PRAXE

č. 1	STŘEDNÍ PRŮMYSLOVÁ ŠKOLA CHOMUTOV	David Herko
10. 12. 2024	ROBOT	V4

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 (zamyslete se nad efektivností zvoleného formátu)

- Zopakování manuálně naučeného pohybu uloženého v souboru

- Průběžná a použitelná indikace aktuální operace a stavu zařízení na monitoru

Teorie

Robot je napájen +5V/0,5A pro logickou část a +12/1,5A pro motory. Maximální taktovací kmitočet je 450Hz, což odpovídá dvěma milisekundám. Motory jsou aktivní v logické 0, směr otáčení se nastavuje zvlášť a je jiný u každého motoru (viz tabulka níže). Pokud je pozice na IR závoře, bude aktivní logická 0. Pokud jsou všechny závory v nule, je robot ve výchozím stavu.

SMĚR OTÁČENÍ	**Log. 1**	**Log. 0**
Horizontální otáčení	Po směru hodinových ručiček	Proti směru hodinových ručiček
Hlavní rameno	dolů	nahoru
Chapadlo	zavřít	otevřít
Rameno s chapadlem	nahoru	dolů

Popis řešení

Po zapnutí se program připojí k PCI desce. Robot se přesune do výchozí polohy: hýbe s částmi, dokud se neaktivuje IR závora. Otáčení základny nejprve zkusí jet 90 stupňů po směru hodinových ručiček, a následně proti směru dokud se neaktivuje závora. Po načtení výchozí pozice se otevře grafické rozhraní s tlačítky pro pohyb.

Uživatel může nastavit počet kroků pomocí `NumericUpDown` prvku. Po zmáčknutí jakéhokoliv tlačítka na pohyb se nastaví jedna z proměnných určující který pohyb a jakým směrem se má vykonat (viz tabulka níže). Nastaví se status text, a tikne se hodinami pomocí voidu `clockTick()`. Ten pomocí smyčky a nastavených proměnných volá voidy v classe `RobotController`, čímž se vykonají jednotlivé akce. Po skončení smyčky a příkazů se proměnné a status text resetují. Robot následně čeká na další příkazy.

Pokud chce uživatel vykonat akce znovu, může je uložit jako soubor s příkazy (formát v podstatě CSV, jenom používá jinou příponu, aby uživatel neotevřel tabulku) a ten následně načíst.

Rozbor proměnných a metod

Form1.cs - hlavní okno a část programu

`int rotate`	Žádaný směr otáčení základny
`int mainArm`	Žádaný směr hlavního ramene
`int secondArm`	Žádaný směr ramene s chapadlem

`int grabber`	Žádaný směr chapadla
`String pastActions`	Ukládá předchozí akce pro jejich znovu spuštění
`Form1()`	Inicializace
`void moveToStart()`	Robot se přesune na výchozí pozici
`void clockTick(decimal steps)`	Tikne hodinami [steps] krát
`void setStatus(String text)`	Nastaví status text

`btnTurnClockwise_MouseDown(...)`
`btnTurnCounter_MouseDown(...)`
`btnTurn_MouseUp(...)`
	Tlačítka pro otáčení základny

`btnMainUp_MouseDown(...)`
`btnMainDown_MouseDown(...)`
`btnMain_MouseUp(...)`
	Tlačítka pro hlavní rameno

`btnSecUp_MouseDown(...)`
`btnSecDown_MouseDown(...)`
`btnSec_MouseUp(...)`
	Tlačítka pro rameno s chapadlem

`btnGrabOpen_MouseDown(...)`
`btnGrabClose_MouseDown(...)`
`btnGrab_MouseUp(...)`
	Tlačítka pro chapadlo

`btnSave_Click(...)`	Tlačítko pro uložení do souboru
`btnLoad_Click(...)`	Tlačítko pro načtení ze souboru
`parseFile(String controlScript)`	Načítání ze souboru

