PRAXE

Datum: 07.11. 2023	Střední průmyslová škola, Chomutov, Školní 50, příspěvková organizace	Třída: V4
Číslo úlohy: 1.	Robot	Jméno: Tomáš Bartoš

Zadání:

S využitím IO karty sestavte v jazyku C# program pro ovládání modelu robota. Dle svého uvážení zvolte buď prostředí konzolové aplikace nebo grafické prostředí.

Výsledné řešení musí obsahovat následující funkcionalitu:

- Automatické nastavení výchozí polohy všech pohonů
- Manuální ovládání robota prostřednictvím ovladače s ukládáním provedeného pohybu do souboru v libovolné podobě
- Zopakování manuálně naučeného pohybu uloženého v souboru
- Průběžná a použitelná indikace aktuální operace a stavu zařízení na monitoru

Teorie:

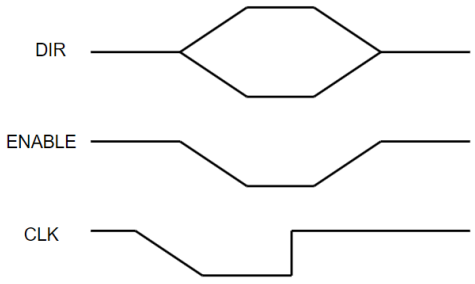
Řešení úlohy zahrnujícího robotickou paži je postaveno na 2 hlavních částech. První z částí je program pro obsluhu hardwaru, tudíž ruky samotné. Druhá část je pak software obslužné aplikace, jejíž primárním účelem je poskytnout uživateli informace o aktuálním stavu a zároveň umožnit mu jednoduchou interakci za pomocí nahrávání/ukládání provedených pohybů.

Díky nutnosti ovládání pohybu robota jsou v modelu použity krokové motory. Každý krokový motor se sestává ze statoru a rotoru, přičemž stator je tvořen sadou cívek a rotor může být tvořen buď feromagnetikem nebo magnetem. Krok se na krokovém motoru provádí za pomocí přivedení elektrického proudu na sadu cívek, kolem který v moment přivedení elektrického proudu vznikne elektromagnetické pole, které díky základním vlastnostem magnetismu přitáhne/odpudí část rotoru, což vyvolá otoční o 1 krok.

Z důvodu zjednodušení práce je model robota opatřen o abstrakční vrstvu ve formě driveru všech motorů, který pohyb jednotlivých motorů redukuje pouze na 6 vstupních signálů:

- enable základna
- enable rameno
- enable rameno chapadla
- enable chapadlo
- taktovací pulz
- signál směru otáčení

Pohyb libovolného motoru robota musí být pak prováděn následovně:



Při pohybech motorem je pak důležité krom nastavování logických hodnot na linky brát v potaz i maximální frekvenci, se kterou se může motor otáčet a na základě ní přizpůsobit interní delay programu. U tohoto konkrétního robota je f_{MAX} rovno 450Hz.

Při pohybu každého robota je také důležité nastavení výchozí polohy. U tohoto konkrétního robota je toto nastavování prováděno pomocí série IR LED hradel a kovových plíšků, přičemž v moment, kdy je motor ve výchozí poloze plíšek přeruší IR paprsek a výstupní daného hradla přejde ze stavu log.1 na log. 0.

Z hlediska softwarové části je velmi podstatnou částí řešení získávání dat z herního ovladače. Ten je za účelem poskytnutí lepší user-experience k počítači připojen bezdrátově za pomocí USB receiveru. Princip jeho funkce je pak založen na principu, kdy pokaždé, kdy je se změní jakékoliv tlačítko je z ovladače odeslán report o aktuálních stavech, kde každí tlačítko má svou vlastní hexadecimální „adresu“ pro získání aktuálního stavu.

Popis programu:

Fáze inicializace:

Po spuštění programu se vytváří instance třídy Form1, která slouží jako informační a zároveň ovládací prvek robota. Jakožto reakce na tvoru instance této třídy se spustí metoda SetupObjects, která má za úkol přiřadit globálním proměnným instance pomocných tříd, stejně tak se v metodě nastavují parametry časovače controllerTimer, sloužícího pro periodické získávání reportů od ovladače. Následně je vyvolána metoda ConnectController pomocné třídy Controller, která se pokusí za užití knihovny HidLibrary a vendorID připojit k ovladači, přičemž neúspěch je indikován v uživatelském zobrazení rozsvícením panelu „nepřipojeno“. Proto, aby finální zobrazení obsahovalo všechny náležitosti (data o použitelných souborech pohybu, stav inicializace motorů a správné zobrazení rychlostního módu), je do procesu inicializace zařazena i metoda SetupComponents, jejíž účinky jsou čistě vizuální. Posledním krokem inicializace je asynchronní spuštění metody pro inicializaci motorů InitializeMotors, která spadá pod pomocnou třídu RobotModel.

U metody inicializace je nejdůležitější částí inicializace motoru základny, tj. díky nestandartnímu umístění IR brány, konkrétněji je na rozdíl od ostatních umístěna jinde než u fyzického dorazu motoru. Z tohoto důvodu je nejlepším způsobem inicializace stanovit si pevně daný počet kroků (BASE_CROSS_STEPS), které jsou nutné podniknout k přerušení paprsku v případě, že je výchozí poloha na levém dorazu. Jamile se totiž provede tento počet kroků a nebyl přerušen paprsek, nachází se motor základny v blízkosti dorazu pravého a tudíž inicializace dosáhneme smyčkou vyvolávající pohyb základy směrem vlevo, přičemž ukončení smyčky proběhne v moment přerušení paprsku.

Hlavní „smyčka“ programu:

Jakmile je dokončena inicializace jak motorů, tak softwaru, zobrazí se okno obsluhy a počne „nekonečný“ proces získávání vstupu od uživatele na základě přerušení generovaných pomocí časovače controllerTimer a interakce obsluhy s formulářem.

Vždy když uplyne předem stanovená doba a controllerTimer vygeneruje přerušení je volaná metoda UpdateControllerData, která za pomocí delegáta do vlákna hlavního programu vyvolá metodu HandleControllerLogic. Daná metoda pak za pomocí získání globální proměnné controller gobálního objektu Controller zjistí, zda je ovladač pořád připojen a na základě této informace aktualizuje panel signalizující připojenost/nepřipojenost ovladače. Následně je aktuální stav ovladače přidán jako argument metodě GetPressedBtnText pro „překlad“ aktuálního stavu ovladače na text použitelný k indikaci aktuálně stisknutého tlačítka a metodě HandleMovement, která pomocí větvení programu determinuje, jaký motor a směr mají být předány následně volané metodě Move.

Metoda Move je metodou pouze přechodnou, to znamená, že sama prohyb nevykonává, ale slouží spíše jako abstrakční vrstva. Tato metoda totiž nejdříve zobrazí data o aktuálně pohybovaném motoru pomocí metody ShowMotorInfoData třídy Form1 a následně zjistí určí počet nutných kroků k provedení pohybu rychlostí zvolenou uživatelem. Informace o počtu kroků je pak použita ve smyčce s pevným počtem opakování, kde je n krát volána metoda Step, mající na starosti samotné zapisování do portů. Z důvodu poskytnutí funkcionality zápisu dat o pohybu do souboru smyčka obsahuje větvící operaci, která při pravdivosti proměnné recording krom zavolání metody Step zavolá i metodu Write, patřící do funkce fileOperations.

Samotná metoda Step nejdříve nastaví proměnou označující příslušný výstupní port na log. 1 a poté je tato proměnná na základě vstupních parametrů metody modifikováno. Konkrétní modifikace se provádí pomocí metody NullBit z pomocné třídy BitOperations v případě nastavení pravé strany otáčení a u všech ostatních modifikací výstupní proměnné je to metoda SetBit ze stejné pomocné třídy. Důležitým prvkem při volání obou metod je jejich poslední parametr typu boolean, který definuje, zdali se jedná o systém s inverzní logikou řízení, v tomto případě tomu tak je, proto je všude nastaveno true. Posledním krokem této metody je zavolání metody Tick.

Metoda Tick ihned po zavolání ukládá stav portu do lokální proměnné portOnCall a následně opět provádí bitové operace na tomto portu. Na rozdíl od metody Step je tu ale manipulováno jen s jedním bitem, jež je definován proměnnou RobotWiring.Out.TACT. Byte adresující jeden taktovací bit je za užití metody InvertByte pomocné třídy BitOperations převrácen tak, že z log. 1 je log. 0 a naopak. Následně se takto invertovaný byte nastaví na port za užitím metody SetByte z pomocné třídy Board a počká se polovinu doby periody za užití funkce DelayMiliseconds z pomocné třídy PreciseDelay. Ve druhé části taktu je opět přečtena hodnota výstupního portu, ale v tomto případě je ale taktovací bit přiveden na úroveň log. 1 za užití metody NullBit a inverzního parametru.

Z hlediska interaktivity a možnosti ovládání robota samotným softwarem jsou zde metody HandleFileIO, keyDown a move_button_click, kde HandleFileIO inicializuje proces nahrávání, či ho ukončí; keyDown slouží jako spouštěcí mechanismus nahrávání a move_button_click slouží pro provádění pohybů ze souboru.

HandleFileIO po zavolání vyhodnotí aktuální fázi nahrávání díky globální proměnné fileWriter z pomocné knihovny FileOperatios a pokud je daný objekt nulový, započne nahrávací proces, tudíž nastaví box pro zadávání jména na zapnutý a vybere první charakter v textboxu pro jméno, kde je již předvyplněna přípona „.bot“. V opačném případě se pouze nastaví globální proměnná recording na false a vyvolá se metoda BreakConnection z pomocné třídy FileOperations, jež jednoduše nastaví FileWriter objekt na null.

keyDown metoda je zavolána pokaždé, když je stisknuta jakákoliv klávesa, ale jediná klávesa, u které je vykonána obslužná akce je enter, zbytek je vyfiltrován. Po filtraci je získán zadaný název souboru a pomocí metody ValidFileName zkontrolována platnost, pokud je neplatný je zobrazena chyba a uživatel je instruován k zadání nového jména, následně je vyvolána metoda InitFile s parametrem jména zapisovaného souboru, tato metoda soubor vytvoří a do globální proměnné fileWriter se uloží odkaz na tento soubor. Následuje inicializace všech motorů a nastavení proměnné recording na log. 1 se společným vyvoláním metody PanelRecording, která signalizuje uživateli, že se aktuálně nahrávají jeho pohyby s robotem.

