



**FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH** **ústav automatizace
TECHNOLOGIÍ** **a měřicí techniky**

LABORATORNÍ ÚLOHA

Závěrečný projekt

AUTOR 1: Jan Tomšej (256421)

AUTOR 2: - (-)

ROČNÍK: 2

PŘEDMĚT: Prostředky průmyslové automatizace

DATUM: -

Zadání

Naším úkolem bylo zprovoznit řídicí systém na bázi PLC s vizualizací. Měli jsme k dispozici virtuální technologické zařízení FactorySim. Toto virtuální zařízení je simulací automatizovaného skladování – pokládá bedny s plechovkami na zadanou pozici v regálu. Pozice v regálu je zadávána prostřednictvím ID následujícím způsobem:

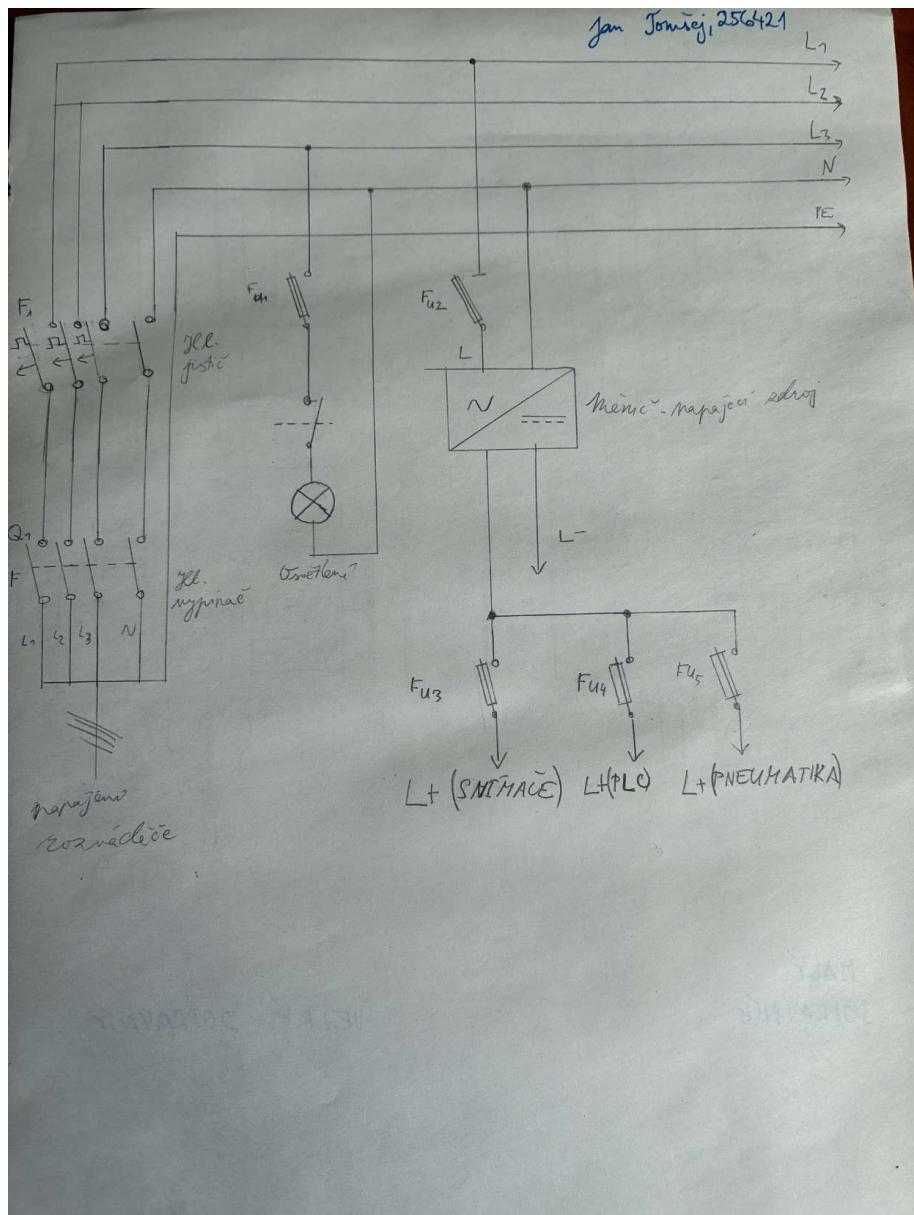
- Jednotlivé číslice odpovídají rádkům v regálu, tj. horní řádek odpovídá první číslici, spodní řádek odpovídá šesté číslici
- Sloupec odpovídá číslu na dané pozici ID, tj. pozice 0 odpovídá obsazení bednou v prvním sloupečku, 9.

