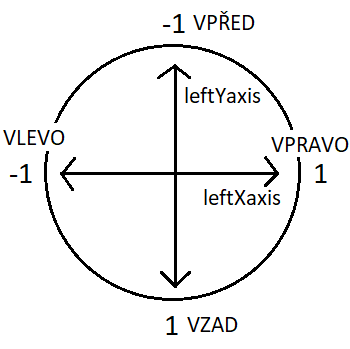
Bakalářská práce – průběžné poznámky

# Původní návrh Matematiky počítání rychlostí motorů na Raspberry Pi v krocích

Plán ovládání podvozku byl takový, že bude ovládán analogovým ovladačem určeným pro herní konzole a jiné platformy. Od tohoto návrhu se ustoupilo hlavně z toho důvodu, že závislost na specifickém ovládacím zařízení je nežádoucí a nepraktická.

Kroky jsou určeny především pro levou páčku určující pohyb podvozku.

Předpoklad je ten, že uživatel pohnul jednou, nebo druhou páčkou. V opačném případě se nic neděje.

Hodnoty do daných směrů páček ovladače jsou popsány na obrázku.

# 1.krok

## Výpočet velikosti vektoru (jak daleko je páčka od jejího středu)

*magnitude = math.sqrt( math.pow(leftXaxis, 2) + math.pow(leftYaxis, 2) )*

# 1,5.krok-Důležitý mezikrok

## Ujištění se nepřekročení maximální hodnoty velikosti vektoru magnitude

Pokud se povede vypočítat velikost vektoru (dále jen magnitude) větší než 1, přiřadíme mu 1… Nesmíme překračovat hranice maximálních hodnot – otáčky motorů by mohly překročit hranici.

*if magnitude > 1: magnitude = 1*

# 2.krok

## VÃ½sledek obrÃ¡zku pro direction of vectorUrčení směru vektoru páčky (direction of vector)

Směr vektoru je velikost úhlu od vodorovné osy X – nad osou je úhel kladný, pod osou záporný. Vzorec je dělení dvěma čísly, přičemž ani u jednoho se nepřipouští nulová hodnota, protože by rozbila celý výpočet. Minimální hodnotu blízko nule kterémukoliv číslu musíme oželit, i když to bude velice malá nepřesnost.  
(Výsledek směru vektoru se v matematice označuje písmenkem THETA.)

*if leftXaxis == 0: leftXaxis = 0.0000001  
if leftYaxis == 0: leftYaxis = 0.0000001*

*direction = math.atan( leftYaxis/leftXaxis )*

# 2,5.krok – Důležitý mezikrok

## Přepočet směru vektoru od 0° a nebo 180°(PI)

Zde přepočet vždy vychází správně pouze do 90° a v mé matematice vždy vycházela čísla pouze v úhlech vpravo od svislé osy Y. Musím zjistit, zda se posouvám vlevo nebo vpravo, a podle toho určit směr vektoru od 0° nebo 180°.

if leftXaxis < 0: direction = math.pi – direction

elif leftXaxis > 0: direction = 0 – direction

# 3.krok

## Určení vektorových komponent Vx a Vy.

Využívá se zde směru páčky, který jsme vypočítali. Tyto vektorové komponenty poté ve finálním kroku slouží jako konstanty pro jasné určení jak rychlosti otáček jednotlivých motorů tak orientace jejich otáčení (vpřed nebo vzad).

*vectorX = math.cos(direction) \* magnitude  
vectorY = math.sin(direction) \* magnitude*

# 4.krok

## Určení rychlostí jednotlivých motorů

Motor1 se pohybuje pouze na ose X, tedy počítání s komponentou Vy nemá význam.  
Motor2 i Motor3 se pohybují v obou osách, vzorec je tedy o něco složitější.

*motor1Speed = -1 \* (vectorX)  
motor2Speed = 0.5\*vectorX + math.sqrt(3/2)\*vectorY  
motor3Speed = 0.5\*vectorX - math.sqrt(3/2)\*vectorY*

# 5.krok

## Ujištění se maximálních hodnot pro motory a jejich vynásobení pro reálné přenesení na hodnoty pro jejich rotaci

Rychlost každého motoru se vynásobí 255ti, protože Arduino operuje s motory na hodnotách od 0 do 255. Poté se číslo zaokrouhlí na celé, aby přenos po sběrnici na Arduino nebyl moc velký.

