Workshop I – Programování robotů

LEGO MINDSTORMS EV3 pomocí softwaru EV3

část 2

FAKULTA ELEKTROTECHNICKÁ ČVUT V PRAZE

robosoutěž

Lektoři: Martin Hlinovský
Martin Šrámek
Matěj Štětka

Podklady poskytl Ing. Jaroslav Honců, CSc. - Jaroslav.Honcu@seznam.cz

Algoritmizace problémů

Algoritmus

Algoritmus je přesný návod či postup, kterým lze vyřešit daný typ úlohy. Pojem algoritmu se nejčastěji objevuje při programování, kdy se jím myslí teoretický princip řešení problému (oproti přesnému zápisu v konkrétním programovacím jazyce). Obecně se ale algoritmus může objevit v jakémkoli jiném odvětví. Jako jistý druh algoritmu se může chápat i např. kuchařský recept

Obecně - stanovení jednoznačného postupu počítačového zpracování dat,

- v robotice jednoznačný postup řešení programového řízení robota

Vlastnosti algoritmů

***Konečnost* (finitnost)**

Každý algoritmus musí skončit v *konečném* počtu kroků. Tento počet kroků může být libovolně velký (podle rozsahu a hodnot vstupních údajů), ale pro každý jednotlivý vstup musí být konečný.

Obecnost (hromadnost, masovost, univerzálnost)
Algoritmus neřeší jeden konkrétní problém (např. „jak spočítat 3×7“), ale obecnou třídu obdobných problémů (např. „jak spočítat součin dvou celých čísel“), má širokou množinu možných vstupů.

Determinovanost
Každý krok algoritmu musí být *jednoznačně* a *přesně* definován; v každé situaci musí být naprosto zřejmé, co a jak se má provést, jak má provádění algoritmu pokračovat, takže pro stejné vstupy dostaneme pokaždé stejné výsledky. Protože **běžný jazyk** obvykle neposkytuje naprostou přesnost a jednoznačnost vyjadřování, byly pro zápis algoritmů navrženy **programovací jazyky**, ve kterých má každý příkaz jasně definovaný význam.

(Vyjádření výpočetní metody v programovacím jazyce se nazývá **program**).

***Výstup* (resultativnost)**

Algoritmus má alespoň jeden *výstup*, veličinu, která je v požadovaném vztahu k zadaným vstupům, a tím tvoří odpověď na problém, který algoritmus řeší (algoritmus vede od zpracování hodnot k výstupu)

Elementárnost

Algoritmus se skládá z konečného počtu jednoduchých (elementárních) kroků.

Algoritmizace

výchozí fáze tvorby složitějších počítačových programů

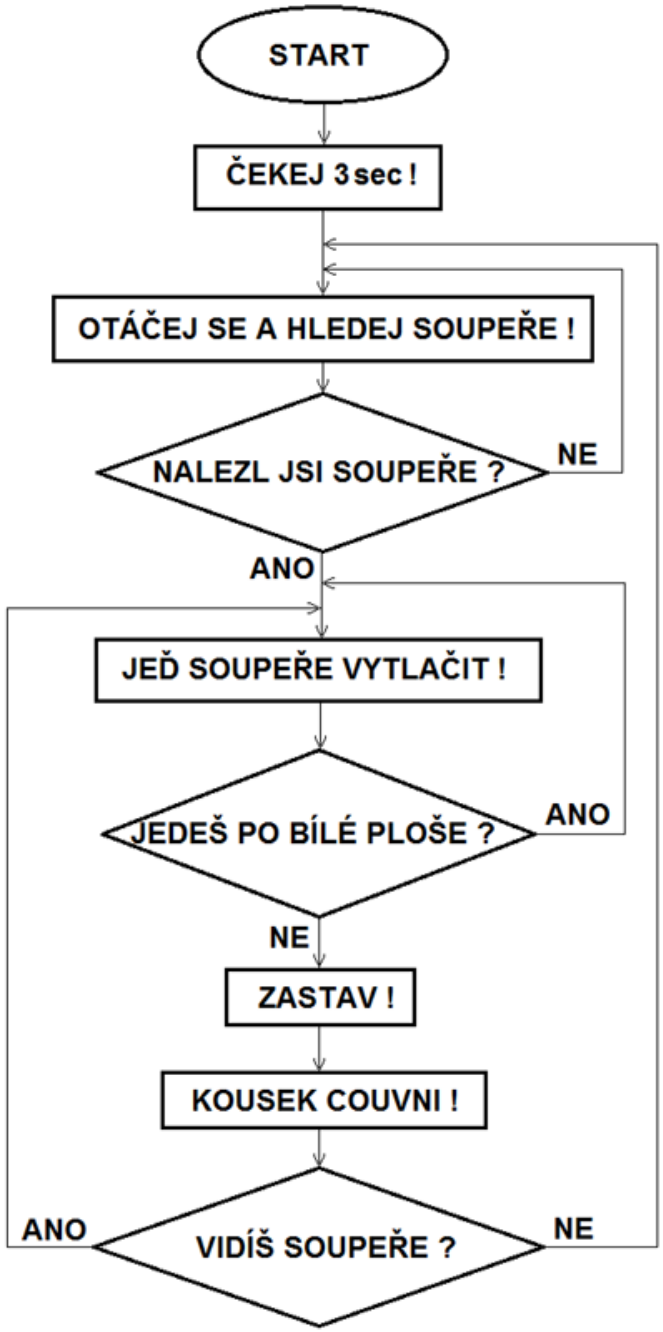
Fáze algoritmizace

- analýza úlohy, upřesnění zadání a stanovení koncepce řešení
- rozdělení úlohy na jednotlivé kroky či podúlohy
- stanovení jednoznačné návaznosti kroků a podúloh
- vytvoření dokumentace pro tvorbu počítačového programu v zadaném jazyce

Zápis algoritmu

přehledné zachycení algoritmu
v textové nebo grafické formě
(viz např. vývojový diagram)

Příklad: *jedna z možností vývojového diagramu programu robota pro soutěž Robosumo*



Metody návrhu

Algoritmus je možno navrhnout několika způsoby:

shora dolů – postup řešení rozkládáme na jednodušší operace, až dospějeme k elementárním krokům

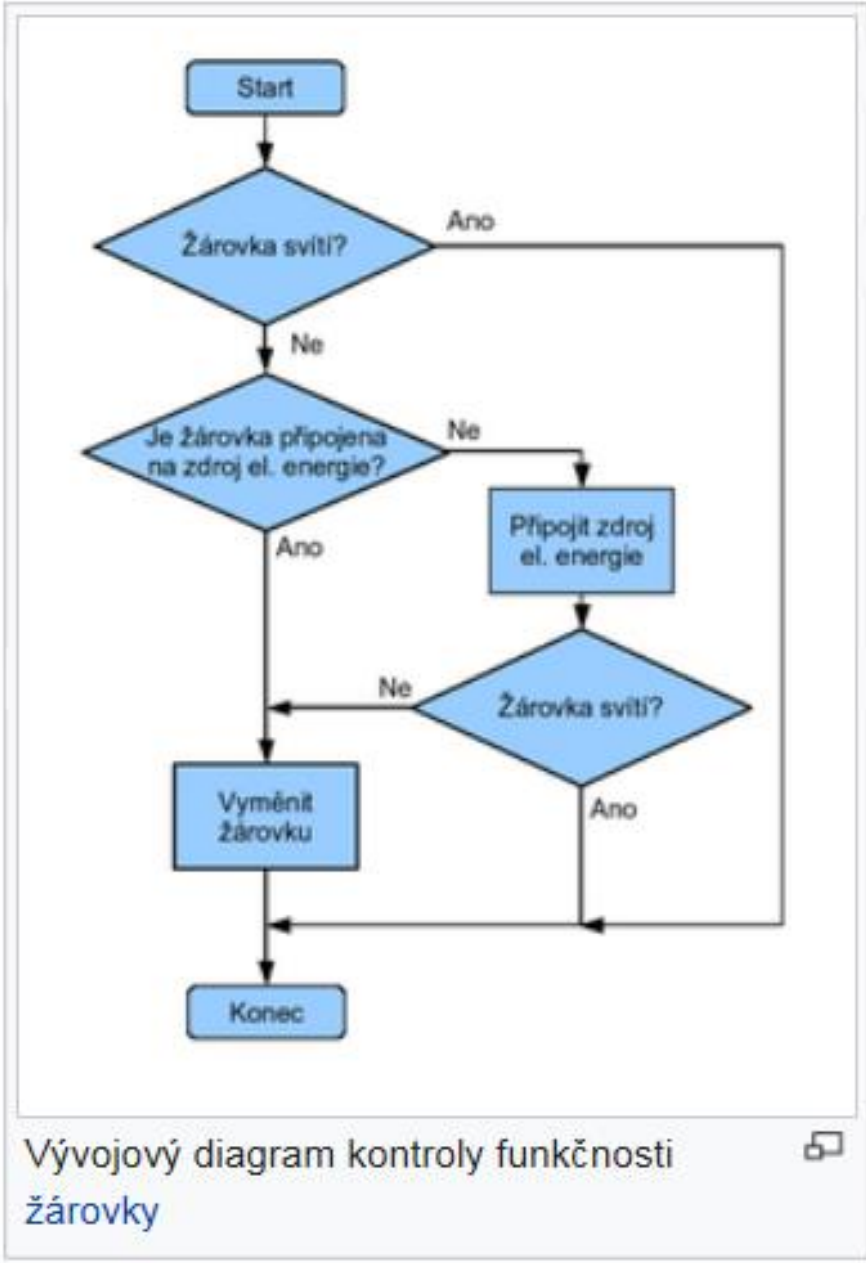
zdola nahoru – z elementárních kroků vytváříme prostředky, které nakonec umožní zvládnout požadovaný problém

kombinace obou – obvyklý postup shora dolů doplníme "částečným krokem" zdola nahoru tím, že se například použijí knihovny funkcí, vyšší programovací jazyk nebo systém pro vytváření programů

Vývojové diagramy

Vývojový diagram

Vývojový diagram je druh diagramu, který slouží ke grafickému znázornění jednotlivých kroků algoritmu, pracovního postupu nebo nějakého procesu. Vývojový diagram obsahuje obrazce různého tvaru (obdélníky, kosočtverce, aj.), navzájem propojené pomocí šipek. Obrazce reprezentují jednotlivé kroky, šipky tok řízení. Vývojové diagramy standardně nezobrazují tok dat, ten je zobrazován pomocí data flow diagramů. Vývojové diagramy jsou často využívány v informatice během programování pro analýzu, návrh, dokumentaci nebo řízení procesu.



Vývojový diagram kontroly funkčnosti žárovky

Význam a použití vybraných symbolů vývojových diagramů

šipka — určuje směr zpracování (zobrazuje „řídící tok“ - šipka směřující z jednoho symbolu a končící u druhého naznačuje, že řídící tok přechází z jednoho symbolu na druhý)

- svislé nebo vodorovné čáry
- mohou se křížit nebo spojovat
- směr dolů a doprava je prioritní, v tomto případě není nutné použít šipky
- šipky se používají jenom v případě, že tento směr je jiný, nebo když je třeba směr toku informace zvýraznit

obdélník s popisem — definuje dílčí krok zpracování (algoritmu)

kosočtverec — podmíněný výraz (větvení postupu v závislosti na splnění podmínky). Je reprezentován kosočtvercem, používá se tam, kde je zapotřebí nějakého rozhodnutí. Má většinou podobu otázky a její odpovědi ve tvaru *Ano/Ne* nebo P*ravda/Nepravda*. Ze symbolu podmíněného výrazu vychází dvě a více šipek, každá z šipek může obsahovat odpověď na danou otázku (doporučuje se odpovědi k šipkám nadepisovat).

obdélník se zaoblenými rohy — počátek nebo konec zpracování, obvykle obsahuje popis „Start“ nebo „Konec“, či podobnou frázi

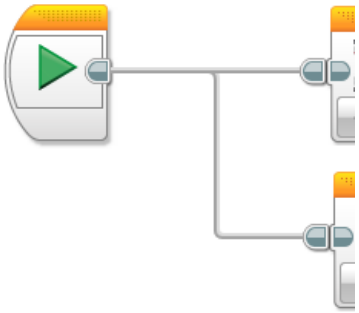
obdélník se svislými čarami po stranách — podprogram - zobrazení skupiny kroků procesu pomocí jednoho symbolu (podprogram je taková část algoritmu, která se může opakovat a která by mohla být tvořena samostatným vývojovým diagramem)

Tvorba programů s paralelními procesy

paralelní procesy (Paralel Process)

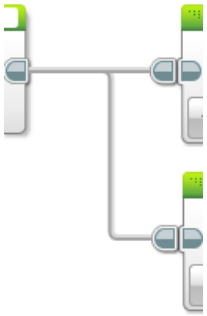
V některých případech můžeme potřebovat paralelní běh několika tzv. vláken programu, např. běh motorů současně s přehráváním melodie.
Je to možné, je nutno však brát ohled na souběh, především akčních členů (motorů apod.), aby na ně nebyly v jednotlivých vláknech programu kladeny současně protichůdné požadavky!

Způsoby větvení programu:



nebo

nebo i



Tvorba programů s datovými spoji

datové spoje (Data Wires)

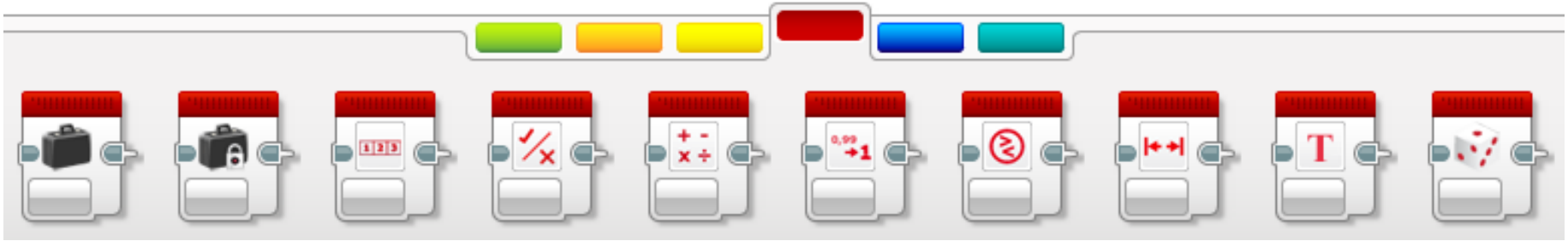
Type	Block Input	Block Output	Block Output Data Wire
Logic			
Numeric			
Text			
Numeric Array			
Logic Array			

Slučitelnost dat

- datový spoj musí spojovat stejný nebo slučitelný typ vstupu a výstupu
- slučitelnost typů dat zajišťuje automatická konverze dle tabulky:

výchozí typ	**cílový typ**	**výsledek**
logický	číslo	False = 0 / True = 1
logický	text	False = "0" / True = "1"
logický	logické pole	pole s jedním prvkem
logický	číselné pole	pole s jedním prvkem (0 nebo 1)
číslo	text	text zobrazující číslo (například "3,5“)
číslo	číselné pole	pole s jedním prvkem
logické pole	číselné pole	pole stejné délky (s prvky o hodnotě 0 nebo 1)

Práce s datovými bloky



Datové bloky (Data Blocks, Data Operations)
(zleva doprava)

- proměnné (Variable)
- konstanty (Constant)
- operace s poli (Array Operations)
- logické operace (Logic Operations)
- matematické operace (Math)
- zaokrouhlení (Round)
- porovnávání (Compare)
- rozsah (Range)
- text (Text)
- náhodné (Random)

Proměnné (Variable)

Blok umožňuje zápis a čtení tzv. proměnné (režimy: Text - Číslo – Logika - Číselné pole - Logické pole).
Umožňuje též tvorbu nových proměnných a jejich pojmenování.
Jde o vyhrazenou část paměti, jejíž obsah lze měnit (tzn. i zapisovat) i při běhu programu (v případě zápisu nových dat jsou předešlá data nahrazena novými).

Konstanty (Constant)

Blok umožňuje zápis a čtení tzv. konstanty (režimy: Text - Číslo – Logika - Číselné pole - Logické pole).
Umožňuje též tvorbu nových konstant a jejich pojmenování.
Jde o vyhrazenou část paměti, jejíž obsah lze při běhu programu pouze číst

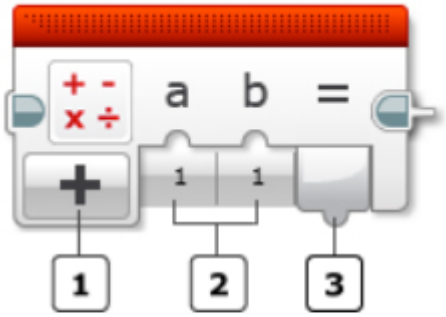
Operace s poli (Array Operations)

Blok umožňuje operace s číselnými poli (Numeric Array) a logickými poli (Logic Array).

Logické operace (Logic Operations)

Blok umožňuje tzv. logické operace, tj. operace s dvouhodnotovou informací (na vstupu je tzv. „logická jednička“ („1“, True, pravda) nebo „logická nula“ („0“, False, lež), výstup je též dvouhodnotový.
K dispozici jsou i logické funkce AND, OR, XOR a NOT.

Matematické operace (Math)



1. Výběr režimu
2. Vstupy
3. Výstup

Režim	Vstupy	Výsledek
Sčítání	A, B	A + B
Odečítání	A, B	A - B
Násobení	A, B	A x B
Dělení	A, B	A : B
Absolutní hodnota	A	|A|
Odmocnina	A	√A
Mocnina	A (základ), N (exponent)	A^N
Pokročilé	A, B, C, D	

další datové bloky:

Zaokrouhlení (Round) 

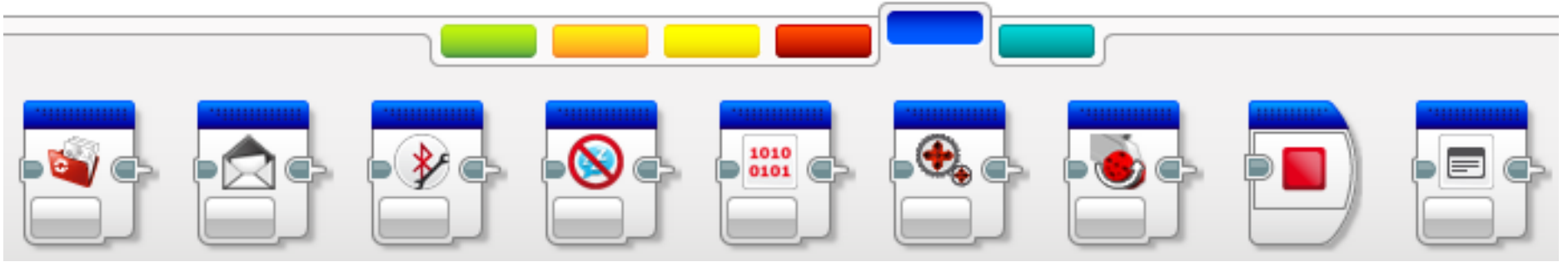
Porovnávání (Compare) 

Rozsah (Range) 

Text (Text) 

Náhodné (Random) 

Práce se speciálními bloky



Speciální bloky (Advanced Blocks, Advanced)
zleva doprava

- zpracování souborů
- zprávy
- Bluetooh připojení
- omezení usínání
- výchozí (neupravená) hodnota senzoru
- neregulovaný motor
- invertování motoru
- stop programu
- komentář

Vytváření vlastních bloků

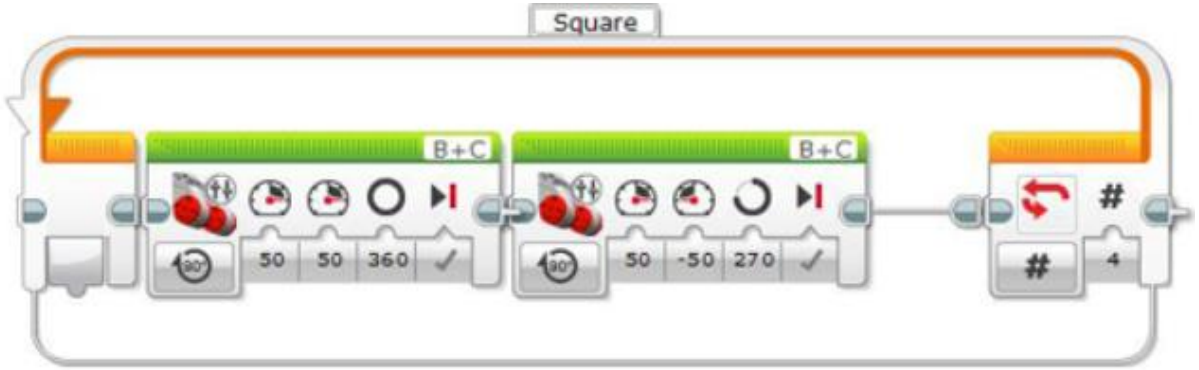
Moje bloky (My Blocks)

Editor vlastních bloků

V případech opakování určité programové sekvence na více místech programu je výhodné ji nahradit jediným blokem. To umožňuje My Block Builder, nový blok je pak zařazen do palety My Blocks.

