis desky je v kapitole [6.4](#).

### 3.7.4 Arduino

Arduino je otevřený projekt pro snadné programování čipů a snadné připojování periferií k čipům. Klíčová výhoda všech desek rodiny Arduino je, že

se dají koupit levně už hotové. Hlavní web projektu je [zde](#) (EN). Český rozcestník pro Arduino je [zde](#). Všechny tři následující desky pracují s napětím 5 V. Na desky Arduino existuje řada rozšíření, tzv. **shieldů**, stačí do Google zadat *arduino shield*. Protože jde o otevřený projekt, existuje řada kopií jak desek, tak shieldů Arduina, obvykle s totožnými vlastnostmi, ale často (především z Číny) s výrazně nižší cenou.

Řídících desek Arduino je více [2, strana 3–10], my používáme následující:

### Arduino Mega

[Arduino Mega](#) patří mezi velké desky z rodiny Arduino. Má vyvedených 100 pinů a může obsluhovat tři sériové linky. Hodí se pro větší projekty a jako hlavní řídící deska na robota. Schéma zapojení pinů je například [zde](#).

### Arduino Uno

[Arduino Uno](#) je základní deska z rodiny Arduino. Je vhodná na testování různých zapojení a jako řídící pro jednodušší projekty. Schéma zapojení pinů je například [zde](#).

### Arduino Nano

[Arduino Nano](#) je nejmenší deska z rodiny Arduino (z námi používaných). Hodí se tam, kde je málo místa a pro řešení jednodušších, relativně samostatných úloh. Schéma zapojení pinů je například [zde](#).

## 3.8 Senzory a malé desky

### 3.8.1 Ultrazvukový senzor HC-SR04

Ultrazvukový senzor **HC-SR04**<sup>11</sup> je cenově dostupný a běžně používaný při amatérské stavbě robotů. Jeho zapojení a oživení je [zde](#).

---

<sup>11</sup><https://www.sparkfun.com/products/13959>

### 3.8.2 Odrazový infrasenzor QRD1114

Tato součástka má v jednom pouzdře vysílací infračervenou LED a přijímací fototranzistor<sup>12</sup>. Umístěná několik mm nad hřiště a zastíněná od okolí je ideální pro rozpoznání černé čáry na bílém podkladu a podobně. Dá se také použít pro detekci blízkého předmětu (dojde k zastínění). Můžeme použít například QRD1114. Může rozlišit vzdálenosti v rozsahu 0,75–10 mm nebo rozlišit černý a bílý povrch. Na adrese <https://learn.sparkfun.com/tutorials/qrd1114-optical-detector-hookup-guide> najdeme schéma zapojení i příklad kódu pro Arduino. Ještě jednodušší [příklad kódu](#).

### 3.8.3 Bluetooth

Moduly **bluetooth** slouží ke komunikaci mezi dvěma čipy nebo počítačem a čipem (nebo jiným zařízením). Mohou být napájeny 5 V nebo 3,3 V.

Propojujeme vždy pin Rx na jednom čipu s pinem Tx na druhém čipu.

#### Připojení k počítači pomocí bluetooth

Nový bluetooth (zub) se musí napoprvé vyhledat a aktivovat v počítači. Pokaždé se musí připojit a zkонтrolovat – když je komunikace v pořádku (aktivována, ale nemusí se přenášet data), svítí LED na zubu. Když je v pořádku modul v počítači, tak bliká.

Dále musí být spojená země zuba a čipu.

### 3.8.4 Převodník napěťových úrovní

**Převodník napěťových úrovní** je sympatická věcička, které umožňuje propojit zařízení pracující na 5 V se zařízením, které pracuje na 3,3 V. Například připojení některých bluetooth nebo serv k ESP32, propojení ESP32 s Arduinem Uno nebo Mega a podobně. Vypadá [takhle](#). Podrobný výklad k převodníku včetně schématu zapojení je [zde](#).

---

<sup>12</sup>Fototranzistor je součástka, která zesiluje signál podle toho, jak je osvětlená.

## 3.9 Napájení a baterie

Napájení robota je obvykle složitější, než by se mohlo zdát.

Vše na robotovi napájíme stejnosměrným proudem. Pro jeho zajištění používáme různé typy článků baterií.

Všechny tyto články jsou nabíjecí a pro každý typ potřebujeme nabíječku.

Používají se obvykle zapojené do série podle výše potřebného napětí.

**Všechny obvody robota (řídící deska, motory, serva, další desky) musí mít společnou zem.**

Pro řídící desky **musí** být určené samostatné články, jiné než pro pohony motorů a serv. Důvod je ten, že motory při práci produkují napěťové špičky, které by se přes společné kably dostaly do čipu a způsobily by, že čip přestane fungovat.

### 3.9.1 Přehled typů článků

#### **AAA nabíjecí baterie (mikrotužky)**

Jeden článek má napětí 1,2 V, plně nabitý i 1,4 V. Používáme pro pohon malých motorků a malých robotů (3pi). Nabíjíme klasickými nabíječkami.

#### **AA nabíjecí baterie (tužky)**

Jeden článek má napětí 1,2 V, plně nabitý i 1,4 V. Používáme pro pohon malých motorků a malých robotů, když je na robotovi dost místa na baterie. Nabíjíme také klasickými nabíječkami.

#### **Li-On články**

Nejčastěji se používají ve velikosti 18650. Jeden článek má asi 3,7 V, plně nabitý i 4,2 V. Nabíjíme je například nabíjecí deskou [TP4056](#), kterou na jedné straně připojíme k USB portu a na druhé straně pomocí drátků připájíme k držáku baterie. Používáme 2 články zapojené do série pro pohon středních robotů a napájení řídících desek, které mají vlastní stabilizaci napětí (RBControl).

#### **Ni-Cd články**

Jeden článek má napětí 1,2 V, plně nabitý i 1,4 V. Mohou dodávat desítky ampér a nelze je snadno zničit velkým odběrem. Používáme obvykle 6 nebo 10 článků v baterii pro pohon velkých robotů.

### **Li-Pol články**

Jejich výhody (nízká hmotnost a malé rozměry) při stavbě robotů obvykle nevyužijeme a kvůli relativně složitému nabíjení, určité nebezpečnosti a snadnému zničení podvýbitím je nepoužíváme.

### **Power-banky**

Poskytují stabilizované napětí 5 V a proto jsou ideální pro napájení řídících desek. Nabíjejí se přímo z USB portu.

### **USB port**

Při stavbě a tréninku často nabíjíme řídící desky přímo z USB portu, který poskytuje stabilizovaných 5 V. Podle specifikace má poskytovat 100 mA, ale běžně z něj lze odebírat 500 mV, aniž by mu to vadilo. Některé porty zvládnou i 1000 mA, ale to doporučujeme zkoušet pouze tam, kde vám nevadí, že port shoří. On by tedy shořet neměl, měl by mít ochranné pojistky, ale jeden nikdy neví ...

## **3.10 Napětí potřebná pro různé části robota a doporučené baterie**

Motory potřebují cca 6 – 12 V podle toho, jaký výkon po nich chceme. Obvykle se používá 7,2 V pro pohon středních robotů a 12 V pro pohon velkých robotů.

Servomotory se napájí 5 V, snesou i 6 V, při vyšším napětí nejspíš shoří.

Pro řídící desky obecně potřebujeme **stabilizované** napětí.

Rídící desky klonu Adruino potřebují 5 V stabilizovaného napětí. Tomu vychovuje běžná Power-banka nebo USB port.

ESP32 a jeho shieldy (ALKS) potřebují 3,3 V stabilizovaného napětí, ale protože mají na sobě stabilizátor z 5 V na 3,3 V, můžeme je také napájet z Power-banky nebo USB portu.

Pokud bychom chtěli kvůli komunikaci (například přes sériovou linku) propojit desky ESP32 a Arduino, musíme mezi ně zařadit tzv. převodník napěťových úrovní, který zajistí převod signálu z 3,3 V na 5 V a zpět.

Deska RBCcontrol má na sobě stabilizátor 7805 a tzv. step-downy pro serva, takže stačí ji připojit na 2 Li-On články zapojené do série, případně na jiný zdroj napětí 8 – 10 V.

## 3.11 Sběrnice

Sběrnice neboli **(komunikační) rozhraní** jsou domluvené postupy/systémy, jak se dva čipy nebo dvě různá zařízení dorozumívají mezi sebou. Sběrnice jsou různých typů, pro naše účely stačí znát **SPI**, **I2C** a **USART/UART**.

### 3.11.1 USART/UART

Sběrnice UART je nejjednodušší pro propojení dvou zařízení. Obě zařízení musí mít společnou zem. Pro přenos se používají piny Rx a Tx. Přitom platí, že Rx prvního zařízení se musí připojit na Tx druhého zařízení a obráceně. Na začátku se musí na obou zařízeních nastavit stejná rychlosť přenosu a další parametry: tzv. parita, start bit a stop bit. Pokud propojujete dvě Arduino desky nebo Arduino a ESP32, mají výchozí nastavení stejné. Příklad programu pro USART je v [2, strana 144]. Další příklad je v kapitole 5.4.5. Pozor, pokud propojujeme zařízení na 5 V se zařízením na 3,3 V, musíme použít [Převodník napěťových úrovní](#).

### 3.11.2 I2C

Rozhraní **I2C** umí pomocí dvou pinů (a společné země) připojit k čipu až 127 zařízení. Komunikovat s čipem (tzv. master) může vždy pouze jedno zařízení (tzv. slave), ostatní „poslouchají“ na lince, až s nimi čip zahájí komunikaci. Sběrnice I2C se neosvědčila tam, kde je větší vzdálenost mezi zařízeními než desítky cm, ideální je, když jsou všechna komunikující zařízení na jedné (řídící) desce. Pro větší vzdálenosti mezi zařízeními je vhodná sběrnice **SPI**. Příklad programu pro I2C je v [2, strana 152].

## 3.12 Osciloskop

### 3.12.1 Než začnete

Osciloskop měří velmi rychle napětí. Dokáže si napětí pamatovat, zobrazit závislost napětí na čase a v zobrazeném průběhu napětí je možné změřit řadu parametrů, např. frekvenci.

Máme možnost pracovat s digitálním paměťovým osciloskopem **Agilent/Keysight DSO-X 2024A**. Tento osciloskop má čtyři analogové vstupy (=kanály), takže lze současně měřit a zobrazovat čtyři signály. Každý kanál má svou barvu, se kterou se signál zobrazuje. Na každém kanálu může měřit napětí až do 300 V.

Dále je možnost měřit pomocí digitální sběrnice až 8 digitálních vstupů, přitom digitální vstupy umí pouze zobrazit a měřit časové parametry, dekódovat digitální sběrnici umí pouze analogové vstupy.

Přitom na vodorovné ose se zobrazuje čas, na svislé ose měřené napětí.

Maximální zobrazované napětí se nastavuje pro každý kanál zvlášť, čas je pro všechny kanály vždy stejný.

Osciloskop není stavěný pro přesné měření napětí, spíše orientační, protože na vstupu je pouze 8 bitový převodník napětí, ale měří přesně časy.

Všechna ovládací kolečka na osciloskopu se dají také stisknout.

Když podržíte libovolné tlačítko nebo kolečko 2 sekundy, objeví se k němu podrobná náповěda (anglicky).

**Menu** ke každému tlačítku se zobrazuje vždy dole na obrazovce a ovládá se tlačítky pod obrazovkou.

Tlačítko **Help** (dole uprostřed) zobrazí menu, které obsahuje mimo jiné položky **Getting started** a **Training singals**.

**Panel** osciloskopu je pravá část osciloskopu plná ovládacích prvků.

**Vzorkovací frekvence** – počet měření za sekundu. Lze nastavit až 2 GSa/s (G – giga, Sa – sample = jednotlivé měření ).

Vzorkovací frekvence se doporučuje nejméně 10x analogová šířka pásma, aby se ze singálu dalo něco poznat. Jinými slovy, pro měření signálu o frekvenci 50 kHz potřebuji nastavit vzorkovací frekvenci minimálně 500 kSa/s.

### 3.12.2 Sondy

**Sonda** (probe) je měřící kabel připojený k osciloskopu.

Sondy připojujte k osciloskopu tak, aby měly stejnou barvu jako kanál, který měří (např. první kanál má žlutou barvu).

Sondy mají klobouček s háčkem pro snadné uchopení měřeného drátu. Když se klobouček sundá, lze měřit dotekově hrotom. Každá sonda má také po-boční drát zakončený „krokodýlem“. Ten se připojuje vždy na zem měře-

ného obvodu. Pokud ho nepřipojíte, bude se vám na kabelu sondy indukovat šum z okolí, který často překryje vlastní signál.

Červený křížek na sondách slouží ke zkálibrování sond pomocí vestavěného otočného kondenzátoru a signálu **Probe**, který generuje osciloskop. Kalibraci sond obvykle provádět nemusíte, stačí ji udělat při prvním použití sond.

Sondy jsou obvykle nastavené tak, že dělí vstupní signál 10 (sonda x 10), v osciloskopu se pak nastavuje opětovné vynásobení, aby se zobrazovala správná hodnota.

Osciloskop si pamatuje poslední nastavení, takže při zapnutí není nutné sondy znova nastavovat.

### 3.12.3 Zahájení měření

Skoro všechna tlačítka v této podkapitole najdete na panelu vpravo nahoře v sekci **Run Control**

Tlačítko **Run/stop** – červená neměří, zelená měří. Osciloskop ukládá naměřené hodnoty do naplnění paměti, potom nejstarší hodnoty zahazuje a přidává nejnovější. Tlačítko **Single** – žlutá svítí – osciloskop udělá právě jednu sadu měření, kterou naplní obsah paměti a dál neměří.

Měření na sondě se zapíná a vypíná tlačítkem s číslem sondy (mezi velkým a malým otočným kolečkem na panelu dole v sekci **Vertical**).

Aby se vám měřený signál správně zobrazil, potřebujete mít optimálně nastavené rozlišení jak času, tak napětí. Pokud si nejste jistí nastavením osciloskopu, stiskněte tlačítko **Auto Scale** a osciloskop se pokusí rozlišení nastavit sám.

Dva signály, které nejsou ze stejných hodin (stejného čipu), se obvykle na monitoru posunují vůči sobě. V takovém případě vypněte měření (tlačítko **Stop**) a změřte Off-line na obrazovce, co potřebujete.

#### Nastavení času

Všechny ovládací prvky v této podkapitole najdete na panelu osciloskopu vlevo nahoře v sekci **Horizontal**

- velké kolečko – nastavení šířky periody (hodnota pro jeden dílek mřížky se zobrazuje nahoře mírně vpravo )

- malé kolečko – posun zobrazení signálu vlevo nebo vpravo (stisk: návrat do původní polohy)
- tlačítko lupa – hodně zvětší
- tlačítko **Horiz** vyvolá **menu** mimo jiné položkami:
  - **Time mode**  
volba **normal** – obvyklé měření volba **XY** umožňuje zobrazit na ose *x* napětí nebo jiné veličiny,  
volba **Roll** nastavuje měření v reálném čase – vhodné pro pomalá měření napětí (točením potenciometrem )
  - **Time Ref** – počátek měření času je umístěn vlevo, na střed, vpravo

### Nastavení napětí

Ovládací kolečka z této podkapitoly najdete na panelu osciloskopu dole v sekci **Vertical** ). Logika jejich ovládání je podobná jako u měření času.