RobotController.cs - ovládání robota samotného

`RobotController(Controller c)`	Konstruktor
`void setup()`	Nastaví výchozí hodnoty
`void turnClockwise()`	Otočí základnu po směru hodinových ručiček
`void turnCounter()`	Otočí základnu proti směru hodinových ručiček
`void turnStop()`	Zastaví otáčení
`void mainArmUp()`	Nahoru s hlavním ramenem
`void mainArmDown()`	Dolů s hlavním ramenem
`void mainArmStop()`	Zastaví hlavní rameno
`void secArmUp()`	Nahoru s ramenem s chapadlem

`void secArmDown()`	Dolů s ramenem s chapadlem
`void secArmStop()`	Zastaví rameno s chapadlem
`void grabberOpen()`	Otevírá chapadlo
`void grabberClose()`	Zavírá chapadlo
`void grabberStop()`	Zastaví chapadlo
`bool getTurnStop()`	Vrátí true, pokud se nachází na závoře otáčení
`bool getMainArmStop()`	Vrátí true, pokud se nachází na závoře hlavního ramene
`bool getSecArmStop()`	Vrátí true, pokud se nachází na závoře ramene s chapadlem
`bool getGrabberStop()`	Vrátí true, pokud se nachází na závoře chapadla

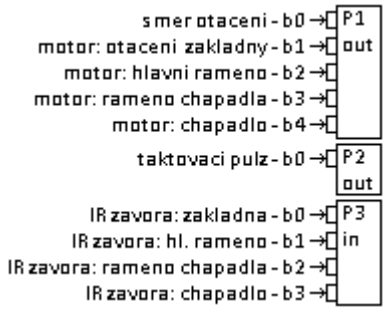
Controller.cs - komunikace s periferií

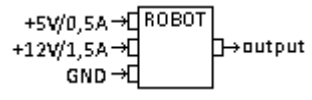
`InstantDoCtrl ioDO` `InstantDiCtrl ioDI`	Ovládací deska
`byte[] portData`	Data na portech

`public int port1out = 0;`
`public int port2out = 1;`
`public int port3in = 0;`
`public int port4in = 1;`
	Čísla portů

`Controller(InstantDoCtrl ioDO, InstantDiCtrl ioDI)`	Konstruktor
`void write(int port, byte input)`	Psaní do portů
`byte read(int port)`	Čtení z portů