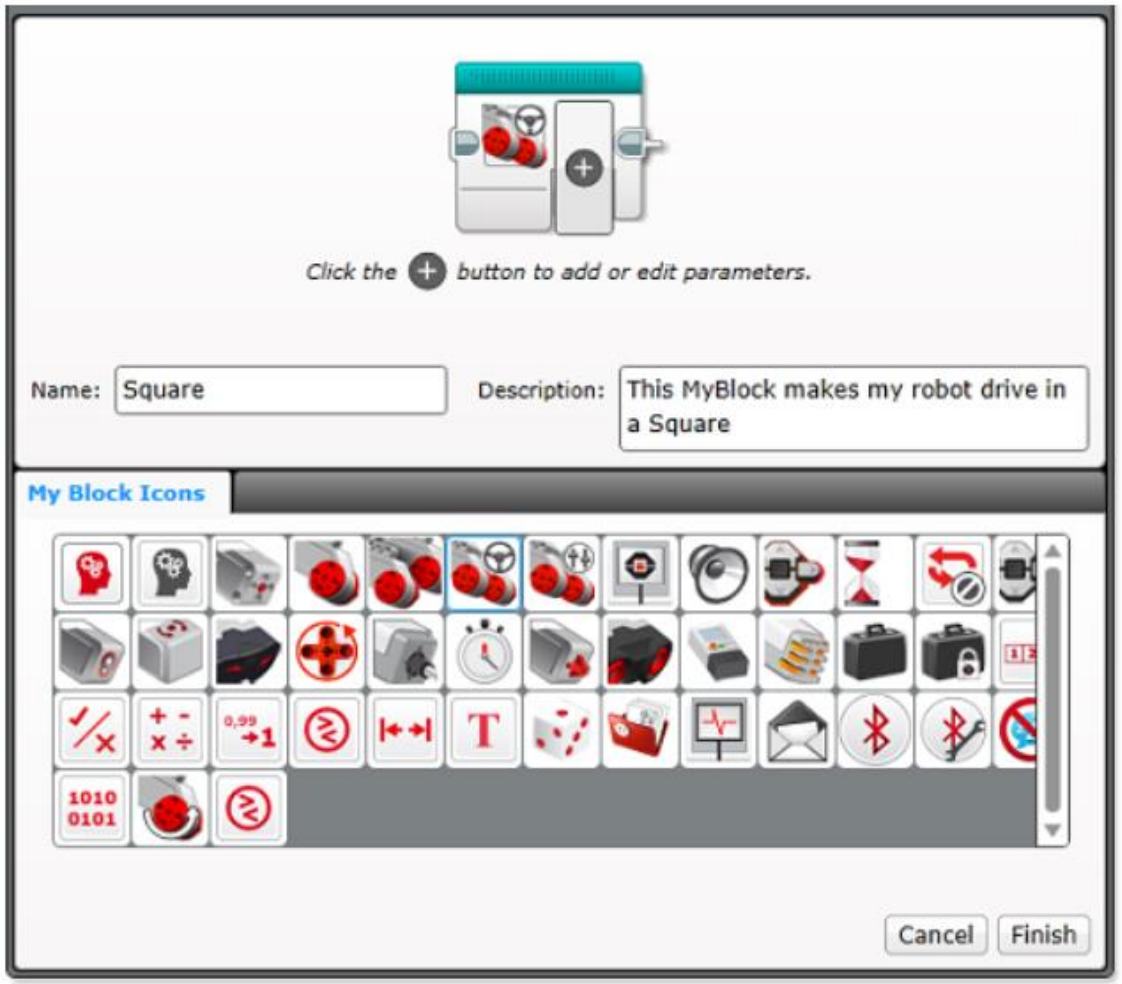
Vytvoření vlastního bloku (příklad):

Máme např. část programu podle kterého robot jezdí do čtverce:



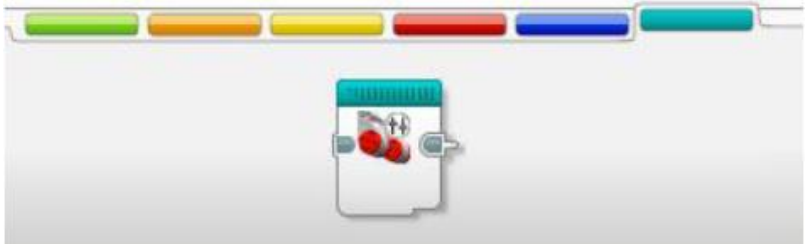
Vytvoření vlastního bloku:

- použijeme „šipku (Select)“ a zvýrazníme všechny bloky které chceme zahrnout do vytvářeného bloku
- klepneme na „My Block Builder“ z „Tools“



- vytvářenému bloku dáme „jméno (Name)“ a přidáme stručný „popis (Description)“
- vybereme ikonu z nabídky „My Block Icons“ pro snadnější identifikaci bloku (pro uvedený příklad robota jezdícího do čtverce budou vhodné ikony s motory)
- klepneme na „Finish (konec)“

Nový blok bude automaticky zařazen do palety My Blocks a bude k dispozici pro všechny programy příslušného projektu.



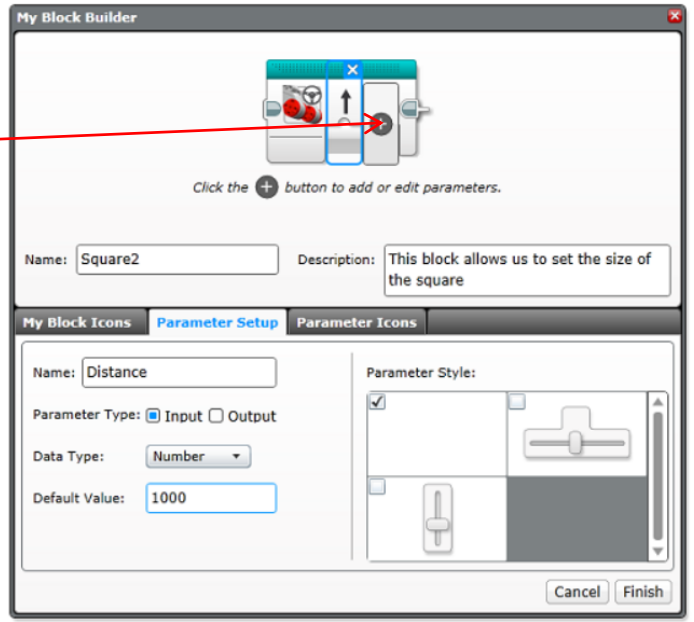
Nyní již můžete používat nový blok v programech, např.:

Po dvojkliku na ikonku bloku se tento otevře jako samostatný program a je možno jej jakkoliv upravovat. Zavře se jako jakýkoliv jiný program (klikem na křížek jeho záložky v horní liště, úprava parametrů zůstane zachována).

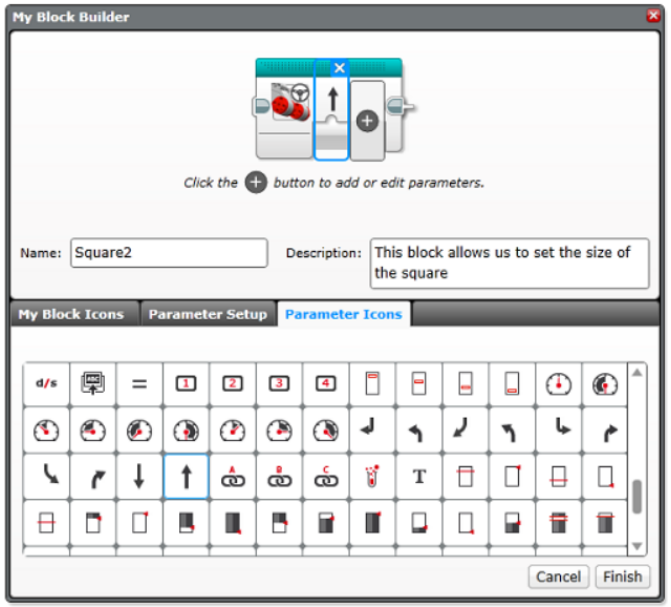
Vlastní blok s parametry

Pro případy častých úprav **vlastního bloku** existuje pohodlnější způsob a to tzv. **vlastní blok s parametry**.
Postup při vytváření takového bloku je shodný s předchozím popisem až ke kroku „Finish“,

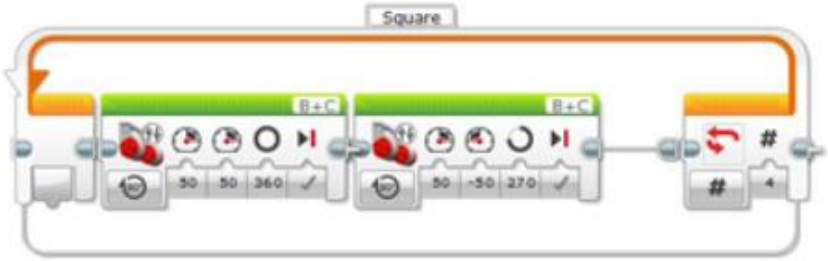
místo toho klikneme na symbol **+** na ikonce.



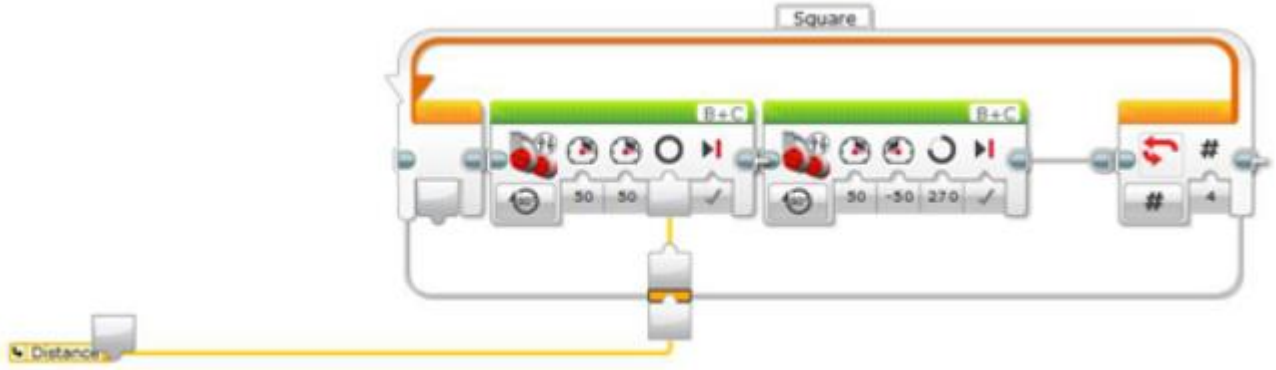
V oddílu „Parameter Setup (nastavení parametrů)“vyplníme či zvolíme příslušné informace, tj. „Name“, „Input/Output“, „Data Type“, „Parameter Style“. Nastavíme i „Default Value (základní hodnota)“ (v tomto případě , při nastavování velikosti čtverce, např. 1000), pro případ že by jsme zapomněli ji nastavit při použití tohoto bloku ve svém programu).
V „Parameter Icons (ikony parametrů)“ vybereme pro tento parametr příhodnou ikonku. V tomto případě, protože budeme nastavovat délku strany objížděného čtverce, bude vhodná např. šipka.



Teď již máme připraveno nastavování parametru, můžeme tedy kliknout na „Finish“.



Nový parametr “Distance” musí být ještě připojen do příslušného vstupu.

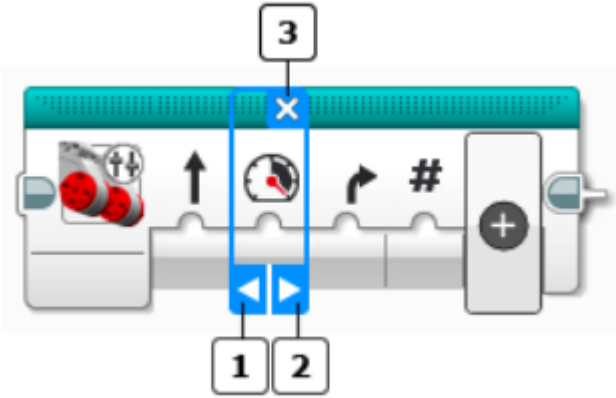


V tomto případě připojíme parametr “Distance” do vstupu „úhel natočení (Degrees)“, tím můžeme nastavovat délku hrany objížděného čtverce.

Zavře se jako jakýkoliv jiný program (klikem na křížek jeho záložky v horní liště, úprava parametrů zůstane zachována).

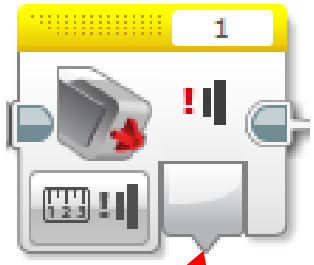
Poznámka: v případě více parametrů je možno v rámci zadávání měnit i pořadí jejich umístění:

1. přesun parametru doleva
2. přesun parametru doprava
3. vymazání parametru



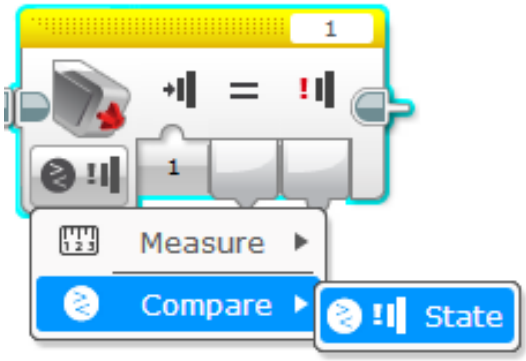
Touch Sensor (dotykový senzor)

Touch Sensor



V módu **Measure** rozeznává **dva stavy**:
stisknuto – výstup True,
uvolněno – výstup False

výstup tzv. datovým spojem, viz kurz dále



V módu **Compare** je možno rozlišit **tři stavy**:
stisk, dotek (Pressed),
uvolnění (Released),
náraz tj. stisk a okamžité uvolnění (Bumped)

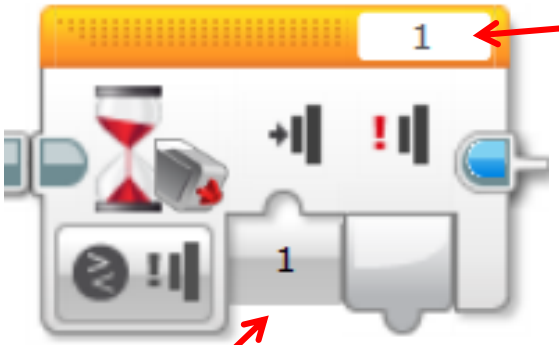
Dotykový senzor lze použít, tak jako většinu senzorů, samostatně (např. k měření) nebo v čekání (***Wait***), ve smyčce (***Loop***) či v přepínači (***Swith***).

Příklady:

mód Wait/Touch Sensor (čekání na stav dotykového senzoru)

Wait 

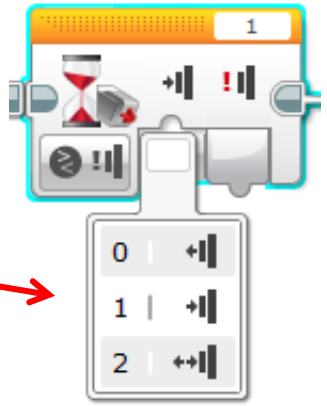
Wait/Touch Sensor



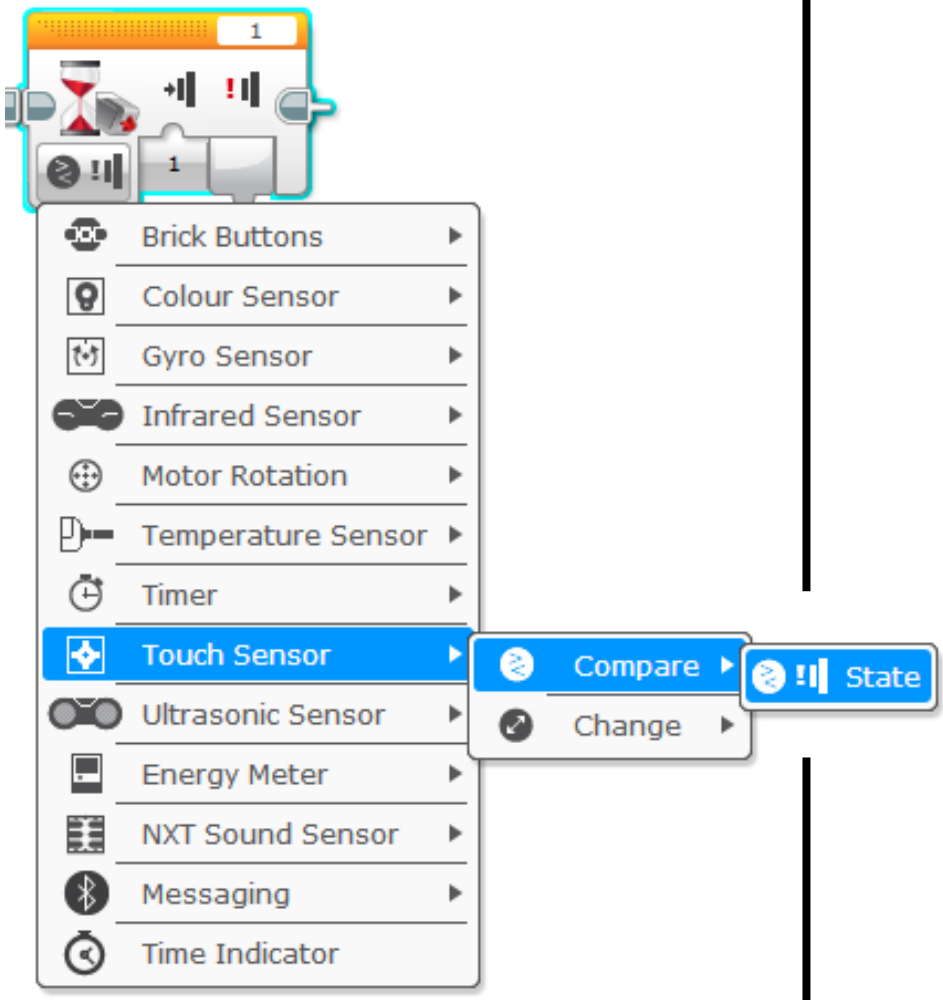
port

Jedná se vlastně o čekání na stav „tlačítka“, podobně jako u tlačítek kostky.
Lze tedy testovat:

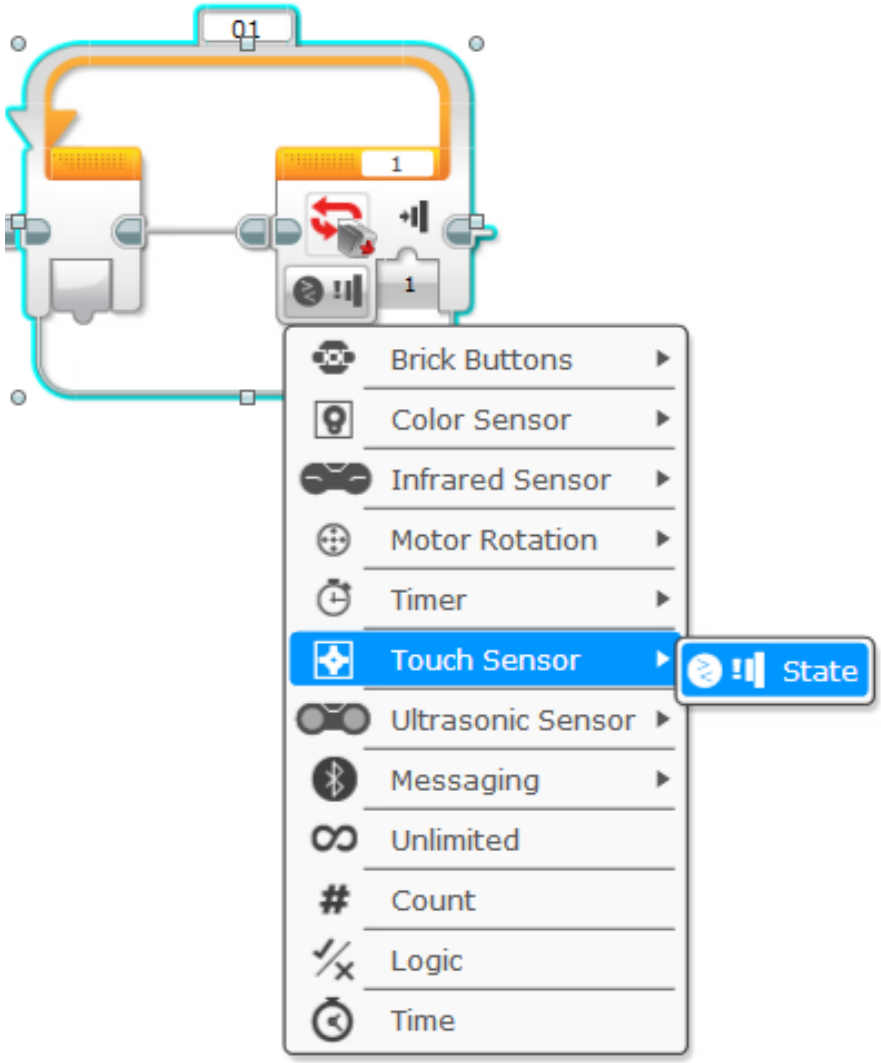
- 0 - uvolnění (Released)
- 1 - stisknutí (Pressed)
- 2 - stisknutí a uvolnění (Bumped)