Move_button_click je motoda vyvolaná v moment, kdy uživatel vybere soubor s daty o pohybu a svůj výběr potvrdí. Jakmile je metoda vyvolaná, vybere se vybraný soubor z combo-boxu jako řetězec. V případě, že je řetězec nesprávná je o tom uživatel informován pomocí messageboxu s chybovou hláškou a celý proces pohybu ze souboru je přerušen. V opačném případě je ale vyvolána metoda MoveByLine z pomocné třídy RobotModel. Tato metoda zinicializuje motory a poté iteruje skrz každý řádek souboru a na základě dat v něm vyvolává metodu Move s příslušnými parametry. Právě ono procházení skrz jednotlivé řádky je prováděno díky metodě ReadFile, jež prvotně pomocí metody Combine třídy File vygeneruje cestu k čtenému souboru a následně ho za pomocí globální proměnné fileReader a motedy ReadLine postupně čte. Čtení je prováděno až do momentu, kdy je souboru roven null a data jsou z funkce vracena v podobě řetězcového pole vraceného za užití yield operátoru, který na rozdíl od return vrací data ihned v moment vyvolání s tím, že běh funkce, jež data vrátila, probíhá dále. Yield tak zaručuje nízkou prodlevu mezi čtením a reálným vykonáváním instrukcí ze souboru.

Rozbor Proměnných a metod:

Metody:

Form1.cs			
Návratový typ	**Název metody**	**Parametry**	**Účel**
void	Form1	None	Inicializuje třídu Form1
void	SetupObjects	None	Inicializuje globální metody a časovač
void	Form1_Load	object sender, EventArgs e	Inicializace objektu formuláře
void	HandleControllerLogic	None	Zobrazuje aktuální stav ovladače
void	HandleMovement	Controller controller	Ovládání motoru na základě vstupu z ovladače
void	HandleFileIO	None	Supštění či zastevení nahrávání pohybu
void	SwitchMode	None	Přepíná režim řízení a mění zobrazení
void	UpdateSpeedPanel	None	Řídí logiku panelu souvisejícího s rychlostí
void	ShowMotorInitData	bool baseMotorPanel, bool armMotorPanel, bool neckMotorPanel, bool gripperMotorPanel	Zobrazuje aktuální stav inicializace robotické paže
void	ShowMotorInfoData	char motor, char dir	Zobrazuje informace o aktuálním motoru a směru
bool	ValidFileName	string name	Posouzení platnosti názvu souboru
void	keyDown	object sender, KeyEventArgs e	Zpracování názvu zapisovaného souboru
void	move_button_Click	object sender, EventArgs e	Zpracování vybraného pohybového souboru

BitOperations.cs			
Návratový typ	**Název metody**	**Parametry**	**Účel**
byte	SetBit	byte value, int bit, bool inversedLogic=false	Nastaví konkrétní bit v daném bytu.
byte	NullBit	byte value, int bit, bool inversedLogic=false	Anuluje konkrétní bit v daném bytu.
byte	InvertByte	byte value	Invertuje všechny bity v bytu.
byte	ChangeBit	byte value, int bit, bool inversedLogic=false	Přepíná konkrétní bit v daném bytu.
byte	MergeBytes	byte byte0, byte byte1	Spojení 2 bytů za pomocí log. AND operace
bool	IsHigh	byte value, int bit, bool inverseLogic=false	Kontrola, zda je daný bit v log. 1
bool	IsLow	byte value, int bit, bool inverseLogic=false	Kontrola, zda je daný bit v log.0

Board.cs			
Návratový typ	**Název metody**	**Parametry**	**Účel**
void	InitObjects	None	Inicializuje objekty související s IO deskou.
void	PortSetup	None	Nastavuje výchozí hodnoty portů.
void	SetByte	int port, byte data	Nastavuje byte na daný port.
bool	GetBit	int port, int bitAddress	Zjišťuje stav konkrétního bitu na daném portu.
byte	GetPortState	int port	Získává stav celého portu.
void	ShowError	string error	Zobrazuje chybové hlášení pomocí MessageBoxu.

Controller.cs			
Návratový typ	**Název metody**	**Parametry**	**Účel**
bool	ConnectController	None	Spojenís ovladačem pomocí HID knihovny.
void	OnControllerReport	HidReport report	Volá aktualizaci proměnných a říká si o další report.
void	UpdateControllerState	HidReport report	Aktualizuje hodnoty globálních proměnných na základě reportu.
void	ControllerDisconnected	None	Ukončí spojení s ovladačem.

FileOperations.cs			
Návratový typ	**Název metody**	**Parametry**	**Účel**
bool	InitFile	string FileName	Navazuje spojení s konkrétním souborem pro zápis.
void	Write	int mode, char motor, char direction	Zapisuje data o pohybu do předdefinovaného souboru.
IEnumerable<string[]>	ReadFile	string fileName	Čte soubor a vrací data postupně pomocí iteratoru.
List<string>	GetAvailableFiles	None	Vrací seznam použitelných souborů pohybů.
void	BreakConnection	None	Ukončuje "připojení" k souboru (nastavuje fileWriter a fileReader na null).
void	ShowError	string msg	Zobrazuje chybové hlášení pomocí MessageBox.

RobotModel.cs			
Návratový typ	**Název metody**	**Parametry**	**Účel**
void	Step	char motor, char dir	Provede jedno kolo s motory ve specifikovaném směru.
void	Move	char motor, char dir, int modelID	Pohne se ve steps a záznam o pohybu uloží do souboru.
void	MoveByLine	string file	Pohne robotem podle načtených dat z určeného souboru.
void	InitializeMotors	None	Inicializuje motory, dostane robota do počáteční pozice.
void	Tick	int port	Poskytuje motorům hodiny (Clock Pulse).

Proměnné:

Form1.cs		
Typ	**Název**	**Účel**
Timer	controllerTimer	Timer pro pravidelné aktualizace dat z gamepadu
BitOperations	bitOperations	Instance třídy BitOperations pro operace s bity
PreciseDelay	preciseDelay	Instance třídy PreciseDelay pro přesné zpoždění
Controller	controller	Instance třídy Controller pro ovládání gamepadu
RobotModel	robotModel	Instance třídy RobotModel pro řízení robota
FileOperations	fileOperations	Instance třídy FileOperations pro práci se soubory
int	controllerPullTimer	Interval času pro aktualizace dat z ovladače
int	currentMode	Aktuální režim pohybu robota (0 - 2)

Board.cs		
Typ	**Název**	**Účel**
DeviceInformation	dev	Objekt označující IO desku
InstantDoCtrl	DO	Objekt pro ovládání výstupů na IO desce
InstantDiCtrl	DI	Objekt pro ovládání vstup na IO desce
BitOperations	bitOperations	Instance třídy pro manipulaci s bity

Controller.cs		
Typ	**Název**	**Účel**
bool	buttonOne	Stav tlačítka s popiskem 1
bool	buttonTwo	Stav tlačítka s popiskem 2
bool	buttonThree	Stav tlačítka s popiskem 3
bool	buttonFour	Stav tlačítka s popiskem 4
bool	buttonLeft	Stav tlačítka s levou šipkou
bool	buttonRight	Stav tlačítka s pravou šipkou
bool	buttonTop	Stav tlačítka s spodní šipkou
bool	buttonBottom	Stav tlačítka s horní šipkou
bool	buttonFrontL	Stav předního velkého tlačítka na levo
bool	buttonFrontR	Stav předního velkého tlačítka na pravo
JoystickState	joystickLeft	Stav levého joysticku (nahnutí v ose X a Y)
JoystickState	joystickRight	Stav pravého joysticku (nahnutí v ose X a Y)
int	vendorID	ID výrobce pro identifikaci zařízení
double	zeroOffset	Hodnota pro odstranění deadbandu joysticku
bool	connected	Stav připojení k gamepadu
HidDevice	controller	Instance třídy HidDevice pro práci s ovladačem

FileOperations.cs

Typ	Název	Účel
string	dirPath	Cesta k adresáři pro ukládání pohybových dat
StreamWriter	fileWriter	Objekt pro zápis do souboru
StreamReader	fileReader	Objekt pro čtení ze souboru

RobotModel.cs

Typ	Název	Účel
BitOperations	bitOperations	Objekt pro manipulaci s bity
PreciseDelay	preciseDelay	Objekt pro přesné časování
Board	board	Objekt reprezentující desku robota
FileOperations	fileOperations	Objekt pro operace se soubory
Form1	form	Reference na hlavní formulář aplikace
bool	recording	Indikátor, zda probíhá záznam pohybů

RobotWiring - General

Typ	Název	Účel
int	FREQ	Maximální frekvence (450Hz)
long	TOT_DELAY	Celkový časový odstup mezi kroky v mikrosekundách
long	HALF_DELAY	Časový odstup mezi každou fází kroku
int	BASE_CROSS_STEPS	Počet kroků potřebných k přerušení IR brány
int	STRADA_STEP	Pomalý pohyb robota
int	SPORT_STEP	Rychlejší pohyb robota
int	CORSA_STEP	Extrémně rychlý pohyb robota