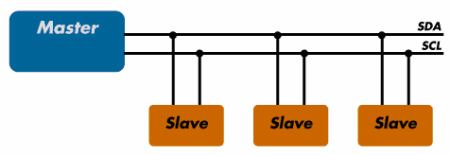
*if abs(motor1Speed) <= 1:  
 motor1Speed = int(round(motor1Speed \* 255, 0))  
else:  
 motor1Speed = int(motor1Speed) \* 255  
  
if abs(motor2Speed) <= 1:  
 motor2Speed = int(round(motor2Speed \* 255, 0))  
else:  
 motor2Speed = int(motor2Speed) \* 255*

*if abs(motor3Speed) <= 1:  
 motor3Speed = int(round(motor3Speed \* 255, 0))  
else:  
 motor3Speed = int(motor3Speed) \* 255*

**Nyní už probíhá pouze přenos hodnot pro motory na Arduino.**

# Charakteristika I2C sběrnice

Sběrnice I2C používá společný hodinový signál (data se tedy přenáší synchronně), ovšem komunikace je poloduplexní – v jeden okamžik může existovat pouze jedno vysílající zařízení a libovolný počet zařízení (většinou však jen jedno), která data přijímají. Komunikace může probíhat oběma směry, sběrnice je tedy sériová. U sběrnice I2C také není použit výběr zařízení typu slave pomocí zvláštních signálů (CS), protože každému uzlu může být přiřazena jednoznačná adresa – kromě elektrických charakteristik je totiž ve specifikaci přesně stanoven i základní komunikační protokol, což je další – a to dosti podstatný – rozdíl oproti výše popsané sběrnici SPI. Obecně je možné říci, že I2C je sice poněkud složitější, ale zato flexibilnější sériovou externí sběrnicí, která se velmi často používá i pro komunikaci na delší vzdálenosti (řádově metry).

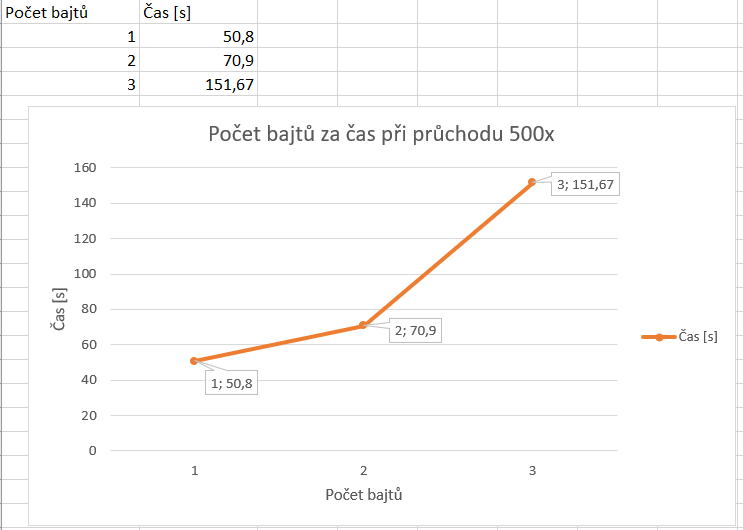
Na obrázku je zobrazen princip propojení několika uzlů pomocí sběrnice I2C. Sběrnice je tvořena dvojicí signálových vodičů. První signálový vodič slouží pro obousměrný přenos dat (SDA – serial data), druhým vodičem pak zařízení typu master posílá všem ostatním zařízením hodinové signály (SCK – serial clock). V praxi je navíc nutné k této dvojici vodičů přidat i společnou signálovou zem (GND – ground). Ani to však ještě pro úspěšnou komunikaci nestačí, protože je navíc nutné oba signálové vodiče, tj. jak SDA, tak i SCK, připojit přes pull-up rezistory o odporu cca 1,5 kΩ na napájecí napětí. Důvod je jednoduchý – ve chvíli, kdy jsou všechny uzly nečinné (mohou být klidně i odpojeny od sběrnice), zvedají tyto dva rezistory napětí na obou signálových vodičích na úroveň logické jedničky, což je normou stanovený klidový stav. Sběrnice může v tomto stavu zůstat po libovolně dlouhou dobu – některé mikrořadiče dokonce mohou přejít do režimu „spánku“ (nízké spotřeby, snížení vyzařování) s tím, že při zahájení komunikace se automaticky aktivují.