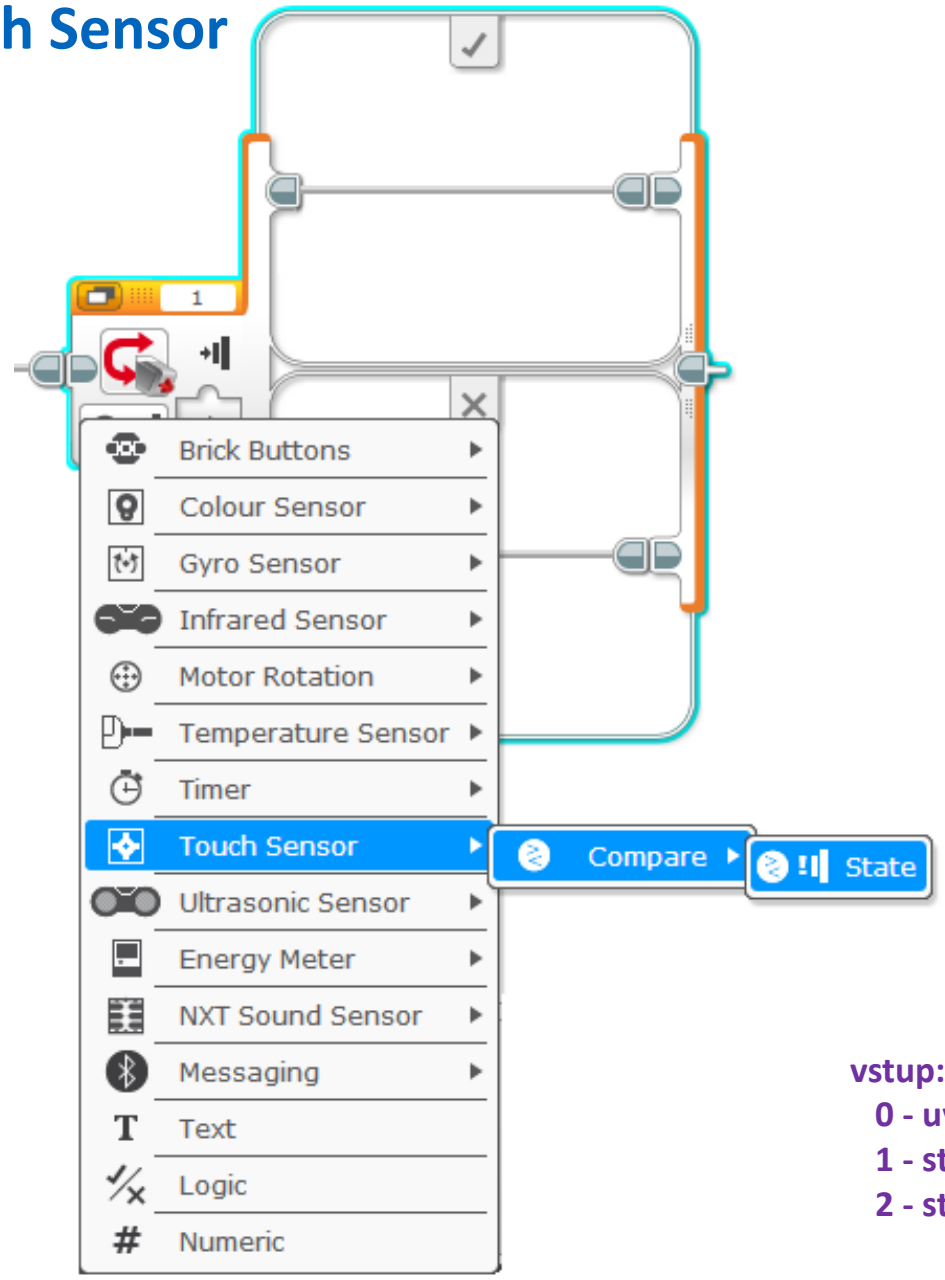
Wait/Touch Sensor

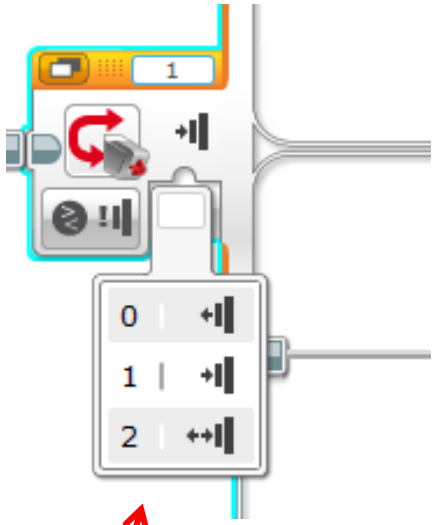


Loop/Touch Sensor



Swith/Touch Sensor





vstup:

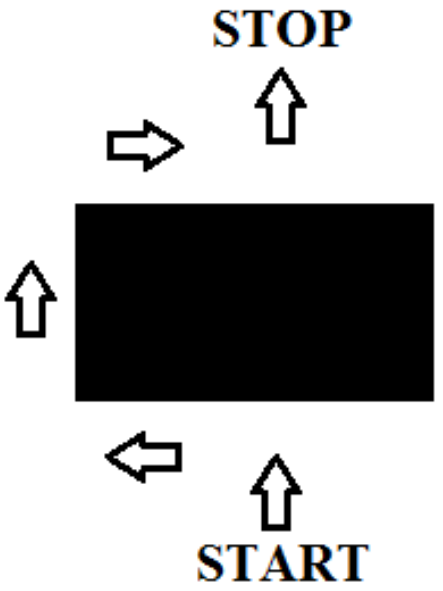
0 - uvolnění (Released)

1 - stisknutí (Pressed)

2 - stisknutí a uvolnění (Bumped)

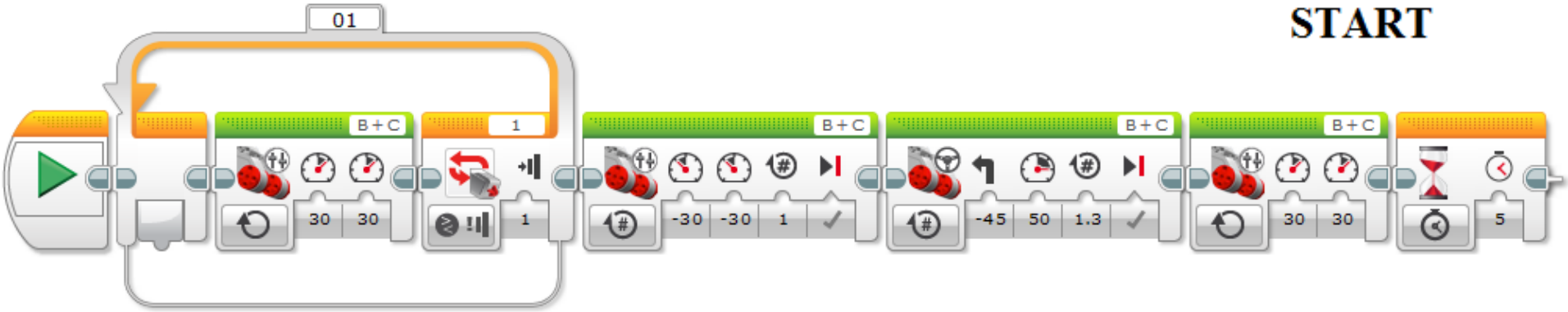
Naprogramujte robota, který pojede dopředu tak dlouho až narazí na překážku, potom trochu couvne, pootočí se a překážku objede!

Pochopitelně - parametry programu budou záviset na použitém objektu coby překážky!



Možné řešení:

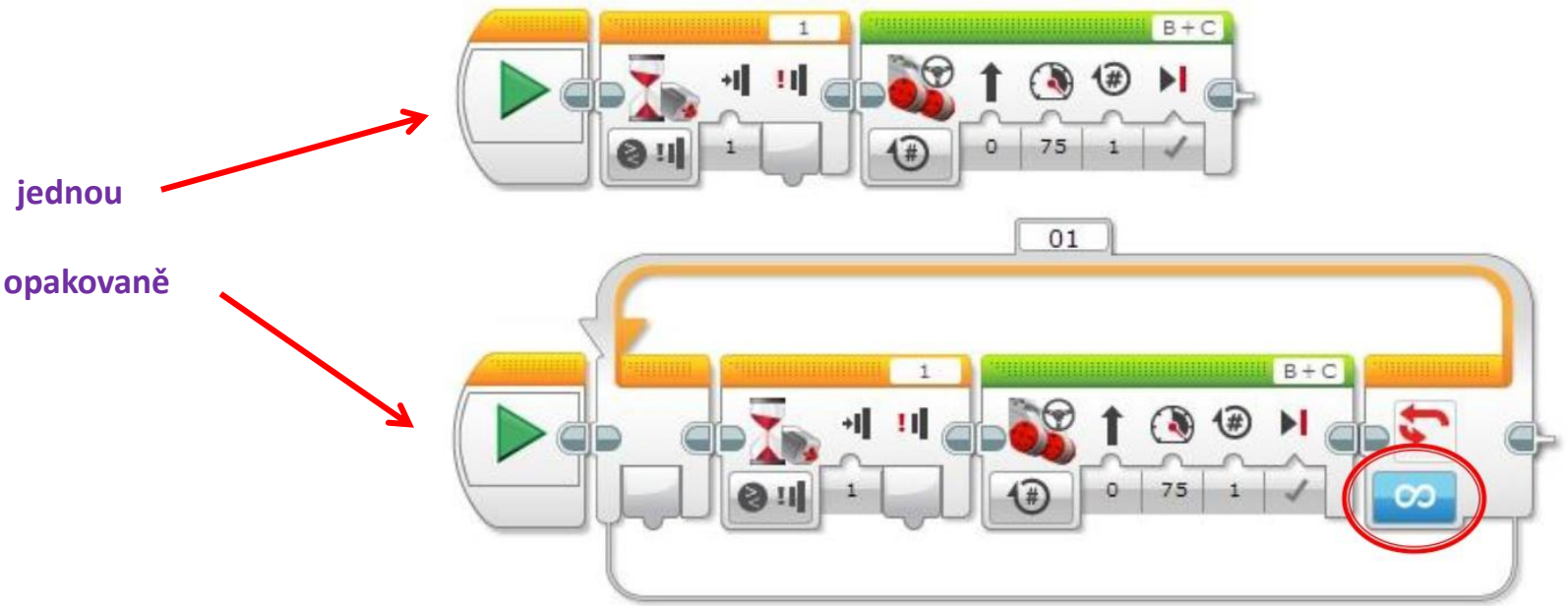
(pouze část řešení – jede, narazí, couvne, pootočí se a kousek popojede)



Připojte ke svému robotu, přes kabel, další dotykový senzor, který bude sloužit jako tlačítko dálkového ovládání! Použijte nejdelší kabel! Naprogramujte robota který zpočátku nebude nic dělat - po stisku připojeného tlačítka pak popojede kousek dopředu a hned se zase zastaví!

Robota tedy budeme pomocí tlačítka jakoby postrkávat (resp. po stisku tlačítka robot poposkočí).

Možné řešení:



Vytvořte a ověřte funkci (postupně) programů:

A *- robot jede jen v případě stisknutého tlačítka, jinak stojí!*

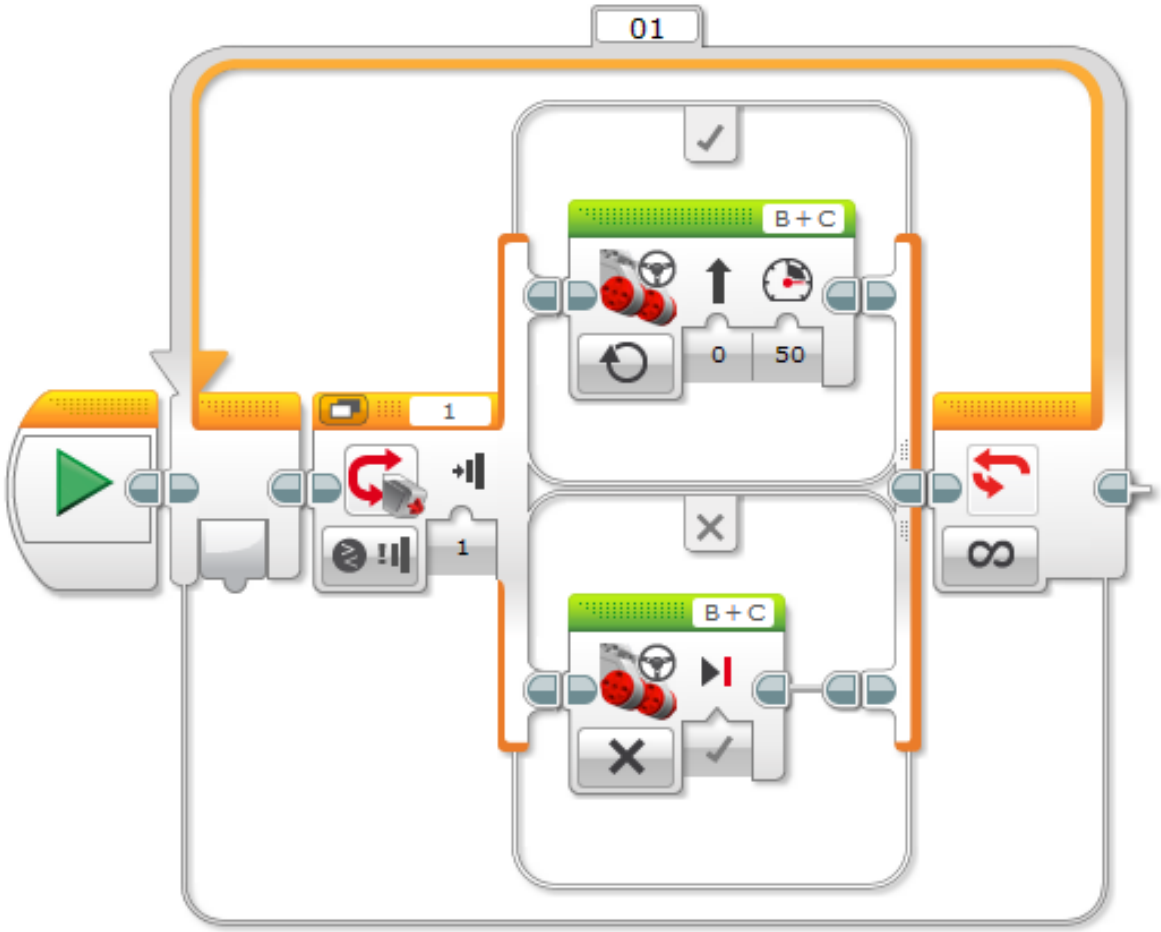
B *- robot po spuštění programu jede, při stisku tlačítka začne zatáčet (tj. jedno kolo se přestane otáčet), po uvolnění tlačítka jede robot opět rovně!*

C *- vylepšete variantu* ***B****: robot po spuštění programu stojí, čeká na krátký stisk tlačítka! Pak se rozjede a tlačítkem se ovládá zatáčení!*

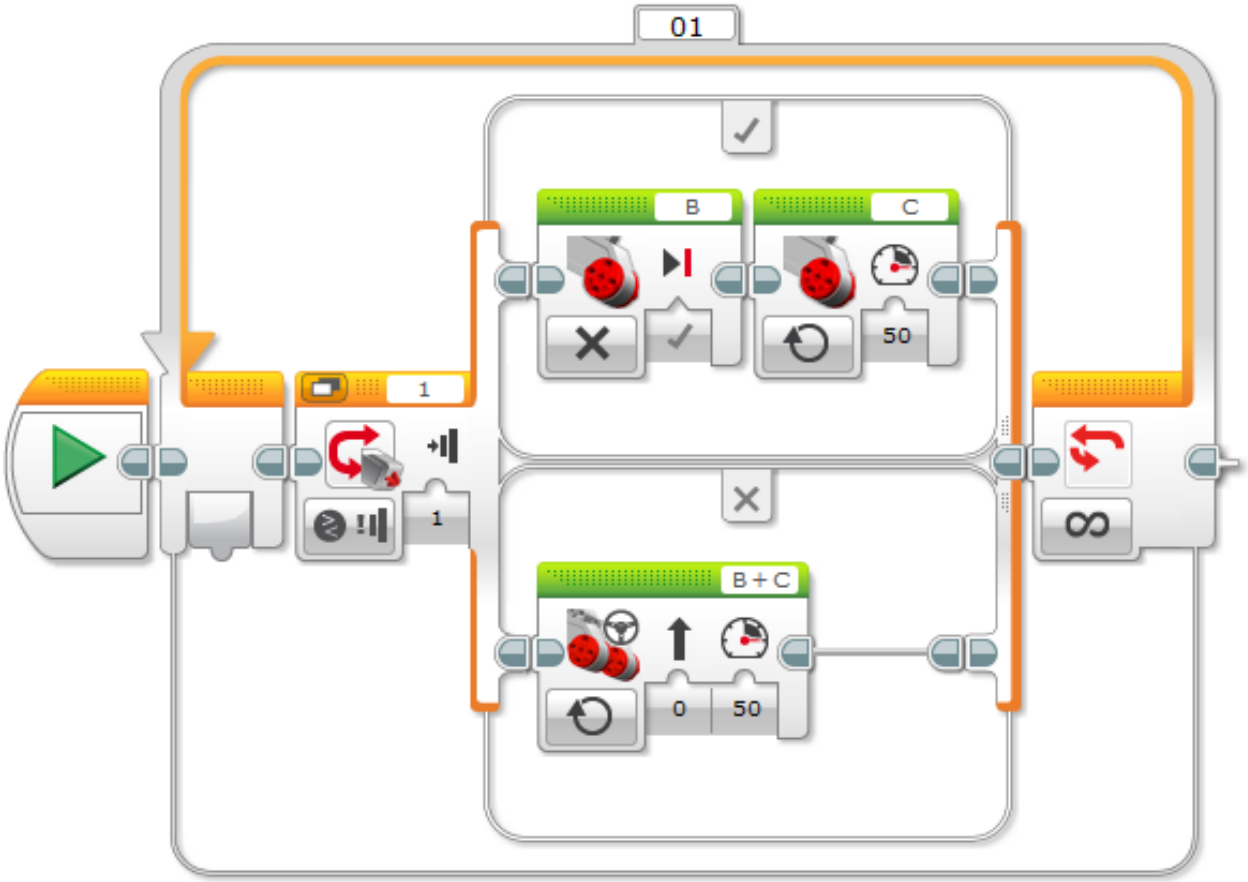
(stisknuté – zatáčí na jednu stranu,
uvolněné – zatáčí na druhou stranu)

Nápověda: použijte Swith

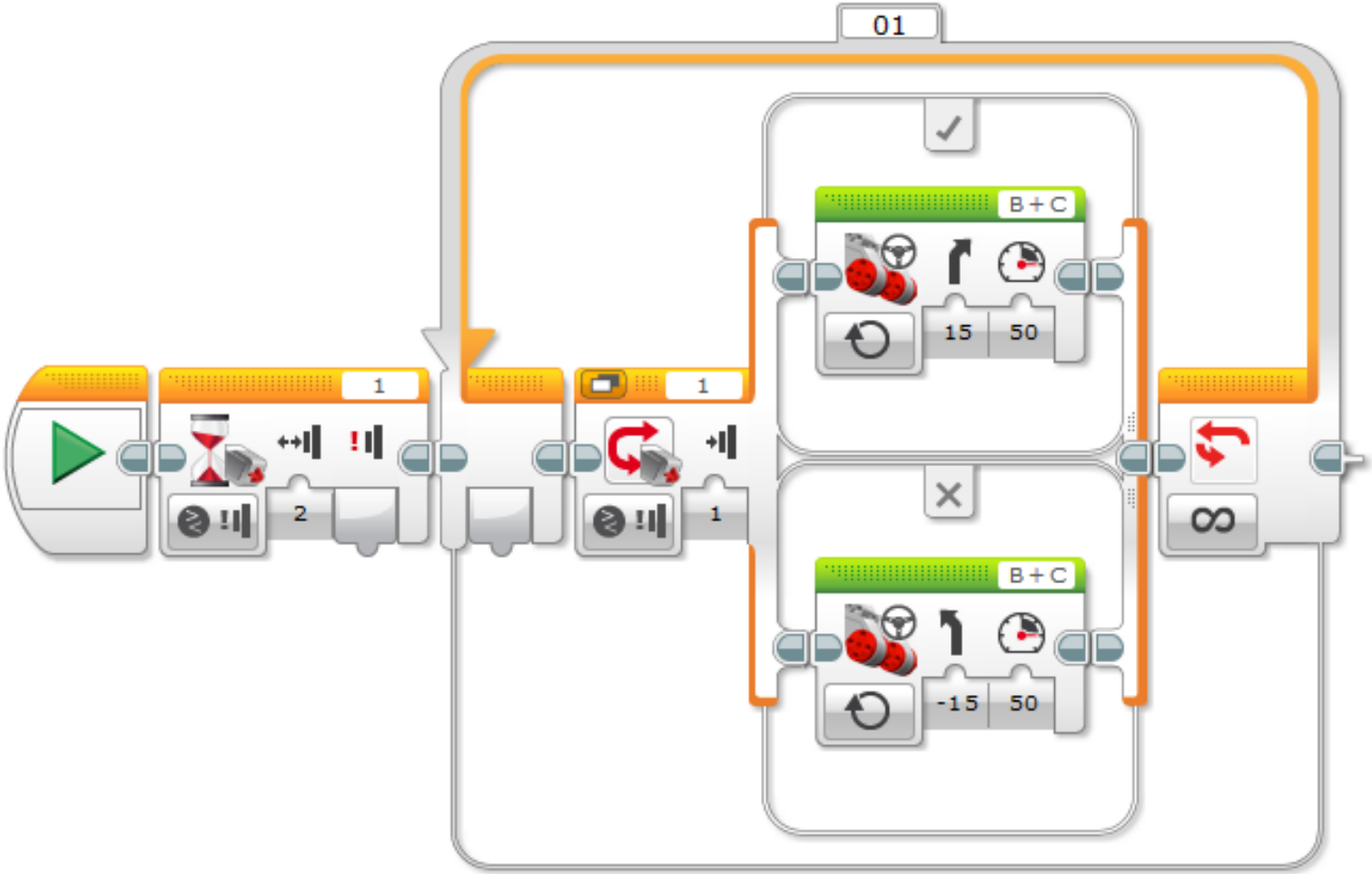
Možné řešení (robot jede jen v případě stisknutého tlačítka, jinak stojí):