- velké kolečko – zesílení signálu (hodnota pro jeden dílek mřížky se zobrazuje nahore vlevo )
- malé kolečko – posun zobrazení signálu nahoru nebo dolů, tzv. offset (stisk: návrat do původní polohy)

### 3.12.4 Trigger

Nastavení, od kdy přesně má osciloskop začít měřit, je v mnoha případech klíčové.

**Trigger** – říká: teď začni měřit. Trigger může mít pouze jeden vstup, který lze velmi různě navolit pomocí tlačítka **Trigger**.

tlačítko Mode Coupling tlačítko Force Trigger – okamžitě zahájí měření (v normal módu)

### 3.12.5 Význam některých tlačítek – heslovitě

Lze uložit až 10 svých nastavení osciloskopu a podle potřeby se k nim vracet.

Tlačítko **Wave gen** – modře svítí – zapnuto / nesvítí vypnuto. Generátor funkcí je popsán v návodě osiloskopu (stiskněte **Wave gen** a držte 2 sekundy).

Tlačítko **Meas** – menu měření. Zde si nastavíte, co všechno chcete měřit (až 4 veličiny zaráz).

Tlačítko **Cursors** (=pravítka, dvě vodorovná a dvě svislá) – lze tím měřit zcela manuálně cokoliv na obrazovce. Nejčastější použití – sledování PWM a signálů na sběrnicích.

Tlačítko **Refs** (referenční signály) – umí si pamatovat dva signály a srovnávat s nimi aktuální průběh

Tlačítko **Math** umí arit. výpočty se dvěma signály, taky umí Fourierovu analýzu signálu.

Tlačítko **Digital** – nastavení měření na digitálních vstupech.

Tlačítko **Serial** – serial decode mode

### 3.12.6 Ostatní

#### Menu sondy

**probe**: dělička v sondě (musí se nastavit stejně jak na sondě), obvyklá hodnota je 10:1

zapnu střídavou vazbu: odstraní stejnosměrnou složku signálu

**invert**: zobrazuje kladnou složku dolů místo nahoru

**BW limit** potlačuje signály nad 20 MHz (tuto hodnotu na tomto osc. nelze měnit) → redukuje šumy, které nás obvykle nezajímají

# Kapitola 4

## Software

Veškerý zde popisovaný a doporučovaný software je (minimálně pro vzdělávací účely) freeware.

### 4.1 Onshape

#### 4.1.1 Úvod, přihlášení, nový dokument

**Onshape** [onšejp] je relativně jednoduchý CAD program pro navrhování 3D modelů. Jeho ovládání je podobné programu SolidWorks.

#### Možnosti použití

K čemu je pro nás Onshape dobrý? Můžeme v něm vyrobit:

- hrubý návrh robota bez uvedení rozměrů pro debatu o konstrukci: „bude to vypadat asi takhle a dělat asi tohle“
- podrobný návrh včetně všech rozměrů a výkresů pro vypálení dílů na laseru
- cokoliv mezi tím

Onshape je pro vzdělávací účely zdarma s tím, že všechno, co si v něm vytvoříte, je veřejně dostupné. Je dostupný přes webový prohlížeč (Opera, Firefox, Chrome) a proto funguje na všech operačních systémech. Podmínka je, aby prohlížeč měl zprovozněné WebGL rozhraní, což některé staré grafické karty nezvládají.

Onshape je pouze anglicky, překlad některých pojmu je dále v textu.  
Pokud s Onshape začínáte, je nutné si vytvořit účet. Na webu [onshape.com](https://onshape.com) klikněte na **Sign in** (přihlášení) a po otevření přihlašovacího okna na **Sign up** (založení nového účtu).

## Nový dokument

**Dokument** je v Onshape obálka pro všechny soubory, které se týkají daného projektu.

**Projekt** je pro nás například konstrukce nového robota – 3D model robota složený z jednotlivých dílů, vazby mezi těmito díly a výkresy všech dílů.

Po prvním přihlášení do Onshape klepněte vlevo nahoře na **Create** a zvolte **Document...**

Zadejte název nového dokumentu (použijte pouze anglická písmena a číslice!) a potvrďte **OK**.

Pozn.: Stejně tak při pojmenovávání čehokoliv dalšího **používejte pouze anglická písmena a číslice**.

## Popis pracovního prostředí

Otevře se hlavní okno programu. Nahoře jsou ikony pro úpravy dílů. Vlevo je panel se seznamem všech geometrických prvků v projektu (díly, skicky, pomocné roviny atd.). Uprostřed jsou tři hlavní roviny a počátek souřadnic (origin). Vpravo spíše nahoře je „kostka“, která ukazuje, jak je vytvářený díl nebo sestava právě otočená.

## Nastavení pracovního prostředí

Nastavte si v Onshape stejné ovládání, jako je v SolidWorks: klikněte vpravo nahoře na svoje jméno, zvolte **My account**, vlevo **Preferences** a níže na stránce **View manipulation**.

## Návody, návodě

Vpravo nahoře je tlačítko **Learning Center**. Obsahuje velké množství krátkých videí, které vás programem krok za krokem provedou. **Videa** jsou pouze anglicky, ale dobře srozumitelná. Pod každým videem je napsané všechno,

co je ve videu řečeno. Pokud jenom trochu umíte anglicky, doporučuji je shlédnout a nebo přečíst, dá vám to hodně.

Kompletní přehledná nápověda k Onshape je [zde](#).

Pro všechny ikony v Onshape platí, že když na ně najedete myší, objeví se jejich název. Když počkáte několik sekund, objeví se nápověda.

## Označení

Označení provedete kliknutím myši a tažením. Vytvoří se obdélník. Když táhnete myší zleva doprava, označí se pouze to, co je zcela uvnitř obdélníka. Když táhnete zprava doleva, označí se vše, co je alespoň částečně uvnitř obdélníka. Odznačení všech označených dílů zajistí mezerník (klávesa **Space**).

## Posunutí, otočení a přiblížení

Posunutí prvku: **Shift** + šipka směru, kam chceme posouvat nebo **Ctrl** + stisknuté kolečko myši.

Prvky je možné přiblížit nebo oddálit otáčením kolečka myši. Přitom se přiblížujeme k bodu, na který právě myš ukazuje.

Stisknutím kolečka a posunem myši se otáčí dané prvky.

Chování popsané zde odpovídá nastavení **SolidWorks** (to doporučujeme, protože SolidWorks budete na SPŠ Sokolská časem probírat).

## Plocha, objem, hmotnost

V části díly (Parts) vlevo dole označíte díly. Vpravo dole se objeví ikona „váhy“. Klepnutím na ni se otevře okno, kde je spočtená plocha povrchu (Surface area) a objem (Volume) označených dílů. Pokud je zadaný materiál (klikněte pravým tlačítkem na díl v seznamu vlevo dole, z menu vyberte **Assign material...**), zobrazí se i hmotnost (mass) a další parametry.

### 4.1.2 Postup práce v Onshape

1. pro každý díl vytvoříte skicu ve 2D – kapitola [4.1.3](#)
2. ze skicy vytvoříte díl ve 3D – kapitola [4.1.5](#)

3. díly poskládáte do sestavy a zkонтrolujete, že k sobě správně pasují – kapitola [4.1.6](#)
4. vytvoříte DWG soubor ze všech dílů → podklad pro řezání na laseru – kapitola [4.1.7](#)

### 4.1.3 Výroba skici

Skica (Sketch) je dvourozměrný podklad pro tvorbu dílů ve 3D.

Klepnutím zvolte rovinu, ve které chcete skicu vytvářet. Klepnutím na tlačítko **Sketch** vlevo nahoře založte ve zvolené rovině novou skicu. Zároveň se ikony nahoře změní z ikon pro úpravy dílů na ikony pro úpravy skici. Zvolte z nich např. kružnici, klepněte na počátek a tažením vytvořte kružnici na skici. Podobně můžete vytvořit úsečku nebo obdélník. Pomocí dalších ikon lze vytvořit mnohem složitější tvary – více v kapitole [4.1.4](#). Nejvíce se ale naučíte, když si všechny ikony vyzkoušíte.

Pomocí „kostky“ vpravo nahoře zvolte vhodnou orientaci skici.

Abychom mohli ze skici nebo modelu vyrobit výkres, musí být tzv. úplně určená.

Skica je **úplně určená**, když má zadané všechny rozměry a také polohu (vzdálenost) od počátku nebo od bodu nebo čáry, která je vztažená k počátku. Úplně určená skica je černá, dokud není úplně určená, má neurčené čáry modré.

Rozměry zadáváte pomocí ikony „kóta“. Při zadávání rozměrů a polohy se velmi doporučuje využívat vazeb a proměnných – více viz [4.1.4](#).

Rozměry se vloží tak, že se číslo napíše ihned po dokončení daného geom. prvku do skici nebo se může doplnit později po kliknutí na ikonu kóty (dimensions).

Rozměry se průběžně zobrazují vpravo dole po klepnutí na prvek (úsečka, plocha), jehož rozměr nás zajímá.

Hotovou nebo rozpracovanou skicu uzavřeme pomocí zeleného zatržítka.

Mimochodem, Onshape nemá Save, vše je automaticky ukládáno do cloudu. Při větším počtu skic a dílů doporučuji skici a díly smysluplně přejmenovávat (opět pouze anglická abeceda a čísla): klikněte pravým tlačítkem na název dílu a zvolte **Rename**.

#### 4.1.4 Možnosti skici

##### Proměnné

Onshape umí přiřadit název danému číslu – vytvořit **proměnné** (variable). To znamená, že na začátku práce si dané číslo (např. tloušťka překližky) nazvete v menu pro díly pomocí ikony **Variable**, např. **tloustka 3 mm**. Tento název potom používáte všude v dokumentu. Název musí být zavedený před jeho prvním použitím (musí být v menu vlevo výše, než všechny skici nebo díly, které ho používají). Když se posléze ukáže, že potřebujete překližku o síle 4 mm, stačí tuto hodnotu změnit na jediném místě. To je obrovská pomoc, pokud například při konstrukci používáte tzv. zámečky – a to byste měli, pokud má robot držet pohromadě a být pevný.

##### Vazby

Vazby ve skice můžete zobrazit nebo skrýt pomocí zatržítka **Show constraints** – třetí rádek pod zeleným zatržítkem, které uzavírá skicu.

Pro vložení vazby (constrain) do skici jsou určeny ikony vpravo od ikony kóty. Přehled důležitých vazeb (ikony zleva doprava):

**shodnost** (Coincident) = dvě stejné entity (body, kružnice, ...) se sloučí do jedné

**středová souměrnost** (dvou kružnic) Concentric

**rovnoběžnost** (Parallel)

**tečna** (Tangent)

**vodorovný směr** (Horizontal)

**svislý směr** (Vertical)

**kolmost** (Perpendicular)

**stejný rozměr** (Equal)

**střed úsečky** (Midpoint)

Další ikony se už moc nepoužijí s vyjímkou **osové souměrnosti** (Symetric), kde se nejdřív vybírá osa, potom čáry, které se mají zrcadlit.

Například když víte, že budete mít v podvozku 8 stejných děr pro uchycení sloupků, zadáte rozměr pouze první z nich a ostatním zadáte vazbu stejný rozměr. Když je pak nutné změnit průměr sloupu, provedete změnu pouze na jednom místě.

## Další možnosti

Některé další možnosti při úpravě skici:

trim – vystříhnutí dané křivky "od bodu k bodu"

fillet – zaoblení

offset – zdvojení hran a jejich odsazení

mirror – zrcadlení = osová souměrnost – nejdřív se vybírá osa, potom čáry, které se mají zrcadlit

linear pattern – dvourozměrné lineární pole, pod ním je ještě kruhové pole a otočení/transformace

### 4.1.5 Výroba dílu – poznámky

Nový díl založíte pomocí tlačítka + vlevo dole. Z menu vyberete **Create Part Studio**. Otevře se nová záložka, ve které pomocí skici začnete tvorit nový díl.

Ze skici vytvoříte díl (part) ve 3D, nejčastěji pomocí příkazu vytažení (Extrude) – první ikona zleva na panelu ikon.

U skic lze při výrobě dílu průběžně zapnout a vypnout viditelnost (ikona „očičko“ vpravo u názvu dílu).

Editaci dílu uložíme klepnutím na zelené tlačítko podobně jako u skici.

Barva dílu: klikněte na díl (seznam vlevo) pravým tlačítkem a z menu vyberte **Edit appearance**.

V jedné záložce „Part studio“ můžete vytvořit více dílů – to se doporučuje, pokud budou spolu díly úzce souviseť a rozměry jednoho dílu využijete při návrhu druhého dílu.

### 4.1.6 Výroba sestavy

Novou sestavu (assembly) založíte pomocí tlačítka + vlevo dole. Z menu vyberete **Create Assembly**.

Každá sestava může být **podsestavou** (částí) jiné sestavy.

Díly, skici, povrchy nebo podsestavy se vkládají do sestavy pomocí tlačítka **insert** vlevo nahore. První díl v sestavě se musí ručně upevnit (fix) vůči počátku sestavy. Umístěte díl podle potřeby, potom klikněte pravým tlačítkem na díl a zvolte **fix**.

Do sestavy můžete vkládat díly, skici, povrhy nebo sestavy, svoje i kohokoliv jiného.

Pokud se stejný díl vkládá do jedné sestavy vícekrát (šroubky, kola, atd. ), nazývají se jednotlivé vložené části **instance** dílu. Pokud změníte díl, změní se všechny instance dílu v sestavě. Další instance stejného dílu se vloží takto: vlevo v soupisu dílů klikněte pravým tlačítkem na díl a zvolte **Copy**. Potom klikněte opět pravým tl. do plochy, kde tvoříte sestavu a zvolte **Paste**. Můžete taky použít klasické **Ctrl+C, Ctrl+V**.

Pohled na díly v řezu (section view): klikněte na „malou kostku vpravo“ a vyberte **Turn section view on**, následně vyberte rovinu, podle které má řez probíhat.

### Vložení dílů ze SolidWorks

Do sestavy jde vložit i díly vymodelované v SolidWorks (přípona .SLDPRT). Tyto se napřed musí importovat (klik na **Onshape** vlevo nahoře, potom na **Create** pod ním a zvolit **Import files...**) Importovaný díl vytvoří vlastní dokument. Potom se musí u tohoto dokumentu vytvořit alespoň jedna verze (poklepání otevřete dokument, vlevo nahoře mezi **Onshape** a názvem projektu jsou tři ikony, klikněte na prostřední a použijte tlačítko **Create version** ).

### Vazby v sestavě

Díly na skutečném robotovi jsou spolu spojeny, nejčastěji napevno nebo se mohou navzájem otáčet. V Onshape se takové upevnění zadává pomocí tzv. **vazby** (mate). Jde o jiné vazby, než ve skice a angličtina pro ně má jiný název. Pokud byste vazby nezadali, budou díly v sestavě „plavat“ nezávisle jeden na druhém.

Přehled možných vazeb mezi díly a jejich použití je [zde](#). Využijete především pevné spojení (Fastened mate) a otáčení kolem osy (Revolute mate). Ikony pro všechny vazby najdete na horním panelu ikon (pokud jste přepnutí na záložku sestavy).