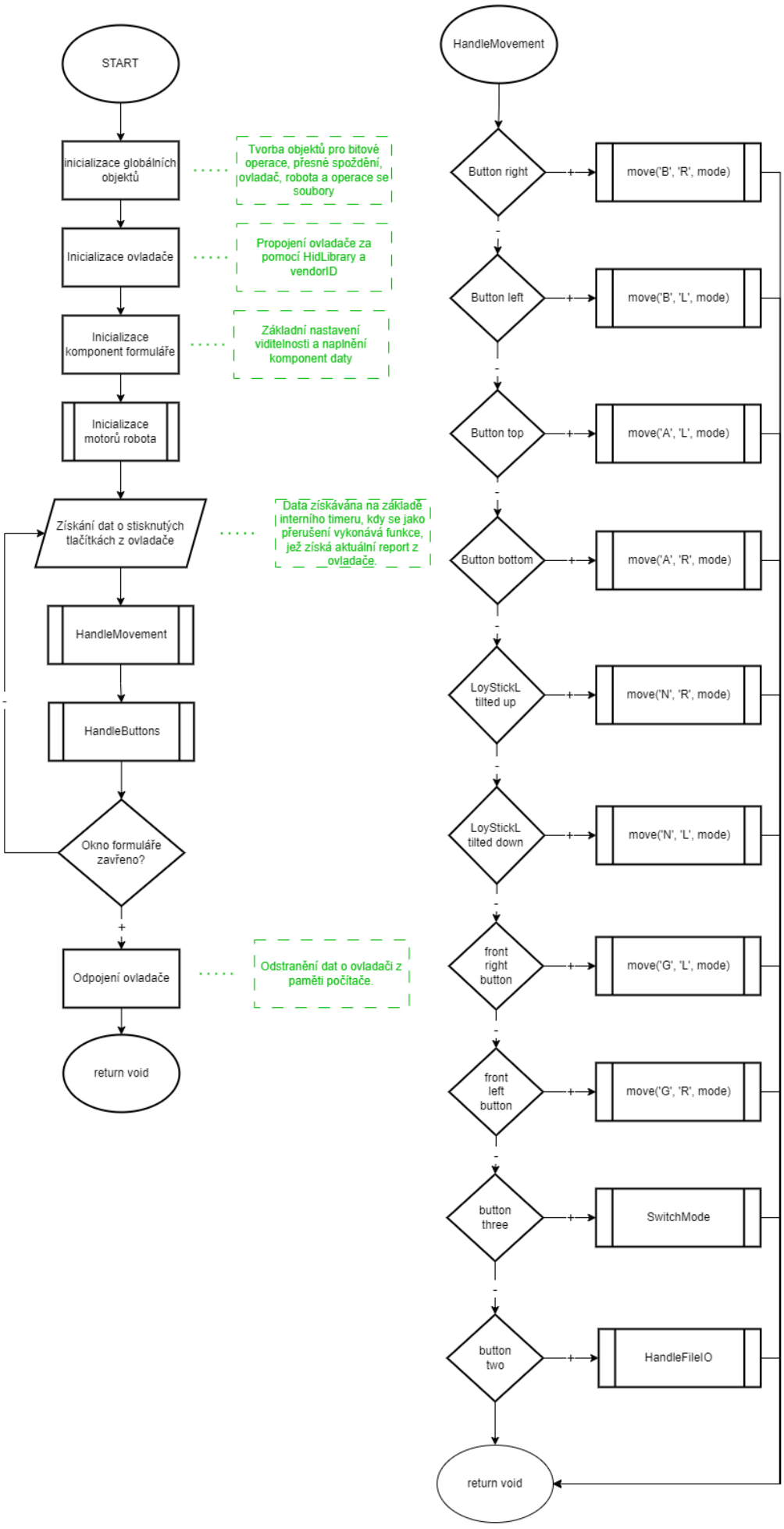
RobotWiring.cs - Out

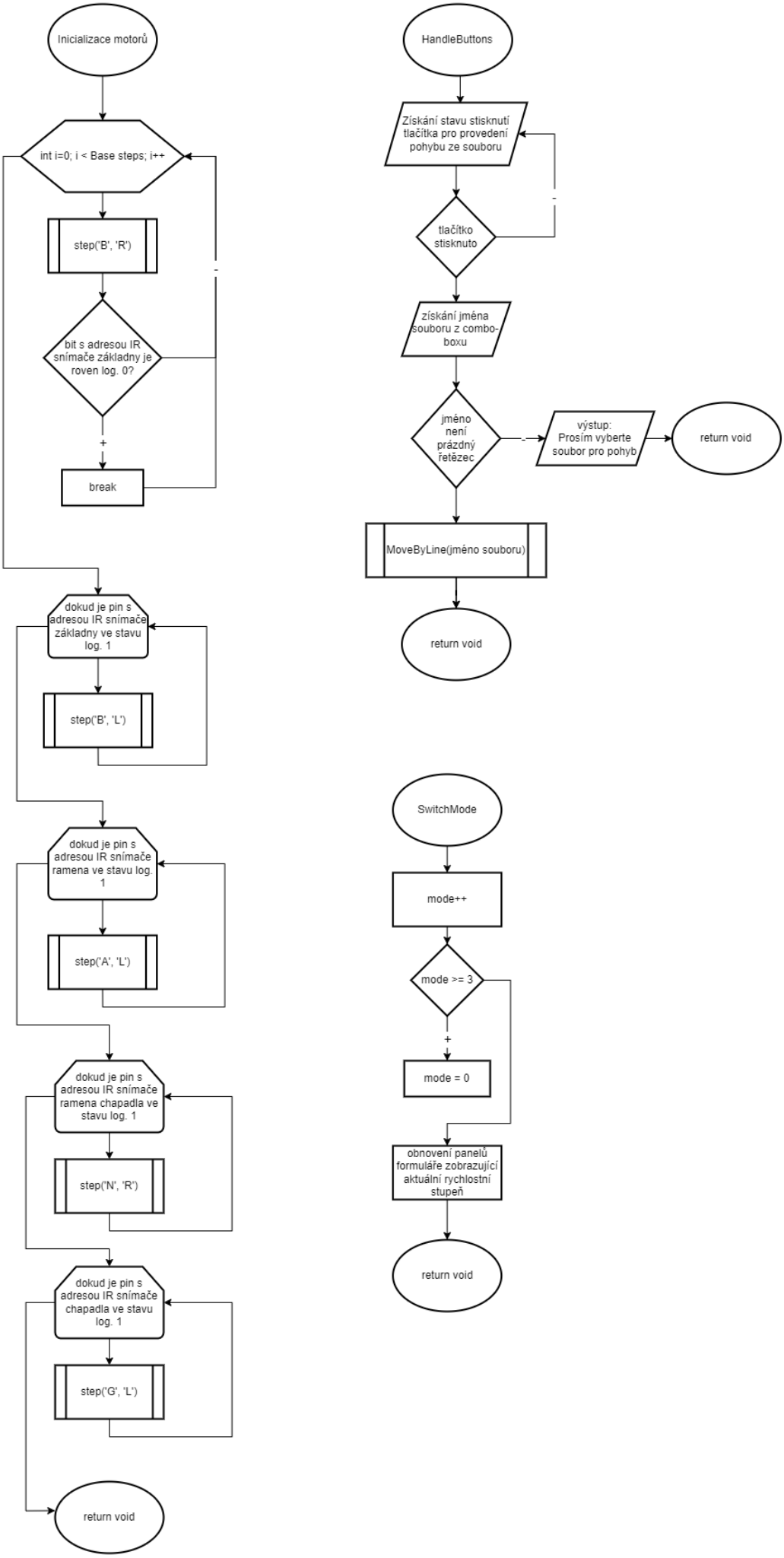
Typ	Název	Účel
int	PORT	Číslo portu pro výstupní signály
byte	BASE	Signál pro motor základny
byte	ARM	Signál pro motor hlavní paže
byte	NECK	Signál pro motor paže chapdla
byte	GRIP	Signál pro motor sevření
byte	TACT	Signál pro posílání hodinových signálů
byte	DIR	Signál pro nastavení směru