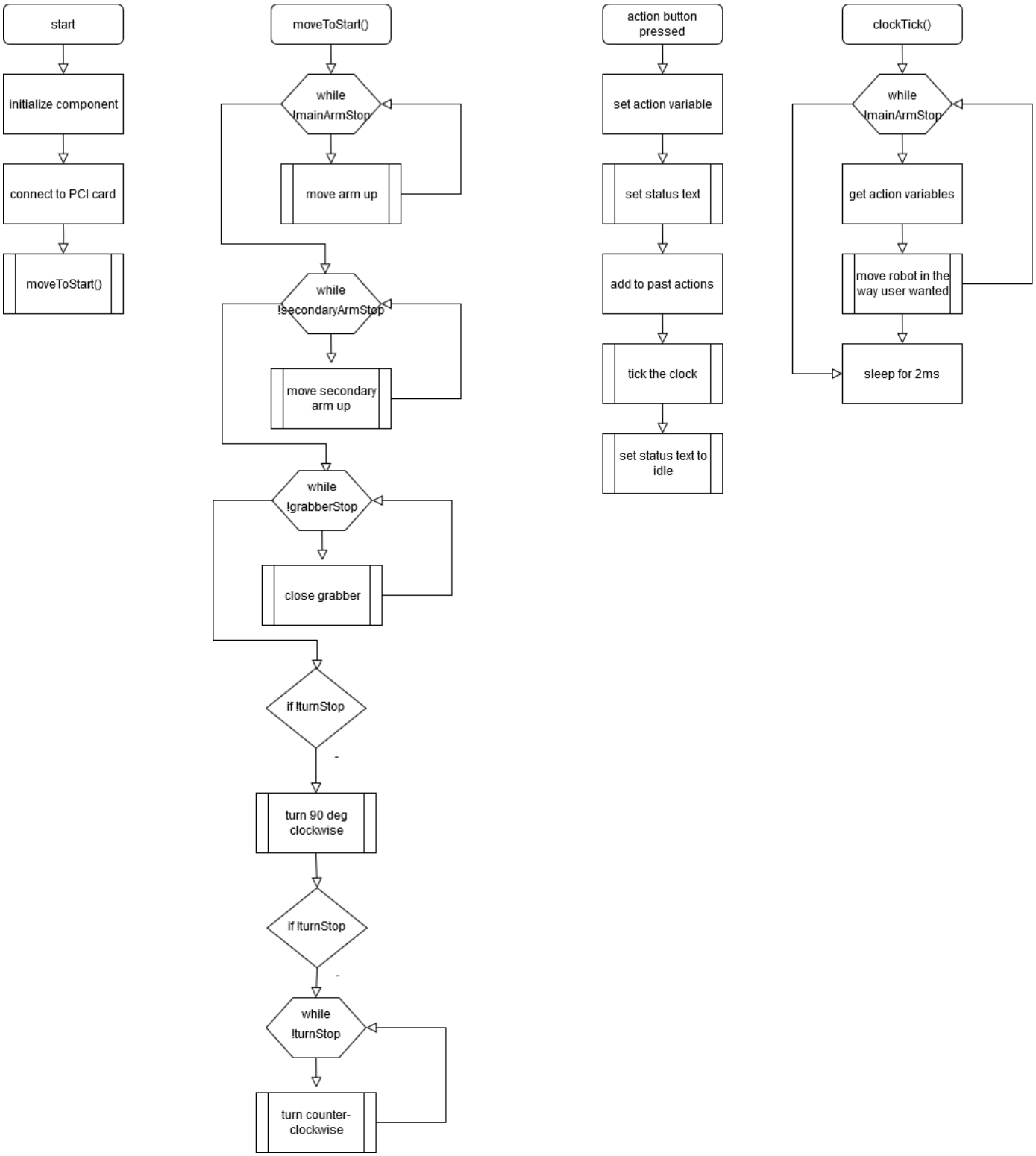
Schéma zapojení





Vývojový diagram

Form1.cs



Komentovaný výpis kódu

Form1.cs

```
using System;
using System.IO;
using System.Linq;
using System.Threadi    ng;
using System.Windows.Forms;
using Automation.BDaq;
using Dejvoss.BitShift;

namespace Herko.Robot
{
    public partial class Form1 : Form
    {
        public Controller c = null;
        public RobotController rc = null;

        //variables from which the loop knows what to do
        public int rotate = -1;
        public int mainArm = -1;
        public int secondArm = -1;
        public int grabber = -1;

        //saving past commands for redoing them
        public String pastActions = "";

        public Form1()
        {
            InitializeComponent();

            //window title
            this.Text = "Robot controller v0.3-dev | (c) DEJVOSS Productions 2024";

            //window style
            this.MaximizeBox = false;
            this.MinimizeBox = false;
            this.FormBorderStyle = FormBorderStyle.FixedSingle;

            //device info
            DeviceInformation di = new DeviceInformation();
            di.Description = "PCI-1756,BID#0";
            di.DeviceMode = AccessMode.ModeWrite;

            InstantDoCtrl ioDO = new InstantDoCtrl();
            ioDO.SelectedDevice = di;
            ioDO.LoadProfile("profile.xml");

            InstantDiCtrl ioDI = new InstantDiCtrl();
            ioDI.SelectedDevice = di;
            ioDI.LoadProfile("profile.xml");

            //controller
            c = new Controller(ioDO, ioDI);
            rc = new RobotController(c);

            //setup
```

```
        rc.setup();
        moveToStart();

        setStatus($"IDLE");
    }

    //move to default position
    public void moveToStart()
    {
        while (!rc.getMainArmStop()) //main arm
        {
            mainArm = 0;
            clockTick(1);
        }
        mainArm = -1;

        while (!rc.getSecArmStop()) //secondary arm
        {
            secondArm = 1;
            clockTick(1);
        }
        secondArm = -1;

        while (!rc.getGrabberStop()) //grabber arm
        {
            grabber = 0;
            clockTick(1);
        }
        grabber = -1;

        //rotation
        if (!rc.getTurnStop())
        {
            //try moving 90 degrees clockwise
            rotate = 1;
            for (int i = 0; i < 900; i++)
            {
                if (!rc.getTurnStop())
                    clockTick(1);
                else
                    break;
            }

            //if didn't hit -> move counterclockwise until it hits the mark
            if (!rc.getTurnStop())
            {
                while (!rc.getTurnStop())
                {
                    rotate = 0;
                    clockTick(1);
                }
            }
        }
        rotate = -1;
    }
    //tick the clock and call commands
    void clockTick(decimal steps)
    {
```

```
            for (int i = 0; i < steps; i++)
            {
                //clock
                c.write(c.port2out, EightBit.setAt(c.portData[c.port2out], 0, 1));
                Thread.Sleep(2);

                //rotation
                if (rotate == 1)
                {
                    rc.turnClockwise();
                }
                else if (rotate == 0)
                {
                    rc.turnCounter();
                }
                else if (rotate == -1)
                {
                    rc.turnStop();
                }

                //main arm movement