#### 4.1.7 Výroba výkresů/příprava pro laser

#### 4.1.8 Slovníček pro Onshape

assembly – sestava nebo podsestava, například celý robot nebo podvozek

constrain – vazba v rámci skici  
dimensions – kóty = rozměry  
extrude – vytažení  
fillet – zaoblení  
linear pattern – dvourozměrné lineární pole  
mate – vazba v rámci sestavy (složené z dílů)  
mirror – zrcadlení = osová souměrnost - nejdřív se vybírá osa, potom čáry, které se mají zrcadlit  
offset – zdvojení hran a jejich odsazení  
origin – počátek soustavy souřadnic  
part – díl = součástka  
part studio – tady se vytváří nové součástky  
sketch – skica = nákres  
trim – vystříhnutí dané křivky „od bodu k bodu“

## 4.2 Visual Studio Code

**Visual Studio Code** (zkráceně VS Code) je pokročilý textový editor od Microsoftu, speciálně navržený pro programátory čehokoliv. Jde o program, který toho hodně umí sám a ještě mnohem víc se toho může naučit, pokud do něj doinstalujeme další rozšíření, tzv. **pluginy**, například [PlatformIO](#), což je plugin speciálně zaměřený na programování čipů.

### Postup

Pokud s programováním čipů začínáme, čekají nás tyto úkoly:

1. nainstalovat prostředí *Visual Studio Code*
2. do Visual Studio Code nainstalovat doplněk *PlatformIO*
3. přes PlatformIO založit nový projekt
4. napsat zdrojový kód, přeložit a dostat jej do čipu

Toto vše podrobněji probereme na dalších řádcích.

### 4.2.1 Nainstalujte Visual Studio Code

Instalujte podobně jako každý jiný program, stahujte zde: <https://code.visualstudio.com/>

**POZOR:** Pokud jméno vašeho uživatelského účtu na PC obsahuje diakritiku (čárky, háčky), speciální znaky (znaky mimo ASCII tabulku) nebo i mezery, můžete mít problémy s používáním PlatformIO.

Prakticky jediné funkční řešení tohoto problému je vytvořit si na PC nový uživatelský účet a používat ho pro práci s **VS Code** a **PlatformIO**.

### Nainstalujte PlatformIO

**PlatformIO** (zkráceně PIO) je ten software, který umožní program v C++ přeložit tak, aby ho čip pochopil a taky ho do čipu umí nahrát. Instalace podle návodu zde: <http://docs.platformio.org/en/latest/ide/vscode.html#installation>

### Založte nový projekt

Program (ne)píšete jen do jednoho souboru, ale aby vše fungovalo, potřebujete vícero dalších souborů, které dohromady tvoří tzv. **projekt**. Tyto soubory jsou mezi sebou hodně provázané, takže v případě přesunu projektu například z kroužku domů je potřeba zkopirovat celý adresář projektu.

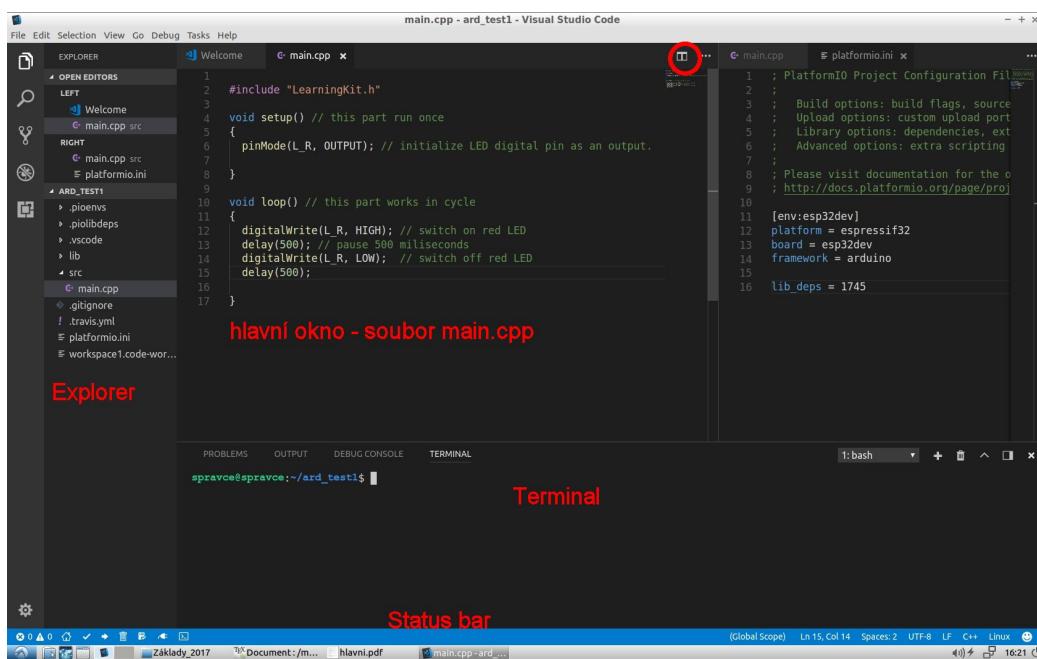
1. Založte nový projekt z ikonky „domeček“ – viz <http://docs.platformio.org/en/latest/ide/vscode.html#quick-start>.
2. Do kolonky *Board* se musí vybrat správná deska. Desek je přes 400 a jsou rozdělené do sekcí řazených podle abecedy vyznačených šedivou barvou. Vám ale stačí kliknout na kolonku *Board* a na klávesnici napsat:
  - (a) *Espressif ESP32 Dev Module* a vybrat tuto desku, pokud používáte samotnou vývojovou desku ESP32 DevKitC s modulem ESP32-WROOM32
  - (b) *ALKS ESP32* a vybrat tuto desku, pokud využíváte výukový kit ALKS (viz sekce [3.7.2](#))

Kolonka *Framework* se potom vyplní automaticky.

- Zbývá vybrat adresář, do kterého bude projekt uložen. Tento adresář si předem vytvořte, s adresářem vytvářeným za pochodu má VS Code kdovíproč problém. Odškrtněte zatržítko *Use defalut folder* a zvolte vámi vytvořený adresář. Pokud si zapomenete nastav vlastní složku pro projekt, PIO projekt vytvoří někde v rámci složky Dokumenty. Nejjednodušší je vytvořit znova nový projekt.

Pro Linux Lubuntu: projekt musí být uložen na pevném disku, ne na flešce, jinak prostě nepojede, netuším proč.

### Napiště zdrojový kód, přeložte jej a nahrejte jej do čipu



Obrázek 4.1: Rozložení oken v programu *VS Code*

Obrázek 4.1 na straně 47 ukazuje rozložení oken v rámci projektu. Hlavní okno rozdělte na dvě části pro zobrazení dvou upravovaných souborů pomocí ikony v kroužku. Začínáme v okně **Explorer**, kde je umístěna adresářová struktura projektu<sup>1</sup>. Otevřete soubory *platformio.ini* a v adresáři *src* soubor *main.cpp*.

<sup>1</sup>Pokud toto okno není vidět, zobrazíte ho v menu *View* položka *Explorer*

Pro pohodlnou práci s deskou ALKS byla napsaná tzv. knihovna *ArduinoLearningKitStarter*. Aby fungovala, musí být do souboru *platformio.ini* dopsán řádek `lib_deps = 1745` (bez mezery na začátku řádku) a do záhlaví souboru *main.cpp* doplňte `include "ALKS.h"` Dále dopište do souboru *main.cpp* kód, který bliká červenou LED. Vše je vidět na obrázku 4.1. Celý zdrojový kód tohoto prvního programu (obsah souboru *main.cpp*) je uveden v kapitole 5.4.

Ted' budou potřeba další dvě části VS Code: terminál (okno vpravo dole) a stavový řádek (Status bar – proužek pod terminálem). Na stavovém řádku klikněte na ikonu šipky<sup>2</sup> (pátá zprava) a *PlatformIO* se pokusí váš program přeložit a nahrát do čipu. Pokud chcete program pouze přeložit, klikněte na ikonu zatržítka<sup>3</sup> hned vedle.

Při prvním pokusu nahrát program do čipu na Linuxu může mít *PlatformIO* problém, že nenajde USB spojení na desku s čipem a vyžaduje ho doinstalovat. Zpráva<sup>4</sup> se objeví v terminálu včetně návodů,<sup>5</sup> jak to udělat. Návod je ale tak podrobná, že to středně poučený linuxový laik s pomocí internetu zvládne. Při všech dalších překladech už to nebude problém.

Další programy budou uvedeny v kapitole 5.4.

## 4.3 Cpp4Robots

**Cpp4robots** je rozšíření do vývojového IDE *Microsoft Visual Studio* (to není *Visual Studio Code*) určené pro programování robotů řízených „Lego kostkou“ EV3 v jazyce C++.

Instaluje se takto:

- nainstalujte Microsoft Visual Studio Community 2017
- pokud nemáte, nainstalujte Java
- nainstalujte Cpp4Robots
- zprovozněte Cpp4Robots

---

<sup>2</sup>totéž provede *Ctrl+Alt+U*

<sup>3</sup>totéž provede *Ctrl+Alt+B*

<sup>4</sup>Warning! Please install ‘99-platformio-udev.rules’

<sup>5</sup><https://raw.githubusercontent.com/platformio/platformio/develop/scripts/99-platformio-udev.rules>

- **zapněte** zobrazení nástrojové lišty pro Cpp4Robots; součástí této lišty je také ikona nápovědy

Kostka EV3 lze programovat přes USB kabel nebo Bluetooth. Aby robot fungoval autonomně i bez kabelu, musí být v kostce Micro SD karta, doporučená velikost je 8 – 32 GB.

Oficiální dokumentace na [www.cpp4robots.cz](http://www.cpp4robots.cz).

## 4.4 Lorris

**Lorris**<sup>6</sup> je rozsáhlá sada nástrojů, které mají společný cíl – pomáhat při vývoji, ladění a řízení zejména robotů, ale i jiných elektronických zařízení. V současnosti neexistuje jiná volně dostupná aplikace, která by umožňovala dostatečně jednoduše v téměř libovolném formátu zobrazit data přicházející z čipů nebo i data ze souborů.

### Hlavní možnosti použití:

- grafické zobrazování, přiřazování a analýza (binárních) dat z čipů
- vykreslování příchozích dat v grafu
- zpracování dat z jednoho zdroje ve více modulech současně
- zobrazení dat z více zdrojů na jednom místě – ideální pro ladění komunikace mezi více zařízeními
- možnost libovolných úprav příchozích i odchozích dat pomocí skriptů v jazyce Python
- simulace chování plánovaného robota – hledání a ladění strategií
- simulace dat ze senzorů (zatím) neexistujícího robota pro psaní a ladění programu
- vytváření vlastních ovládacích prvků – například ovládání robota joystickem z počítače

Lorris naprogramoval Vojtěch Boček a popsal ji podrobně ve své práci SOČ: <http://soc.nidv.cz/archiv/rocnik35/obor/18>.

Video s krátkým představením Lorris: <http://www.youtube.com/watch?v=LkmFn40BbX8>.

---

<sup>6</sup><http://tassadar.github.io/Lorris/cz/index.html>

Příklad posílání dat pro Lorris.

## 4.5 L<sup>A</sup>T<sub>E</sub>X

### 4.5.1 Proč používat L<sup>A</sup>T<sub>E</sub>X

Tento text je psán v sázecím systému L<sup>A</sup>T<sub>E</sub>X[leitech]. Jeho silná stránka je především matematická sazba (bohužel nevyužijeme) a snadné zpracování obsahu, rejstříků, seznamů obrázků a tabulek a podobně, což dramaticky urychluje přípravu dokumentu.

Čas, který vložíte do nastavení a učení se systému, se vrátí v rychlosti práce → jedná se o řešení vhodné pro delší dokumenty, např. pro soutěž SOČ nebo dlouhodobou maturitní práci<sup>7</sup>.

Návody pro L<sup>A</sup>T<sub>E</sub>X lze najít na internetu, pro úvodní zorientování doporučuji text *L<sup>A</sup>T<sub>E</sub>X pro pragmatiky*<sup>8</sup>.

Hodně vám také může pomoci zdrojový text této dokumentace, především hlavní soubor, kde je nastavení podrobně komentované.

Příkazy L<sup>A</sup>TEXu podobně jako C++ a systémy typu Linux rozlišují velká a malá písmena.

### 4.5.2 Instalace a editory pro L<sup>A</sup>T<sub>E</sub>X

Podobně jako Linux, je i L<sup>A</sup>TEX dostupný v řadě distribucí. Doporučuji buď TeXLive<sup>9</sup> nebo MiKTeX<sup>10</sup>.

Jako editory ve WinXP používám PSpad<sup>11</sup>, v linuxu TeXstudio<sup>12</sup>. Oba editory umí zavolat překlad do pdf pomocí klávesové zkratky, zobrazit výsledný pdf a barevné zvýraznění syntaxe. TeXstudio má navíc velké možnosti pro zrychlení práce.

---

<sup>7</sup><https://github.com/RoboticsBrno/Latex-slideshow-czech>.

<sup>8</sup><http://mirrors.nic.cz/tex-archive/info/czech/latex-pro-pragmatiky/latex-pro-pragmatiky.pdf>

<sup>9</sup><https://www.tug.org/texlive>

<sup>10</sup><https://miktex.org/>

<sup>11</sup><http://www.pspad.com/>

<sup>12</sup><https://www.texstudio.org/>

Další možností je tvořit latexové dokumenty online bez nutnosti instalace, například pomocí služby [overleaf](#).<sup>13</sup> Overleaf je online služba, která vám umožňuje psát, sdílet a komentovat LaTeXové dokumenty – ideální pro psaní SOČ. Úvod do možností služby je [zde](#).

### 4.5.3 LATEX a SOČ

Jarek Páral vytvořil na Overleafu LaTeXovou šablonu<sup>14</sup> pro SOČ, kde je řada prvků sazby už optimálně přednastavená. Šablona navíc obsahuje informace o tom, jak a co psát, o citacích a dalších věcech – vřele doporučuji.

Pro inspiraci také doporučuji práce Vojty Bočka, Honzy Mrázka, Jarka Párala, Bédi Saidy a Martina Sýkory, dostupné v archivu SOČ<sup>15</sup>, ročníky 32. - 37., kategorie informatika a elektro.

### 4.5.4 Další inspirace k SOČ

Jak na psaní SOČ:

- <http://www.soc.cz/soc-krok-za-krokem/>
- <http://www.jcmm.cz/cz/podpora-soc.html>

Šablony SOČ pro Word

- [http://www.soc.cz/dokumenty/sablona\\_SOČ.docx](http://www.soc.cz/dokumenty/sablona_SO%C4%8D.docx)
- [http://www.jcmm.cz/data/sablona\\_pro\\_sockare.docx](http://www.jcmm.cz/data/sablona_pro_sockare.docx)

Doporučuji si obě šablony přečíst, i když je třeba nepoužijete – jsou v nich zajímavé a užitečné rady.

## 4.6 Git

### 4.6.1 Základní pojmy

Git je program, který umí uchovávat jednotlivé verze souborů, zobrazovat

<sup>13</sup><https://www.overleaf.com/>

<sup>14</sup>Tady je k dispozici šablona, kterou si můžete na Overleafu zkopirovat jako vlastní projekt a začít tvořit: <https://www.overleaf.com/read/gvqvqzwgdtwk>

<sup>15</sup><http://www.soc.cz/archiv-minulych-rocniku/>

rozdíly mezi nimi, slučovat změny více uživatelů a tak dál. Pro vývoj softwaru pro roboty je to naprosto nezbytný nástroj.