Možné řešení (robot po spuštění programu jede, při stisku tlačítka začne zatáčet (tj. jedno kolo se přestane otáčet), po uvolnění tlačítka jede robot opět rovně):



Možné řešení (vylepšení varianty B: robot po spuštění programu stojí, čeká na krátký stisk tlačítka. Poté se rozjede a tlačítkem je možno ovládat zatáčení.):



Ultrazvukový senzor, mobilní robot s UZ senzorem, vyhledávání překážky, bránění kolize s překážkou

Ultrasonic Sensor (ultrazvukový dálkoměr)

Pozor: *ikonka* ***Ultrasonic Sensor*** *je velmi podobná ikonce* ***Infrared Sensor*** *a je možné je snadno zaměnit!*

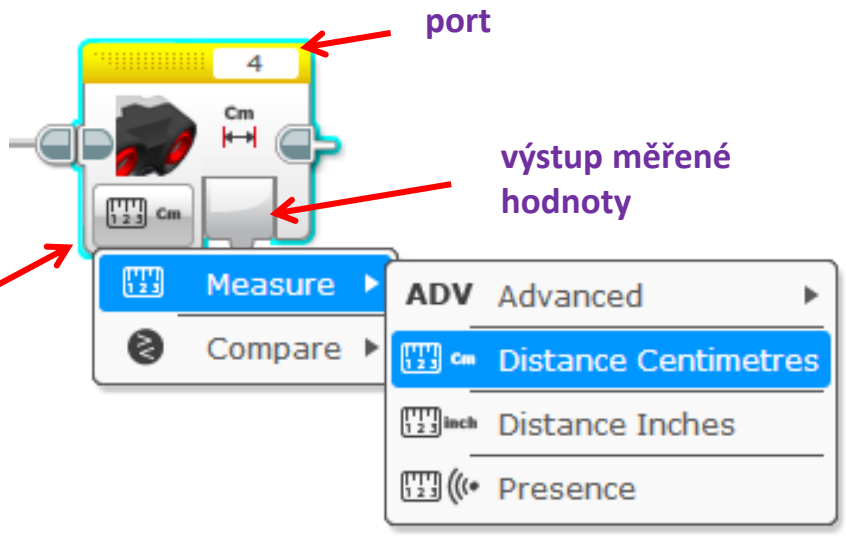
US

IR

Poznámka:

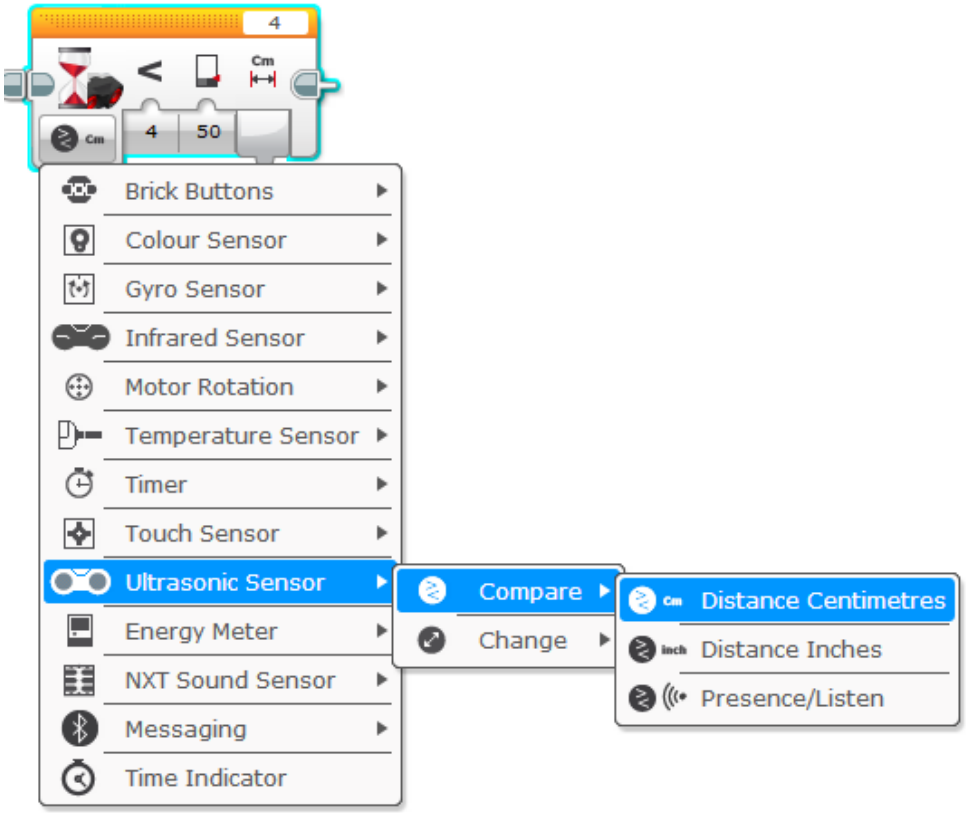
US senzor funguje podobně jako netopýr, ale mnohem hůř. Záleží na překážce (a odrazu od ní) - krabici od bot detekuje spolehlivě, kulatý míč hůře, chlupatého nepravidelného medvěda úplně nejhůře. Pokud je poblíž několik US senzorů, mohou se navzájem ovlivňovat - tím je možno při soutěžích ROBOSUMO protivníkům i škodit.

Ultrasonic Sensor

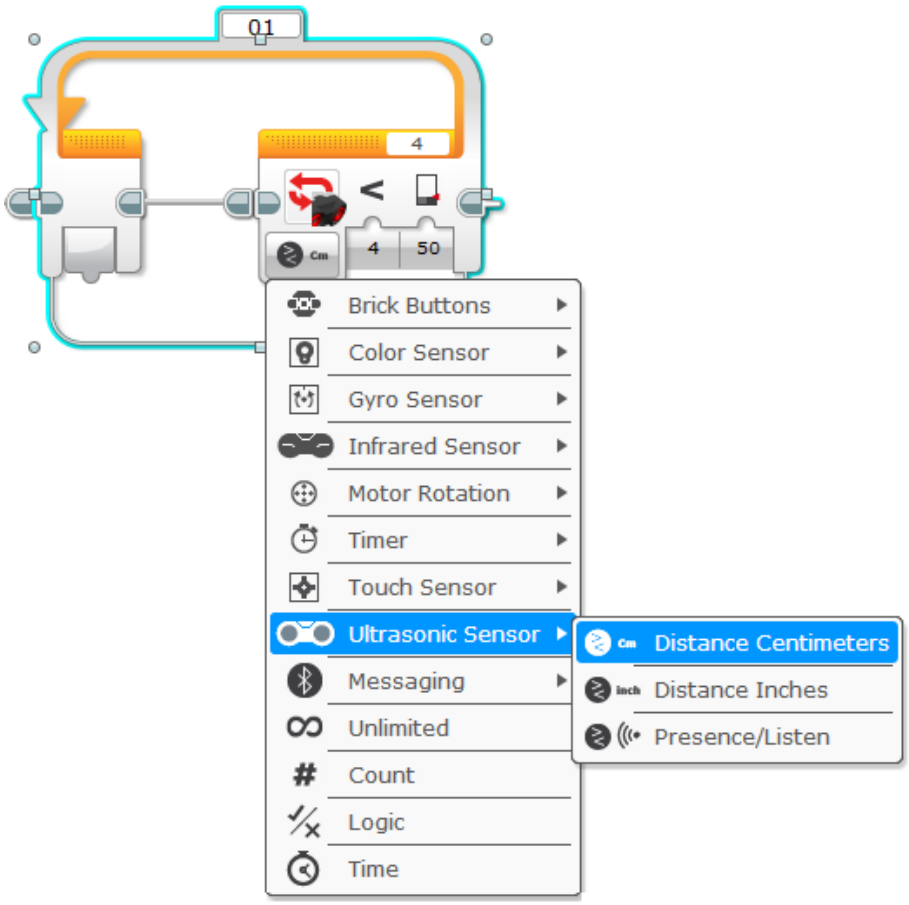


Příklady:

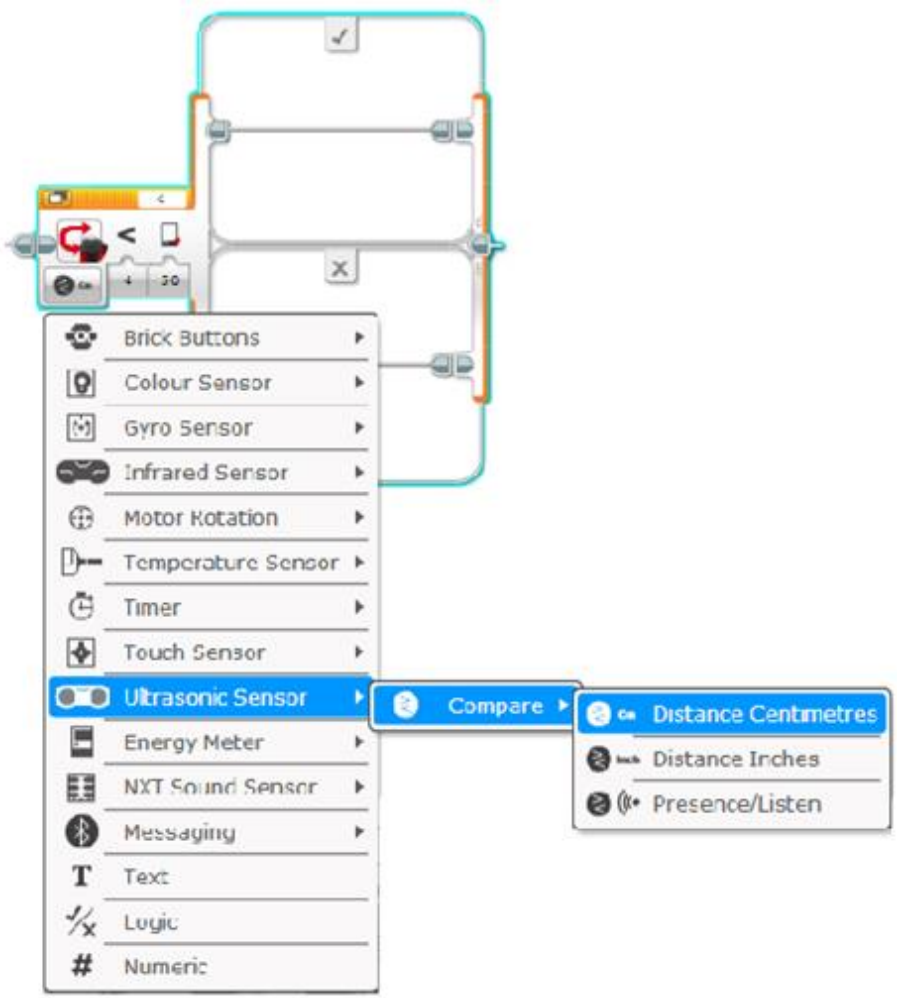
Wait/Ultrasonic Sensor



Loop/Ultrasonic Sensor



Swith/ Ultrasonic Sensor



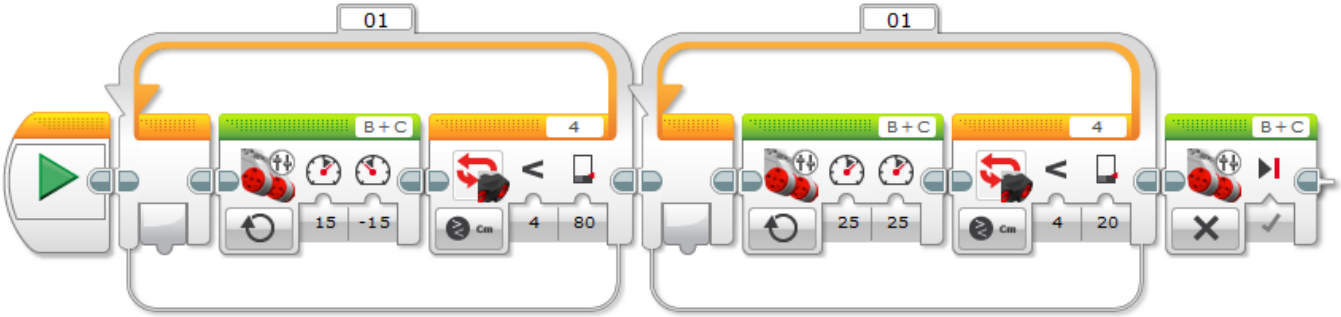
Práce s ultrazvukovým senzorem - robot vás má najít! Když vás najde, s radostí se k vám rozjede a kousek od vás zastaví!

Podrobné zadání: *robot se musí otáčet na místě. Když uvidí překážku (třeba ve vzdálenosti menší než 80cm) přestane se točit, rozjede se dopředu a ve vzdálenosti třeba 20cm od překážky zastaví!*
Problém: akce je jednorázová, po zastavení robota program končí!

Vylepšete tedy předchozí variantu tak, že robot za vámi jede jen v případě, že vás vidí (tj. jede rovně jen pokud UZ senzor detekuje překážku). Jakmile robot nikoho nevidí, zastaví se a opět se začne otáčet na místě *(budete asi muset použít podmínku řízenou UZ senzorem - když je vzdálenost menší než 80cm jede robot rovně, jinak se na místě otáčí)****. Zastaví, pokud se přiblíží k překážce na vzdálenost menší než 20cm*** *(zmizí-li překážka, pokračuje hledáním)****!***

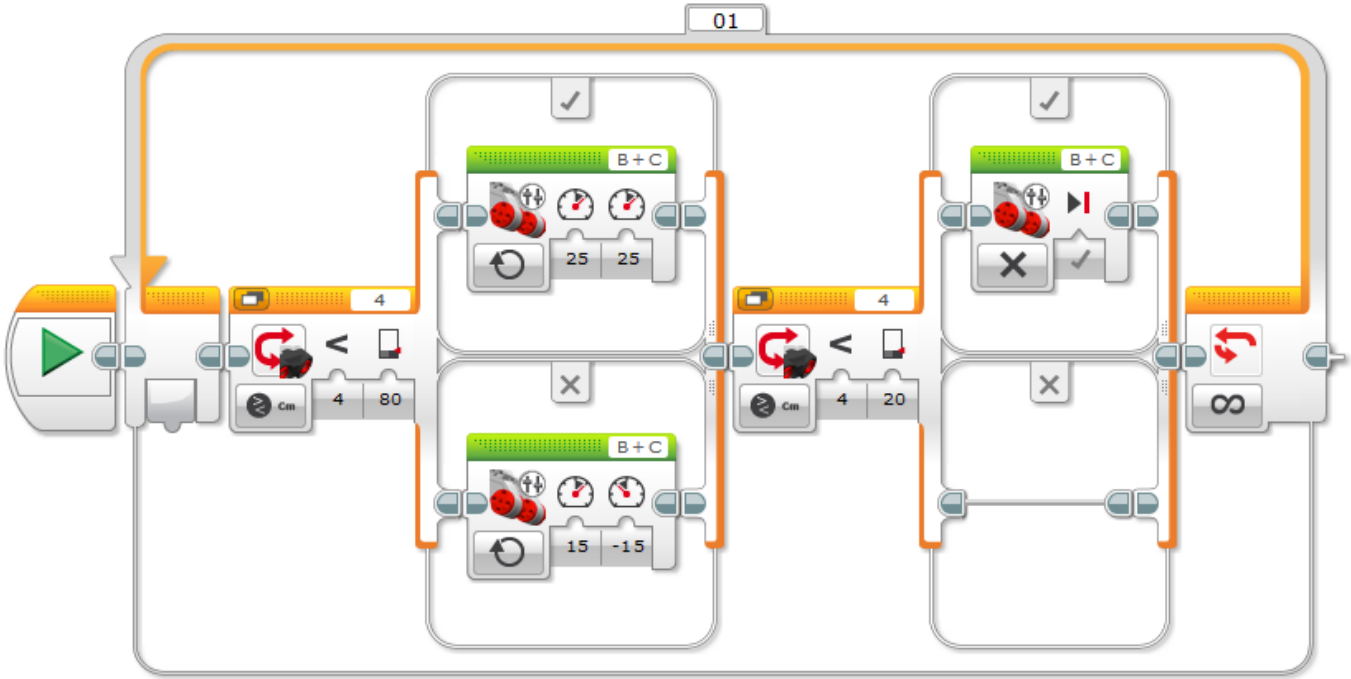
Možné řešení:

Robot se musí otáčet na místě. Když uvidí překážku (ve vzdálenosti menší než 80cm), přestane se točit, rozjede se dopředu a ve vzdálenosti 20cm od překážky zastaví!



Vylepšená varianta:

Jakmile robot nikoho nevidí, zastaví se a opět se začne otáčet na místě a hledat. Pokud se přiblíží k překážce na vzdálenost menší než 20cm, zastaví.



Naprogramujte robota tak, aby hrál tón podle vzdálenosti ve které „vidí“ překážku. Změnou vzdálenosti překážky (třeba i ruky) od UZ senzoru můžete tedy na robota „hrát“!

Nápověda: převod vzdálenosti překážky od UZ senzoru na tón

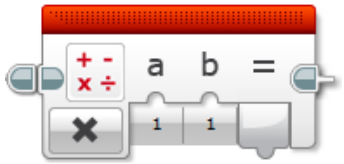
tón [Hz] = vzdálenost [cm] x N + P

kde **N** (násobek) je vhodné volit v rozmezí 5 až 30

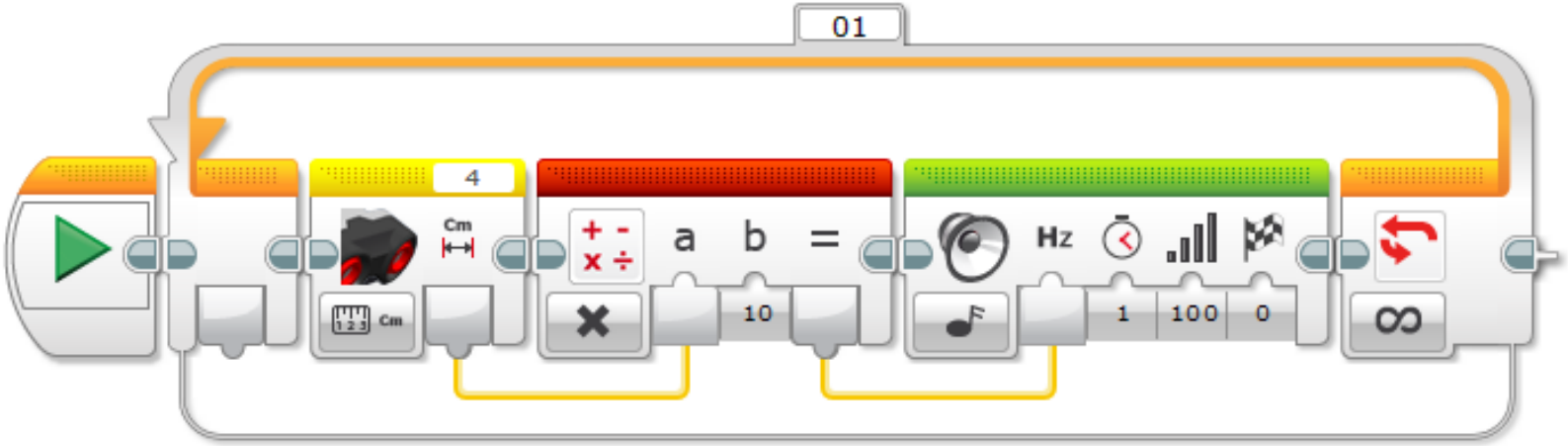
a **P** (posun) je vhodné volit v rozmezí 0 až 1000

Např. pro převod vzdálenosti 44cm na tón 440Hz (komorní A) platí N=10 (při P=0).