Zdroj: <https://www.root.cz/clanky/komunikace-po-seriove-sbernici-isup2supc/>

# Komunikace Raspberry Pi a Arduina přes sběrnici I2C

Pro přesnou komunikaci a výměnu dat je nutné mít domluvený formát dat, který se bude mezi Raspberry Pi a Arduinem posílat. Pro všechny možné příkazy musí vyjít dostatek místa pro data, ale zároveň čím více je dat, tím pomalejší je přenos. Rozhodnutí tedy padlo na přenos několika bytů. Jednotlivé skupiny bitů v těchto bytech budou reprezentovat hodnotu, kterou požadujeme.

Byla provedena měření na přenos jednoho, dvou a tří bytů v určitém počtu opakování. Tyto byty obsahovaly naše požadovaná data. Posílání dat jedním bytem je málo, všechna data se nám na tak malý prostor nevejdou. Výsledek posílání dvěma byty se zvýšil časově od jednoho jen o 20 sekund, což ještě není špatné. Na 500 opakování je zpoždění 4 setiny vteřiny na každý přenos, což je čas, který je v reálném světě při komunikaci postradatelný. Výsledek posílání tří bytů po sběrnici ovšem nemile překvapil. Očekávání bylo, že čas se zvedne lineárně, tak jako mezi jedním a dvěma byty. Čas výsledku tohoto průběhu je více než dvojnásobný. Proběhly i pokusy snížit čekání na Raspberry Pi, aby se instrukce odesílaly rychleji, ale sběrnice v průběhu přenosu selhala a vyhodila chybovou hlášku, protože přenos nestíhala. Kdyby výsledek přenosu tří bytů byl lineární, nebo i trošku více, rozhodnutí by bylo těžké, možná i přiklánějící se přenosu tří bytů. Do tří bytů by se nám vlezly všechny potřebné hodnoty s opravdu vysokou přesností. Bylo by místo pro možná rozšíření v budoucnu. Graf s výsledky rozhodnul, že přenos dvou bytů je nejefektivnější varianta. Je rychlá a pro odesílaná data dostačující.



Ovšem s výběrem omezeného množství bytů přichází i řada problémů, které se musí vyřešit. Nevlezou se do bytů všechna čísla instrukce v původních hodnotách. Byte je číslo, které může nabývat pouze hodnot od 0 do 255. Obsahuje 8 bitů. Nemůžeme tedy přenášet informaci o úhlu, který může nabývat hodnoty až do 360 stupňů jen tak v jednom bytu. Prvním rozhodnutím bylo, že se 360 stupňů konstantou upraví na hodnotu nepřesahující maximum 255, abychom ve dvou bytech ušetřili místo. Z důvodu přebytku množství informací a nedostatku místa je nutné byty zkombinovat. Konec jednoho, a začátek druhého bytu bude obsahovat právě tuto upravenou hodnotu úhlu, který bude poté opět převeden na Arduinu stejným způsobem nazpět.

## Nastavení přenosu I2C

Raspberry bez dodatečných nastavení přes I2C komunikovat nemůže. Naší implementaci předchází řada nastavení na Raspberry, než jej můžeme začít používat. Musí se i2c smazat z blacklistu konfigurací, do /etc/modules se musí vepsat modul i2c-dev, nainstalovat balíček i2c-tools, přidat přístup k I2C sběrnici systémovému uživateli pi. Poslední věc pro možnost používání i2c rozhraní je nyní už pouze zkontrolovat, zda je jako interface (rozhraní) k používání povoleno. Po zadání příkazu raspi-config v terminálu se dostaneme do nejrůznějších nastavení Raspberry Pi. Naše hledaná záložka je Interfaces. Zde je právě možnost I2C Interface, na který klikneme, a Raspberry se nás zeptá, zda chceme tento interface povolit. Po těchto krocích Raspberry zrestartujeme. Poté už příkazem můžeme zobrazit tabulku s adresami, na kterých jsou připojená I2C zařízení. Nainstaluje se nyní knihovna pro jazyk Python (python-smbus) pomocí linuxového instalátoru balíčků apt-get install, která nám umožní ke sběrnici v kódu přistupovat.