Podrobnější představení gitu je [zde](#).

**Github<sup>16</sup>** je v současnosti jeden z nejpoužívanějších webů pro tvorbu a správu repozitářů.

**Repozitář** je skupina souborů nějakého projektu, která navíc obsahuje komentovanou historii všech změn projektu. Používá se pro zálohování programů a sdílení a společné týmové práci na rozsáhlejších projektech, především programátorských a textových.

Repozitář, ve kterém je i tato dokumentace, je na adrese <https://github.com/RoboticsBrno/RobotikaBrno-guides/tree/RoboticsManual>.

Pro práci s repozitáři je důležitý pojem **commit** – je to jeden „kus hotové práce“. Například naprogramování nové funkce, přidání kapitoly do textu a podobně. Ke každému commitu píšeme při vytvoření anglicky komentář, aby bylo jasné, čeho se daný commit týká.

Postup práce je následující: na webu github.com si vytvoříte repozitář. Na svůj počítač si nainstalujete git. Stáhnete k sobě na počítač aktuální verzi repozitáře, upravíte, co potřebujete, vyrobíte commit a upravené soubory nahrajete zpět na server. Podrobněji v návodech níže. Neustálému komentovanému ukládání jednotlivých verzí se říká **verzování**. Je to vlastně podstatně vylepšené zálohování.

Verzování na web github lze provádět z příkazové řádky (terminálu) nebo s pomocí různých programů, například prostředí **VSCODE** – viz dále. Také lze soubory upravovat přímo na serveru github.com pomocí ikony „tužka“.

Návody pro github jsou například zde:

- Github na příkazovém řádku v Linuxu – článek o základech
- Oficiální help (EN)
- Podrobná knížka o gitu (CZ)
- Další knížka o gitu (CZ)
- Velmi názorný web a tutoriál o gitu (EN)
- Nastavení repozitáře (EN)

---

<sup>16</sup>[www.github.com](http://www.github.com)

#### 4.6.2 Instalace gitu a stažení repozitáře – Linux

1. Na webu <https://github.com> si vytvořte účet a přihlaste se. Vpravo můžete založit nový repozitář (new repository) nebo se můžete přepnout do už existujících repozitářů.
2. Na svém počítači si nainstalujte git. Napište do terminálu: `sudo apt-get install git`.
3. Vytvořte si lokální složku a stáhněte do ní repozitář. Přepněte se do složky, ve které chcete repozitář mít a napište do terminálu: `git clone <cesta_k_repozitáři_na_webu>`. Git vytvoří v aktuální složce pod-složku s kopíí repozitáře z webu. Cestu potřebnou pro příkaz `clone` zjistíte tak, že se na webu [github.com](https://github.com) přepnete do požadovaného repozitáře a kliknete na zelené tlačítko **Clone or Download**. Objeví se okno s cestou k repozitáři a také s tlačítkem **Download ZIP**, které umožnuje stáhnout celý repozitář jako ZIP soubor.
4. Pokud má repozitář více tzv. větví, přepněte se do požadované větve příkazem `git checkout <název_větve>`, například `git checkout RoboticsManual`.

#### 4.6.3 Instalace gitu a stažení repozitáře – Windows

1. Na webu <https://github.com> si vytvořte účet a přihlaste se. Vpravo můžete založit nový repozitář (new repository) nebo se můžete přepnout do už existujících repozitářů.
2. Na svém počítači si nainstalujete git. Stáhněte instalační program z adresy <https://git-scm.com/download> a instalujete jako libovolný jiný program.
3. Dále stáhněte a nainstalujete program [TortoiseGit](#).
4. Po restartu se v programech pro práci se soubory (například *Průzkumník*, *Total Commander*, *Free Commander*, *Altap Salamander* a podobně) objeví v kontextovém menu (po stisku pravého tlačítka nad zvoleným souborem nebo adresářem) nabídka pro práci s Gitem.
5. Klikněte pravým tlačítkem na adresář, kam chcete stáhnout repozitář, a z nabídky vyberte **Git clone...**
6. Do kolonky URL zadáte cestku k repozitáři. Cestu k repozitáři zjistíte tak, že se na webu [github.com](https://github.com) přepnete do požadovaného repozitáře a kliknete na zelené tlačítko **Clone or Download**. Objeví se okno s

cestou k repozitáři a také s tlačítkem Download ZIP, které umožňuje stáhnout celý repozitář jako ZIP soubor. Kliknutím na OK stáhnete repozitář.

#### 4.6.4 Vytvoření commitu a nahrání změn na server pomocí terminálu – Linux

1. Změny, které chcete zahrnout do commitu, musíte do nejprve přidat příkazem `git add <vybrané_zmeny>`. Příkaz `git add .` přidá všechny nové změny. Všimněte si, že před tečkou je v příkazu `git add .` mezera. Příkaz je možné libovolně opakovat.
2. Pokud si nejste jistí, co vše už je nebo není přidáno, použijte příkaz `git status`
3. Příkaz `git commit` vytvoří nový commit ze všech změn přidaných příkazem `add`. Co nepřidáte, to v commitu nebude! Protože chceme vědět, co jsme v minulosti dělali, použijeme tvar: `git commit -m "Tady je napsáno, co tento commit obsahuje."` I commitů můžeme vytvořit více za sebou. Všechny zůstávají na lokálním počítači, dokud je nepošlete na server. ih
4. Příkaz `git push` pošle všechny vytvořené a dosud neposlané commity na server. Vyžaduje přihlášení a heslo. Nezapomeňte, že v Linuxu se heslo nezobrazuje, ani ve formě teček.
5. Pokud už na serveru provedl změny někdo jiný a potřebujete je stáhnout na svůj počítač, použijete `git pull`.

#### 4.6.5 Vytvoření commitu a nahrání změn na server pomocí TortoiseGit – Windows

1. Nové soubory nebo soubory zkopiované odjinud musíte do repozitáře ručně přidat: klikněte pravým tlačítkem na nový anebo zkopiovaný soubor a zvolte `<Git><Add...>` Úpravy existujících souborů přidávat nemusíte, TortoiseGit je přidá před commitem automaticky. Smazání souboru z repozitáře také zjistí a provede automaticky.
2. Vytvořte commit: klikněte pravým tlačítkem na adresář s repozitářem a z nabídky zvolte `Git Commit -> 'master' ...` Objeví se formulář

pro commit. Do kolonky **Message** zadejte komentář ke commitu. **Komentář je povinný a nutný!** Je nutný pro orientaci ve změnách nejen pro vás, ale i pro další, kdo repozitář používají (spolužáci, učitel, vedoucí kroužku, ... ) a nevidí vám do hlavy!

3. Dole ve formuláři klikněte na tlačítko **Commit**. Commitů můžeme vytvořit více za sebou. Všechny zůstávají na lokálním počítači, dokud je nepošlete na server.
4. Všechny vytvořené a dosud neposlané commity na server pošlete tak, že ve formuláři pro commit (viz předchozí bod) kliknete na šipku vedle tlačítka **Commit** a z nabídky vyberete **Commit & Push**. Po přihlášení a zadání hesla se commity odešlou na server.

#### 4.6.6 Formát Markdown (\*.md)

Formát **Markdown (\*.md)** umožňuje velmi snadné a rychlé základní formátování textů. Web Github podporuje psaní textů v tomto formátu. Pokud jej použijete, zobrazí se text na webu už zformátovaný. Přehled příkazů formátu \*.md je například [zde](#). Mimochodem, toto formátování podporuje i editor Visual Studio Code.

### 4.7 Další software

#### 4.7.1 Arduino IDE

**Arduino IDE**<sup>17</sup> je textový editor vhodný pro začátky v programování řídicích desek klonu Arduino. Je přímo od tvůrců Arduina. Integruje všechny základní funkce a je používán u většiny tutoriálů (návodů), ovšem většinou není problém použít jiné IDE (Atom, VSCode) s danými tutoriály. Instaluje se [standardně](#), po [doinstalování](#) zvládne i desky ESP32. Jiný návod pro doinstalování ESP32 do Arduino IDE je [zde](#) nebo [zde](#).

#### 4.7.2 Atom

[Atom](#) je výkonný textový editor (IDE), hodně podobný programu [Visual Studio Code](#). Také se podobně instaluje, napřed nainstalujte samotný program a

---

<sup>17</sup>IDE – Integrated Development Environment – integrované vývojové prostředí

potom rozšíření **PlatformIO**, které je určené pro samotné programování čipů.

### 4.7.3 Cpp4Robots

**Cpp4robots** je rozšíření programu *Microsoft Visual Studio* (to není *Visual Studio Code*) určené pro programování robotů řízených „Lego kostkou“ EV3 v jazyce C++.

Instaluje se takto:

- nainstalujte **Microsoft Visual Studio Community 2017**
- pokud nemáte, nainstalujte **Javu**
- nainstalujte **Cpp4Robots**
- zprovozněte **Cpp4Robots**
- **zapněte** zobrazení nástrojové lišty pro Cpp4Robots; součástí této lišty je také ikona nápovědy

Kostka EV3 se programuje pomocí USB kabelu. Aby robot fungoval autonomně i bez kabelu, musí být v kostce SD micro karta, doporučená velikost karty je 8 – 32 GB.

### 4.7.4 Proficad

Proficad<sup>18</sup> je software určený původně pro snadné a rychlé kreslení elektronických schémat a v této oblasti je vynikající. Lze jej použít i jako jednoduchý vektorový editor obrázků.

SPŠ Sokolská zakoupila plnou multilicenci pro Proficad, takže studenti i učitelé jej mohou používat bez omezení.

Ovládání programu je velice intuitivní a návodů prakticky nepotřebujete – s jedinou výjimkou, a tou je nastavení rastru. Po instalaci je rastr zobrazení automaticky nastaven na 2 mm. To znamená, že součástky můžete umisťovat například 10 mm nebo 12 mm od kraje, ale nic mezi tím. Většinou se to hodí – součástky máte na schématu pěkně zarovnané – ale někdy je prostě potřeba rastr například vypnout neboli nastavit na nulu. Nastavení rastru je schované zde: *soubor/nastavení/dokument/obsah/rastr*.

---

<sup>18</sup>Instalační soubor seženete v kroužku nebo na webu.

### 4.7.5 Python

Programovací jazyk Python použijete, když chcete například napsat program pro počítač, který bude komunikovat s robotem. Nepoužívá se pro programování čipů v robotovi. Návody pro Python jsou například [zde](#) a [zde](#).

### 4.7.6 Linux

Tato kapitolka není úvodem do Linuxu (materiálů na toto téma je na webu spousta). Jsou zde poznámky, které hodí, když uživatel přechází z Windows na Linux.

#### Některé rozdíly Linux – Windows

- Linux **rozlišuje velká a malá písmena**. V terminálu, v názvech souborů, všude.
- Když si chce uživatel nainstalovat Linux, vybírá ne operační systém, ale tzv. distribuci = operační systém + spousta předinstalovaných programů. Kritérií pro výběr je spousta, pár tipů, které používáme na Robotárně:
  - distribuce *Mint* má podobné rozložení grafických ovládacích prvků jak windows a je nenáročná na hardware
  - distribuce *Ubuntu* je standardní volba s velkou výbavou programů. Její odlehčená verze pro starší počítače je *Lubuntu*.
- V Linuxu se celkově mnohem víc (a účiněji) používá příkazová řádka neboli program **terminál**. Spouští se klávesovou zkratkou **Ctrl+Alt+T** nebo v Lubuntu spusťte správce souborů (2. ikona dole vlevo), přepnete se do požadovaného adresáře a stiskněte **F4**.
- Adresářová struktura Linuxu je jednotná pro všechny disky v počítači. Akce typu: „*přepni se na disk e:*“ se v terminálu provede jako „*přepni se do adresáře /media/<uzivatel>/ ...*“. Nebo se přepnete pomocí jiného programu, např. Správce souborů.
- Některé programy fungují pod Linux i Windows, u jiných se při přechodu z Windows na linux musí na webu najít odpovídající nahradu, opět pár tipů:

- Ve WinXP se pro prohlížení pdf souborů osvědčil Foxit Reader verze 2.3, v linuxu Evince.
- Irfan view jede i v Linuxu<sup>19</sup>.

## Co se hodí vědět

- Heslo v Linuxu se při zadávání v terminálu nevypisuje ani tečkami (ale počítač o něm ví!).
- Znak zavináč se napíše **AltGr+v**, znak vlna se napíše **AltGr+a**.
- Anglické znaky na české klávesnici napíšete, když použijete **AltGr**.
- Drivery do tiskáren mají příponu ppd. Před jejich použitím se musí nainstalovat program cups.
- Snímek obrazovky uložíte stiskem klávesy **PrintScreen** do adresáře `/home/<prihlaseny uzivatel>`.
- Znak # znamená řádek na terminálu (zastupuje vypsání aktuálního adresáře).
- Zkratka **Ctrl+C** v terminálu není pro kopírování, ale ukončení běžícího programu. (Mimo terminál kopírování funguje jako ve windows). Pokud chceme v terminálu kopírovat a vkládat, použijeme **Ctrl+Shift+c** **Ctrl+Shift+v**.
- V nabídce Start je také tzv. *Centrum softwaru pro Ubuntu*. Alternativně můžete Centrum spustit z příkazové řádky příkazem **software-center**.

---

<sup>19</sup><http://www.boekhoff.info/install-irfan-view-on-linux/>

# Kapitola 5

## Programování čipů v C++

### 5.1 Základní doporučení pro programování

5.1.1 title

5.1.2 title

5.1.3 Událostmi řízené programování

### 5.2 Základy syntaxe v jazyce C

5.2.1 Základní pojmy

Upozornění: V jazyce C a C++ se **rozlišují velká a malá písmena – jazyk je case sensitive**.

#### Bity a bajty

Každá číslice ve **dvojkové** soustavě reprezentuje jeden **bit** (nabývá hodnot 0 nebo 1).

Osm bitů dohromady tvoří **bajt**. Nejnižší bit v bajtu leží vpravo (tzv. **nultý bit**), další je nalevo od něj (první bit), až do sedmého bitu, který je nejvíce vlevo.

**příkaz** – povel pro procesor, říká procesoru, co má dělat. Příkazy probíhají postupně, nejprve se udělá jeden a pak další, který je na řadě. Za příkazy dáváme středník (;).