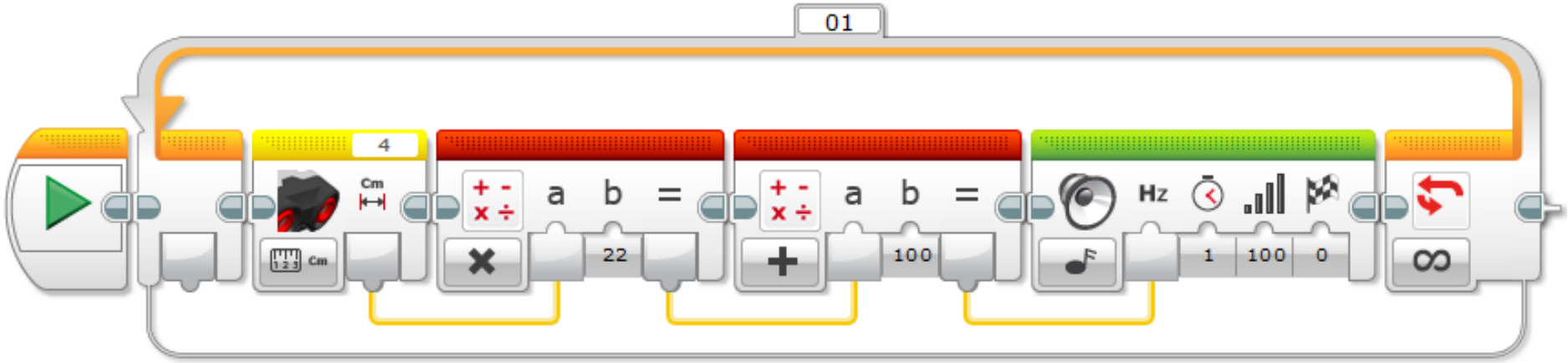
součet 

násobení 

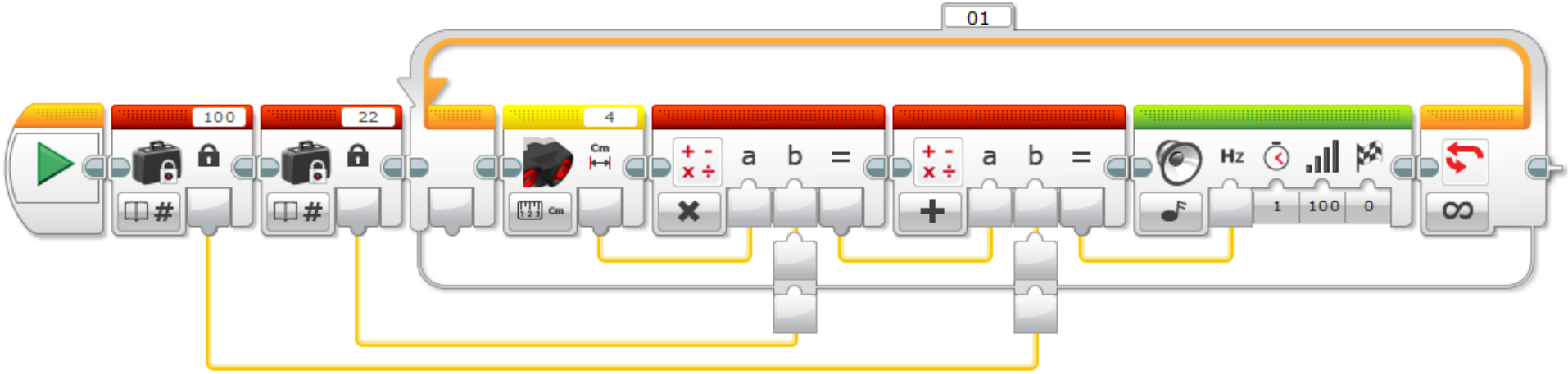
Možné řešení (asi nejjednodušší, volíme N = 10, P = 0):



Jiné možné řešení (volíme N = 22, P = 100):



Jiné provedení předchozího řešení:



Práce se senzorem odraženého světla

Colour Sensor *(čidlo barev, světelný senzor)*

Senzor barvy dokáže detekovat barvu nebo intenzitu světla prostupujícího malým objektivem na čelní straně senzoru.

Senzor lze použít ve třech režimech:

1. **Režim barvy (Color Mode)**
2. **Režim intenzity odraženého světla (Reflected Light Intensity Mode)**
3. **Režim intenzity okolního světla (Ambient Light Intensity Mode)**

V **režimu barvy** senzor barvy rozpoznává sedm barev:

černou, **modrou**, **zelenou**, **žlutou,** **červenou**, bílou, **hnědou** (a žádnou).

V režimu intenzity odraženého světla senzor měří intenzitu odraženého červeného světla ze zdroje v čelní části senzoru (pod objektivem).
Senzor používá škálu od 0 (velmi tmavá) do 100 (velmi světlá).
Umožňuje programovat robota tak, aby se pohyboval po bílém povrchu, dokud nedetekuje černou čáru, nebo aby rozpoznal barevně označenou identifikační kartu.

V režimu intenzity okolního světla senzor měří intenzitu světla přicházejícího do objektivu z okolního prostředí, např. světla slunečního nebo světelného kuželu svítilny.
Senzor používá škálu od 0 (velmi tmavá) do 100 (velmi světlá).
Umožňuje naprogramovat robota tak, aby vypnul budík, když ráno vyjde slunce nebo zastavil akci, když zhasnou světla.

K dosažení nejvyšší přesnosti v režimu barvy a odrazu světla musí být senzor kolmo a v blízkosti zkoumaného povrchu (nesmí se ale dotýkat)!

Blok senzoru barvy může pracovat (samostatně) ve třech módech:

měření (Measure),
porovnávání (Compare)
a **kalibrace (Calibrate)**

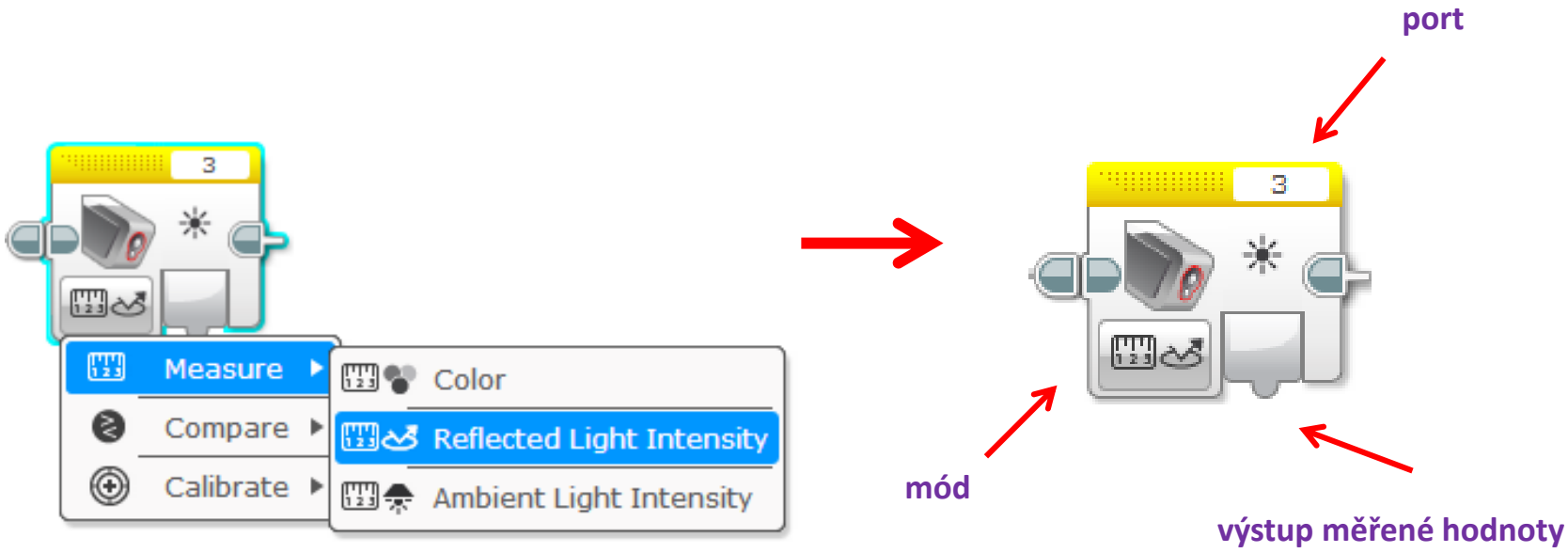
Měření (Measure)
– Color
– Reflected Light Intensity
– Ambient Light Intensity

Porovnávání (Compare)
– Color
– Reflected Light Intensity
– Ambient Light Intensity

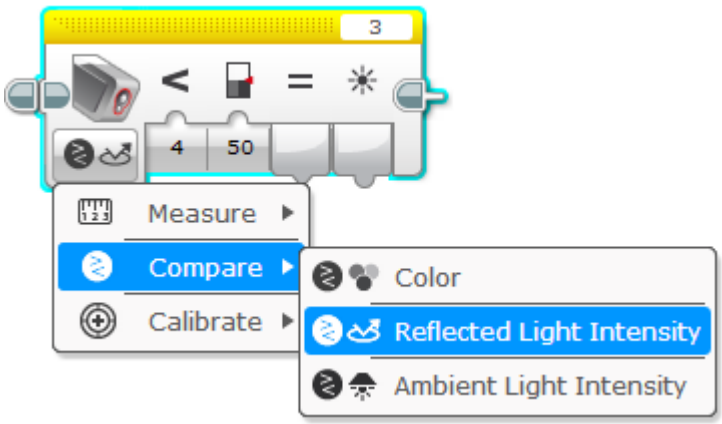
Kalibrace (Calibrate)
– Minimum
– Maximum
– Reset

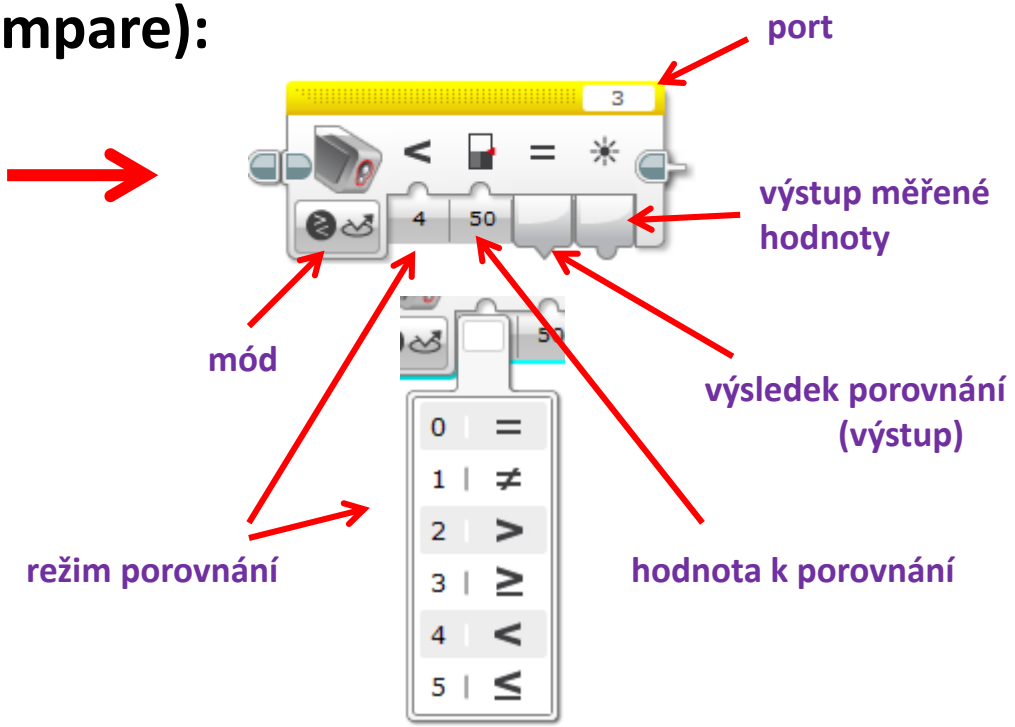
I senzor barev lze použít, tak jako naprostou většinu senzorů, **samostatně** (např. k měření) nebo **v čekání** (Wait), **ve smyčce** (Loop) či **v přepínači** (Swith):

samostatně, měření (Measure):

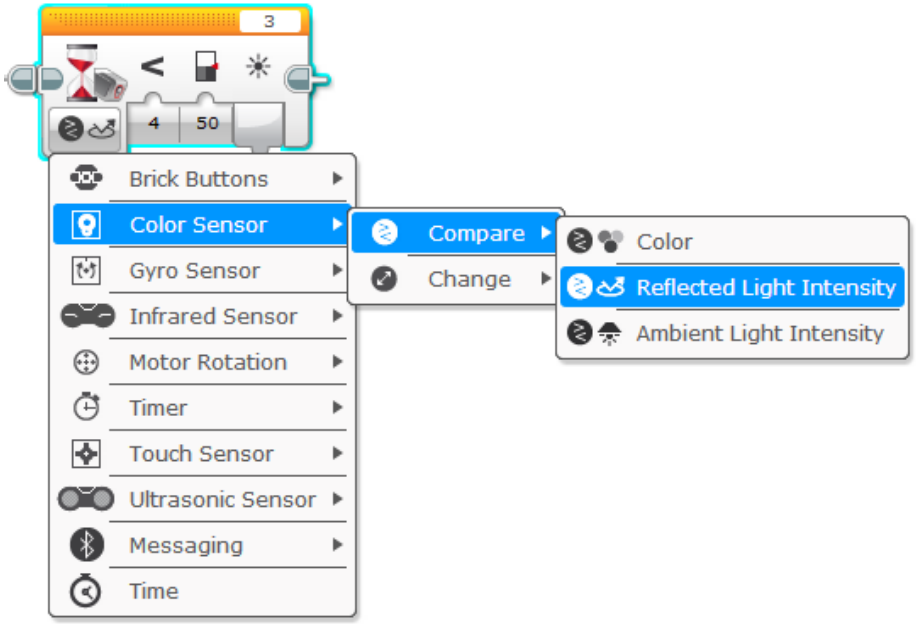


samostatně, porovnávání (Compare):

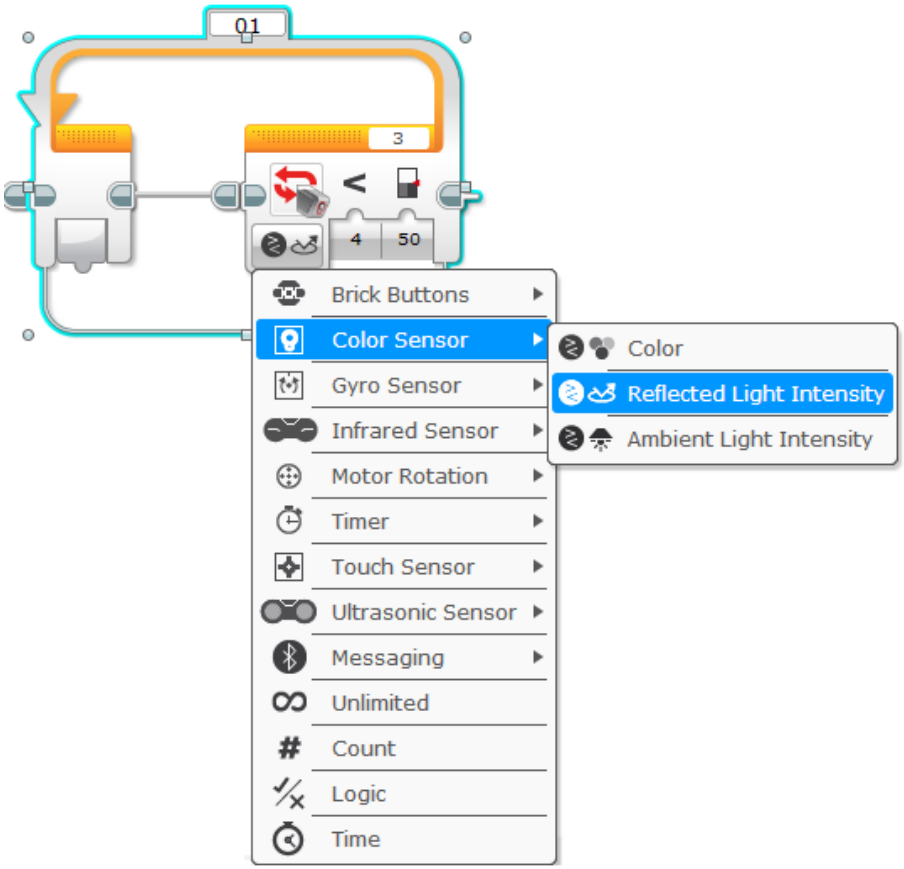




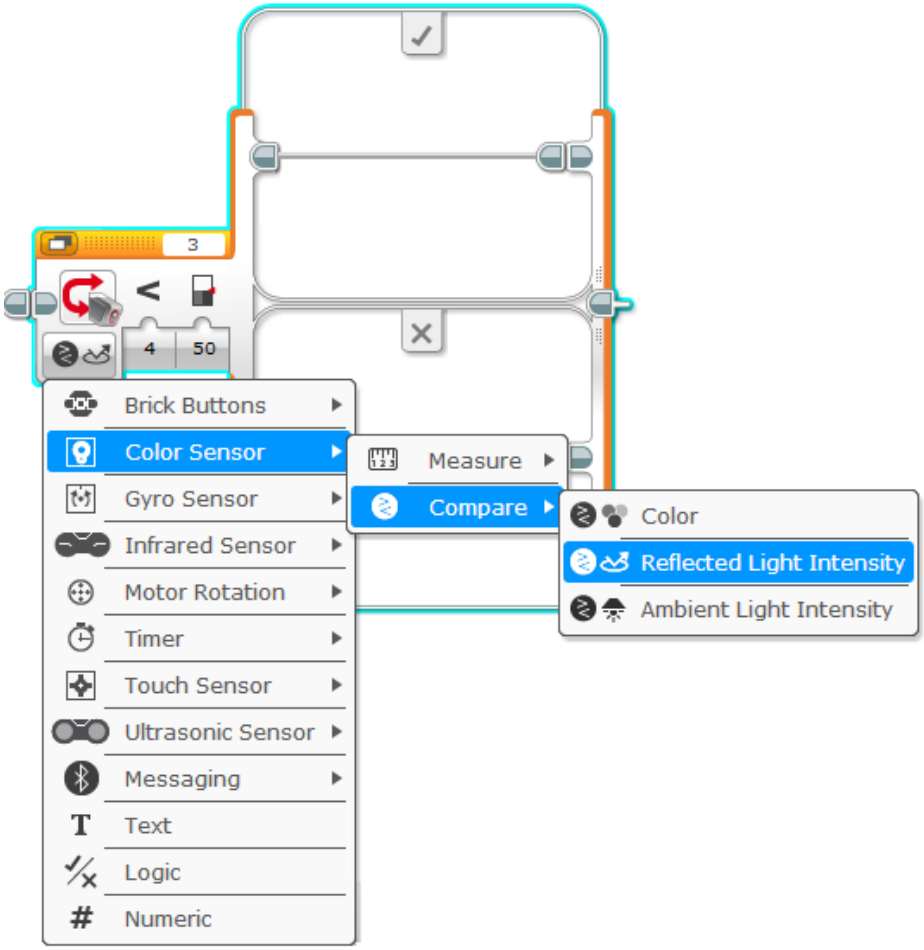
v čekání (Wait):



ve smyčce (Loop):

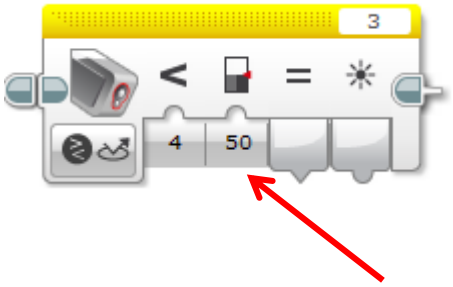


v přepínači (Swith):



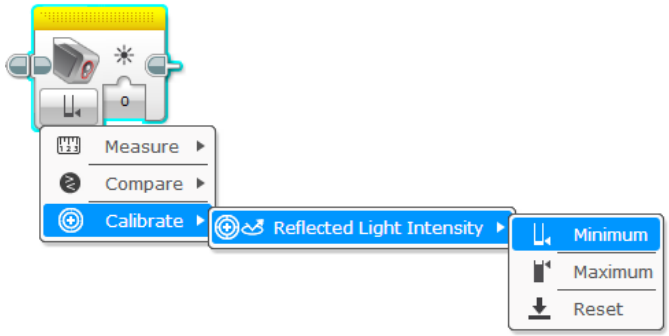
Nastavení hodnoty k porovnávání

(např. v módu **Compare** – Reflected Light Intensity, tj. režim černá/šedá/bílá)**:**

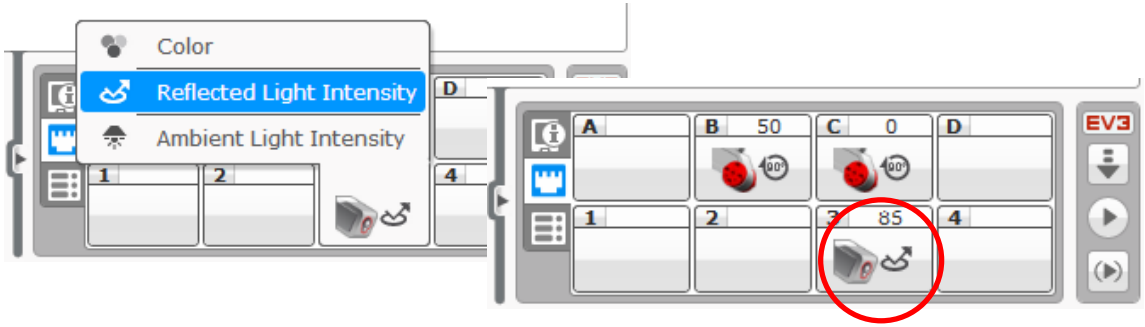


hodnota k porovnání

Je možno použít mód k**alibrace (Calibrate)**, ale výsledky mohou být nepoužitelné.



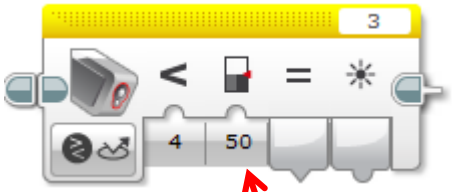
Vhodnější se jeví odečtení hodnoty intenzity odraženého světla z hardwarového okénka a následný výpočet



Příklad: - bílá se např. odrazí v hodnotě **85**

- černá se např. odrazí v hodnotě **5**

takže:



(hodnota bílé + hodnota černé)/2 = střed (hodnota pro komparaci (porovnání))

t.j. (85 + 5)/2 = 45

hodnota k porovnání, sem zapsat!

Úloha sledování černé čáry, tzv. „robota stopaře“ („Line Follower“)

Častou úlohou při programování Lego Mindstorms robotů je stavba a programování robota „stopaře“ („Line Follower“). Robot má za úkol sledovat kontrastní stopu. Soutěží se v jízdě na čas.