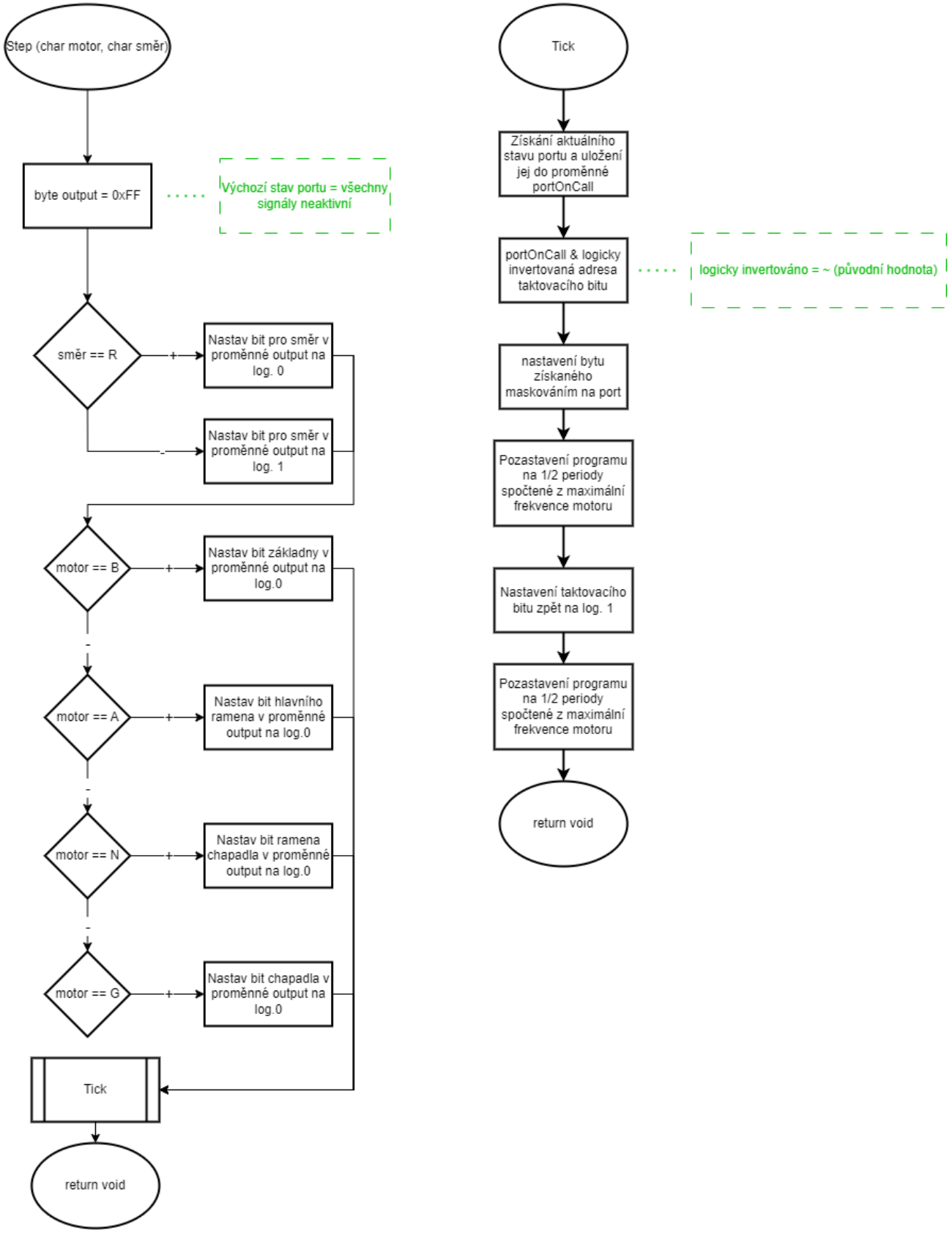
RobotWiring.cs - In

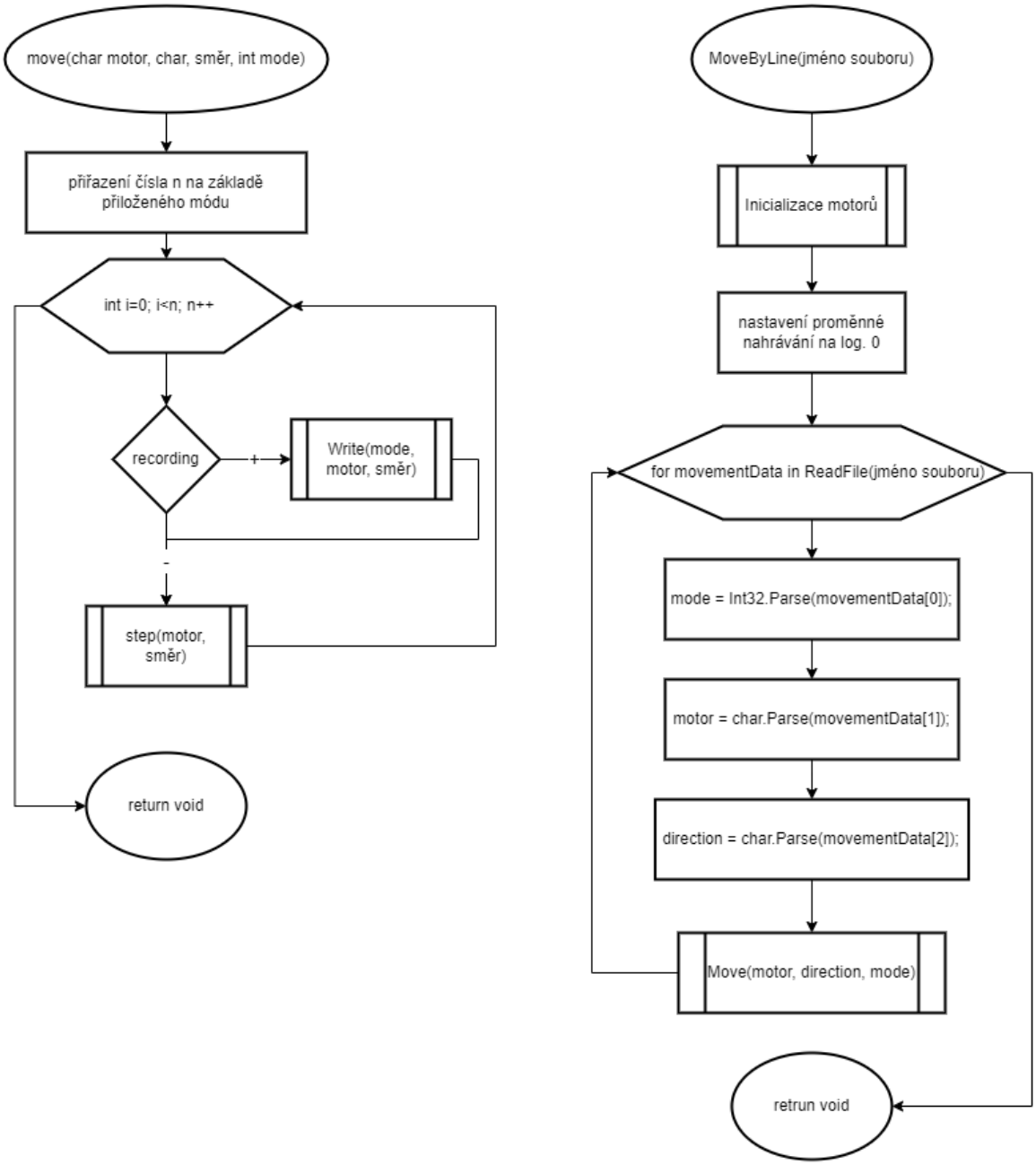
Typ	Název	Účel
int	PORT	Číslo portu pro vstupní signály
byte	BASE_IR	Signál pro IR senzor u základny
byte	ARM_IR	Signál pro IR senzor u hlavní paže
byte	NECK_IR	Signál pro IR senzor u paže chapadla
byte	GRIP_IR	Signál pro IR senzor u sevření chapadla

Vývojový diagram:




Inicializace motorů
int i=0; i < Base steps; i++
step('B', 'R')
bit s adresou IR snímače základny je roven log. 0?
break
dokud je pin s adresou IR snímače základny ve stavu log. 1
step('B', 'L')
dokud je pin s adresou IR snímače ramena ve stavu log. 1
step('A', 'L')
dokud je pin s adresou IR snímače ramena chapadla ve stavu log. 1
step('N', 'R')
dokud je pin s adresou IR snímače chapadla ve stavu log. 1
step('G', 'L')
return void
HandleButtons
Získání stavu stisknutí tlačítka pro provedení pohybu ze souboru
tlačítko stisknuto
získání jména souboru z combo-boxu
jméno není prázdný řetězec
výstup: Prosím vyberte soubor pro pohyb
return void
MoveByLine(jméno souboru)
return void
SwitchMode
mode++
mode >= 3
mode = 0
obnovení panelů formuláře zobrazující aktuální rychlostní stupeň
return void


Step (char motor, char směr)
byte output = 0xFF
Výchozí stav portu = všechny signály neaktivní
směr == R
+
Nastav bit pro směr v proměnné output na log. 0
-
Nastav bit pro směr v proměnné output na log. 1
motor == B
+
Nastav bit základny v proměnné output na log.0
-
motor == A
+
Nastav bit hlavního ramena v proměnné output na log.0
-
motor == N
+
Nastav bit ramena chapadla v proměnné output na log.0
-
motor == G
+
Nastav bit chapadla v proměnné output na log.0
Tick
return void
Tick
Získání aktuálního stavu portu a uložení jej do proměnné portOnCall
portOnCall & logicky invertovaná adresa taktovacího bitu
logicky invertováno = ~ (původní hodnota)
nastavení bytu získaného maskováním na port
Pozastavení programu na 1/2 periody spočtené z maximální frekvence motoru
Nastavení taktovacího bitu zpět na log. 1
Pozastavení programu na 1/2 periody spočtené z maximální frekvence motoru
return void


move(char motor, char, směr, int mode)
přiřazení čísla n na základě přiloženého módu
int i=0; i<n; n++
recording
+
Write(mode, motor, směr)
-
step(motor, směr)
return void
MoveByLine(jméno souboru)
Inicializace motorů
nastavení proměnné nahrávání na log. 0
for movementData in ReadFile(jméno souboru)
mode = Int32.Parse(movementData[0]);
motor = char.Parse(movementData[1]);
direction = char.Parse(movementData[2]);
Move(motor, direction, mode)
retrun void