**syntaxe** – „pravopis“ programovacího jazyka, říká, jak psát příkazy programovacího jazyka tak, aby nám počítač rozuměl

**preprocesor** – před samotným překladem programu do \*.hex souboru proběhnou tzv. direktivy preprocesoru (něco jako příkazy, podrobně )

Příklady: `#include <>` – pro vkládání hlavičkových souborů

`#include ""` – pro vkládání vlastních hlavičkových souborů

`#define KONSTANTA HODNOTA_KONSTANTY` – pro definici konstanty, všude kde se v kódu vyskytne text KONSTANTA bude tento text nahrazen HODNOTA\_KONSTANTY

**proměnná** – je místo v paměti, kde jsou uložena nějaké data

**datový typ** – říká nám, v jaké podobě jsou data v proměnné uložena. V proměnné jsou pouze jedničky a nuly, pomocí datového typu mikrokontrolér pozná, jestli ta data jsou čísla, nebo znak, v jakém rozsahu jsou čísla, atd... používané datové typy:

`uint8_t` – celá čísla od 0 do 255

`int8_t` – celá čísla od -127 do 127

`uint16_t` – celá čísla od 0 do 65535

`int16_t` – celá čísla od -32767 do 32767

`uint32_t` – celá čísla od 0 do 4294967296

`int32_t` – celá čísla od -2147483647 do 2147483647

**operandy** – jsou čísla nebo výrazy, které „vstupují do operace“

**operátory** – nám říká, jaké operace provedeme s operandy (co s nimi uděláme)

Příklad: `5 + 2`, přitom 5 a 2 jsou operandy a znaménko plus je operátor, který nám říká, že čísla chceme sečítat (provést operaci sečítání)

**poznámka** – cokoliv se objeví v kódu mezi znaky /\* a \*/, tak je poznámka a bude před překladem odstraněno z kódu.

Pokud chceme zapoznámkovat pouze jeden řádek, použijeme //

Příklady:

```
/* Vše co je napsáno zde
```

```
je poznámka a nebude překládáno  
*/  
  
// Toto je poznámka, která může popsat funkci nějakého úseku kódu
```

### 5.2.2 Výrazy a operátory

Výraz je něco, co nabývá nějaké hodnoty, např.: `5 + 3` je výraz, který nabývá hodnoty `8`. `5 > 3` je výraz nabývající hodnoty `true` neboli pravda, `x <= 10` je výraz který je pravdivý, pokud proměnná `x`(kterou musíme mít deklarovanou) menší nebo rovna číslu `10`.

#### Aritmetické operátory

- + sčítání
- odčítání
- \* násobení
- / dělení

#### Operátory inkrementace a dekrementace

pokud chceme zvýšit hodnotu proměnné o jedničku, napišeme `++název_proměnné`;  
pokud chceme snížit hodnotu proměnné o jedničku, napišeme `--název_proměnné`;  
Příklad:

```
uint8_t i = 5; // Vytvoření proměnné i a dosazení hodnoty 5 do ní.  
uint8_t j = 42; // Vytvoření proměnné j a dosazení hodnoty 42 do ní.  
  
++i; // Hodnota i se zvýší o jedničku na číslo 6.  
--j; // Hodnota j se sníží o jedničku na číslo 41.
```

#### Logické výrazy a operátory

Přehled logických operátorů:  
`==` porovnání - rovnost  
`!=` porovnání - nerovnost

```

&& logický součin
|| logický součet
! negace
< menší než
<= menší nebo rovno
> větší než
>= větší nebo rovno

```

**Logický výraz** může nabývat pouze dvou hodnot – pravda (true) a nepravda (false). Obvykle je nepravda reprezentovaná nulou a pravda každým nenulovým číslem. Logické výrazy se používají v podmínkách (viz např. 5.2.3)

**Logické operace** se používají při vyhodnocování logických výrazů.

Příklady:

a == b výraz je pravdivý, pokud se a rovná b  
a != b výraz je pravdivý, pokud se a nerovná b lze to napsat i konkrétněji, např.: a != 5, výraz je pravda pokud se a nerovná 5, pokud se rovná výsledkem výrazu je nepravda(false) b > c výraz je pravdivý, pokud je b větší než c  
b >= d výraz je pravdivý, pokud je b větší nebo rovný d

!(a > b) výraz v závorce je pravdivý, pokud je a větší než b, ale pak je negováno (z pravdy se stává nepravda a naopak), tj. celý výraz je nepravdivý, pokud je a větší než b, pokud je a menší nebo rovno, tak je výraz pravdivý.

!(a == b) výraz (a == b) je negován znaménkem !, to znamená, že výraz je pravdivý, pokud se výraz a nerovná výrazu b, v podstatě se to dá napsat i takto: a != b

Pozor: negace výrazu (a > b), tj. !(a > b) není to samé jako výraz (a < b), ale správně je to (a <= b)<sup>1</sup>

Logický součin **&&** se používá, pokud budeme potřebovat spojit dva nebo více výrazů dohromady, např.: (a > b)&&(c == d), výsledkem tohoto výrazu bude logický součin výrazů v závorkách, pro logický součin platí, že je pravda pokud oba výrazy jsou pravdivé, jinak je výsledek nepravda, tj. zde

---

<sup>1</sup>Platí zde určité zákonitosti viz De Morganovy zákony

bude pravda pouze pokud bude **a** větší než **b** a zároveň bude platit, že **c** se rovná **d**. Logický součin použijeme, pokud musí všechny výrazy být pravda. Logický součet **||** je pravdivý, pokud alespoň jeden výraz je pravdivý. Např.: **(e <= f) || (g != 3)** výraz bude pravda, pokud bude platit, že **e** je menší nebo roven **f**, nebo bude platit, že **g** se nerovná **3**, anebo klidně budou platit oba výrazy.

### 5.2.3 Nejčastější příkazy C++

#### Přiřazovací příkaz

Použijeme, pokud chceme, aby se do proměnné uložila nějaká data.

Syntaxe:

**datový\_typ název\_proměnné;**

syntaxe vkládání:

**název\_proměnné = hodnota\_která\_se\_má\_uložit;**

Data do proměnné můžeme vložit rovnou při vytváření proměnných:

**datový\_typ název\_proměnné = hodnota;**

Příklad: **uint8\_t b = 5;** Vytvoří se proměnná pojmenovaná **b** (která je v rozsahu od 0 do 255) a uloží se do ní číslo 5.

**int16\_t B;** Vytvoří se proměnná **B**. **B** není to samé jako **b**, protože jazyk C rozlišuje VELKÁ a malá písmena.

**B = 1024;** Do proměnné **B** se uloží hodnota 1024;

#### Blok příkazů

Pokud chceme někam dát více příkazů, ale můžeme tam dát pouze jeden příkaz, tak je dáme do složených závorek Příklad:

```
{  
    PŘÍKAZ1;  
    PŘÍKAZ2;  
    PŘÍKAZ3;  
}
```

## Podmiňovací příkaz (if)

Použijeme, pokud chceme, aby se program mohl rozhodnout na základě nějaké podmínky.

syntaxe:

```
if(PODMÍNKA)
    PŘÍKAZ1;
else
    PŘÍKAZ2;
```

Pokud platí PODEMÍNKA v kulaté závorce, vykoná se PŘÍKAZ1, pokud neplatí, vykoná se PŘÍKAZ2. Za **if** nebo **else** může být pouze jeden příkaz, pokud jich tam chceme dát více, použijeme blok. Větev **else** je nepovinná.

Příklad:

```
if(a > 5) // Pokud platí, že
            // proměnná a je větší než 5, tak se
{
    b = a; // do proměnné b uloží hodnota, která je v proměnné a
    c = 2; // do proměnné c se uloží číslo 2.
} // Pokud podmínka neplatí, tak se nevykoná nic.
```

podmínka může být výraz např.:

```
a > 5 // podmínka bude platit,
        // pokud proměnná a je větší než 5
```

podmíněné příkazy lze vnořovat - v bloku příkazů může být další if Příklad:

```
if(PODMÍNKA1)
{
    PŘÍKAZ1;
    if(PODMÍNKA2)
    {
        PŘÍKAZ2;
    }
    PŘÍKAZ3;
}
```

```
else
{
    PŘÍKAZ4;
}
```

Pokud platí PODMÍNKA1 tak se vykoná PŘÍKAZ1, pak se zkontroluje PODMÍNKA2, pokud platí, tak se vykoná PŘÍKAZ2, pokud ne, program pokračuje dál a vykoná PŘÍKAZ3. V případě, že neplatí ani PODMÍNKA1, tak se blok příkazů přeskočí a vykoná se blok příkazů za `else`, tj. PŘÍKAZ4.

Je tu možnost také do větve `else` napsat daší `if else`, vypadá to takto:

```
if(PODMÍNKA1)
{
    PRÍKAZ1;
}
else
{
    if(PODMÍNKA2)
    {
        PRÍKAZ2;
    }
    else
    {
        PRÍKAZ3;
    }
}
```

Pokud platí PODMÍNKA1, tak se vykoná PŘÍKAZ1, pokud ne, tak se zkontroluje PODMÍNKA2, pokud platí tak se vykoná PŘÍKAZ2, pokud neplatí, tak se vykoná PŘÍKAZ3. Výše zapsaný kód sice bude fungovat, ale není přehledný, a proto z důvodu lepší orientace v kódu zvolíme tento zápis:

```
if(PODMÍNKA1)
{
    PRÍKAZ1;
}
else if(PODMÍNKA2)
{
```

```
    PRÍKAZ2;  
}  
else  
{  
    PRÍKAZ3;  
}
```

### Příkaz několikanásobného větvení (switch)

Pokud by takto zapsaných podmínek bylo moc a vztahovali by se k jedné řídící proměnné lze použít příkaz několikanásobného větvení neboli switch. Tak například místo zápisu:

```
if(a == 1)  
{  
    PRÍKAZ1;  
}  
else if(a == 2)  
{  
    PRÍKAZ2;  
}  
else if(a == 3)  
{  
    PRÍKAZ3;  
}  
else if(a == 4)  
{  
    PRÍKAZ4;  
}  
else  
{  
    PRÍKAZ5;  
}
```

Lze napsat:

```
switch(a)  
{
```

```

case 1:
    PŘÍKAZ1;
    break;
case 2:
    PŘÍKAZ2;
    break;
case 3:
    PŘÍKAZ3;
    break;
case 4:
    PŘÍKAZ4;
    break;
default:
    PŘÍKAZ5;
    break;
}

```

Čím se to celé zjednoduší.

### Cyklus for (cyklus řízený proměnou )

Tento cyklu použijeme, pokud předem známe kolikrát se má opakovat určitá operace Syntaxe:

```

for(řídící_proměnná; podmínka; zvýšení_nebo_snížení_hodnoty_proměnné )
{
    PŘÍKAZ1;
    PŘÍKAZ2;
}

```

Příklad:

```

for(uint8_t i=0; i<4; i++)
{
    rozsvit_ledku();
    pockej_sekundu();
    zhasni ledku();
}

```

Co tento kód udělá? Dejme tomu, že máme už předem vytvořené funkce `rozsvit_ledku()`, atd... Potom cyklus udělá následující: vytvoří proměnnou `i` datového typu `uint8_t` (celá čísla od 0 do 255), uloží do ní číslo 0. Potom zkонтroluje podmínu, zda je proměnná `i` menší než 4, pak se provedou příkazy či funkce v těle cyklu, tj. rozsvítí se LED-dioda, bude svítit jednu sekundu, a pak zhasne. Pak se provede příkaz `i++`, ten zvýší hodnotu proměnné o jednu, tj. na číslo 1. Potom se kontroluje znova podmínka, zdali je proměnná `i` ( která má teď hodnotu 1) menší než 4, atd... Celkem se tělo cyklu vykoná čtyřikrát.

Cyklus `for` lze použít i bez proměnné, podmínky a zvýšení hodnoty, potom bude probíhat donekonečna. Příklad: `for(;;)`

```
{  
    rozsvit_ledku();  
    pockej_sekundu();  
    zhasni_ledku();  
}
```

Tento cyklus bude dělat to samé, co předchozí, s tím rozdílem, že to bude dělat donekonečna a nebude se ptát na podmínu.

Použití v programu pro robota: Nekonečný cyklus, ve kterém se kontrolují podmínky, zda je na čáře, zda je před ním soupeř, zda má udělat to či ono.

### **Cyklus while (cyklus s podmínkou na začátku )**

Použijeme, pokud nebudeme vědět kolikrát mají proběhnout příkazy a funkce v těle cyklu. Syntaxe:

```
while(podminka)  
{  
    PŘÍKAZ1;  
    ...  
}
```

Příklad:

```
while(!je_tlacitko_stiskle())  
{  
    ujed_1_cm();  
}
```

Tento cyklus bude stát, dokud nezmáčkneme tlačítko. Vyžaduje mít předem naprogramovanou funkci `ujed_1_cm()` a `je_tlacitko_stiskle()`.

### 5.2.4 Funkce a procedury

Pokud se nám v zdrojovém kódu opakují dokola stejné příkazy, můžeme vytvořit funkce nebo procedury, které potom voláme a které nám usnadní kód. syntaxe:

```
datový_typ_který_funkce_vrací název_funkce(datový_typ_parametru1  
parametr1, datový_typ_parametru2 parametr2)  
// Parametry jsou nepovinné  
{  
    PŘÍKAZ1; // Nepovinné příkazy  
    PŘÍKAZ2;  
    return HODNOTA_KTEROU_FUNKCE_VRACI;  
    // Za return se napíše výraz, který funkce vrací.  
}
```

Příklad:

```
// Funkce na sčítání dvou čísel, každé od 0 do 255.  
uint16_t umocni(uint8_t zakl, uint8_t exp)  
{  
    uint16_t vys = 1;  
    if(exp > 0)  
    {  
        for(uint8_t i = 0; i < exp; i++)  
        {  
            vys = vys * zakl;  
        }  
    }  
    return vys;  
}
```

pokud někde v programu použiji funkci `secti(10, 2)`, tak výsledkem bude, jako bych napsal  $10*10$ , tj. 100. Příklad:

```
c = umocni(a, b); // Do proměnné c se uloží b-tá mocnina čísla a.
```

procedura – je vlastně funkce, jen s tím rozdílem, že procedura má datový typ **void** a nemá **return** (tj. nevrací žádnou hodnotu).

Příklad:

```
void rozsvitLedku()
{
    DDRB |= (1<<P0); /* Co dělá tento příkaz nemusíte zatím řešit,
    pak se k tomu dostaneme.
    */
}
```

Pokud někde v programu napíšu **rozsvitLedku()**, tak se rozsvítí LEDka na robotovi.

tělo funkce main – všechny příkazy, které jsou mezi složenými závorkami, se po zahajení programu postupně vykonají

### 5.2.5 Knihovny

#### Vlastní knihovny

prog:knihovna Z funkcí a procedur se mohou vytvářet soubory pro usnadnění práce. Těmto souborům se říká **knihovny**. Soubory mají koncovku **.h**

Příklad: **avr/io.h moje\_knihovna.h**

Vlastní knihovnu poté mohu vložit direktivou **#include** Příklad:

```
#include "moje_knihovna.h"
```

Příklad takové knihovny je v kapitole [5.4.7](#) na straně [76](#).

#### Použité knihovny

Pro práci s mikrokontroléry budeme potřebovat některé už vytvořené knihovny. Příklad: Knihovna **Arduino.h** zavádí příkazy pro práci s čipy ATMega a ESP32. Knihovna **Learningkit.h** rozšiřuje možnosti předchozí knihovny o pohodlnou práci s deskou ALKS.

## 5.3 Příkazy C++ pro čipy

## 5.4 Příklady programů v C++

Ve všech příkladech níže je uveden vždy obsah souboru *main.cpp*. Text předpokládá, že nad příklady budete samostatně přemýšlet a učit se z nich, proto se to, co bylo řečeno u prvního příkladu, už neopakuje u druhého. Doporučuji projít soubory *LearningKit.h* a *LearningKit.cpp* (viz v [Explorera](#) adresář *.piolibdeps/ArduinoLearningKitStarter\_ID1745/src*), protože jsou v nich zkratky typu L\_R a jejich přiřazení pinům čipu.

Zdrojové soubory všech příkladů z této kapitoly jsou umístěny [zde](#).

Další příklady jsou na <http://wall.robotikabrn.cz> a <https://www.arduino.cc/reference/en/>.