Na Raspberry i Arduinu se musí nastavit stejná I2C adresa, přes kterou spolu budou komunikovat. Na Raspberry se nastaví pouze adresa 0x04, na Arduinu musíme definovat, že je Slave, neboli podřízený, a k němu nastavíme stejnou adresu 0x04. Definováním Arduina jako podřízeného řešíme také odlišnosti v hodnotě napájení mezi Raspberry a Arduinem. Raspberry totiž napájí na 3,3V, zatímco Arduino napájí na 5 V, což by mohlo směrem z Arduina na Raspberry způsobovat problémy. Jelikož jsme ale tuto cestu „utlumili“, tak jednosměrných 3,3 V z Raspberry směrem na 5 V Arduino nezpůsobuje žádné problémy. Na Raspberry se oba byty ve správném formátu uloží do pole. Do metody knihovny smbus pro I2C přenos se vkládají tři parametry. I2C adresa Arduina, offset neboli posunutí dat od začátku přenosu a samotná data (v tomto případě naše byty).

## Struktura přenášených bytů

Přenášíme tedy dva byty. Tyto dva byty jsou po vytvoření jejich struktury uloženy do pole a poslány na Arduino. Tyto dva byty mají následující strukturu jednotlivých bitů:

**První byte:**  
**Bity 1-3:** Kód primitiva prováděné instrukce. Podle tohoto kódu se Arduino rozhoduje, jaký typ pohybu se provádí. Např. „Jet“, „Zatočit“, „Zabrzdit“, „Vypnout motory“, „Nastavit rychlost“.  
**Bity 4-8**: První část parametru úhlu prováděné instrukce. Úhel je příliš velký. Je konstantou upraven na hodnotu nepřesahující 255, která se poté na Arduinu stejnou konstantou vrátí nazpět, abychom se opět pohybovali v celých 360 stupních.

**Druhý byte:**  
**Bity 1-3:** Druhá část parametru úhlu prováděné instrukce. Úhel je příliš velký. Je konstantou upraven na hodnotu nepřesahující 255, která se poté na Arduinu stejnou konstantou vrátí nazpět, abychom se opět pohybovali v celých 360 stupních.  
**Bity 4-7**: Hodnota vzdálenosti. Jednotka této hodnoty jsou centimetry. Může označovat, jak daleko má podvozek cestovat, popřípadě poloměr zatáčení atd…  
**Bit 8**: Tzv. „Bezpečnostní bit“. Pokud je po provedení přenosu 1, data nebyla během přenosu poškozena. Pokud bude 0, data mohla být modulována nebo poškozena a příkaz se neprovede.

# Původní návrh získání všech hodnot z bytů na Arduinu

Na Arduinu byl deklarován řetězec, který k sobě jednotlivé vytažené bity přijaté instrukce přidával na konec. Aby k sobě řetězec mohl bity přidat, musely se bity převést z čísla na řetězec. Zde se návrh ukončil, než se vůbec došlo na výpočty a následné kontroly motorů. Práce s řetězcem zabírá velké množství místa i času. Když provádíme operace pouze nad číselnými datovými typy, v tomto případě typem byte, pracujeme obyčejně s jedním bytem, tedy hodnotami od 0 do 255. Když pracujeme s řetězcem, každý jeho znak zabírá 1 byte. Abychom tedy mohli využívat náš výsledný řetězec, který je v tomto případě dlouhý 16 znaků, musíme do něj všech 16 bytů vložit. Hodnoty z tohoto řetězce se vybíraly podle indexu začátku určité hodnoty, a indexu konce určité hodnoty. Například pro získání délky v centimetrech, kolik má náš podvozek ujet, musíme z řetězce vybrat hodnoty na pozicích 12-15. Tato operace je zdlouhavá a nežádoucí. Přešlo se tedy na bitové operace, které se používají u číselných datových typů.

# Použitý návrh získání hodnot z bytů na Arduinu

**Binární číslo** je reprezentováno posloupností binárních hodnot (jedniček a nul) a takto reprezentovaná čísla se používají v digitální technice.

**Bitová operace** (anglicky bitwise operation) je operace, která aplikuje určitou logickou operaci nikoliv na jednotlivé bity, ale na celé vektory bitů. Většina procesorů má strojové instrukce, které provádějí bitové operace. Tyto operace typicky nejsou pomalejší než provádění sčítání a odečítání.  
Dalšími bitovými operacemi, které jsou dostupné jako strojové instrukce, jsou operace bitových posuvů a rotací. Posuv o jeden bit doleva lze interpretovat jako znásobení operandu číslem dvě, posuv o jeden bit doprava jako vydělení dvěma. Posuvy o více bitů pak jako násobení nebo dělení příslušnou mocninou čísla dvě. Posuvy o jeden bit jsou mnohem rychlejší než operace násobení a dělení. Posuvy o větší počet bitů se na některých procesorech provádějí opakovaným posuvem o jeden bit. Co se ovšem týče výpočtové náročnosti, naše číselné posuvy jsou mnohem rychlejší než transformace přes dlouhé řetězce.