                if (mainArm == 1)
                {
                    rc.mainArmDown();
                }
                else if (mainArm == 0)
                {
                    if (rc.getMainArmStop() == false)
                    {
                        rc.mainArmUp();
                    }
                    else
                    {
                        setStatus("MAIN ARM HIT STOP MARK");
                        rc.mainArmStop();
                        break;
                    }
                }
                else if (mainArm == -1)
                {
                    rc.mainArmStop();
                }

                //secondary arm movement
                if (secondArm == 1)
                {
                    if (rc.getSecArmStop() == false)
                    {
                        rc.secArmUp();
                    }
                    else
                    {
                        setStatus("SECOND ARM HIT STOP MARK");
                        rc.secArmStop();
                        break;
                    }
                }
                else if (secondArm == 0)
```

```
            {
                rc.secArmDown();
            }
            else if (secondArm == -1)
            {
                rc.secArmStop();
            }

            //grabber movement
            if (grabber == 1)
            {
                rc.grabberClose();
            }
            else if (grabber == 0)
            {
                if (rc.getGrabberStop() == false)
                {
                    rc.grabberOpen();
                }
                else
                {
                    setStatus("GRABBER HIT STOP MARK");
                    rc.grabberStop();
                    break;
                }
            }
            else if (grabber == -1)
            {
                rc.grabberStop();
            }

            //clock 2 electric boogaloo set back
            c.write(c.port2out, EightBit.setAt(c.portData[c.port2out], 0, 0));
            Thread.Sleep(2);
        }
    }

    //sets robot status text
    private void setStatus(String text)
    {
        lblStatus.Text = $"STATUS: {text}";
        lblStatus.Update();
    }

    //BELOW: pressing buttons sets variables and runs clock
    private void btnTurnClockwise_MouseDown(object sender, MouseEventArgs e)
    {
        rotate = 1;
        setStatus("TURNING CLOCKWISE");
        pastActions += $"ROT;1;{numSteps.Value}\n";
        clockTick(numSteps.Value);
        setStatus($"IDLE");
    }

    private void btnTurnCounter_MouseDown(object sender, MouseEventArgs e)
    {
        rotate = 0;
        setStatus("TURNING COUNTER-CLOCKWISE");
```

```
            pastActions += $"ROT;0;{numSteps.Value}\n";
            clockTick(numSteps.Value);
            setStatus($"IDLE");
        }

        private void btnTurn_MouseUp(object sender, MouseEventArgs e)
        {
            rotate = -1;
        }

        private void btnMainUp_MouseDown(object sender, MouseEventArgs e)
        {
            mainArm = 0;
            setStatus("MAIN ARM UP");
            pastActions += $"MAA;0;{numSteps.Value}\n";
            clockTick(numSteps.Value);
            setStatus($"IDLE");
        }

        private void btnMainDown_MouseDown(object sender, MouseEventArgs e)
        {
            mainArm = 1;
            setStatus("MAIN ARM DOWN");
            pastActions += $"MAA;1;{numSteps.Value}\n";
            clockTick(numSteps.Value);
            setStatus($"IDLE");
        }

        private void btnMain_MouseUp(object sender, MouseEventArgs e)