### 5.4.1 Blikání LED

Program bliká červnenou LED.

```
#include "LearningKit.h"

void setup() // this part run once
{
    pinMode(L_R, OUTPUT); // initialize LED digital pin as an output.
}

void loop() // this part works in cycle
{
    digitalWrite(L_R, HIGH); // switch on red LED
    delay(500); // pause 500 miliseconds
    digitalWrite(L_R, LOW); // switch off red LED
    delay(500);
}
```

### 5.4.2 LED zapínaná tlačítkem

Žlutá LED zapínaná tlačítkem.

```

#include "LearningKit.h"

void setup()
{
    pinMode(L_Y, OUTPUT); // initialize LED digital pin as an output

    digitalWrite(L_Y, HIGH);
}

void loop()
{
    if ((digitalRead(SW1)) == LOW)
        {digitalWrite(L_Y, LOW);}
    else {digitalWrite(L_Y, HIGH);}

}

```

### 5.4.3 Nejjednodušší PWM

PWM umožnuje (ve spolupráci s [drivery](#)) řídit motory, serva a podobně. Zde je použito na stmívání LED pomocí potenciometru.

```

#include "LearningKit.h"

void setup()
{
    pinMode( L_Y, OUTPUT );
    ledcSetup(0, 1000, 10);
        // ledcSetup(channel, freq, resolution)
        // channel = 0 - 15
        // resolution = 10 => 2^10 => 1024
    ledcAttachPin(L_Y, 0); // ledcAttachPin(pin, channel)
}

void loop()
{
    ledcWrite(0, analogRead(POT1)/4); // potentiometer connect
}

```

Výše uvedený kód funguje pro čip ESP32. Pro čipy řady ATMega, které jsou na deskách Arduino uno, Arduino nano a Arduino Mega je možné použít například následující kód:

```
#include "Arduino.h"

int ledPin = 9;          // LED connected to digital pin 9
int analogPin = 3;        // potentiometer connected to analog pin 3
int val = 0;              // variable to store the read value

void setup()
{
    pinMode(ledPin, OUTPUT); // sets the pin as output
}

void loop()
{
    val = analogRead(analogPin); // read the input pin
    analogWrite(ledPin, val / 4); // analogRead values go from 0 to 1023
}
```

Funguje stejně, ale místo příkazu `ledcWrite()` je použit `analogWrite()`.

#### 5.4.4 Infrasenzor na čáru

Použijeme součástku [QRD1114](#), kolektor (C) je připojený na A0. Zdrojový kód je pro Arduino Uno.

```
#include <Arduino.h>

void setup()
{
    Serial.begin(115200);
    pinMode(A0, INPUT);
}

void loop()
{
```

```

    Serial.write(analogRead(A0)/4);
    delay(100);
}

```

#### 5.4.5 Posílání dat pro Lorris po sériové lince

Příklad ukazuje posílání dat do Analyzéru pro [Lorris](#). Posílá simulovaný bárový senzor, potenciometr, tlačítka binárně a tlačítka po bytech.

Zdrojový kód je tentokrát pro desku [Arduino nano](#) umístěnou na desce [ALKS](#) a používá knihovny [LearningKit.h](#) a [LearningKit\\_nano.h](#).

Poznámka: pokud chceme data poslat místo Analyzéru do terminálu, použijeme místo funkce `Serial.write()` funkci `Serial.print()` nebo `Serial.println()`. Rozdíl je v tom, že funkce `Serial.write()` posílá byty „tak jak jsou“, zatímco funkce `Serial.print()` převádí jednotlivé byty na řetězce, které představují posílaná čísla – vyzkoušejte.

```

#include "LearningKit_nano.h"

char a = '0';
int stav_tlac_2 = 0;
int stav_tlac_3 = 0;
int stav_tlac_4 = 0;
int stav_vsech_tlac = 0;

void setup() {
    Serial.begin(115200);
    setup_alks();
}

void loop() {
    delay(100); // nezahltit lorris
    Serial.write(0x80); // hlavicka - zacatek posilani dat
    Serial.write(0x0A); // ID_RGB
    Serial.write(0x03); // delka paketu RGB v bytech
    Serial.write(analogRead(POT_LEFT)/4); // posila binarne R
}

```

```

Serial.write(analogRead(POT_RIGHT)/4); // posila binarne G
Serial.write(0x70); // simulace treti barvy B

Serial.write(0x80); // hlavicka - zacatek posilani dat
Serial.write(0x0B); // ID_POT
Serial.write(0x01); // delka paketu POT v bytech
Serial.write(analogRead(POT_LEFT)/4); // posila binarne POT

stav_tlac_2 = digitalRead(2);
stav_tlac_3 = digitalRead(3);
stav_tlac_4 = digitalRead(4);
stav_vsech_tlac = stav_tlac_2 | ( stav_tlac_3 << 1 ) | ( stav_tlac_4 << 2 );

Serial.write(0x80); // hlavicka - zacatek posilani dat
Serial.write(0x0C); // ID_TL
Serial.write(0x01); // delka paketu TL v bytech
Serial.write(stav_vsech_tlac); // posila binarne vsechny tlacitka

Serial.write(0x80); // hlavicka - zacatek posilani dat
Serial.write(0x0D); // ID_TL_bytove
Serial.write(0x03); // delka paketu TL v bytech
Serial.write(stav_tlac_2); // posila tlacitko 2 jako byte
Serial.write(stav_tlac_3);
Serial.write(stav_tlac_4);

analogWrite(LED_Y, analogRead(POT_LEFT)/4); // kontrola na ALKS ,
}

```

#### 5.4.6 Ultrazvukový senzor HC-SR04

Zde je kompletní popis senzoru **HC-SR04** včetně zapojení a funkčního zdrojového kódu, který si můžete upravit podle potřeby.

### 5.4.7 Řízení serva

Jednoduchý program pro řízení **serva** pomocí potenciometru.

Pro desku Arduino nano nasazenou na ALKS se program použije tak, jak je.

Pro desku ESP-32 se musí použít jiná knihovna **Servo.h** (ale se stejným názvem, proto se samotný program nemění). Co se mění, je soubor **platformio.ini**, který kromě **knihovny pro ALKS** obsahuje navíc informaci o knihovně pro servo. Do souboru **platformio.ini**, který vám nachystá VSCode při **založení nového projektu**, je proto potřeba přidat tyto řádky:

```
lib_deps = 1745
          ServoESP32
```

Přitom první písmeno „S“ na druhém řádku musí být přesně pod číslicí „1“ na prvním řádku. Na začátku prvního řádku nesmí být mezery.

```
#include "LearningKit.h"
#include <Servo.h>

static const int servoPin = S1;
static const int PotPin = 32;

Servo servo1;

void setup() {
    servo1.attach(servoPin);
}

void loop() {
    int servoPosition = map(analogRead(PotPin), 0, 4096, 0, 180);
    servo1.write(servoPosition);
    delay(20);
}
```

### 5.4.8 Řízení motorů s použitím VNH2SP30 – základní

Příklad ukazuje jednoduché řízení silnějších motorů pomocí přídavné desky<sup>2</sup> k deskám Arduino uno nebo Arduino mega s dvěma **drivery vnh2sp30**. Zá-

---

<sup>2</sup><http://www.instructables.com/id/Monster-Motor-Shield-VNH2SP30>

roven se jedná o pěkný příklad využití knihovny. Zde se knihovna jmenuje vnh2sp30.h a je ke stažení [zde](#).

```
#include <Arduino.h>
#include "vnh2sp30.h"

void setup()
{
    motorSetup();

    Serial.begin(115200); // Initiates the serial to do
    Serial.println("Begin\u2022motor\u2022control");
    Serial.println(); //Print function list for user selection
    Serial.println("Enter\u2022number\u2022for\u2022control\u2022option:");
    Serial.println("1.\u2022STOP");
    Serial.println("2.\u2022FORWARD");
    Serial.println("3.\u2022REVERSE");
    Serial.println("4.\u2022READ\u2022CURRENT");
    Serial.println("+.\u2022INCREASE\u2022SPEED");
    Serial.println("-.\u2022DECREASE\u2022SPEED");
    Serial.println();
}

void loop()
{
    char user_input;
    while(Serial.available())
    {
        user_input = Serial.read(); //Read user input and trigger approp
        digitalWrite(EN_PIN_1, HIGH);
        digitalWrite(EN_PIN_2, HIGH);

        if (user_input =='1')
        {
            Stop();
        }
        else if(user_input =='2')
        {
            Forward();
        }
    }
}
```

```

    }
    else if(user_input == '3')
    {
        Reverse();
    }
    else if(user_input == '+')
    {
        IncreaseSpeed();
    }
    else if(user_input == '-')
    {
        DecreaseSpeed();
    }
    else
    {
        Serial.println("Invalid option entered.");
    }
}

```

#### 5.4.9 Přerušení

Co je přerušení? Procesor může zvládnout pouze jednu operaci na jeden tisk krystalu, postupuje od jednoho příkazu k druhému a nemůže jen tak všechno nechat a věnovat se něčemu jinému, občas je to ale potřeba. Při přerušení procesor všechno nechá a bude se věnovat přerušení, potom co skončí se bude věnovat dál programu tam, kde přestal.

### 5.5 Desítková soustava a dvojková soustava

V počítači jsou všechna data uložena pouze v podobě jedniček a nul. A proto se nám bude hodit, když se budeme aspoň trochu orientovat v převodech mezi dvojkovou a desítkovou soustavou.

### 5.5.1 Desítková soustava

V **desítkové soustavě** je základem číslo deset.

Příklad: číslo 156 si můžeme rozložit jako  $1 \cdot 10^2 + 5 \cdot 10^1 + 6 \cdot 10^0$ , přitom  $10^0=1$ ,  $10^1=10$ ,  $10^2=10 \cdot 10=100$ ,  $10^3=10 \cdot 10 \cdot 10=1000$  atd ..., takže naše číslo potom vypadá takto:  $1 \cdot 100 + 5 \cdot 10 + 6 \cdot 1 = 156$ .

### 5.5.2 Dvojková soustava

Ve **dvojkové soustavě** je základem číslo dva.

Příklad: číslo 11001010 rozložíme na

$1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$ , přitom

$2^0=1$ ,  $2^1=2$ ,  $2^2=2 \cdot 2=4$ ,  $2^3=2 \cdot 2 \cdot 2=8$ ,  $2^4=2 \cdot 2 \cdot 2 \cdot 2=16$ ,  $2^5=32$ , atd..., proto naše číslo má hodnotu:

$1 \cdot 128 + 1 \cdot 64 + 0 \cdot 32 + 0 \cdot 16 + 1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 = 202$ .

### 5.5.3 Převod z desítkové soustavy do dvojkové

Jak na to, když chceme převádět číslo z desítkové soustavy do dvojkové?

Například převedeme číslo 97.

Nejprve je potřeba, abychom znali násobky čísla dva:

2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, atd...

Napíšeme jedničku.

1

Najdeme nejbližší nižší násobek dvou k našemu číslu, v našem případě je nejbližše 64. Odečteme si od 97 číslo 64,  $97 - 64 = 33$ . Odečítali jsme číslo 64, další v pořadí je číslo 32, je 32 obsaženo v 33? Ano, proto napíšeme další jedničku.

11

Dále  $33 - 32 = 1$ , další číslo je 16, je 16 obsaženo v 1? Ne, proto napíšeme nulu.

110

Další číslo je 8, také není obsaženo v 1, takže napíšeme nulu.

1100

Čísla 4 a 2 také nejsou obsažena v číslu 1, proto napíšeme nuly.

110000

A nakonec, je číslo jedna obsaženo v čísle jedna? Ano, napíšeme jedničku.

Náš binární zápis čísla 97 vypadá takto:1100001. Čísla v binární soustavě by měla mít počet cifer rovno 8, 16, 32, podle toho s kolika bitovými čísly počítáme. V našem případě musíme doplnit jednu nulu. Pokud bychom počítali s 16 bitovými čísly doplníme o 8 nul více. Nuly doplňujeme zleva. Zapanujeme si to snadno v desítkové soustavě se číslo 128 také nezmění pokud před něj dáme nuly, 000128. Nuly navíc nebudeeme psát v desítkové soustavě, ale v dvojkové nám to pomůže se lépe orientovat. Takže to bude vypadat: 01100001 No a to ještě není všechno. Aby mělo číslo správný tvar, musí mít před číslem uvedeno 0b(nula a malé B), tím říkáme v jazyce C, že číslo je v (binární) dvojkové soustavě.

Pozor: Čísla 101 a 0b101 nejsou to samé! Je to jednoduché, pokud 0b neuvědeme bude překladač číslo brát jakož je v desítkové soustavě. Tj. 101 => číslo sto jedna, 0b101 => číslo pět.

#### Další příklad:

Převedeme číslo 237.

Nejbližší nižší číslo je 128.

1

237-128=109 je číslo 64 obsaženo v čísle 109? ANO, další jednička.

11

109-64=45 je číslo 32 obsaženo v 45? ANO, jednička.

111

45-32=13 je 16 obsaženo v čísle 13? NE, nula.

1110

je číslo 8 obsaženo v 13? ANO, jednička.

11101

13-8=5 je 4 obsaženo v 5? ANO, jednička.

111011

5-4=1 je obsažena v 1? NE, nula.

1110110

je 1 obsažena v 1? ANO, jednička.

11101101 Binární zápis čísla 237 vypadá takto:11101101.

#### 5.5.4 Převod z dvojkové soustavy do desítkové

Vezmeme si například binární číslo 1011. Očíslujeme si pozice zprava doleva od nuly, tj.:

1 0 1 1

3 2 1 0

Pohybujeme se v dvojkové soustavě, tj. základ bude 2. Vezmeme nultou číslici a vynásobíme s ní číslo  $2^0$ . Dostáváme  $1 \cdot 2^0 = 1$ . Dále první číslici vynásobíme číslem  $2^1$ , tj.  $1 \cdot 2^1 = 2$ . Dále druhou číslici vynásobíme číslem  $2^2$ , tj.  $0 \cdot 2^2 = 0$ . A to samé provedeme s číslicí na třetí pozici:  $1 \cdot 2^3 = 8$ . A nyní čísla sečteme  $8 + 0 + 2 + 1 = 11$ , takže binární číslo 1011 převedené do desítkové soustavy je 11.

Příklad 2: Převeďte binární číslo 11101001 do desítkové soustavy.

$1 \cdot 2^0 + 0 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 + 0 \cdot 2^4 + 1 \cdot 2^5 + 1 \cdot 2^6 + 1 \cdot 2^7$  To je:

$1 + 0 + 0 + 8 + 0 + 32 + 64 + 128 = 233$  Binární číslo 11101001 převedeno do desítkové soustavy je 233.

Příklad 3: Převeďte binární číslo 00010101.

$1 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 + 0 \cdot 2^5 + 0 \cdot 2^6 + 0 \cdot 2^7$ , saozřejmě členy  $2^x$ , které násobíme nulou budou nulové a tudíž je můžeme beztrestně vynechat, tj.

$1 \cdot 2^0 + 1 \cdot 2^2 + 1 \cdot 2^4 = 1 + 4 + 16 = 21$  Binární číslo 00010101 má v desítkové soustavě hodnotu 21.

### 5.5.5 Bitové výrazy a práce s nimi

Bitové operace se od logických operací (viz 5.2.2) liší tím, že se neaplikují na číslo jako celek, ale bit po bitu, viz níže.

My budeme potřebovat následující bitové operace: bitový součet, bitový součin, bitovou negaci, bitový posun doleva.

