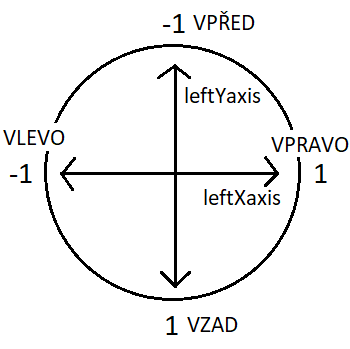
Bakalářská práce – průběžné poznámky

# Původní prvotní 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:

# 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, te

dy 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 motorů na Arduino.

# 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. Grafy sekce tohoto měření rozhodly, že nejoptimálnější řešení, co se času týče, je posílání dvou bytů.

Ovšem jsou tu další problémy, které hned přichází vhod vyřešit. 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 celou hodnotu úhlu.

## 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. Po těchto krocích se Raspberry zrestartuje. Poté už příkazem můžeme zobrazit tabulku s adresami, na kterých jsou připojená I2C zařízení. Nainstaluje se nyní knihovna pro jazyk Python (python-smbus), 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.

**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.  
**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.

# 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.  
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.

# Aktuální přepracovaný návrh Matematiky počítání rychlostí motorů na Arduinu v krocích

Arduino Uno přijímá 2 bajty za sebou přes sběrnici I2C.