        {
            mainArm = -1;
        }

        private void btnSecUp_MouseDown(object sender, MouseEventArgs e)
        {
            secondArm = 1;
            setStatus("SECOND ARM UP");
            pastActions += $"SEA;1;{numSteps.Value}\n";
            clockTick(numSteps.Value);
            setStatus($"IDLE");
        }

        private void btnSecDown_MouseDown(object sender, MouseEventArgs e)
        {
            secondArm = 0;
            setStatus("SECOND ARM DOWN");
            pastActions += $"SEA;0;{numSteps.Value}\n";
            clockTick(numSteps.Value);
            setStatus($"IDLE");
        }

        private void btnSec_MouseUp(object sender, MouseEventArgs e)
        {
            secondArm = -1;
        }

        private void btnGrabOpen_MouseDown(object sender, MouseEventArgs e)
        {
```

```
        grabber = 0;
        setStatus("GRABBER OPENING");
        pastActions += $"GRB;0;{numSteps.Value}\n";
        clockTick(numSteps.Value);
        setStatus($"IDLE");
    }

    private void btnGrabClose_MouseDown(object sender, MouseEventArgs e)
    {
        grabber = 1;
        setStatus("GRABBER CLOSING");
        pastActions += $"GRB;1;{numSteps.Value}\n";
        clockTick(numSteps.Value);
        setStatus($"IDLE");
    }

    private void btnGrab_MouseUp(object sender, MouseEventArgs e)
    {
        grabber = -1;
    }

    //save past commands
    private void btnSave_Click(object sender, EventArgs e)
    {
        SaveFileDialog sfd = new SaveFileDialog();
        sfd.Filter = "Robot controller script (*.rcs)|*.rcs|All files (*.*)|*.*";
        if (sfd.ShowDialog() == DialogResult.OK)
        {
            File.WriteAllText(sfd.FileName, pastActions);
        }
    }

    //load commands from file
    private void btnLoad_Click(object sender, EventArgs e)
    {
        OpenFileDialog ofd = new OpenFileDialog();
        ofd.Filter = "Robot controller script (*.rcs)|*.rcs|All files (*.*)|*.*";
        if (ofd.ShowDialog() == DialogResult.OK)
        {
            parseFile(File.ReadAllText(ofd.FileName));
        }
    }

    //parses command script file and executes it
    public void parseFile(String controlScript)
    {
        setStatus($"LOADING SCRIPT");

        pastActions += controlScript;
        String[] cmds = controlScript.Split('\n');
        int count = 0;
        foreach (String cmd in cmds)
        {
            //set status
            setStatus($"RUNNING SCRIPT {count + 1}/{cmds.Count() - 1}");

            //get variables
            String[] vars = cmd.Split(';');
```

```
                if (vars.Count() == 3)
                {
                    String type = vars[0];
                    int dir = int.Parse(vars[1]);
                    int steps = int.Parse(vars[2]);

                    //set items to direction
                    if (type == "ROT")
                        rotate = dir;
                    else if (type == "MAA")
                        mainArm = dir;
                    else if (type == "SEA")
                        secondArm = dir;
                    else if (type == "GRB")
                        grabber = dir;

                    //tick
                    clockTick(steps);

                    //reset everything
                    rotate = -1;
                    mainArm = -1;
                    secondArm = -1;
                    grabber = -1;

                    //update script count
                    count++;
                }
            }

            setStatus($"IDLE - SCRIPT FINISHED");
        }
    }
}
```