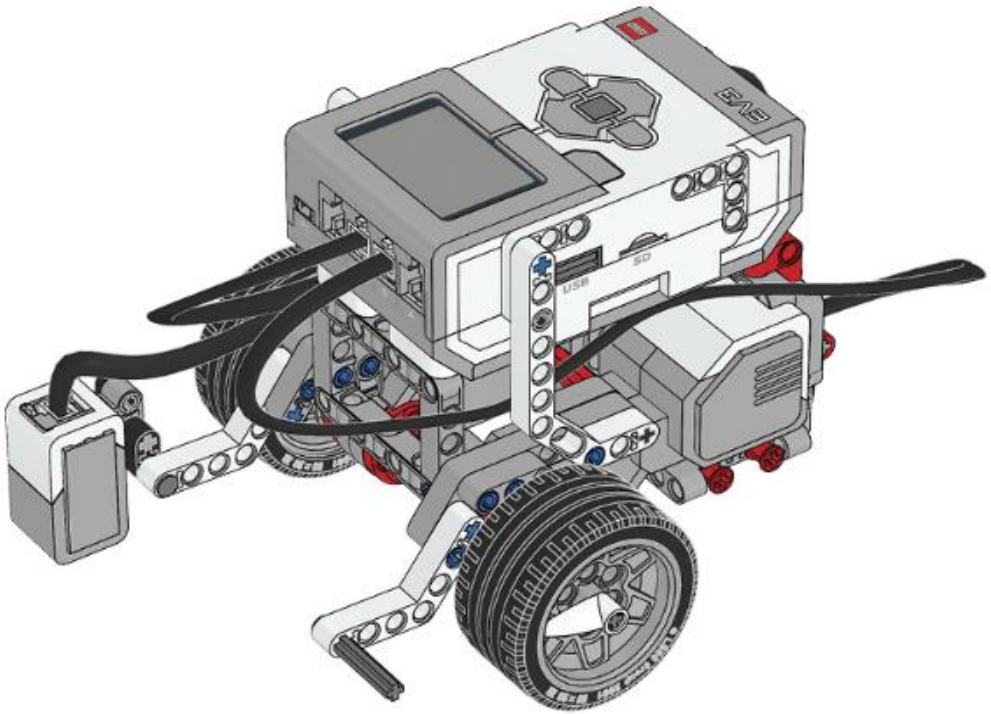
Šikovnost robota stopaře není dána jen šikovností programátora, ale závisí také na konstrukci robota. Robot musí včas reagovat na změnu směru linie a sledovat i ostré zatáčky při dostatečné rychlosti.

Často se stopař staví jako **trojkolka** (aby mohl reagovat i na ostré zatáčky) s hnanými předními koly a jedním opěrným kolečkem (či kuličkou) vzadu (ale existují i jiné konstrukce).

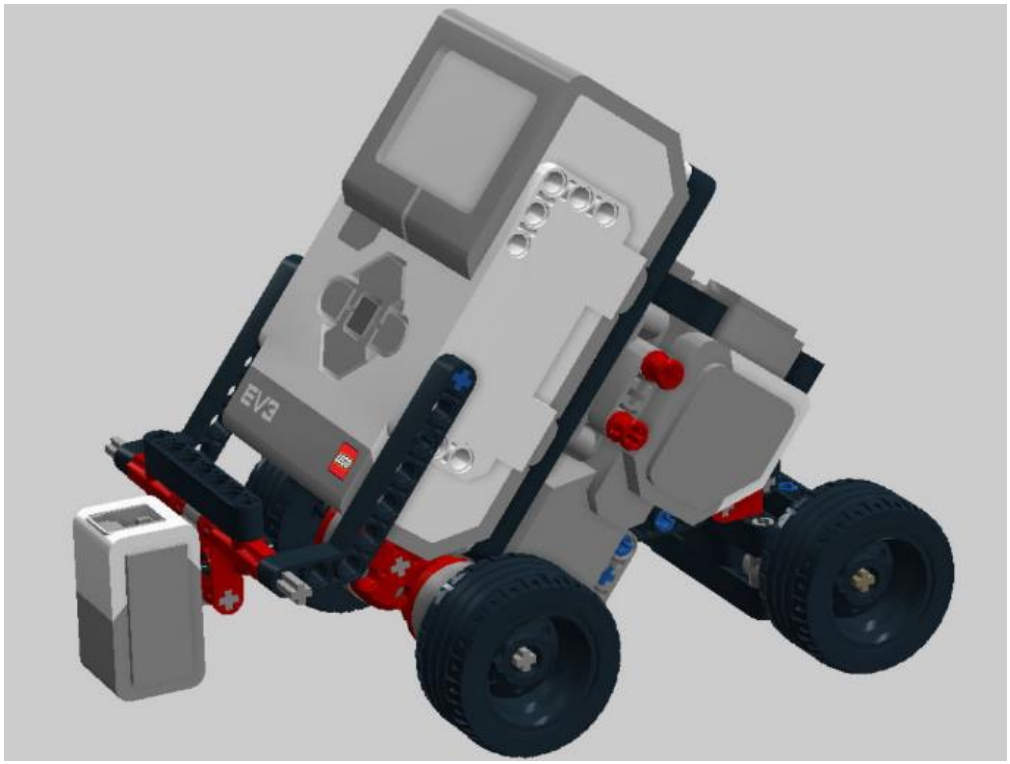
Jak tedy vytvořit šikovného robota stopaře?

1. např. podle návodu přiloženého ke stavebnici (v návodu kroky 1 až 40, str. 7 až 38, s přidáním senzoru barev směřujícího dolů (v návodu kroky 1 až 5, str. 69 až 71) proti návodu ale umístěného v ose robota)!

(nebo totéž viz*:* ***LEGO MINDSTORMS EV3 Education Edition › Robot Educator › Building Instructions › Building Ideas › Driving Base kroky 1 až 45*** + ***Colour Senzor Down - Driving Base kroky 1 až 6***)



nebo **2.** stopař **čtyřkolka**



Stavební návod:
http://robowiki.spsnome.cz/uploads/Roboti/stopar-ev3-4kola.htm

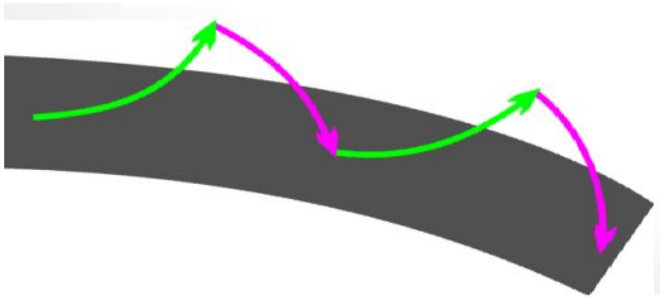
3. **Pochopitelně je možno sestavit vlastní konstrukci!**

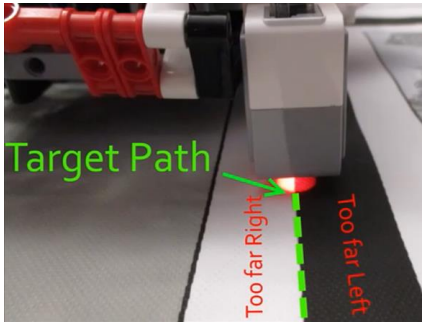
Sledování čáry pomocí cik-cak algoritmu (dvoupolohového regulátoru), sledování čáry pomocí lineárního regulátoru

Naprogramujte robota, který bude sledovat černou čáru na bílém podkladu (tzv. robota stopaře) pomocí tzv. cik-cak algoritmu!

Nápověda: *jestliže máme k dispozici pouze jeden snímač barev, můžeme vlastně sledovat pouze rozhraní bílá/černá, tj. hranu čáry*

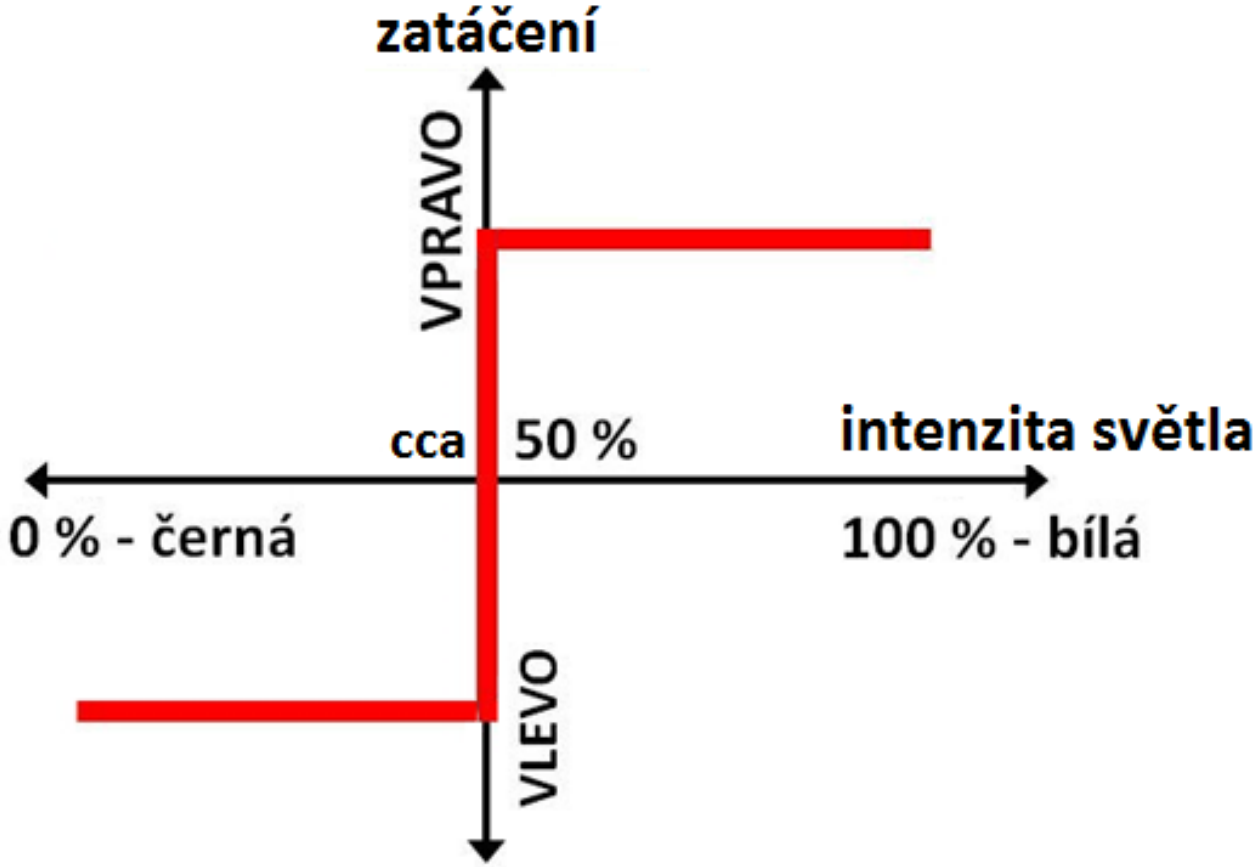
Doporučení: *kvůli stabilitě sledování nastavujte spíše menší výkony motorů!*





Informace: cik-cak algoritmus představuje z hlediska teorie řízení tzv. nelineární (nespojitou) dvoupolohovou regulaci

Názorný příklad pro sestavení cik-cak algoritmu pro řízení robota stopaře:



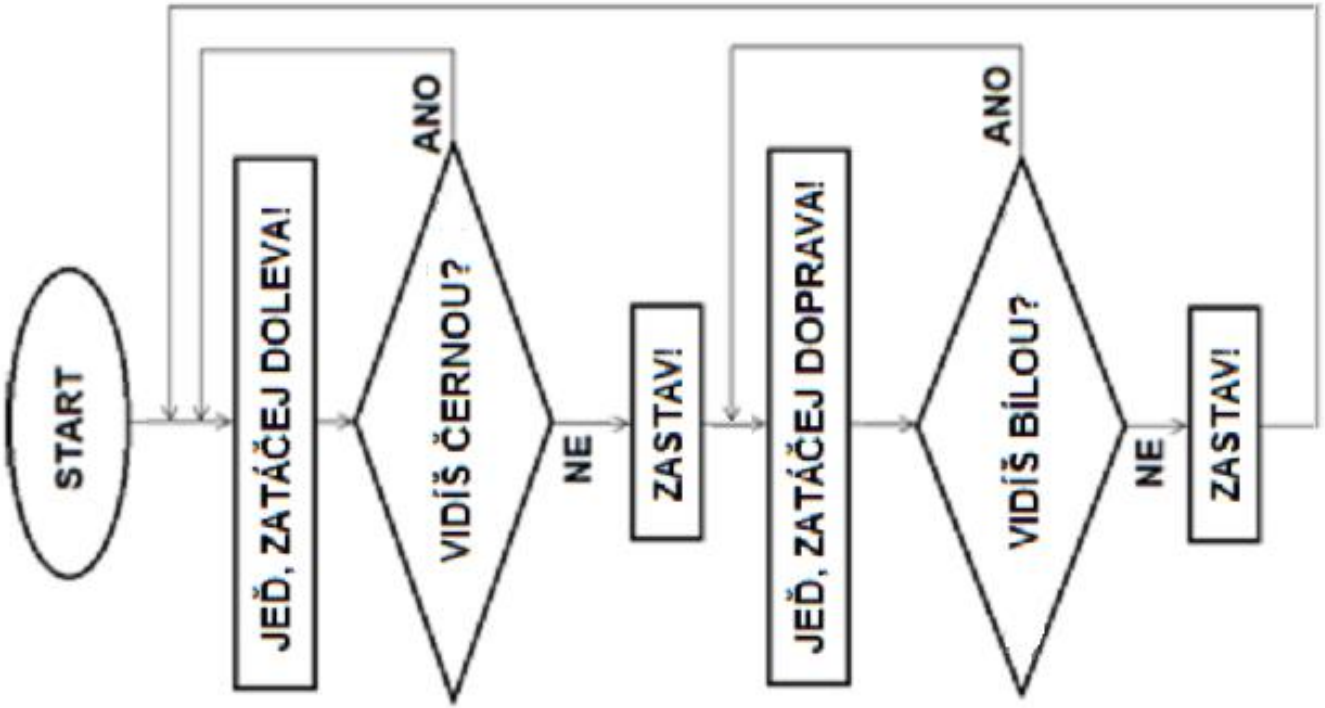
Nápověda: ***cik-cak algoritmus pro řízení robota stopaře***

- zvol si hranu, kterou bude robot sledovat

např. pro levou hranu:

- jeď a zatáčej doleva, dokud vidíš černou (čáru)
- až uvidíš bílou (tj. opustíš čáru), zastav
- jeď a zatáčej doprava, dokud vidíš bílou (tj. dokud jsi mimo čáru)
- až uvidíš černou (čáru), zastav
- opakuj od začátku

Vývojový diagram cik-cak algoritmu pro řízení robota stopaře:

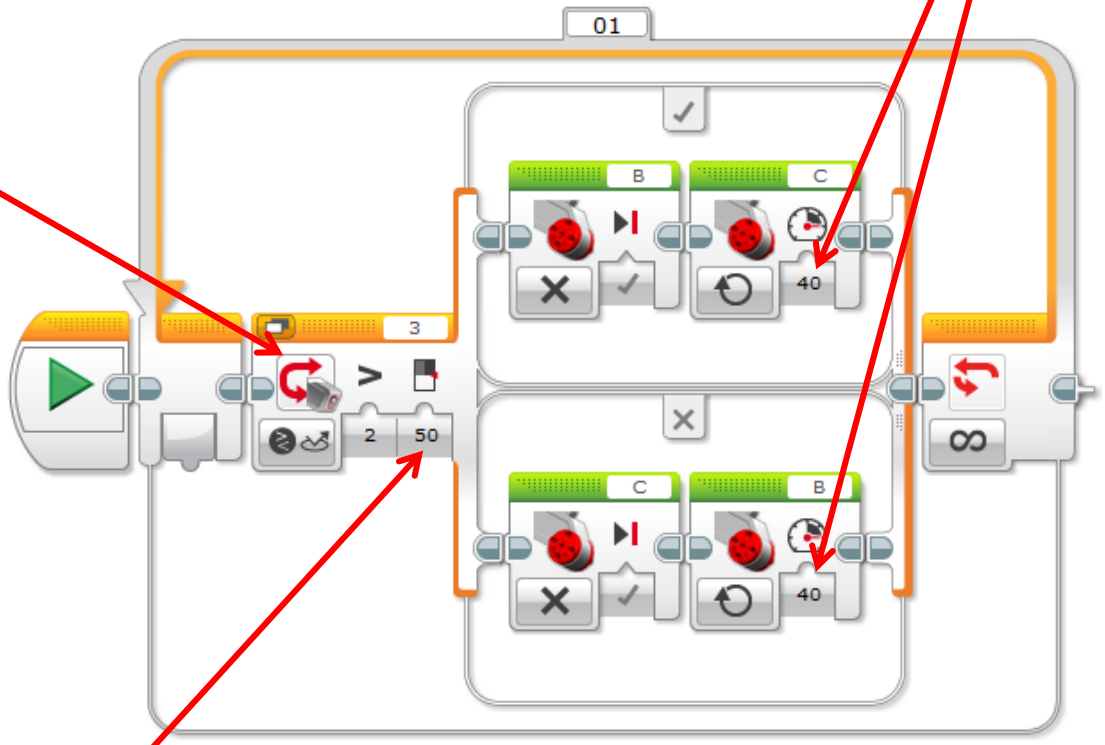


Možné řešení :

(s řízením motorů samostatně)

Nastavení rychlosti zatáčení, nutno vyzkoušet!

Přepínač se senzorem barev v režimu „Compare › Reflected Light Intensity“

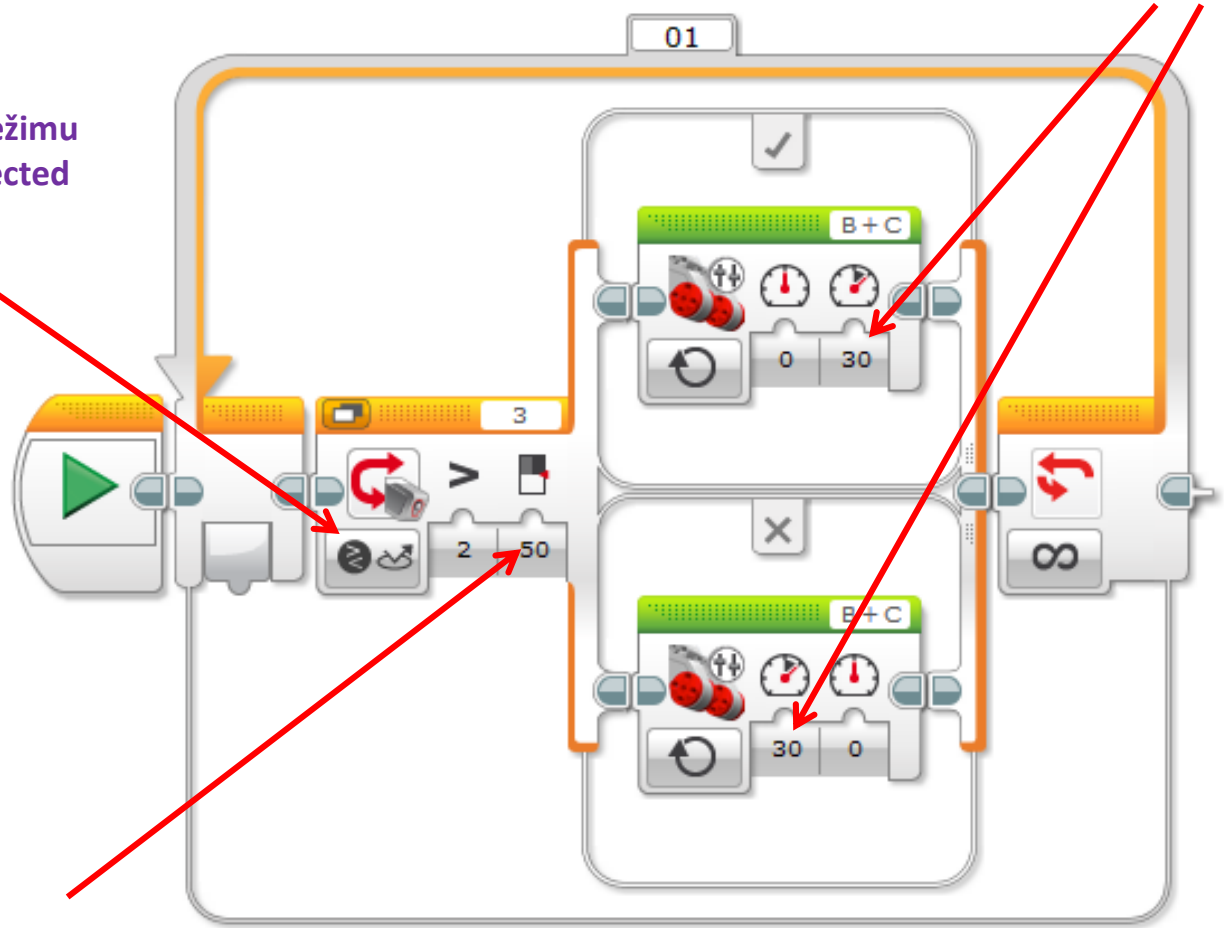


Hodnota pro komparaci získaná kalibrací, v konkrétním provedení robota se bude lišit!