#### Bitové operátory

- | bitový součet
- & bitový součin
- ~ bitová negace
- ^ bitový exluzivní součet
- << bitový posuv doleva
- >> bitový posuv doprava

## Bitový součet

Platí, že  $0|0=0$ ,  $0|1=1$ ,  $1|0=1$ ,  $1|1=1$ , sčítají se vždy pouze bity, které leží pod sebou. Zkrátka pokud je aspoň jeden bit jedna, výsledek je také jedna. Příklad: máme dvě binární čísla 00000100 a 01100001 a chceme je bitově sečíst.

Když sečteme nulté bity zadaných čísel, dostaneme  $0|1=1$ . Pak provedeme bitový součet i pro první, druhé a další bity. Zapíšeme: (0b00000100 | 0b01100001); Nebo pokud budeme mít čísla v proměnných:

```
uint8_t a = 0b00000100;  
uint8_t b = 0b01100001;  
uint8_t c;  
c = (a|b);
```

Do proměnné **c** se zapíše výsledná hodnota.

```
00000100 - první číslo  
01100001 - druhé číslo  
-----  
01100101 - výsledek po bitovém logickém součtu
```

Výsledek: 01100101.

## Bitový součin

Platí, že  $0&0=0$ ,  $0&1=0$ ,  $1&0=0$ ,  $1&1=1$ , tady platí, že výsledek je jedna, pokud oba bity jsou jedna. Příklad: bitový součin čísel 00110011 a 01000001 Zapíšeme: (0b00110011 & 0b01000001); Nebo pokud budeme mít čísla v proměnných:

```
uint8_t c = 0b00110011;  
uint8_t d = 0b01000001;  
uint8_t e;  
e = (c&d);
```

Výsledná hodnota se zapíše do proměnné **e**.

```
00110011 - první číslo  
01000001 - druhé číslo  
-----  
00000001 - výsledek bitového logického součinu
```

## Bitová negace

Tam, kde byla jednička, bude nula, a kde byla nula, bude jednička. Příklad: znegujte číslo 00110101 Zapíšeme:  $\sim(0b00110101)$ ; Nebo pokud budeme mít číslo v proměnné:

```
uint8_t a~= 0b00110101;  
uint8_t b;  
b = ~a;
```

Do proměnné **b** se zapíše výsledná hodnota.

```
00110101 - číslo, které bude logicky znegováno  
-----  
11001010 - výsledek
```

## Bitový exklusivní součet

Výsledkem exklusivního součinu je pravda, pokud se hodnoty liší, tj. nejsou stejné. Takže platí  $0^0=0$ ,  $0^1=1$ ,  $1^0=1$ ,  $1^1=0$

Zapíšeme:  $(0b00110011 \ ^ 0b01000001)$ ; Nebo pokud budeme mít čísla v proměnných:

```
uint8_t c = 0b00110011;  
uint8_t d = 0b01000001;  
uint8_t e;  
e = (c^d);
```

Výsledná hodnota se zapíše do proměnné **e**.

```
00110011 - první číslo  
01000001 - druhé číslo  
-----  
01110010 - výsledek bitového exklusivního součtu
```

## Bitový posun doleva

Vezmeme celé osmibitové číslo a posuneme ho o určitý počet míst doleva. Přitom čísla vlevo zmizí a na na uprzedněné místo vpravo přijdou vždy nuly. Zapíšeme: (0b01000101 << 3); Nebo pokud budeme mít číslo v proměnné:

```
uint8_t c = 0b01000101;  
uint8_t d;  
d = (c << 3);
```

Do proměnné d se přiřadí výsledek operace. Pokud například už nebudeme potřebovat původní hodnotu proměnné c, lze to napsat i takto:

```
uint8_t c = 0b01000101;  
c = (c << 3);
```

Hodnota v proměnné c se nejprve posune o tři pozice a pak se výsledná pozice zapíše na stejnou pozici v paměti mikrokontroléru.

Příklad: proveděte bitový posun doleva o tři s číslem 00000101

```
01000101  
-----  
10001010 - posun o jedno doleva  
00010100 - znova o jedno doleva (celkem o dvě místa)  
00101000 - celkový posun o tři bity doleva
```

## Bitový posun doprava

Bitový posun doprava je podobný jako operace bitový posun doleva, liší se jen ve směru posunu, tj. posouvá se na opačnou stranu. Příklad 1: Proveděte bitový posun doprava o tři pozice s binárním číslem 00000101. Zapíšeme: (0b00000101 >> 3); Výsledkem bude toto:

```
00000101  
-----  
00000010 - posun o jednu pozici doprava  
00000001 - posun o druhou pozici  
00000000 - posun o třetí pozici
```

Příklad 2: Proveďte bitový posun o dvě pozice doprava a potom o tři doleva s binárním číslem 11110111. Zapíšeme: ((0b11110111 >> 2) << 3); Výsledkem bude toto:

```
11110111
-----
00111101 - posun o dvě pozice doprava
11101000 - posun o tři pozice doleva, toto je výsledek.
```

Příklad 3: Ale co když zadám číslo do proměnné v desítkové soustavě, například 189 a provedu bitový posun o dvě pozice doprava? Co dostanu? Dostanu snad 1? Ne, i když jsme číslo zadali v desítkové soustavě, v mikrokontroléru je stejně uloženo jako jedničky a nuly, takže číslo musíme nejprve převést do dvojkové soustavy.  $189 = 0b10111101$  A teď posun o dvě pozice doprava: 00101111 a to je číslo 47. Takže 47 je výsledek bitového posunu čísla 189 o dvě pozice doprava.

Zapíšeme takto:

```
uint8_t a~= 189;
uint8_t b;
b = (a~>> 2);
```

No a výsledek(tj. číslo 47 ) se uloží do proměnné b.

# Kapitola 6

## Přílohy

### 6.1 Vývojový deník robota pro Ketchup House aneb stručná historie výroby

Vývojový deník je v podstatě kronika stavby robota. Silně doporučuji jej alespoň heslovitě psát s konkrétními daty. Při porovnání s **plánem stavby** (tj. s vizí, jak rychle se co udělá, což taky doporučuji sepsat) je velmi cenným zdrojem zkušeností. Pro představu, velmi hrubý plán stavby tohoto robota byl následující: září - listopad: plán a stavba mechanická konstrukce, prosinec - leden: zapojení elektroniky a senzorů, únor - duben: programování a testování, květen - červen: rezerva. Závěr: to se v pohodě stihne.

O tom, jaká byla realita, čtěte dále.

#### **září - říjen:**

- první návrh strategie
- první návrh mechaniky a konstrukce
- vize ohledně senzorů
- jako řídící jednotka byla zvolena deska Arduino Mega, především kvůli velkému počtu pinů, cíl: vystačit si s jednou deskou

#### **listopad:**

- druhý návrh strategie
- výroba první konstrukce, jsou použity vrtačkové motory
- pokusy o zprovoznění/naprogramování driverů pro motory, byly použity VNH32, které známe ze starších strojů, tyto drivery se

vypínaly kvůli nízkému napětí baterií + byla potřeba úprava driverů (odpájení Zenerovy diody)

- motory se ukazují jako příliš silné a obtížně ředitelné

**prosinec:**

- druhá konstrukce, místo vrtačkových byly použity čínské modelářské motory

**leden:**

- pololetní klasifikace

**únor:**

- osazování senzorů QRD1114 pro čáry
- první verze programu QRD1114 z Číny se neosvědčily → znova koupit a zapájet senzory z GM Electronic
- řešení problémů s chybným zapojením senzorů
- kalibrace senzorů

**březen:**

- zprovoznění serva pro vypouštění plechovek
- testování senzorů HS-04 pro ultrazvukovou detekci soupeře

**duben:**

- čtení z ultrazvuků je příliš pomalé a robot nestihá detekovat čáru → přidání druhé ATmega desky pro čtení ultrazvuků a enkodérů
- rozchození komunikace mezi deskami
- rozchození enkodérů → robot konečně jede rovně

**květen:**

- QRD nečtou správně → posun QRD níž
- přechod na algoritmus typu stavový automat
- robot pořád nechce jet po čáře
- robot se hodně zpomaluje, pokud veze více plechovek najednou (se čtyřmi plechovkami už stojí) → snaha o programové řešení pomocí zvýšení výkonu motorů při zpomalení robota

**červen:**

- intenzivní snaha dodělat a naprogramovat robota, provázená průběžně poruchami všeho druhu; tyto poruchy byly často způsobeny nezkušeností a nedotaženými detaily z předchozích měsíců
- do termínu soutěže 10.6. se nepodařilo naprogramovat potřebné schopnosti robota, robot se přesto soutěže zúčastnil

- v den soutěže se spálila základní deska a její výměna vzala skoro veškerý zbývající čas plánovaný na dotažení programu

### Zkušenosti a závěry

- skutečná stavba a programování trvá nejméně 2x tak dlouho, než předpokládá plán
- (ze zkušenosti i s jinými roboty): vždy se naplánuje několik verzí obtížnosti a prakticky vždy se postaví a zprovozní pouze ta nejjednoduší, více se nestihne, především naprogramovat
- je potřeba buď mít dva stejné roboty nebo vézt celého robota s sebou na soutěž ještě jednou v náhradních dílech; kdybychom neměli náhradní základní desku, tak jsme skončili, ještě než soutěž začala

## 6.2 Požadavky na počítač pro robotiky

Požadavky na počítač se liší podle toho, jestli na něm chceme tvořit programy pro roboty nebo konstrukci v [CADu](#).

Pro pro psaní a ladění programů postačí počítač s Pentium Dual Core 1.8 GHz, RAM 1 GB, HDD 100 GB, Linux Lubuntu nebo Windows 7. U ještě slabších strojů se to musí vyzkoušet, ale například na [IBM T23](#) už se Linux ani nepodařilo nainstalovat. Na Windows XP nepojede [VSCode](#) ani další programy, například [Lorris](#).

Pro návrh konstrukce potřebujete výkonější stroj, za rozumné minimum se dá považovat Intel I3-2328M 2,2GHz, 2 core, 4 logické procesy, 4GB RAM, HDD 250 GB, ale na Windows 10 se zde programy spouští pořád pomalu, třeba půl minuty i déle. Samotný provoz už je potom bez problému.

Přiměřený je například [Lenovo T440](#). Samozřejmě, že čím výkonější počítač, tím líp.

## 6.3 Hodnoty vybraných součástek

### Dioda 1N4148

Maximální napětí: 100 V

Maximální proud: 200 mA

Maximální výkon: 500 mW

Úbytek napětí: 1,8 V

### Dioda 1N4007

Maximální napětí: 1000 V

Maximální proud: 1 A

Maximální výkon: 3 W

Úbytek napětí: 1,1 V

### tranzistor BC337

Max. napětí mezi kol. a emit.  $V_{CEO}$ : 50 V

Max. napětí mezi bází a emit.  $V_{CBO}$ : 5 V

Max. proud tekoucí kolektorem  $I_C$ : 800 mA

Maximální výkon  $P_C$ : 625 mW

Zesílení  $h_{fe}$ : 100 až 600

### tranzistor BC547

Max. napětí mezi kol. a emit.  $V_{CEO}$ : 45 V

Max. napětí mezi bází a emit.  $V_{CBO}$ : 6 V

Max. proud tekoucí kolektorem  $I_C$ : 100 mA

Maximální výkon  $P_C$ : 500 mW

Zesílení  $h_{fe}$ : 110 až 800

### tranzistor BD911

Max. napětí mezi kol. a emit.  $V_{CEO}$ : 100 V

Max. napětí mezi bází a emit.  $V_{CBO}$ : 5 V

Max. proud tekoucí kolektorem  $I_C$ : 15 A

Maximální výkon  $P_C$ : 90 W

Zesílení  $h_{fe}$ : 5 až 250

Důležité je si všimnout, že minimální zesílení<sup>1</sup> je 5. To znamená, že pokud budeme chtít tranzistorem spínat proud 10 A, tak budeme muset nechat bází téct řídící proud 2 A, to znamená, že ho nemůžeme naplno využít, pokud ho budeme řídit mikrokontrolérem, tj. musíme ho spínat jiným tranzistorem. Anebo přijdeme na myšlenku, že pokud budeme chtít řídit velké proudy, budeme potřebovat tranzistory typu MOS-FET, např.: IRF520, IRL3803.

---

<sup>1</sup>Platí pokud bude kolektorem protékat 10 A.

## 6.4 Popis a vlastnosti desky RB3201-RBControl

### 6.4.1 Určení a cíl

RB3201 - RBControl (RBC) je univerzální deska pro stavbu hobby robotů. Jde v podstatě o shield k desce [ESP32](#), který má dva hlavní cíle: rozšířit počet pinů desky ESP32 a umožnit snadné připojení velkého množství různých periférií, především robotických.

### 6.4.2 Hlavní vlastnosti

Deska RBC umožňuje současně ovládat až 8 [motorů](#) (1,5 A trvale, 2 A špičkově každý). Dále umí po osazení spínanými zdroji napájet a ovládat 4 [serva](#) nebo 8 [mikroserv](#), která pracují současně. Maximálně je možné připojit až 32 serv nebo mikroserv. Má vyvedenou I2C sběrnici celkem 6x na 3,3 V a 2x na 5 V. Dále je na desce [expandér pinů](#), který je připojený na I2C a obsluhuje další dva porty A,B po 8 pinech. Na desce jsou vyvedená tři tlačítka, 4 LED a piezo.

### 6.4.3 Další vlastnosti

Po osazení tranzistorem Q3 je deska chráněná proti přepólování. Přímo na desce je možné měřit reálné hodnoty napětí 3,3 V a 5 V rozvedených po desce.

Čip ESP32 má 26 použitelných pinů, z toho 4 pouze vstupní. Přitom na dvou pinech je připojena sériová linka (používá se pro programování čipu přes USB), na dvou pinech je I2C, 3 piny jsou použity pro komunikaci s drivery pro motory, 1 pin měří napětí na baterii. Pro uživatele tedy zbývá 14 pinů.

K RBC s ESP32 je možné připojit další RBC bez ESP32 a rozšířit tak počet periférií a počet ovládaných motorů. U motorů na připojené ovládané RBC desce nelze použít enkodéry. Také je možné připojit k I2C místo 5 V libovolné externí napětí do cca 30 V a provozovat I2C na tomto napětí.

Drivery pro motory se dají paralelizovat (pouze po dvou na stejně desce s drivery), tj. zapnout paralelně dva drivery pro jeden motor a dodávají pak motoru dvakrát větší proud.

#### 6.4.4 Napájení

Napájení desky RBC je ideální ze dvou Li-On baterií, které dodávají asi 8 V a dostatečné proudy. K desce lze připojít napětí až do 10 V, připojení vyššího napětí neumožňují použité drivery pro motory. Do motorů je přiváděn signál PWM na napětí z baterie a 5 V napájení pro enkodéry.

RBC si hlídá napětí na baterii a umí ho měřit do 10 V. V ESP32 je softwarově (v knihovně) nastavené napětí napájení 7,2 V, při kterém ESP32 vypne desku, aby nedošlo k podvybití baterie.

Napětí 5 V pro desku může tvořit buď stabilizátor 7805 nebo spínaný zdroj. Z těchto 5 V se tvoří 3,3 V na dalším stabilizátoru na desce ESP32-DevKitC. Proto při napájení pouze z USB nebude fungovat rozvod 5 V na desce. Čip 7805 dává asi 1 A, asi 0.5 V spotřebuje deska sama, zbytek mohou použít připojené periferie. Když zkuste odebírat více proudu, tak se stabilizátor vypne, neb má v sobě pojistku. Pokud proud pro periferie nestačí, je možné místo 7805 použít spínaný zdroj, který může dodávat 2 A.