RobotController.cs

```
using Dejvoss.BitShift;

namespace Herko.Robot
{
    public class RobotController
    {
        public Controller c;

        public RobotController(Controller c)
        {
            this.c = c;
        }

        //sets default values on card (everything is stopped)
        public void setup()
        {
            c.write(c.port1out, 0b11111111);
            c.write(c.port2out, 0b11111111);
        }
```

```
        //turns robot clockwise
        public void turnClockwise()
        {
            c.write(c.port1out, EightBit.setAt(c.portData[c.port1out], 0, 1)); //turn
clockwise
            c.write(c.port1out, EightBit.setAt(c.portData[c.port1out], 1, 0)); //start
motor
        }

        //turns robot counter clockwise
        public void turnCounter()
        {
            c.write(c.port1out, EightBit.setAt(c.portData[c.port1out], 0, 0)); //turn
counter clockwise
            c.write(c.port1out, EightBit.setAt(c.portData[c.port1out], 1, 0)); //start
motor
        }

        //stops robot from turning
        public void turnStop()
        {
            c.write(c.port1out, EightBit.setAt(c.portData[c.port1out], 1, 1)); //stop
motor
        }

        //moves main arm up
        public void mainArmUp()
        {
            c.write(c.port1out, EightBit.setAt(c.portData[c.port1out], 0, 0)); //move
up
            c.write(c.port1out, EightBit.setAt(c.portData[c.port1out], 2, 0)); //start
motor
        }

        //moves main arm down
        public void mainArmDown()
        {
            c.write(c.port1out, EightBit.setAt(c.portData[c.port1out], 0, 1)); //move
down
            c.write(c.port1out, EightBit.setAt(c.portData[c.port1out], 2, 0)); //start
motor
        }

        //stops robot from moving main arm
        public void mainArmStop()
        {
            c.write(c.port1out, EightBit.setAt(c.portData[c.port1out], 2, 1)); //stop
motor
        }

        //moves secondary arm up
        public void secArmUp()
        {
            c.write(c.port1out, EightBit.setAt(c.portData[c.port1out], 0, 1)); //move
up
            c.write(c.port1out, EightBit.setAt(c.portData[c.port1out], 3, 0)); //start
motor
        }
```

```
        //moves secondary arm down
        public void secArmDown()
        {
            c.write(c.port1out, EightBit.setAt(c.portData[c.port1out], 0, 0)); //move
down
            c.write(c.port1out, EightBit.setAt(c.portData[c.port1out], 3, 0)); //start
motor
        }

        //stops secondary arm
        public void secArmStop()
        {
            c.write(c.port1out, EightBit.setAt(c.portData[c.port1out], 3, 1)); //stop
motor
        }

        //opens grabber
        public void grabberOpen()
        {
            c.write(c.port1out, EightBit.setAt(c.portData[c.port1out], 0, 0)); //open
grabber
            c.write(c.port1out, EightBit.setAt(c.portData[c.port1out], 4, 0)); //start
motor
        }

        //closes grabber
        public void grabberClose()
        {
            c.write(c.port1out, EightBit.setAt(c.portData[c.port1out], 0, 1)); //close
grabber
            c.write(c.port1out, EightBit.setAt(c.portData[c.port1out], 4, 0)); //start
motor
        }

        //stops grabber
        public void grabberStop()
        {
            c.write(c.port1out, EightBit.setAt(c.portData[c.port1out], 4, 1)); //stop
motor
        }

        //gets turning IR stop; true if reached stopped
        public bool getTurnStop()
        {
            if (EightBit.getAt(c.read(c.port3in), 0) == 1)
                return false;
            else
                return true;
        }

        //gets main arm IR stop; true if reached stopped
        public bool getMainArmStop()
        {
            if (EightBit.getAt(c.read(c.port3in), 1) == 1)
                return false;
            else
                return true;
```

```
        }

        //gets secondary arm IR stop; true if reached stopped
        public bool getSecArmStop()
        {
            if (EightBit.getAt(c.read(c.port3in), 2) == 1)
                return false;
            else
                return true;
        }

        //gets grabber arm IR stop; true if reached stopped
        public bool getGrabberStop()
        {
            if (EightBit.getAt(c.read(c.port3in), 3) == 1)
                return false;
            else
                return true;
        }
    }
}
```