# Bitové operace použité na Arduinu

V každém bytu je potřeba zjistit pouze určité skupiny bitů. V tomto případě třeba první tři, nebo prostřední 4 v druhém bytu. Nejrychlejší, a přitom stále jednoduchá operace pro zjištění těchto pár hodnot je operace bitového posuvu.

## Bitový posuv (bitshift)

Při provádění bitového posuvu se vezme celé číslo jako operand a aplikuje se na něj operátor „<<” (posuv vlevo) nebo “>>” (posuv vpravo), a za něj operand, o kolik míst se všechny bity posunou. Existují však různé druhy posuvu podle toho, jakými hodnotami vyplňuje krajní pozice.  
**Aritmetický posuv** používá operátory „<<“ a „>>“. Respektuje kódování čísla, které je posouváno, a proto má i aritmetický význam - posuv o jednu pozici vlevo odpovídá násobení dvěma a posuv vpravo dělení. Opakovaným aritmetickým posuvem lze tedy velmi efektivně násobit a dělit mocninami čísla 2. V případě doplňkového kódu a záporného čísla (tzn. nejvýznamnější bit je 1) jsou směrem od nejvýznamnějšího nasouvány hodnoty 1, jinak 0. Tento typ posuvu je použit v naší implementaci, ovšem vždy se při posuvu generují na kraji nuly, protože nepracujeme se zápornými čísly.  
**Logický posuv** používá operátory „<<<“ a „>>>“. Nemá žádný aritmetický význam, z obou krajních pozic generuje vždy nuly.  
Příklad: Je potřeba zjistit první 3 bity prvního bytu pro zjištění našeho kódu primitive prováděné instrukce. Byte vypadá následovně: “11100101”. Arduino tento byte reprezentuje jako číslo. Tento byte je hodnota 229, ale není potřebná. Pro zjištění prvních tří bitů se provede operace „229 >> 5“. Celé číslo posunulo všechny své bity o 5 míst doprava. Nyní byte vypadá takto: “00000111”. Nyní jsme získali hodnotu, našeho primitva, podle kterého se budeme rozhodovat, jaký pohyb se bude provádět.

## Bitový součet

Bitový součet využívá operátor „|“. Také využívá dva operandy, ale tentokrát jsou jimi obyčejně naše dvě vybraná porovnávaná čísla. Bere dva bitové vzory o stejné délce a vytváří nový bitový vzor, jehož hodnota závisí na hodnotě vstupních bitových vzorů. Porovnává postupně jeden po druhém příslušné bity (první bit prvního vzoru s prvním bitem druhého vzoru, druhý bit prvního vzoru s druhým bitem druhého vzoru atd.) a provádí s každým párem logickou operaci OR (součet).   
Příklad: Potřebuji spojit 5 bitů prvního bytu s prvními třemi bity druhého bytu, a získat tak úhel směru jízdy. Vezmeme si první byte z minulého příkladu: “11100101”. Bitový posuv o tři doprava nám vrátí byte, který bude mít na prvních pěti bitech naši první část bitů, poslední tři budou vždy nuly. Tedy „00101000“ (40). Druhý byte bude: “00101101” (45). Bitový posuv o pět doprava nám vrátí byte, který bude mít na posledních třech bitech naši druhou část bitů, prvních pět bitů budou vždy nuly. Tedy „00000001“ (1). Nyní proběhne součet prvního a druhého upraveného bytu a dostaneme náš správný hledaný úhel od 0 do 255. Proběhne tedy operace „40 | 1“, neboli (40+1). Je to jako obyčejný součet pod sebou 00101000 + 00000001. Tento součet upravené hodnoty musíme definovanou konstantou v programu vrátit zpět do maxima 360 stupňů.

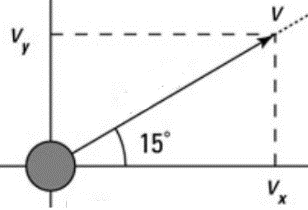
Zdroj pro bitové operace: <http://voho.eu/wiki/bit/>

# Použitý návrh Matematiky pro motory na Arduinu v krocích

O tom, jakým způsobem se podvozek bude chovat, závisí na typu instrukce, kterou po něm budeme požadovat, tedy jaký typ instrukce od Raspberry Pi po I2C sběrnici Arduino přijme. Pro různé typy instrukcí se tedy vykonává jiná posloupnost výpočtů, podle kterých se poté motory, jakožto celý podvozek zachovají.