K tomu musíme implementovat:

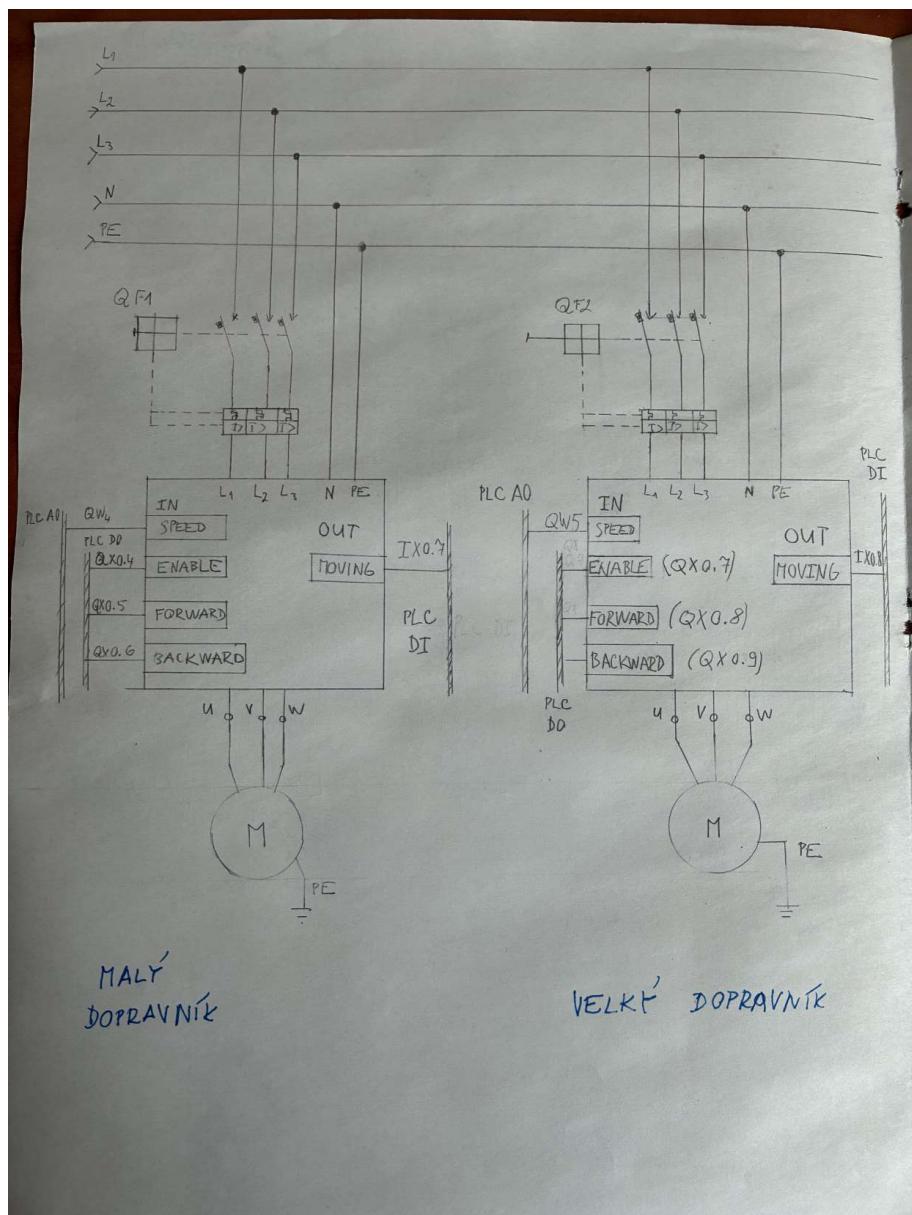
- Schéma elektrického zapojení řídího systému, který obsahuje PLC, HMI, průmyslový napájecí zdroj, frekvenční měniče s příslušnými ochranami pro ovládání motorů, asynchronní motory, pneumatické válce ovládané pomocí ventilů, světelné limitní senzory, robotické kontroléry, senzory polohy.
- PLC program - založení projektu, konfigurace komunikace s FactorySim, definice proměnných, tasky, programy, funkce, funkční bloky, uživatelské datové typy, režimy automat a manual (ovládání jednotlivých zařízení technologie)
- Vizualizace - sledujeme stav celého procesu (stav jednotlivých komponentů, provozní veličiny, vizualizace fáze cyklu, vizualizace režimu - automat/manual) a manuální ovládání technologie (přepnutí režimu automat/manual, start cyklu, manuální ovládání jednotlivých komponent).