Controller.cs

```
using Automation.BDaq;

namespace Herko.Robot
{
    public class Controller
    {
        //controllers
        public InstantDoCtrl ioDO;
        public InstantDiCtrl ioDI;

        //bytes to work with
        public byte[] portData = new byte[4];

        //ports
        public int port1out = 0;
        public int port2out = 1;
        public int port3in = 0;
        public int port4in = 1;

        //construct
        public Controller(InstantDoCtrl ioDO, InstantDiCtrl ioDI)
        {
            this.ioDO = ioDO;
            this.ioDI = ioDI;
        }

        //write [input] to [port]
        public void write(int port, byte input)
        {
            ioDO.Write(port, input);
            portData[port] = input;
        }

        //reads byte at [port]
```

```
        public byte read(int port)
        {
            byte data;
            ioDI.Read(port, out data);

            return data;
        }
    }
}
```

Odpovědi na otázky

1) Pro ovládání krokových motorů je dnes obvyklé používat tzv. drivery krokových motorů. Pokuste se stručně vysvětlit funkcionalitu takového obvodu a pro konkrétní typ popište jeho ovládací rozhraní ze strany počítače / MCU.

> Driver generuje pulzy pro motor, které budí jednotlivé části motoru. Také se mu říká budič.[1] Ovládají se například pomocí USB rozhraní.[2]

2) Některé krokové motory jsou vybaveny tzv. enkodérem. Co to je a o jakou vlastnost do řízení zařízení přináší?

> Enkodér vysílá signály do řídící jednotky o tom, jestli pohyb, který jsme chtěli proběhl správně. Zvyšuje se tím přesnost stroje.[3]

3) Představte si svislé polohování frézy/vrtáku CNC stroje s použitím šroubu se stoupáním závitu 1,5mm a krokového motoru NEMA HT17-269. Pokuste se odvodit nejmenší posun nástroje při použití jednoduchého krokování. (Tip: nakreslete si obrázek a podívejte se základní parametry daného motoru v jeho specifikaci / datasheetu).

> Jestli jsem to správně pochopil, mělo by to být 1,8 stupně, což je uvedeno ve specifikaci.[4]

Závěr

Nestihl jsem dodělat část ovládání robota pomocí ovladače, jinak všechno běží tak, jak má. Možná bych trochu zlepšil GUI pro ovládání, ale nejsem si úplně jist jak, a tohle funguje dostatečně.

---

1 https://dratek.cz/571-drivery-krokovych-motoru/

2 http://www.trestik.cz/rizeni-krokovych-motoru-pomoci-usb-rozhrani-s-moznosti-rucniho-ovladani

3 https://www.hennlich.cz/lin-tech/krokovy-motor-s-metrickym-konektorem-a-enkoderem/

4 https://eu.mouser.com/ProductDetail/Applied-Motion/HT17-269?qs=gYO9MXSt8FPov9BRHibn7w%3D%3D