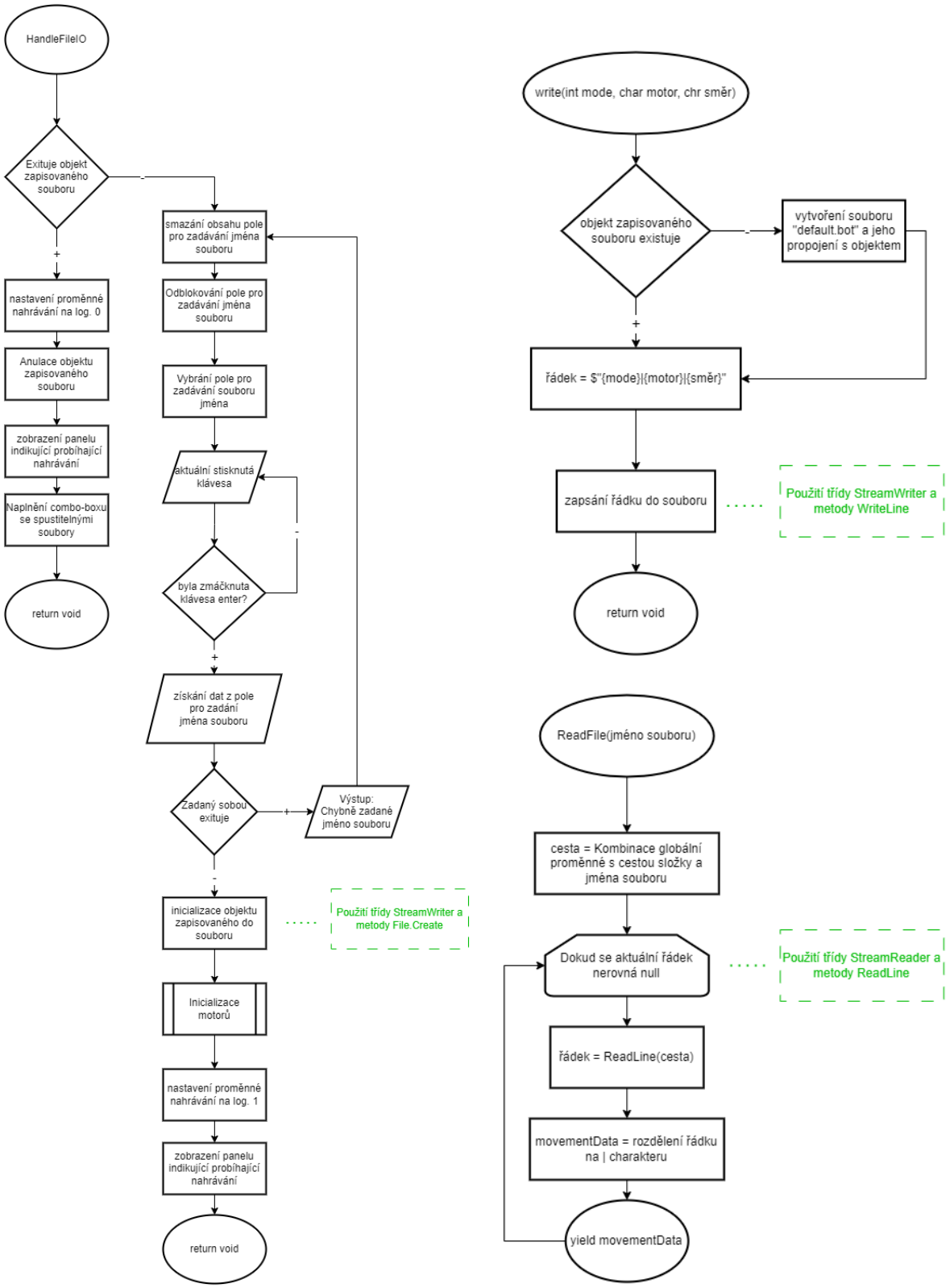
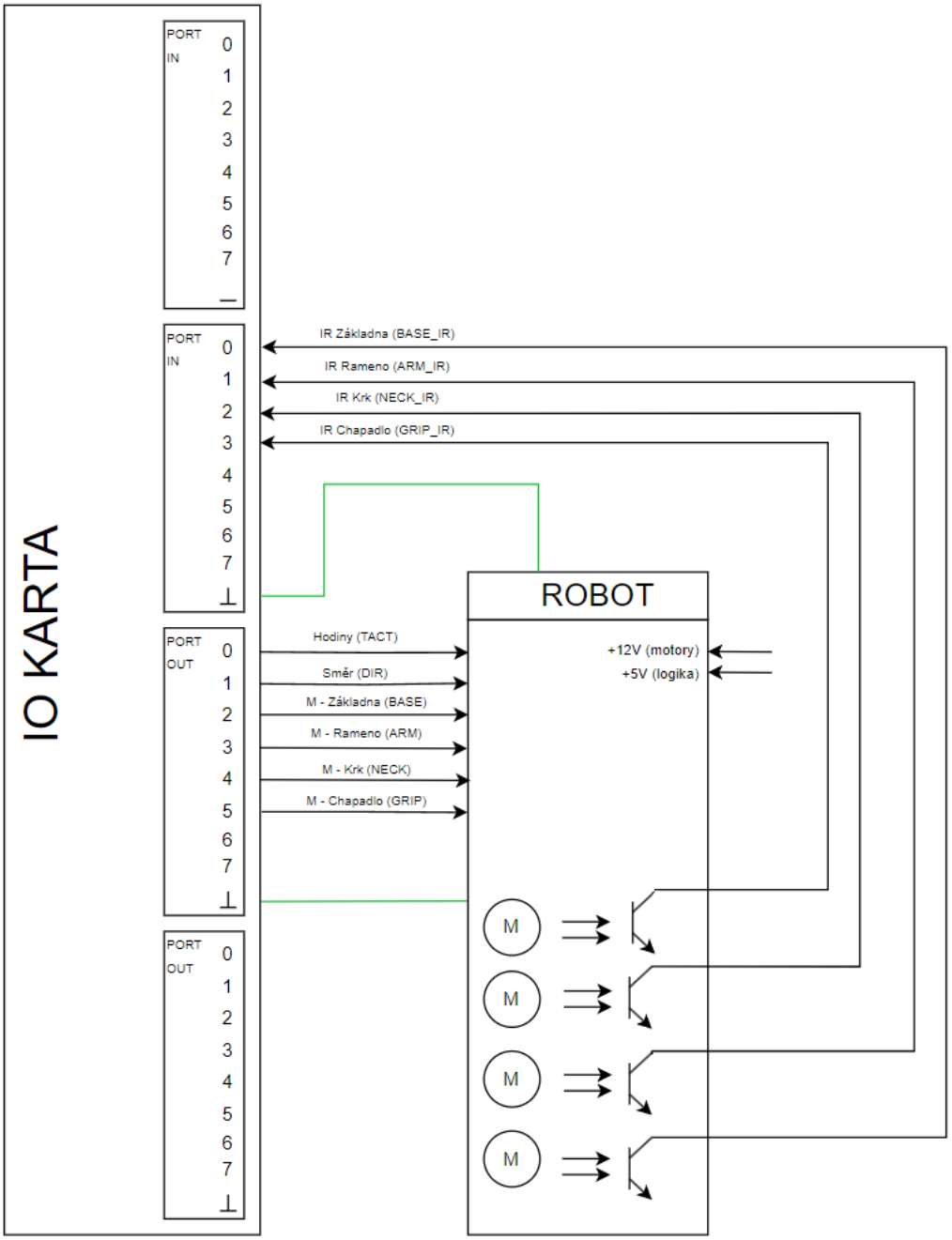

HandleFileIO
Exituje objekt zapisovaného souboru
+
nastavení proměnné nahrávání na log. 0
Anulace objektu zapisovaného souboru
zobrazení panelu indikující probíhající nahrávání
Naplnění combo-boxu se spustitelnými soubory
return void
-
smazání obsahu pole pro zadávání jména souboru
Odblokování pole pro zadávání jména souboru
Vybrání pole pro zadávání souboru jména
aktuální stisknutá klávesa
byla zmáčknuta klávesa enter?
-
+
získání dat z pole pro zadání jména souboru
Zadaný sobou exituje
+
Výstup: Chybně zadané jméno souboru
-
inicializace objektu zapisovaného do souboru
Použití třídy StreamWriter a metody File.Create
Inicializace motorů
nastavení proměnné nahrávání na log. 1
zobrazení panelu indikující probíhající nahrávání
return void
write(int mode, char motor, chr směr)
objekt zapisovaného souboru existuje
-
vytvoření souboru "default.bot" a jeho propojení s objektem
+
řádek = $"{mode}|{motor}|{směr}"
zapsání řádku do souboru
Použití třídy StreamWriter a metody WriteLine
return void
ReadFile(jméno souboru)
cesta = Kombinace globální proměnné s cestou složky a jména souboru
Dokud se aktuální řádek nerovná null
Použití třídy StreamReader a metody ReadLine
řádek = ReadLine(cesta)
movementData = rozdělení řádku na | charakteru
yield movementData

Schéma zapojení:



Komentovaný výpis program:

Form1.cs:

```
using Microsoft.Win32;
using Robot.Essentials;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Runtime.ExceptionServices;
using System.Security.Permissions;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading;
using System.Threading.Tasks;
using System.Timers;
using System.Windows.Forms;


namespace Robot
{
    public partial class Form1 : Form
    {
        System.Timers.Timer controllerTimer { get; set; }
        BitOperations bitOperations { get; set; }
        PreciseDelay preciseDelay { get; set; }

        Controller controller { get; set; }

        RobotModel robotModel { get; set; }

        FileOperations fileOperations { get; set; }

        // global variables
        int controllerPullTimer = 10;
        int currentMode = 0;

        public Form1()
        {
            InitializeComponent();
        }

        private void SetupObjects()
        {
            // function for setting up the global objects and timer

            // setup global objects
            bitOperations = new BitOperations();
            preciseDelay = new PreciseDelay();
            controller = new Controller();
            robotModel = new RobotModel(this);
            fileOperations = new FileOperations();
```

```
        // setup timer object
        controllerTimer = new System.Timers.Timer();
        controllerTimer.Interval = controllerPullTimer;
        controllerTimer.AutoReset = true;
        controllerTimer.Elapsed += UpdateControllerData;
        controllerTimer.Start();

    }

    private void UpdateControllerData(object sender, ElapsedEventArgs e)
    {
        this.Invoke((MethodInvoker)delegate
        {
            // display the current controller data in form
            HandleControllerLogic();
        });
    }

    private void SetupComponents()
    {
        // function for the initial setup of components
        controller_connected_panel.Visible = false;
        controller_disconnected_panel.Visible = true;

        controller_pressed_btn.Text = "";
        motor_info_text.Text = "";
        direction_info_text.Text = "";

        // disable the init panel
        ShowMotorInitData(false, false, false, false);

        // set current mode
        UpdateSpeedPanel();

        // set the file name input to disbaled
        file_name_text.Enabled = false;

        // disbale the move panel
        move_panel.Enabled = true;

        // set sombobox as pure combobox
        available_file_combo.DropDownStyle = ComboBoxStyle.DropDownList;

        // fill the combobox with data
        FillCombo();
    }

    private async void Form1_Load(object sender, EventArgs e)
    {
        // function for inicializing the form and objects related to it
        // enable reading key press
        this.KeyPreview = true;
        SetupObjects();
```

```
            controller.ConnectController();
            SetupComponents();

            // run the motor init in async
            await Task.Run(() => robotModel.InitializeMotors());
        }

        private void Form1_Closing(object sender, System.ComponentModel.CancelEventArgs e)
        {
            // function for disconnecting the controller, so no problems arise
            controller.ControllerDisconnected();
            controller = null;
        }

        private void FillCombo()
        {
            // function that fills the combobox with data about usable files
            available_file_combo.Items.Clear();
            foreach(string file in fileOperations.GetAvailableFiles())
            {
                available_file_combo.Items.Add(file);
            }
        }
        private void HandleControllerLogic()
        {
            // function for displaying controller related info in the form

            // connection status
            if (controller.controller != null)
            {
                controller_connected_panel.Visible = true;
                controller_disconnected_panel.Visible = false;
            }
            else
            {
                controller_connected_panel.Visible = false;
                controller_disconnected_panel.Visible = true;
            }

            // show the latest clicked button
            controller_pressed_btn.Text = GetPressedBtnText(controller);

            // handle movement
            HandleMovement(controller);
        }

        public void HandleMovement(Controller controller)
        {
            // function for deciding which motor to move with

            // MOVEMENT controlls
            if (controller.buttonRight)
            {
                //robotModel.Step('B', 'R');
```

```
                robotModel.Move('B', 'R', currentMode);
            }

            if (controller.buttonLeft)
            {
                //robotModel.Step('B', 'L');
                robotModel.Move('B', 'L', currentMode);
            }

            if (controller.buttonTop)
            {
                //robotModel.Step('A', 'L');
                robotModel.Move('A', 'L', currentMode);
            }
            if (controller.buttonBottom)
            {
                //robotModel.Step('A', 'R');
                robotModel.Move('A', 'R', currentMode);
            }

            if(controller.joystickLeft.accelX == 0 && controller.joystickLeft.accelY == -
1)
            {
                robotModel.Move('N', 'R', currentMode);
            }

            if (controller.joystickLeft.accelX == 0 && controller.joystickLeft.accelY ==
1)
            {
                robotModel.Move('N', 'L', currentMode);
            }

            if (controller.buttonFrontR)
            {
                //robotModel.Step('G', 'R');
                robotModel.Move('G', 'R', currentMode);
            }

            if (controller.buttonFrontL)
            {
                //robotModel.Step('G', 'L');
                robotModel.Move('G', 'L', currentMode);
            }

            // GENERAL controlls
            if (controller.buttonThree)
            {
                // switch drive mode
                SwitchMode();
            }

            if (controller.buttonTwo)
            {
                // start/stop the recording based on the nternal state of variables
```

```
            HandleFileIO();
        }
    }

    public void HandleFileIO()
    {
        // function for starting to record or saving the recorded file

        if(fileOperations.fileWriter == null)
        {
            // the connecion has not been established yet
            move_panel.Enabled = false;
            file_name_text.Enabled = true;
            file_name_text.Text = " .bot"; // reset the value
            file_name_text.Select(0, 1);
        }
        else
        {
            // connsection has been established -> end the recording process
            robotModel.recording = false;
            move_panel.Enabled = true;
            fileOperations.BreakConncetion();

            // refresh the data in combobx
            FillCombo();
            PanelNormal();
        }
    }

    public void SwitchMode()
    {
        // function for swittchin the drive mode var and changing the view

        currentMode++;
        if(currentMode >= 3)
        {
            currentMode = 0;
        }
        UpdateSpeedPanel();
    }

    public void UpdateSpeedPanel()
    {
        // function for handling the speed related panel logic
        strada_panel.Visible = currentMode == 0;
        sport_panel.Visible = currentMode == 1;
        corsa_panel.Visible = currentMode == 2;
    }

    public void PanelRecording()
    {
        // show the correct data when the programme is recording
        recording_text.Text = "ZAPNUTO";
        recording_panel.BackColor = Color.Red;
```

```
            recording_panel.ForeColor = Color.Black;
        }

        public void PanelNormal()
        {
            // show the correct data when the programme is not recording
            recording_text.Text = "VYPNUTO";
            recording_panel.BackColor = Color.Gray;
            recording_panel.ForeColor = Color.White;
        }

        public string GetPressedBtnText(Controller controller)
        {
            // function translating the internal state of buttons to string
            // Check each button and return the name of the pressed button
            if (controller.buttonOne) return "ButtonOne";
            if (controller.buttonTwo) return "ButtonTwo";
            if (controller.buttonThree) return "ButtonThree";
            if (controller.buttonFour) return "ButtonFour";
            if (controller.buttonFrontL) return "ButtonFrontL";
            if (controller.buttonFrontR) return "ButtonFrontR";
            if (controller.buttonLeft) return "ButtonLeft";
            if (controller.buttonRight) return "ButtonRight";
            if (controller.buttonTop) return "ButtonTop";
            if (controller.buttonBottom) return "ButtonBottom";

            // Check if JoystickLeft is tilted
            if (Math.Abs(controller.joystickLeft.accelX) > controller.zeroOffset ||
                Math.Abs(controller.joystickLeft.accelY) > controller.zeroOffset)
            {
                return $"JoyL: X={controller.joystickLeft.accelX},
Y={controller.joystickLeft.accelY}";
            }

            // Check if JoystickRight is tilted
            if (Math.Abs(controller.joystickRight.accelX) > controller.zeroOffset ||
                Math.Abs(controller.joystickRight.accelY) > controller.zeroOffset)
            {
                return $"JoyR: X={controller.joystickRight.accelX},
Y={controller.joystickRight.accelY}";
            }

            // If no button is pressed and no joystick is tilted, return an empty string
or null
            return "";
        }

        public void ShowMotorInitData(bool baseMotorPanel, bool armMotorPanel, bool
neckMotorPanel, bool gripperMotorPanel)
        {
            // function for showing the current status of robot arm inicialization
            init_base_panel.Visible = baseMotorPanel;
            init_arm_panel.Visible = armMotorPanel;
            init_neck_panel.Visible = neckMotorPanel;
```

```
            init_gripper_panel.Visible = gripperMotorPanel;
        }

        public void ShowMotorInfoData(char motor, char dir)
        {
            // function for displaying info about the current motor and direction
            Dictionary<char, string> motorDict = new Dictionary<char, string>{
                { 'B', "BASE" },
                { 'A', "ARM" },
                { 'N', "NECK" },
                { 'G', "GRIPPER" }
            };
            Dictionary<char, string> dirDict = new Dictionary<char, string>{
                { 'L', "LEFT/UP" },
                { 'R', "RIGHT/DOWN" },
            };

            motor_info_text.Text = motorDict[motor];
            direction_info_text.Text = dirDict[dir];
        }

        public bool ValidFileName(string name)
        {
            // function for assessing the validity of a file name with regular expressions
            if (name.EndsWith(".bot"))
            {
                return true;
            }
            return false;
        }

        private void keyDown(object sender, KeyEventArgs e)
        {
            // function for handling key down events
            // in the context of this app, only the filename input will be handeled

            if(e.KeyValue != 13)
            {
                // if not enter
                return;
            }

            // enter pressed -> check if input valid
            string name = file_name_text.Text;
            if (!ValidFileName(name))
            {
                file_name_text.Enabled = true;
                file_name_text.Text = " .bot";
                file_name_text.Select(0, 1);
                ShowError("Invalid file name");
                return;
            }

            bool status = fileOperations.InitFile(name);
```

```
            if (!status)
            {
                // if init went wrong ->  abort
                file_name_text.Enabled = true;
                file_name_text.Text = " .bot";
                file_name_text.Select(0, 1);
                ShowError("Invalid file name");
                return;
            }

            // init the motors to start at a reference point before recording
            robotModel.InitializeMotors();

            robotModel.recording = true;
            PanelRecording();
        }

        public void ShowError(string msg)
        {
            // function for displaying errors by showing a messagebox
            MessageBox.Show(msg, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }

        private void move_button_Click(object sender, EventArgs e)
        {
            // function for handling the user selected file
            string selectedName = available_file_combo.SelectedItem as string;

            if(selectedName == "")
            {
                // if the selected name is blank -> make sure to show an error
                ShowError("Prosím vyberte soubor!");
                return;
            }

            // move the robot accordingly to the file
            robotModel.MoveByLine(selectedName);

        }
    }
}
```

BitOperations.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Robot.Essentials
{
    internal class BitOperations
    {
```

```
        // Function to set a specific bit in the given byte.
        public byte SetBit(byte value, int bit, bool inversedLogic=false)
        {
            if (inversedLogic)
            {
                return (byte)(value & ~(0x01 << bit));
            }
            return (byte)(value | (0x01 << bit));
        }

        // Function to null (set to 0) a specific bit in the given byte.
        public byte NullBit(byte value, int bit, bool inversedLogic=false)
        {
            if (inversedLogic)
            {
                return (byte)(value | (0x01 << bit));
            }
            return (byte)(value & ~(0x01 << bit));
        }

        public byte InvertByte(byte value)
        {
            return (byte)~value;
        }

        // Function to toggle a specific bit in the given byte.
        public byte ChangeBit(byte value, int bit, bool inversedLogic=false)
        {
            if (inversedLogic)
            {
                return (byte)(value ^ (0x01 << bit));
            }
            return (byte)(value ^ (0x01 << bit));
        }

        // Function for merging two bytes together
        public byte MergeBytes(byte byte0, byte byte1)
        {
            return (byte)(byte0 & byte1);
        }

        // Function to check if a specific bit in the given byte is high (1).
        public bool IsHigh(byte value, int bit, bool inverseLogic=false)
        {
            if (inverseLogic)
            {
                return ((value & (0x01 << bit)) == 0);
            }
            return ((value & (0x01 << bit)) != 0);
        }

        // Function to check if a specific bit in the given byte is low (0).
        public bool IsLow(byte value, int bit, bool inverseLogic=false)
        {
```