## Přímá jízda

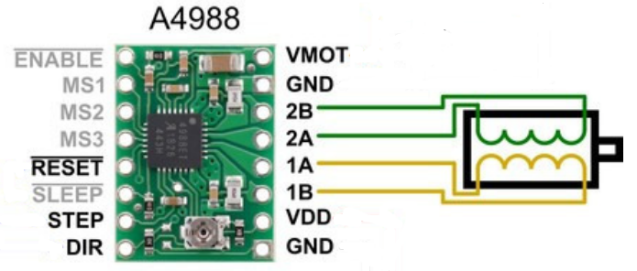
Po přijmutí dat k tomuto typu instrukce víme před začátkem všech výpočtů dvě věci, které Arduino obdrželo, a je potřeba je ve výpočtu využít. První informace je úhel, který po přepočtení má hodnotu 0-360 stupňů. Druhá informace je vzdálenost, kterou podvozek pojede.

**1)** **Zjistíme velikosti vektoru pohybu na ose X a ose Y pro daný úhel jízdy.** Konstanta WHEELS\_MAGNITUDE je nastavena na 1. Je zde pro případnou rychlou a jednoduchou úpravu celkové rychlosti všech motorů.   
*double vectorX = cos(receivedAngle) \* WHEELS\_MAGNITUDE;  
double vectorY = sin(receivedAngle) \* WHEELS\_MAGNITUDE;*

**2) Zjistíme rychlosti otáček za sekundu pro jednotlivé motory.**Motor1 se pohybuje pouze na ose X, tedy počítání s komponentou Vy nemá význam. Motor2 i Motor3 se pohybují v obou osách, vzorec je tedy o něco složitější. Motory jsou od sebe s rozestupem 120 stupňů, proto v druhém vzorci můžeme vidět přičítání 120° v radiánech, a v třetím odečítání pro třetí motor. Když se na podvozek budeme dívat shora, kladné výsledné hodnoty otáčí koly proti směru hodinových ručiček, zatímco záporné po směru hodinových ručiček.  
*double motor1Speed = (-1 \* vectorX) \* MAX\_RPS;  
double motor2Speed = (0.5 \* vectorX + sqrt(3)/2 \* vectorY) \* MAX\_RPS;  
double motor3Speed = (0.5 \* vectorX - sqrt(3)/2 \* vectorY) \* MAX\_RPS;*

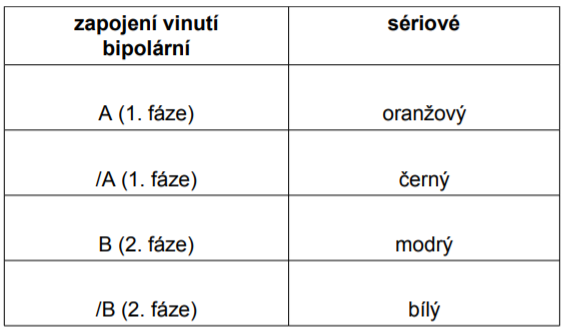
# Díly a součásti použité pro sestavení podvozku

# Zapojení krokového motoru k motor driveru A4988

[](https://www.pololu.com/product/1182)Krokové motory použité v této bakalářské práci jsou bipolární: Vedou z nich tedy čtyři dráty, z každého vinutí dva. Jeden pár drátů kontroluje polaritu jedné cívky elektromotoru, druhý pár kontroluje polaritu cívky druhé. Je tedy zásadní správně tyto dráty zapojit. Pokud bychom dráty zapojili špatně, například bychom pomíchali dráty mezi páry, motor by se v pozici mezi cívkami začal cukat a už by se dál neotočil. Způsobené by to bylo tím, že jsme zapojili jeden drát z každého páru na jednu cívku, což zapříčiní, že cívky jdou proti sobě.

1. Zapojení párů cívek k motor driveru

Zapojení jednotlivých fází těchto konkrétních krokových motorů je zobrazeno v dokumentech výrobce, firmy Microcon.

[](http://www.microcon.cz/zapojenivinuti2012web/zapojenivinutipdf2012/ZV4lw.pdf)