Další možné řešení :

(s motory v režimu „Move Tank“)

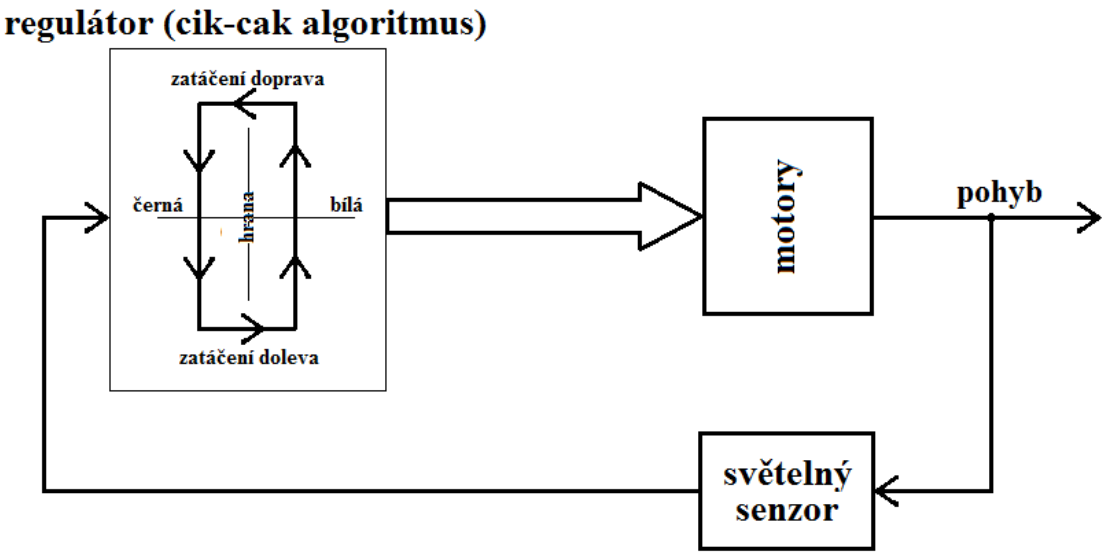
Nastavení rychlosti zatáčení, nutno vyzkoušet!



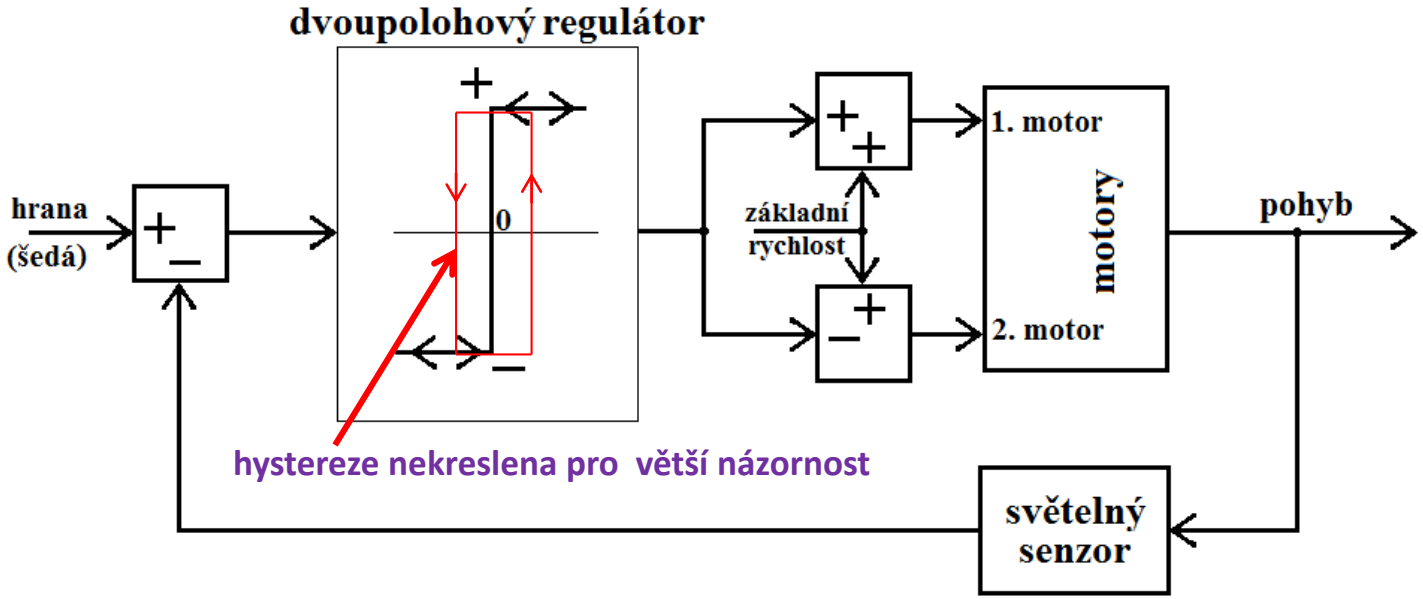
Senzor barev v režimu „Compare › Reflected Light Intensity“

Hodnota pro komparaci získaná kalibrací, v konkrétním provedení robota se bude lišit!

Dvoupolohová regulace - princip

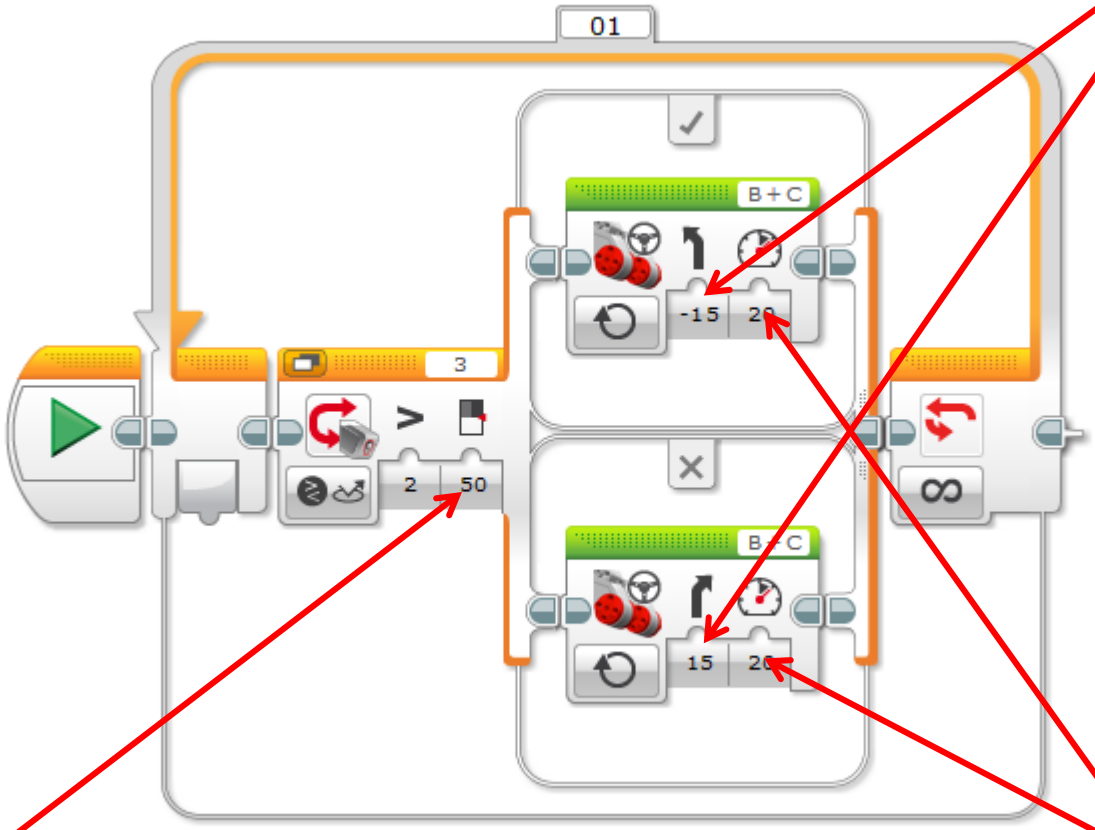


Lepší znázornění:



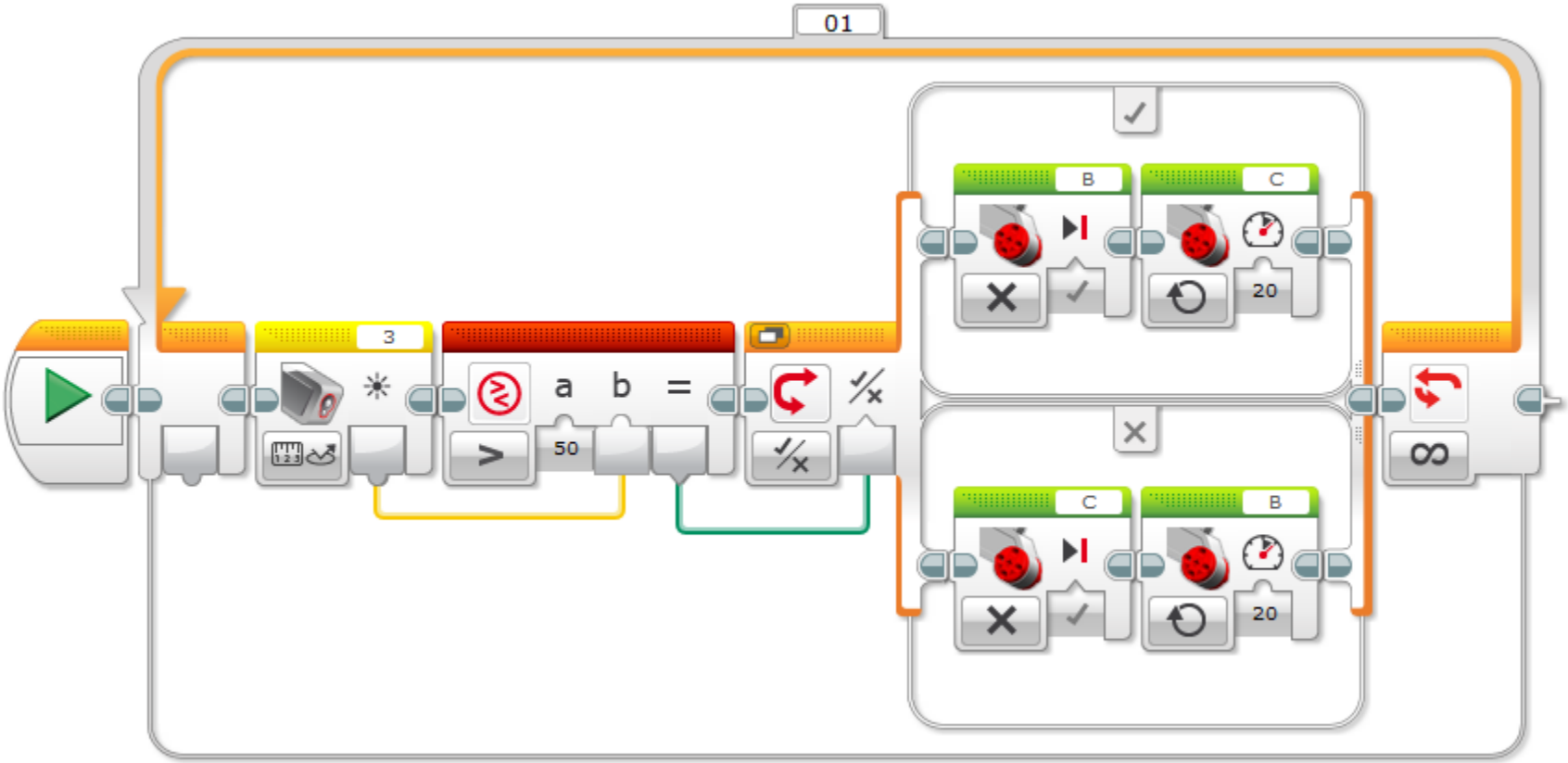
Možné řešení :

(bez zastavování, s motory v režimu „Move Steering“)



A ještě jedno možné řešení :

(s přepínačem jinak)

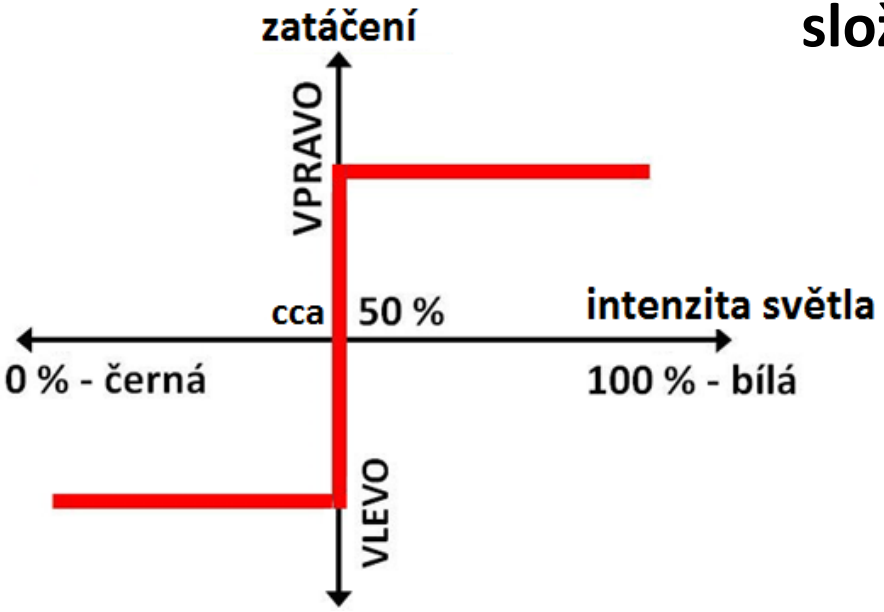


Sledování čáry pomocí lineárního regulátoru

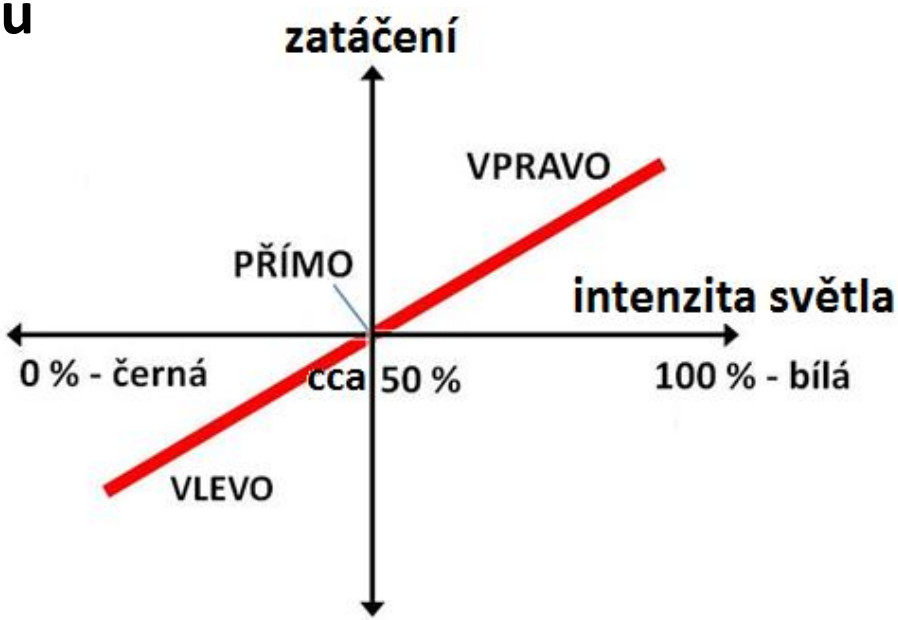
Naprogramujte řízení robota pro sledování čáry pomocí regulační smyčky s lineárním proporcionálním (P) regulátorem!

Porovnání rozdílu mezi nelineárním dvoupolohovým regulátorem a lineárním regulátorem s P (proporcionální) složkou:

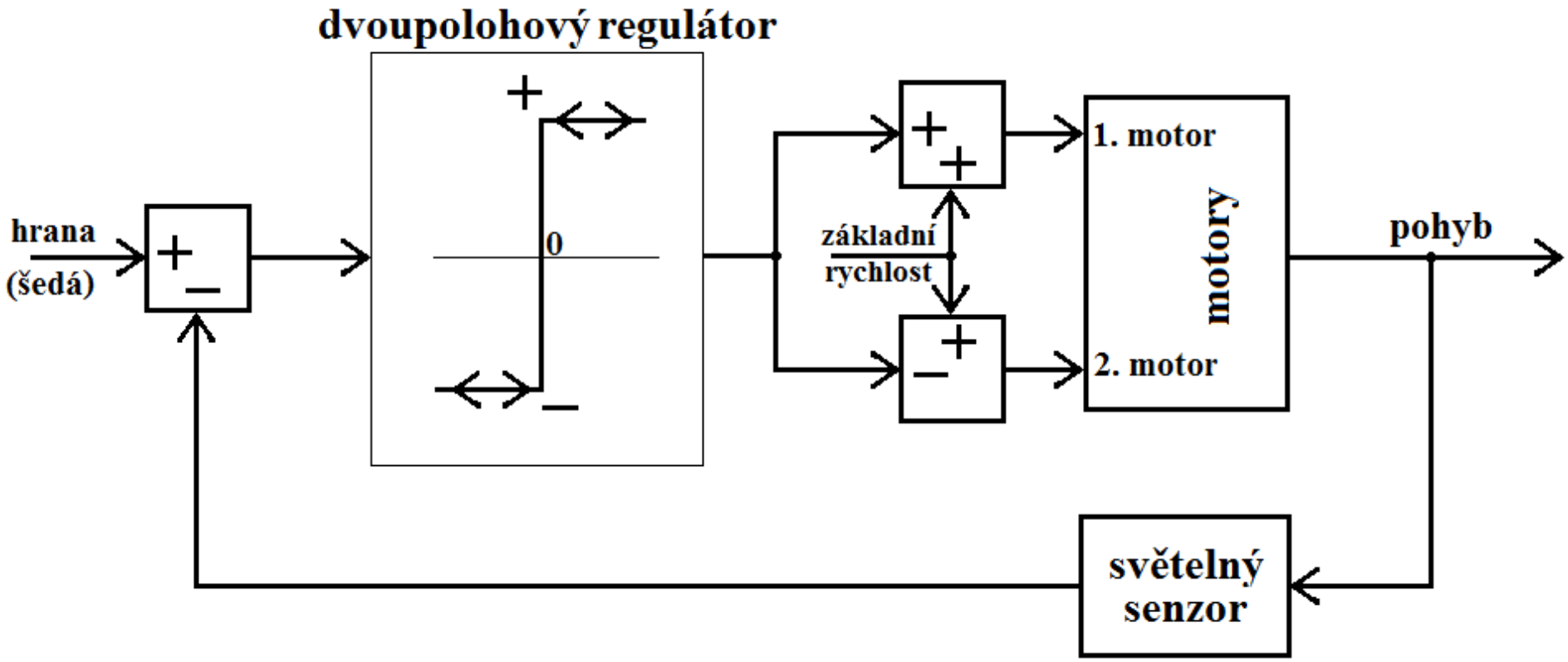
dvoupolohový regulátor



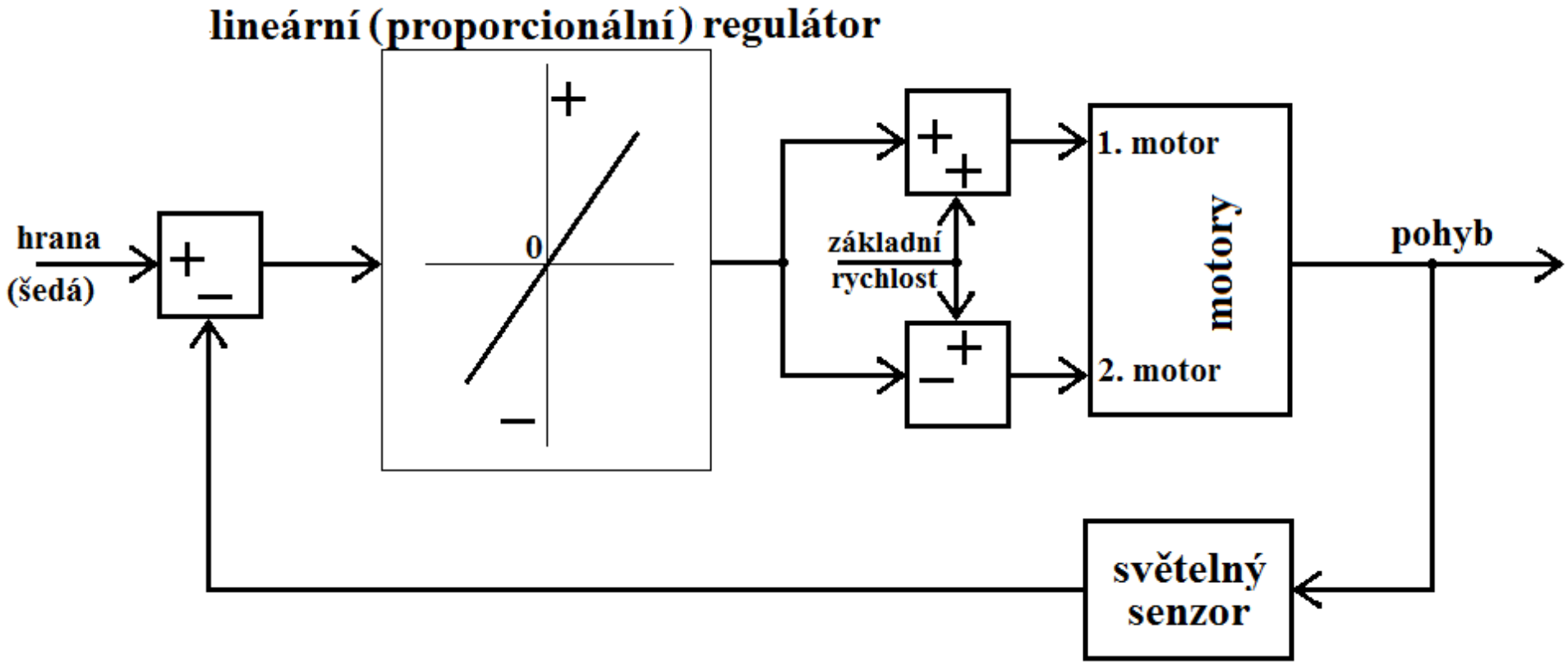
lineární regulátor s P (proporcionální) složkou