```
            if (inverseLogic)
            {
                return ((value & (0x01 << bit)) != 0);
            }
            return ((value & (0x01 << bit)) == 0);
        }
    }
}
```

Board.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Automation.BDaq;

namespace Robot.Essentials
{
    internal class Board
    {
        public DeviceInformation dev { get; set; }
        public InstantDoCtrl DO { get; set; }
        public InstantDiCtrl DI { get; set; }

        public BitOperations bitOperations { get; set; }

        public Board()
        {
            InitObjects();
            PortSetup();
        }

        public void InitObjects()
        {
            // initialize board related objects

            // basic device information
            dev = new DeviceInformation
            {
                Description = "PCI-1756,BID#0",
                DeviceMode = AccessMode.ModeWrite
            };

            // Digitial output information
            DO = new InstantDoCtrl();
            DO.SelectedDevice = dev;
            DO.LoadProfile("CardProfile.xml");

            // Digital input information
            DI = new InstantDiCtrl();
            DI.SelectedDevice = dev;
```

```
            DI.LoadProfile("CardProfile.xml");

            // init of external objects
            bitOperations = new BitOperations();
        }

        public void PortSetup()
        {
            // Set log. 1 to all of the output ports as the model is active in log. 0
            SetByte(0, 0xFF);
            SetByte(1, 0xFF);
        }

        public void SetByte(int port, byte data)
        {
            // fucntion for setting a byte to a desired port
            DO.Write(port, data);
        }

        public bool GetBit(int port, int bitAddress)
        {
            // function that checks if a bit with a certain address is high or low
            byte data;
            DI.Read(port, out data);

            if(bitOperations.IsHigh(data, bitAddress))
            {
                return true;
            }

            return false;
        }

        public byte GetPortState(int port)
        {
            // function that returns the current state of a port
            byte state;
            DO.Read(port, out state);
            return state;
        }

    }
}
```

Controller.cs:

```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using HidLibrary;
```

```
using static System.Windows.Forms.AxHost;


namespace Robot.Essentials
{
    public struct JoystickState
    {
        public double accelX, accelY;
    }
    public class Controller
    {
        public bool buttonOne, buttonTwo, buttonThree, buttonFour;
        public bool buttonLeft, buttonRight, buttonTop, buttonBottom;
        public bool buttonFrontL, buttonFrontR;
        public JoystickState joystickLeft, joystickRight;
        public int vendorID;
        public double zeroOffset;
        public bool connected;
        public HidDevice controller;

        public Controller(int userVedorID = 0x2563, double userZeroOffset = 0.015)
        {
            vendorID = userVedorID;
            zeroOffset = userZeroOffset;
        }

        public bool ConnectController()
        {
            // function for establishing connection with the gamepad
            if(controller != null)
            {
                // already connected
                return true;
            }

            controller = HidDevices.Enumerate(vendorID).FirstOrDefault(); // get
controller instance or null if not connected

            if(controller != null)
            {
                controller.OpenDevice(); // open HID interface for communication
                controller.MonitorDeviceEvents = true; // monitor the actions of the
controller
                controller.Removed += ControllerDisconnected;
                controller.ReadReport(OnControllerReport);
                controller.ReadAsync();// enable reading asynchonously
                connected = true;

            }
            connected = false;
            return controller != null;
        }

        public void OnControllerReport(HidReport report)
```

```
        {
            // function for calling the report parser method

            UpdateControllerState(report);

            if (controller != null)
            {
                controller.ReadReport(OnControllerReport);
            }
        }


        public void UpdateControllerState(HidReport report)
        {
            // function for updating the values of global variables based on the report
the gamepad sends
            if ((report.ReportId != 0) || (report.Data.Length != 27))
            {
                return;     // probably other controller type / report type data passed
            }

            buttonOne = (report.Data[11] == 0xff);
            buttonTwo = (report.Data[12] == 0xff);
            buttonThree = (report.Data[13] == 0xff);
            buttonFour = (report.Data[14] == 0xff);
            buttonFrontL = ((report.Data[15] | report.Data[16]) > 0x80);
            buttonFrontR = ((report.Data[17] | report.Data[18]) > 0x80);
            buttonLeft = (report.Data[8] == 0xff);
            buttonRight = (report.Data[7] == 0xff);
            buttonTop = (report.Data[9] == 0xff);
            buttonBottom = (report.Data[10] == 0xff);
            buttonFrontL = ((report.Data[15] | report.Data[16]) > 0x80);
            buttonFrontR = ((report.Data[17] | report.Data[18]) > 0x80);
            joystickLeft.accelX = Math.Min(report.Data[3] - 127, 127) / 127.0;
            joystickLeft.accelY = Math.Min(report.Data[4] - 127, 127) / 127.0;
            joystickRight.accelX = Math.Min(report.Data[5] - 127, 127) / 127.0;
            joystickRight.accelY = Math.Min(report.Data[6] - 127, 127) / 127.0;

            // Deadband around 0 for joystick position
            if (Math.Abs(joystickLeft.accelX) < zeroOffset) joystickLeft.accelX = 0;
            if (Math.Abs(joystickLeft.accelY) < zeroOffset) joystickLeft.accelY = 0;
            if (Math.Abs(joystickRight.accelX) < zeroOffset) joystickRight.accelX = 0;
            if (Math.Abs(joystickRight.accelY) < zeroOffset) joystickRight.accelY = 0;
        }
        public void ControllerDisconnected()
        {
            // function for cling the connection with the gamepad
            controller.CloseDevice(); // close connection with controller
            controller = null; // reset the controller var
            connected = false;
        }

    }
}
```

FileOperations.cs:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Robot.Essentials
{
    internal class FileOperations
    {
        public const string dirPath = ".\\movements";

        public StreamWriter fileWriter { get; set; }
        public StreamReader fileReader { get; set; }

        public bool InitFile(string FileName)
        {
            // function for establishing connection witht the desired file
            string fullPath = Path.Combine(dirPath, FileName);

            if (File.Exists(fullPath))
            {
                ShowError("Souvour již existuje!");
                return false;
            }

            fileWriter = new StreamWriter(File.Create(fullPath));
            return true;
        }

        public void Write(int mode, char motor, char direction)
        {
            // function for writing movement data to a predefined file
            try
            {
                if (fileWriter == null)
                {
                    InitFile("default.bot");
                }

                string line = $"{mode}|{motor}|{direction}";
                fileWriter.WriteLine(line);
            }
            catch (Exception ex)
            {
                ShowError($"Chyba při ukládání dat o pohybu do soboru: {ex.Message}");
            }
        }
```

```
        public IEnumerable<string[]> ReadFile(string fileName)
        {
            // function for reading a file and yielding the data as it reads it
            string fullPath = Path.Combine(dirPath, fileName);
            string line;

            using (StreamReader fileReader = new StreamReader(fullPath))
            {
                while ((line = fileReader.ReadLine()) != null)
                {
                    line = line.Trim();
                    string[] movementData = line.Split('|');
                    yield return movementData;
                }
            }
        }

        public List<string> GetAvailableFiles()
        {
            // function that returns a list of usable movement files
            List<string> files = new List<string>();
            try
            {
                DirectoryInfo d = new DirectoryInfo(dirPath);
                FileInfo[] Files = d.GetFiles("*.bot");
                foreach (FileInfo file in Files)
                {
                    files.Add(file.Name);
                }
            }
            catch (Exception ex)
            {
                ShowError($"Error while getting available files: {ex.Message}");
            }
            return files;
        }
        public void BreakConncetion()
        {
            // function for "breaking" the connection with the file

            // set the file writer to null as the file is already automatically saved
            fileWriter = null;

            // reset the file reader
            fileReader = null;
        }

        public void ShowError(string msg)
        {
            // function for displaying errors by showing a messagebox
            MessageBox.Show(msg, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
}
```

PreciseDelay.cs:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Robot.Essentials
{
    public static class StopwatchExtension
    {
        // extend the functionality of System.Diagnostics stopwatch by adding function
that gets elapsed microseconds
        public static long ElapsedMicroseconds(this Stopwatch stopwatch)
        {
            return stopwatch.ElapsedTicks / (Stopwatch.Frequency / (1_000_000L));
        }
    }

    internal class PreciseDelay
    {
        public long DelayMicroseconds(long microseconds)
        {
            Stopwatch delayWatch = Stopwatch.StartNew();

            while (delayWatch.ElapsedMicroseconds() < microseconds)
            {
                // Do nothing, just wait
            }

            delayWatch.Stop();
            return delayWatch.ElapsedMicroseconds();
        }
    }
}
```

RobotModel.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Robot.Essentials
{
    internal class RobotModel
    {
        public BitOperations bitOperations { get; set; }
        public PreciseDelay preciseDelay { get; set; }
        public Board board { get; set; }
```

```
        public FileOperations fileOperations { get; set; }
        public Form1 form { get; set; }

        public bool recording = false;

        public RobotModel(Form1 formParam)
        {
            // assing the global objects
            board = new Board();
            bitOperations = new BitOperations();
            preciseDelay = new PreciseDelay();
            fileOperations = new FileOperations();

            // set the local form var to the received form
            form = formParam;
        }

        public void Step(char motor, char dir)
        {
            // function for performing one step with the motors in specified direction
            // set out port as default for this methid
            int port = RobotWiring.Out.PORT;

            byte output = 0xFF; // initial state of the port


            // set direction
            if (dir == 'R')
            {
                // moving right/up
                output = bitOperations.NullBit(output,
RobotWiring.GetPinID(RobotWiring.Out.DIR), true);
            }
            else
            {
                // moving left/down
                output = bitOperations.SetBit(output,
RobotWiring.GetPinID(RobotWiring.Out.DIR), true);
            }

            // select base motor
            if (motor == 'B')
            {
                // BASE motor
                output = bitOperations.SetBit(output,
RobotWiring.GetPinID(RobotWiring.Out.BASE), true);
            }
            else if (motor == 'A')
            {
                // ARM motor
                output = bitOperations.SetBit(output,
RobotWiring.GetPinID(RobotWiring.Out.ARM), true);
            }
            else if (motor == 'N')
```