Kromě hlavního je možné na desku osadit další 4 spínané zdroje pro napájení serv – viz bod č. 20 u popisu desky (kap. 6.4.6).

#### 6.4.5 Expandér

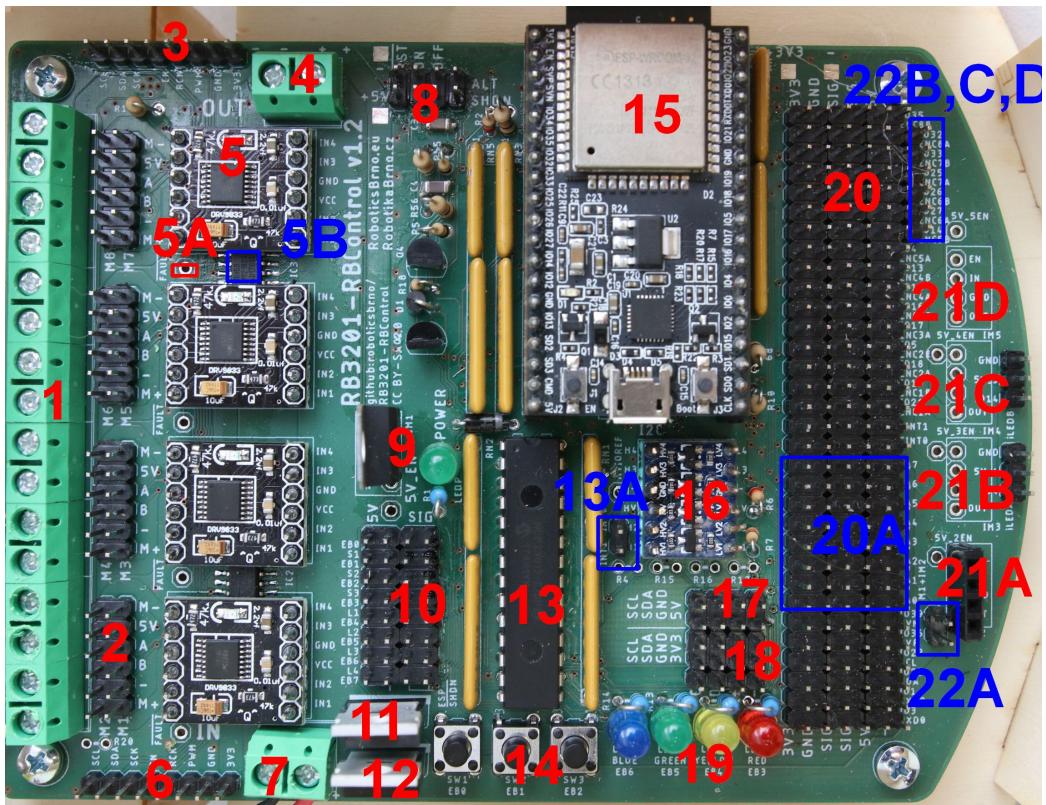
**Expandér pinů** je obvod, který umožnuje připojit osm vstupně-výstupních (I/O) pinů přes I2C sběrnici, tedy pouze pomocí dvou pinů na čipu. Funguje na 3,3 V, má dva porty A, B – každý port má 8 pinů, port A je pro uživatele, port B je pro tlačítka, LED a vypínání desky, ale uživatel může port B také použít.

Expandéry jsou pomalé, proto jsou vhodné např. pro řízení motorů a pod., ale ne pro například modelářské servo.

Expandér pinů na tříbitovou adresu, takže jich může být připojeno 8 tohoto typu (expandéry jsou různých typů, včetně „rychlých“ expandérů, které zvládnou i ovládání modelářských serv).

#### 6.4.6 Rozložení pinů na desce RBC a jejich vlastnosti

Pro snazší orientaci si otočte desku tak, aby oblouk na desce byl vpravo. Při popisu budeme postupovat zleva doprava a shora dolů. Poloha jednotlivých částí desky je vyznačena na obrázku 6.1.



Obrázek 6.1: Rozložení pinů na desce RB3201-RBControl

1. Svorkovnice pro připojení DC motorů
2. Piny pro připojení motorů s enkodéry. Protože ESP32 má málo pinů, je potřeba si zjistit, které piny jsou použité pro enkodéry a nepoužívat je na něco jiného.
3. Výstupní piny, slouží signálovému propojení s další deskou RBC.
4. Výstupní napájení. Použije se v případě připojení další RBC desky jako vstupní napájení pro připojenou desku. Piny SCL, SDA, GND, 3V3 lze použít i samostatně jako další I2C port.
5. **Drivery** pro DC motory. Každý driver poskytuje PWM napájení pro dva motory. Každý driver má vyvedený pin 5A (FAULT), kde je možné měřit, jestli se nenachází v chybovém stavu. Každý driver má ochranu proti přetížení i přehřátí → při překročení mezních hodnot se vypne. Pod drivery jsou posuvné registry 5B, které generují signál pro motory podle pokynů z ESP32.

6. Vstupní piny, slouží signálovému propojení s řídící deskou RBC. Piny SCL, SDA, GND, 3V3 lze použít i samostatně jako další I2C port.
  7. Vstupní napájení z baterie nebo z řídící RBC desky.
  8. Jde o čtveřici jumperů<sup>2</sup>, zleva doprava při propojení provedou následující: Reset RBC, zapnutí RBC, vypnutí RBC, vypnutí RBC v případě, že expandér není osazen nebo nefunguje.
  9. **Stabilizátor** 7805 (dodává cca 1 A, většinu z tohoto proudu spotřebuje RBC pro svůj provoz). Místo 7805 je možné osadit spínaný zdroj, který dodává cca 2 A.
  10. Port B z expandéru. Zleva doprava jsou to piny: 5 V, GND, 2x signálový pin (ten stejný) na 3,3 V. Na tomto portu jsou také připojená tlačítka, všechny LED a **vypínání desky (pin B7)**. Pozor, ve verzi 1.2 jsou popisy pinů posunuté o jeden pin dolů.
  11. **Tranzistor** Q1 – zapíná desku.
  12. Tranzistor Q3. Po osazení tranzistorem Q3 je deska chráněná proti přepólování.
  13. Expandér pinů. 13A jsou piny pro sledování **přerušení** na portu A a portu B expandéru.
  14. Tlačítka S1, S2, S3.
  15. ESP32-DevKitC
  16. Deska, která převádí signál I2C z 3,3 V na 5 V a signál pro připojení inteligenčních LED a podobně – viz č. 17 a 23.
  17. 2x I2C na 5 V
  18. 3x I2C na 3,3 V
  19. 4x LED
  20. Piny pro připojení periferií k ESP32. Součástí jsou piny portu A z expandéru (č. 20A). Zleva doprava: 3,3 V, GND, 2x signálový pin, 5 V, GND. Přitom piny 5 V jsou ve výchozím nastavení připojeny na spínané zdroje (č.21A-D, které samozřejmě musí být osazeny).
- Vyjímka je spodních 8 pinů, které jsou připojeny přímo ke stabilizátoru nebo spínanému zdroji č. 9. To také znamená, že na pinech značených 5 V není 5 V, ale tolik voltů, jak je nastavený stabilizátor, který je připojený k daným pinům. Toto se musí ohlédat speciálně při připojování serv, jinak je můžete snadno spálit.

---

<sup>2</sup>jumper – propojka mezi dvěma piny

Piny IOxx na desce odpovídají pinům na ESP32, piny EXx jsou piny z expandéru pinů, EAx jsou volné piny, EBx jsou použité pro tlačítka a diody (viz popisy na desce a bod č.10) – je potřeba při jejich použití na to dát pozor.

Popis pinů: například vpravo nahoře: vždy dva popisky odpovídají jedné řadě, takže například první řada od shora patří k pinu IO35 a současně je na stejný pin přiveden ENC8B – enkodér pro 8. motor (každý enkodér má dva výstupy, A a B).

Horní dvě řady pinů jsou připojeny na spínaný zdroj 21D (první řada od shora je pouze vstupní), dvě řady pinů pod nimi jsou připojeny na 21C, další dvě řady na 21B a zbývající řady kromě spodních osmi na 21A. Pokud chceme tyto piny připojit na zdroj č. 9, musíme propojit jumpery 22A-D. Přitom piny tvoří kaskádu, tj. musíme nedřív propojit jumper 22A, potom 22B atd.

21. Místo pro osazení spínaných zdrojů (step-down): 21A, 21B, 21C, 21D. Každý zdroj má vyvedený tzv. enable-pin. Zdroje jsou ve výchozím stavu zapnuté a lze je vypnout tak, že na enable-pin přivedu kabelem z jiného pinu potřebný signál.

Na každý spínaný zdroj (step-expanderdown) je možné připojit jedno servo nebo dvě mikroserva. Piny jsou vyvedeny tak, aby bylo možné serva připojit přímo. Přitom zdroj 21D obsluhuje horní dva piny 5 V, zdroj 21C dva piny 5 V pod nimi, zdroj 21B opět dva piny 5 V pod nimi a zdroj 21A všechny zbývající piny 5 V kromě spodních osmi.

22. Jumpty pro připojení spínaných zdrojů z bodu 21, písmeno vždy odpovídá: 22A, 22B, 22C, 22D.

23. Signál pro inteligentní LED nebo podobné zařízení – piny na desce nejvíce vpravo (nejsou vyznačeny).

#### 6.4.7 Programování a návod

Programování desky RBC — viz <https://www.mickoflus.cz/guide.html>, sekce programování. Stahnout, otevřít a přeložit (build) demo projekt z míčkoflusu, ten si stáhne z GitHubu všechny potřebné knihovny a bude připraven k překladu. Příklady kódu: <https://rbcontrol.robotikabruno.cz>.

Dokumentace RBControl knihovny: <https://github.com/RoboticsBrno/RB3201-RBControl-library>.

## 6.5 Řešení některých problémů

### 6.5.1 Chyba při uploadu programu do desky Arduino nano:

Chyba:

```
Please specify 'upload_port' environment or use global '-upload-port'  
option
```

Řešení: pomohlo spustit příkaz `sudo service udev restart` a znovu spustit VS code a odpojit a připojit desku.

# Literatura

- [1] MALÝ, Martin. *Hradla, volty, jednočipy: úvod do bastlení*. Praha: CZ.NIC, z.s.p.o., 2017. CZ.NIC. ISBN 978-80-88168-23-2.  
[https://knihy.nic.cz/files/edice/hradla\\_voltynednocy.pdf](https://knihy.nic.cz/files/edice/hradla_voltynednocy.pdf)
- [2] VODA, Zbyšek. *Průvodce světem Arduina*. Vydání druhé. Bučovice: Martin Stříž, 2017. ISBN 978-80-87106-93-8.  
<https://www.robotikabrnocz/docs/arduino/Pruvodce-svitem-Arduina-CZ.pdf>
- [3] [www.citace.com](http://www.citace.com) [online]

# Rejstřík

- \*.md, 51
- čip, 12
- 1N4007, 85
- 1N4148, 84
- 7805, 13
- ALKS/deska, 24
- Arduino, 25
  - Mega, 25
  - Nano, 25
  - Uno, 25
- Arduino IDE, 51
- Atom, 51
- báze, 11
- bajt, 55
- baterie
  - 18650, 27
  - AA, 27
  - AAA, 27
  - Li-On, 27
  - Li-Pol, 28
  - Ni-Cd, 28
- BCC337, 85
- BCC547, 85
- BD911, 85
- bit, 55
  - nulty, 55
- bluetooth, 26
- cín, 19
- cívka, 12
- CAD, 8
- commit, 48
- Cpp4Robots, 44, 52
- díl, 39
- datasheet, 9
- datový typ, 56
- desítková soustava, 75
- deska
  - ESP 32, 23
- deska/ALKS, 24
- dioda, 11
  - 1N4007, 85
  - 1N4148, 84
  - LED, 11, 16
- direktiva preprocesoru, 56
- driver, 12
- dvojková soustava, 75
- elektrické
  - napětí, 15
- elektrický
  - proud, 14
- emitor, 11
- enkodér
  - rotační, 21
- ESP 32
  - deska, 23
- expander, 87
- Explorer, 43

git, 48  
HC-SR04, 26  
I2C, 29  
IDE, 44, 51  
infrasenzor  
    odrazový, 26  
jumper, 89  
kalafuna, 19  
knihovna, 66  
kolektor, 11  
kondenzátor, 10  
    elektrolytický, 10  
    keramický, 10  
    tantalový, 10  
konstrukce robota, 7  
krystal, 14  
LaTeX, 46  
LearningKit, 24  
LED, 11, 16  
    dioda, 11, 16  
Linux, 53  
logická operace, 58  
logická jednička, 12  
logická nula, 12  
logický výraz, 58  
Lorris, 45  
Měření  
    napětí, 18  
    odporu, 18  
    proudu, 19  
markdown, 51  
menu osciloskopu, 30  
mikrokontrolér, 12  
mikroprocesor, 12  
mikroservo, 23  
MiKTeX, 46  
multimetr, 17  
napěťový dělič, 18  
napětí  
    elektrické, 15  
Nepájivé kontaktní pole, 9  
nultý bit, 55  
ochrana  
    přepolování, 11  
offset, 32  
Ohmův  
    zákon, 15  
Onshape, 8, 34  
operace  
    logická, 58  
osciloskop, 30  
páječka, 19  
pájka, 19  
příkaz, 56  
Přehled soutěží, 5  
přepolování  
    ochrana, 11  
Převodník napěťových úrovní, 27  
paralelní  
    zapojení, 10  
pin, 12  
pinout, 24  
plán stavby, 82  
PlatformIO, 42  
plugin, 41  
podsestava, 39  
preprocesor, 56  
prerusení, 74  
Proficad, 52  
projekt, 42

proměnná, 56  
proměnné, 38  
proud  
    elektrický, 14  
PsPad, 46  
PWM, 22, 68  
QRD1114, 26  
RBControl, 24, 86  
repozitář, 48  
rezistor, 9, 16  
robot  
    autonomní, 7  
    konstrukce, 7  
Robotiáda, 5  
Robotický den, 5  
sériové  
    zapojení, 10  
sběrnice, 29  
senzor  
    ultrazvukový, 26  
Serial.print(), 70  
Serial.println(), 70  
Serial.write(), 70  
servo, 22, 23  
sestava, 39  
shield, 25  
skica, 37  
Solidworks, 8  
sonda osciloskopu, 31  
soustava  
    desítková, 75  
    dvojková, 75  
SPI, 29  
stabilizátor, 13  
stabilizator, 13  
Status bar, 44  
syntaxe, 56  
tepelný  
    výkon, 15  
terminál, 44, 53  
TeXLive, 46  
TeXstudio, 46  
tlumivka, 12  
tranzistor, 11  
    BCC337, 85  
    BCC547, 85  
    BD911, 85  
UART, 29  
ultrazvukový senzor, 26  
USB, 23  
výkon, 15  
    tepelný, 15  
výraz  
    logický, 58  
vývojový deník, 82  
verzování, 47  
Visual Studio Code, 41  
vh2sp30, 13, 72  
vzorkovací frekvence, 30  
zákon  
    Ohmův, 15  
zapojení  
    paralelní, 10  
    sériové, 10