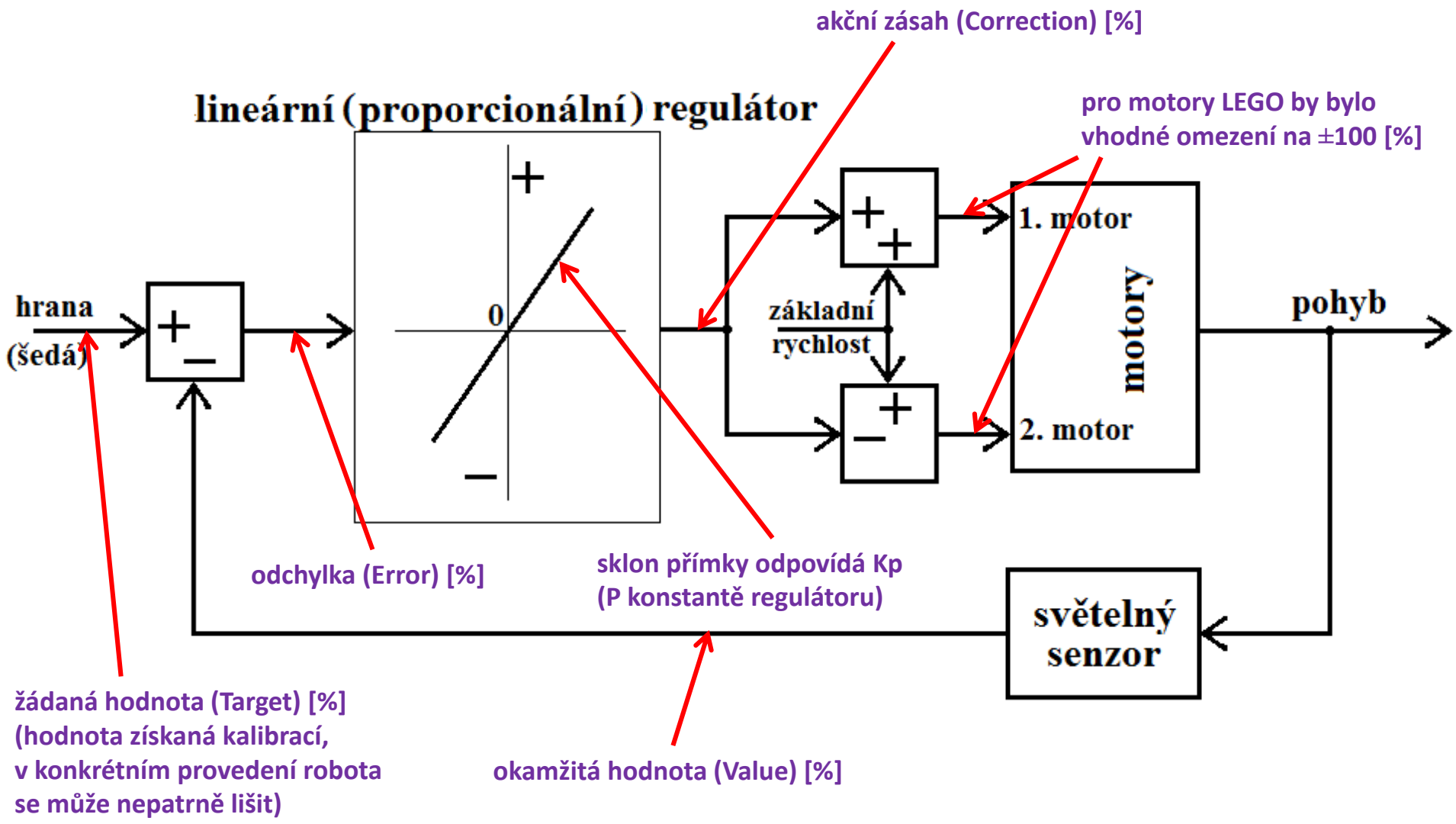
Porovnání rozdílu mezi nelineárním dvoupolohovým regulátorem a lineárním regulátorem s P (proporcionální) složkou:



Porovnání rozdílu mezi nelineárním dvoupolohovým regulátorem a lineárním regulátorem s P (proporcionální) složkou:



Porovnání rozdílu mezi nelineárním dvoupolohovým regulátorem a lineárním regulátorem s P (proporcionální) složkou:



Řešení lineárního proporcionálního regulátoru (P regulátoru) pro robota stopaře

P regulátor:

měření *odchylky (Error)* od *žádané hodnoty (cílové hrany, Target)* pomocí senzoru barev (v módu Compare – Reflected Light Intensity, tj. režim černá/šedá/bílá):

Target (light value) = cca 50 [%]
Value = Light Sensor Value [%]

odchylka = žádaná hodnota – okamžitá hodnota [%; %]

(**Error** = Target – Value) [%; %]

akční zásah = Kp*odchylka [%; -, %]

(Correction = Kp*Error) [%; -, %]

kde ***Kp*** je proporcionální (P) konstanta

Nastavení *Kp* (proporcionální konstanty) regulátoru:

Konstantu *Kp* je možné nastavit zkusmo, lepší je však pomocí nějaké metody, např. Ziegler-Nicholsovy.

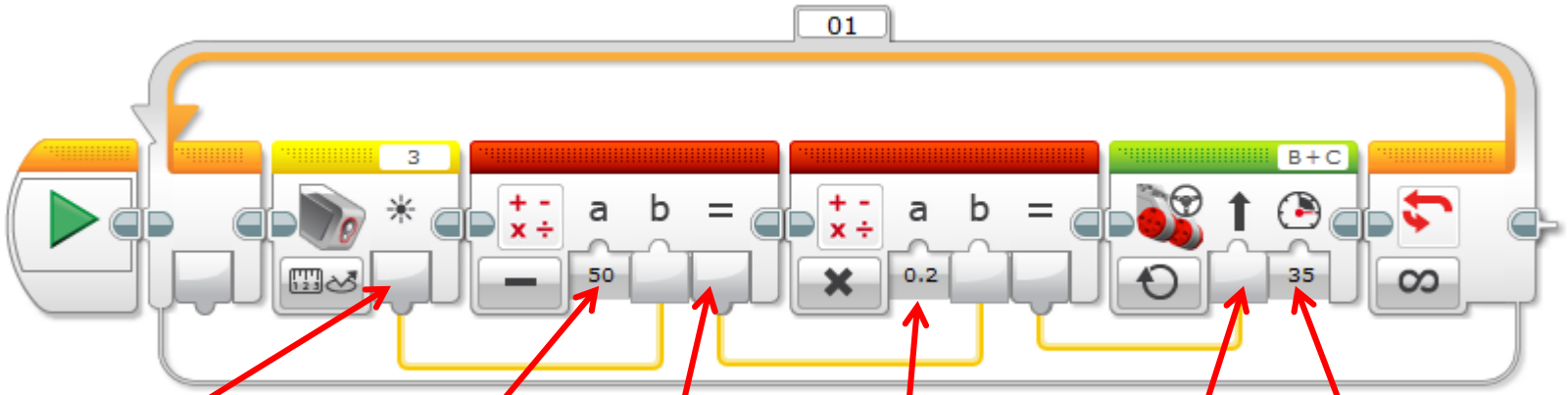
Ziegler-Nicholsova metoda spočívá v postupném zvětšování konstanty *Kp* až do hodnoty *Kk* (kritické), při které systém začne kmitat.

Z hodnoty *Kk* lze poté vypočítat konstantu *Kp* podle vzorce:

Kp = 0,5*Kk

Poznámka: v našem případě se nejedná o lineární analogový (spojitý) ale o číslicový regulátor, nastavení však zůstává obdobné!

Jedno z možných řešení:



okamžitá hodnota (Value) [%]

žádaná hodnota (Target) [%]
(hodnota získaná kalibrací,
v konkrétním provedení robota
se může nepatrně lišit)

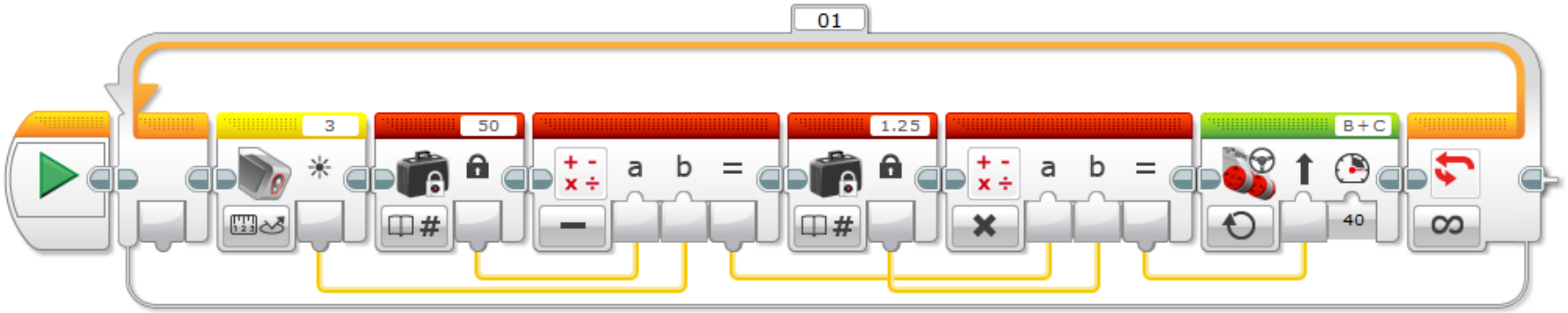
odchylka (Error) [%]

Kp (nastavení P konstanty
regulátoru, nutno vyzkoušet!)

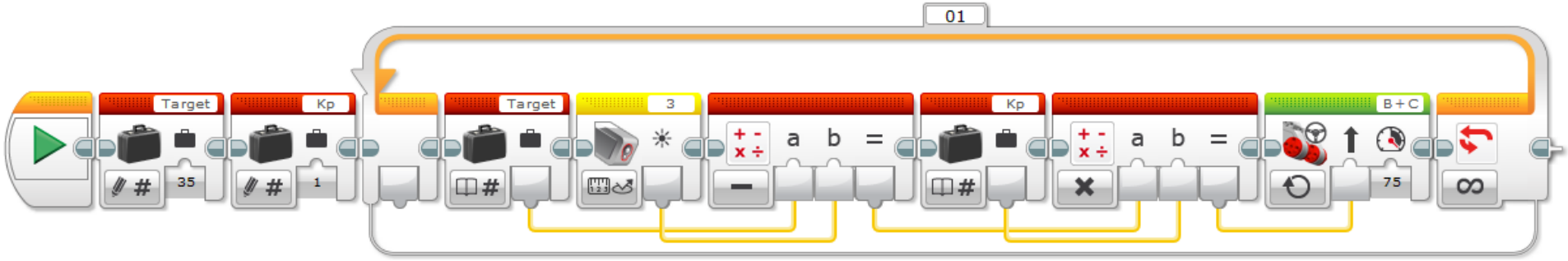
akční zásah (Correction) [%]

nastavení základní
rychlosti motorů,
nutno vyzkoušet!

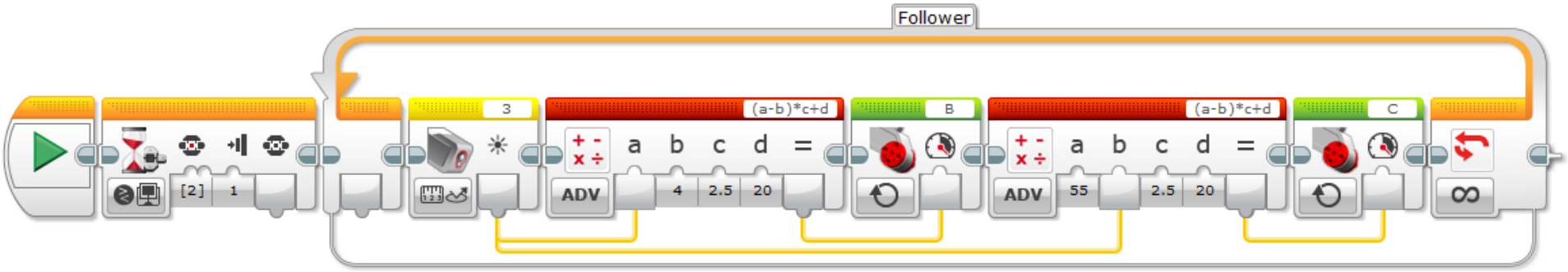
Totéž jinak:



Další řešení (vhodné jako základ pro řízení pomocí PD regulátoru):



A ještě jedno:



Řešení PD regulátoru (proporcionálně diferenčního regulátoru) pro robota stopaře

Proporcionální část

měření *odchylky (Error)* od *žádané hodnoty (Target)* pomocí senzoru barev (v módu Compare – Reflected Light Intensity, tj. režim černá/šedá/bílá):

Target (light value) = cca 50 [%]
Value = Light Sensor Value [%]

Diferenční část

pro řízení v následujícím kroku se využívá rozdíl mezi *minulou* a *okamžitou hodnotou odchylky*

odchylka = žádaná hodnota – okamžitá hodnota [%; %]

(**Error** = Target – Value) [%; %]

diference = odchylka – minulá odchylka [%; %]

(**Difference** = Error – Last Error) [%; %]

akční zásah = (Kp*Error) + (Kp*Kd*Difference) [%; -, %]

(Correction = (Kp*Error) + (Kp*Kd*Difference)) [%; -, %]

kde ***Kp*** je proporcionální (P) konstanta
Kd je diferenční (D) konstanta

Nastavení *Kp* a *Kd* regulátoru:

Tyto konstanty je opět možné nastavit buď zkusmo nebo pomocí nějaké metody, opět např. Ziegler-Nicholsovy.

Ziegler-Nicholsova metoda spočívá v nastavení konstanty *Kd* na nulu a následném postupném zvětšování zvětšování konstanty *Kp* až do hodnoty *Kk* (kritické), při které systém začne kmitat (s konstantní periodou *Tk* [s]).

Z hodnot *Kk* a *Tk* lze poté vypočítat konstanty *Kp* a *Kd* podle vzorců:

Kp = 0,8*Kk [-; -]

Kd = 0,125*Tk [s; s]

Vsuvka:

konstanty podle Ziegler-Nicholsovy metody(*Ku= Kk, Tu=Tk*)

Ziegler–Nichols method[1]			
Control Type	K_p	T_i	T_d
P	$0.5K_u$	-	-
PI	$0.45K_u$	$T_u/1.2$	-
PD	$0.8K_u$	-	$T_u/8$
classic PID[2]	$0.6K_u$	$T_u/2$	$T_u/8$
Pessen Integral Rule[2]	$0.7K_u$	$T_u/2.5$	$3T_u/20$
some overshoot[2]	$0.33K_u$	$T_u/2$	$T_u/3$
no overshoot[2]	$0.2K_u$	$T_u/2$	$T_u/3$

[1]

These 3 parameters are used to establish the correction $u(t)$ from the error $e(t)$ via the equation:

$$u(t) = K_p \left(e(t) + \frac{1}{T_i} \int_0^t e(\tau)d\tau + T_d \frac{de(t)}{dt} \right)$$

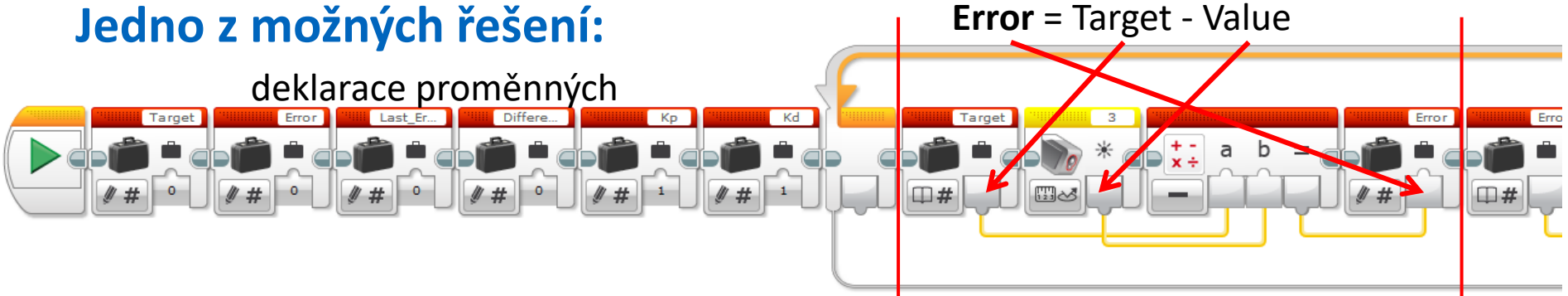
which has the following transfer function relationship between error and controller output:

$$u(s) = K_c \left(1 + \frac{1}{T_i s} + T_d s \right) e(s) = K_c \left(\frac{T_d T_i s^2 + T_i s + 1}{T_i s} \right) e(s)$$

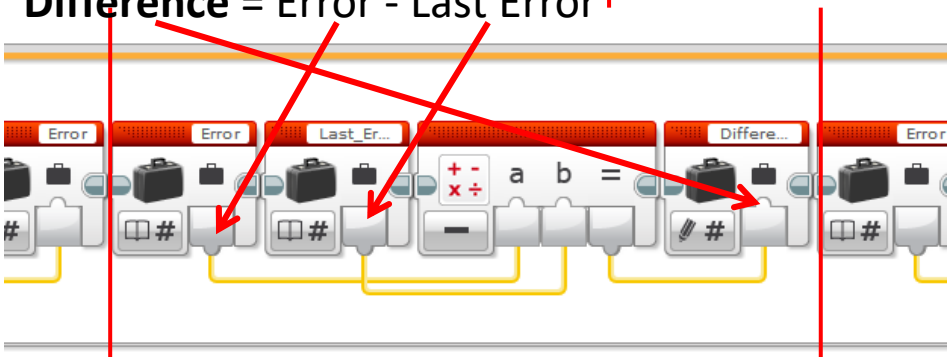
konec vsuvky

Jedno z možných řešení:

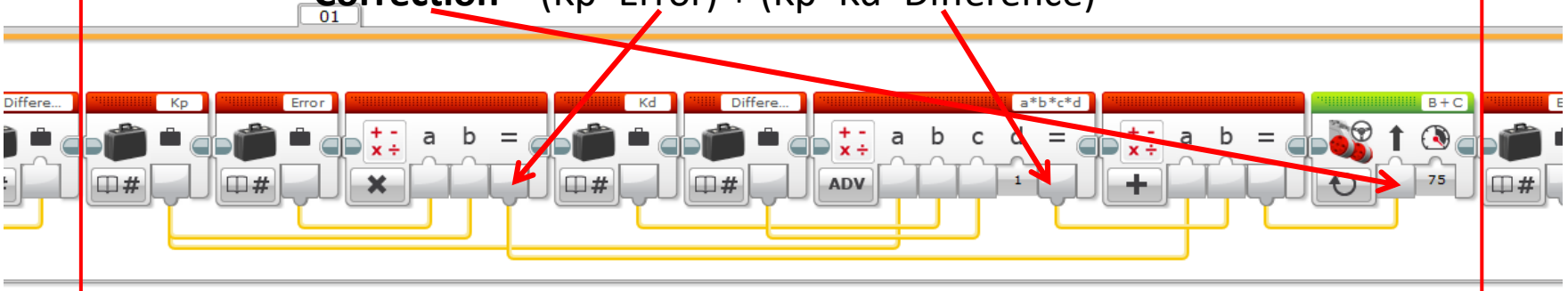
deklarace proměnných



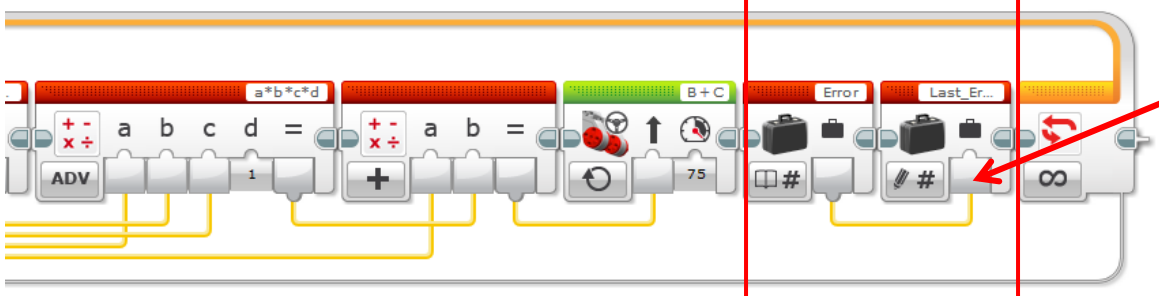
Error = Target - Value



Difference = Error - Last Error



Correction = (Kp*Error) + (Kp*Kd*Difference)



Last Error = Error

Poznámka 1: Podobně je možno řešit regulátor pro robota stopaře jako PS regulátor (proporcionálně sumační regulátor) nebo i PSD regulátor (proporcionálně sumačně diferenční regulátor).

Příklad: jedno z možných řešení s PSD regulátorem:

Poznámka 2: U analogového regulátoru by se jednalo o známější verze - PI regulátor (proporcionálně integrační regulátor) nebo PID regulátor (proporcionálně integračně derivační regulátor).