```
            {
                // NECK motor
                output = bitOperations.SetBit(output,
RobotWiring.GetPinID(RobotWiring.Out.NECK), true);
            }
            else if(motor == 'G')
            {
                // GRIP motor
                output = bitOperations.SetBit(output,
RobotWiring.GetPinID(RobotWiring.Out.GRIP), true);
            };


            // set the motor and direction pins accordingly
            board.SetByte(port, output);

            // tick the clock to make the step happen
            Tick(port);
        }

        public void Move(char motor, char dir, int modeID)
        {
            // function moving in steps and saving the moves to a file

            // show the info to the screen
            form.ShowMotorInfoData(motor, dir);

            // calculate the number of steps based on the current mode of operation
            int n = (modeID == 0) ? RobotWiring.General.STRADA_STEP : (modeID == 1) ?
RobotWiring.General.SPORT_STEP : RobotWiring.General.CORSA_STEP;
            for(int i = 0; i < n; i++)
            {
                if (recording)
                {
                    // if movements are to be recorded -> write them to a file
                    fileOperations.Write(modeID, motor, dir);
                }
                Step(motor, dir);
            }
        }

        public void MoveByLine(string file)
        {
            // function for movinf with the robot accordingly to the read data
            // initialize the motors to always start at some reference point
            InitializeMotors();

            // make sure that the movements are not recorded again
            recording = false;
            foreach (string[] movementData in fileOperations.ReadFile(file))
            {
                // for each line in the file
                int mode;
                char motor, direction;
```

```
                mode = Int32.Parse(movementData[0]);
                motor = char.Parse(movementData[1]);
                direction = char.Parse(movementData[2]);
                Move(motor, direction, mode);
            }
        }

        public void InitializeMotors()
        {
            // function for getting the robot into init position
            // set the default state to disabled

            form.Invoke(new Action(() =>
                form.ShowMotorInitData(false, false, false, false)
            ));

            // input port
            int port = RobotWiring.In.PORT;


            // BASE motor inicialization
            // first phase of base init -> spin right for a set number of steps

            for (int i = 0; i < RobotWiring.General.BASE_CROSS_STEPS; i++)
            {
                Step('B', 'R'); // spin right

                if (!board.GetBit(port, RobotWiring.GetPinID(RobotWiring.In.BASE_IR)))
                {
                    // if initialized break aout of the loop
                    break;
                }
            }


            // the first phase of base init was not successfull -> spin in the other
direction
            while (board.GetBit(port, RobotWiring.GetPinID(RobotWiring.In.BASE_IR)))
            {
                Step('B', 'L'); // spin left
            }

            // edit the form in a different thread
            form.Invoke(new Action(() =>
                form.ShowMotorInitData(true, false, false, false)
            ));

            // ARM motor inicialization
            // get the state of the IR gate -> if high -> already set; if not go up
            // while should do the job
            while (board.GetBit(port, RobotWiring.GetPinID(RobotWiring.In.ARM_IR)))
            {
                // move up till the gate is not crossed
```

```
                Step('A', 'L');
            }

            // edit the form in a different thread
            form.Invoke(new Action(() =>
                form.ShowMotorInitData(true, true, false, false)
            ));

            // NECK motor inicialization
            while (board.GetBit(port, RobotWiring.GetPinID(RobotWiring.In.NECK_IR)))
            {
                // move up while the gate is not croseed;
                Step('N', 'R');
            }

            // edit the form in a different thread
            form.Invoke(new Action(() =>
                form.ShowMotorInitData(true, true, true, false)
            ));

            //GRIP motor inicialization
            while (board.GetBit(port, RobotWiring.GetPinID(RobotWiring.In.GRIP_IR)))
            {
                // while the gate is not crossed open the gripper
                Step('G', 'L');
            }

            //edit the form in a different thread
            form.Invoke(new Action(() =>
                form.ShowMotorInitData(true, true, true, true)
            ));
        }

        public void Tick(int port)
        {
            // function for providing motors with clock pulse
            // get the value of the whole output port when called
            byte portOnCall = board.GetPortState(port);

            // high clock pulse
            // perform an and operation to only change the bits for clock
            byte val = (byte)(portOnCall &
bitOperations.InvertByte(RobotWiring.Out.TACT));
            board.SetByte(port, val);

            // wait for the motor to react
            preciseDelay.DelayMicroseconds(RobotWiring.General.HALF_DELAY);

            // get the state of the port after the first clock pulse
            portOnCall = board.GetPortState(port);

            // invert the previous value on the TACT pin by nulling it
            board.SetByte(port, bitOperations.NullBit(portOnCall,
RobotWiring.GetPinID(RobotWiring.Out.TACT), true));
```

```
            // wait for the motor to react
            preciseDelay.DelayMicroseconds(RobotWiring.General.HALF_DELAY);
        }
    }
}
```

RobotWiring.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Policy;
using System.Text;
using System.Threading.Tasks;

namespace Robot.Essentials
{
    public static class RobotWiring
    {
        public class Out
        {
            public const int PORT = 1;
            // motor signals
            public const byte BASE = 0x04; // the base of the robot
            public const byte ARM = 0x08; // the first moveable arm
            public const byte NECK = 0x10; // the second moveable arm
            public const byte GRIP = 0x20; // the gripper

            // control signals
            public static byte TACT = 0x01; // signal for sending clock signals
            public static byte DIR = 0x02; // signal for setting the direction
        }

        public static class In
        {
            public const int PORT = 0;
            // each of the inputs relates to one IR sensor on the robot
            public const byte BASE_IR = 0x01;
            public const byte ARM_IR = 0x02;
            public const byte NECK_IR = 0x04;
            public const byte GRIP_IR = 0x08;
        }

        public class General
        {
            public const int FREQ = 450; // max frequency = 450Hz
            public const long TOT_DLEAY = 1_000_000 / FREQ;// total delay between steps in
micro seconds (T = (1/f))
            public const long HALF_DELAY = TOT_DLEAY / 2; // delay between each phase of a
step (2 phases -> from 1 to 0)
            public const int BASE_CROSS_STEPS = 1_500; // the number of steps needed to
cross the IR gate
            public const int STRADA_STEP = 1; // slow robot move
```

```
            public const int SPORT_STEP = 10; // faster robot move
            public const int CORSA_STEP = 25; // super-fast robot
        }
        public static int GetPinID(byte pinAddress)
        {
            // function for converting bit address to a single int (ID of the pin on the
port)
            return (int)Math.Log(pinAddress, 2);
        }
    }
}
```

Odpovědi na otázky:

1. V některých případech je pro řízení krokového motoru vhodné využít tzv. closed loop systém. Pokuste se zjistit, jak tento systém řízení krokového motoru funguje a stručně jej popište spolu s nezbytnými komponenty které jsou potřeba.

   Closed loop systém[1] je systém řízení krokového motoru, ve kterém je mimo krokového motoru použit i senzor aktuální polohy motoru (enkodér). Daný senzor je tu jak pro ujištění, že se opravdu motor pohybuje podle zaslaných příkazů, tak z důvodu rozšířených možností uplatnění daného motoru. Přidáním senzoru otáčení se totiž stane z motoru krokového, motor hybridní, to jest motor krokový opatřený o prvky klasického servo motoru.

   Výhody použití korkového motoru se zavřenou smyčkou namísto servo motoru jsou pak: vysoký kroutící moment v malých rychlostech, cena motoru a autokorekce.

   Nezbytné komponenty:

   - Krokový motor
   - Enkodér či zařízení poskytující jakoukoliv jinou zpětnou vazbu
   - Driver motoru

2. Představte si svislé polohování frézy/vrtáku CNC stroje s použitím šroubu se stoupáním závitu 1mm a krokového motoru NEMA HT24-105. Pokuste se odvodit nejmenší posun nástroje při použití jednoduchého krokování.

   Vzhledem k tomu, že by frézka stoupala/klesla o 1mm při otočení závitu o 360 stupňů, dá se minimální posuv spočítat za pomocí úhlu, o který se posune rotor daného krokového motoru při jednom kroku. Ten je roven 1,8 stupňům[2] u konkrétního modelu motoru. Z toho pak vyplývá:

$$posun = Stoupání \cdot \frac{úhel\ kroku\ motoru}{360^o} = 1 \cdot \frac{1{,}8^o}{360^o} = 0{,}005\ mm$$

3. V některých případech není instalace spínačů nulových poloh žádoucí či možná. Pokuste se najít a vysvětlit, jak funguje tzv. Switchless endstop v případě driverů krokových motorů od společnosti Trinamic (např. TMC2209).

   Switchless endstop je koncept, kdy k inicializaci krokových motorů není třeba fyzických switchů. Konkrétní implementace od firmy Trinamic se nazývá StallGuard[3] a je založena na sledování elektrického odporu motoru během vykonávání pohybu. V praxi to znamená, že inicializace motoru probíhá najetím na fyzický doraz, jež díky přetlačování motoru a konstrukce vyústí ve zvýšeném odporu, který je řídící jednotkou vyhodnocen jako počáteční bod daného motoru.

Závěr:

Výsledkem řešení této úlohy je plně funkční obslužný program pro model robotické ruky, který nejenže splňuje základní kritéria zadání, ale je navíc i opatřen funkcionalitou 3 různých rychlostních stupňů, které přidají ovládání robota ten správný drive. I přes to má toto řešení pár nedostatků. Hlavním z nich je nemožnost uložení provedených pohybů do vlastního souboru. Důvodem je chyba paměťového charakteru, kdy se při odkazování na objekt uchovávající aktuální soubor, objekt jeví jako anulovaný, ač se k němu od inicializace nepřistupovalo.

---

[1] https://en.nanotec.com/knowledge-base-article/closed-loop-technology

[2] https://www.applied-motion.com/s/product/step-motor-high-torqueht24105/01t5i000000xzFDAAY?name=HT24-105-NEMA-24-High-Torque-Stepper-Motor

[3] https://www.analog.com/media/en/technical-documentation/data-sheets/TMC5031_datasheet_rev1.14.pdf