Schéma zapojení

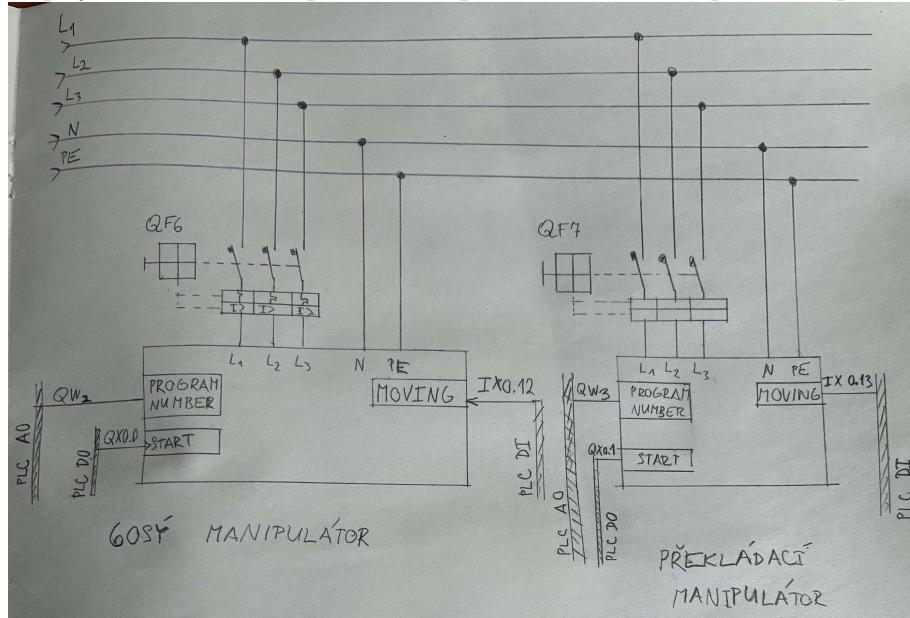
- Rozváděč je zapojený do sítě TN-S, jejíž jmenovité hodnoty jsou 230/400 V, 50 Hz. Proud hlavního jističe a jednotlivých pojistek je možné nadimenzovat, známe-li proudové odběry spotřebičů (především měniče).
- Rozváděč je osvětlen. Osvětlení se samo sputí, jakmile se dveře rozváděče otevřou.
- Měnič převádí síťové napětí buď na 24 V, nebo 12 V - závisí na výběru zařízení. Jeho schéma (včetně dalších zapojení) je připojeno níže.



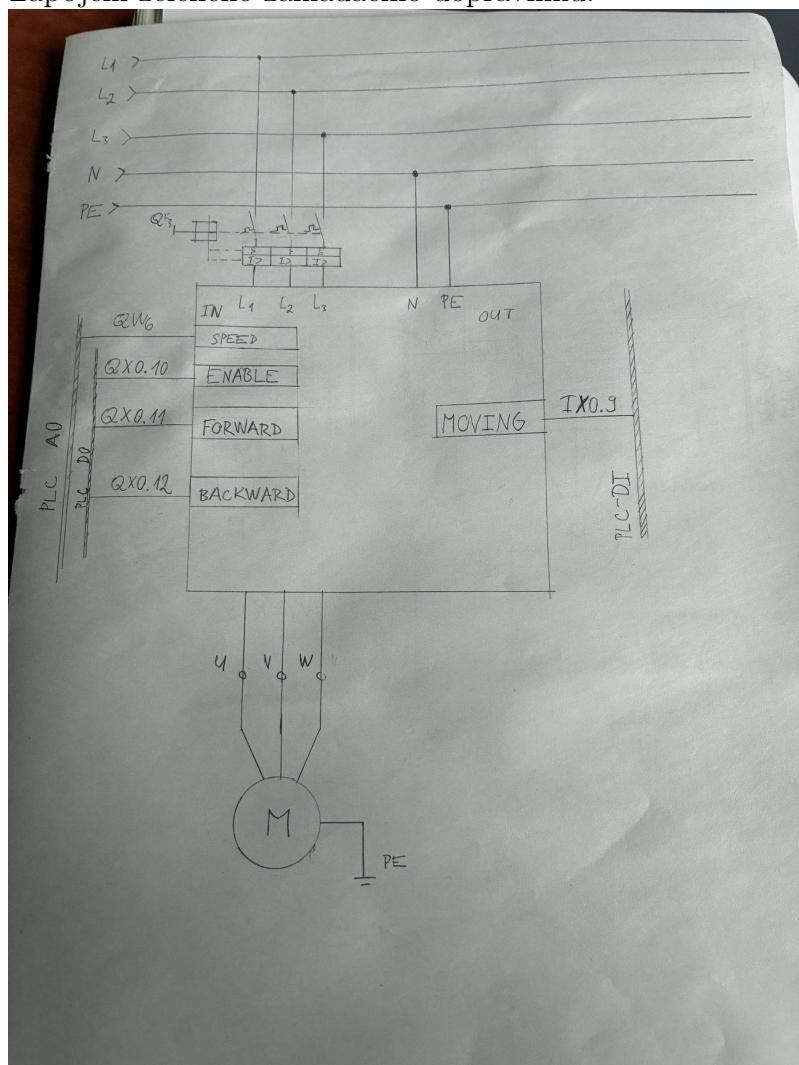
- Zapojení velkého dopravníku beden a malého dopravníku plechovek:



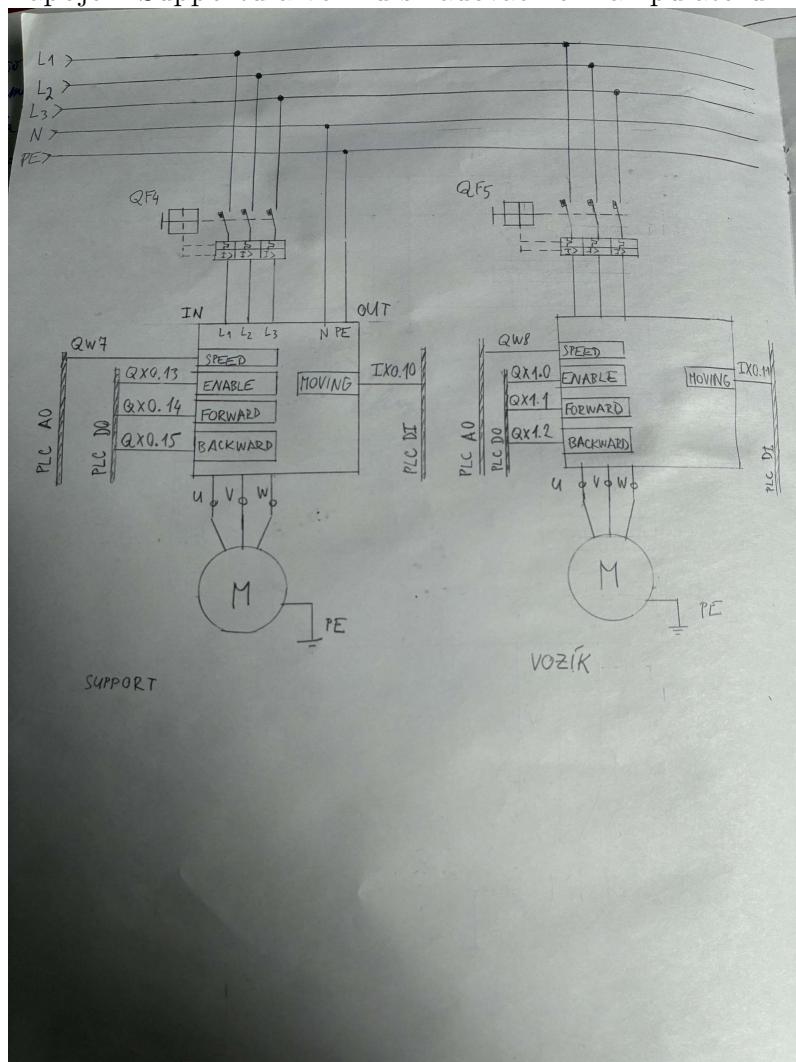
- Zapojení 6osého manipulátoru a překládacího manipulátoru plechovek:



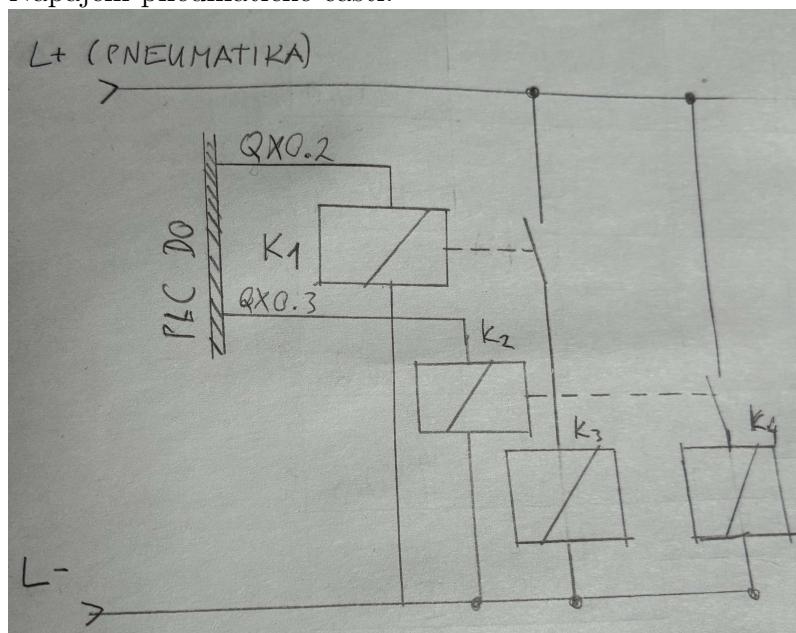
- Zapojení zeleného zakládacího dopravníku:



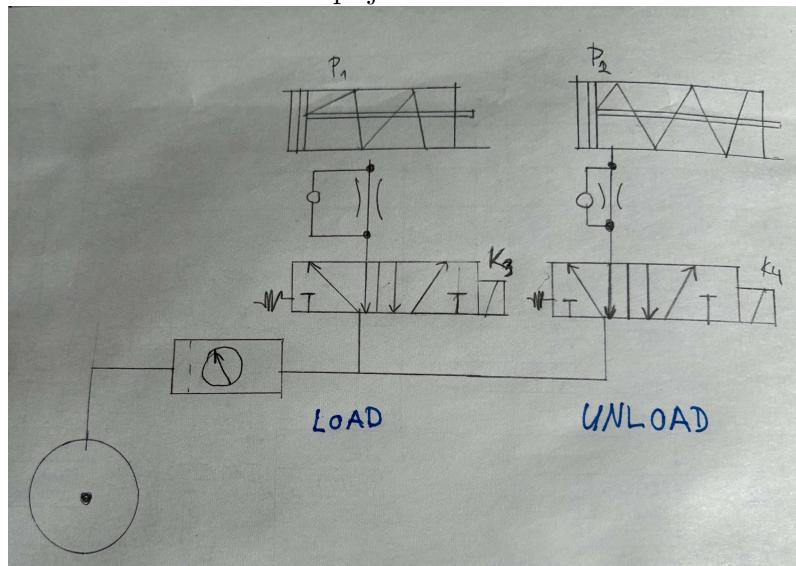
- Zapojení Supportu a vozíku skladovacího manipulátoru:



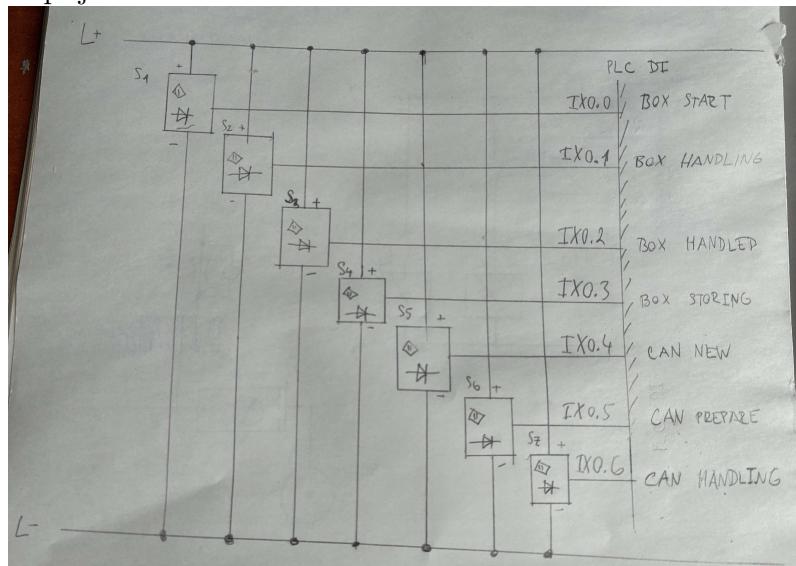
- Napájení pneumatické části:



- Pneumatické schéma zapojení:

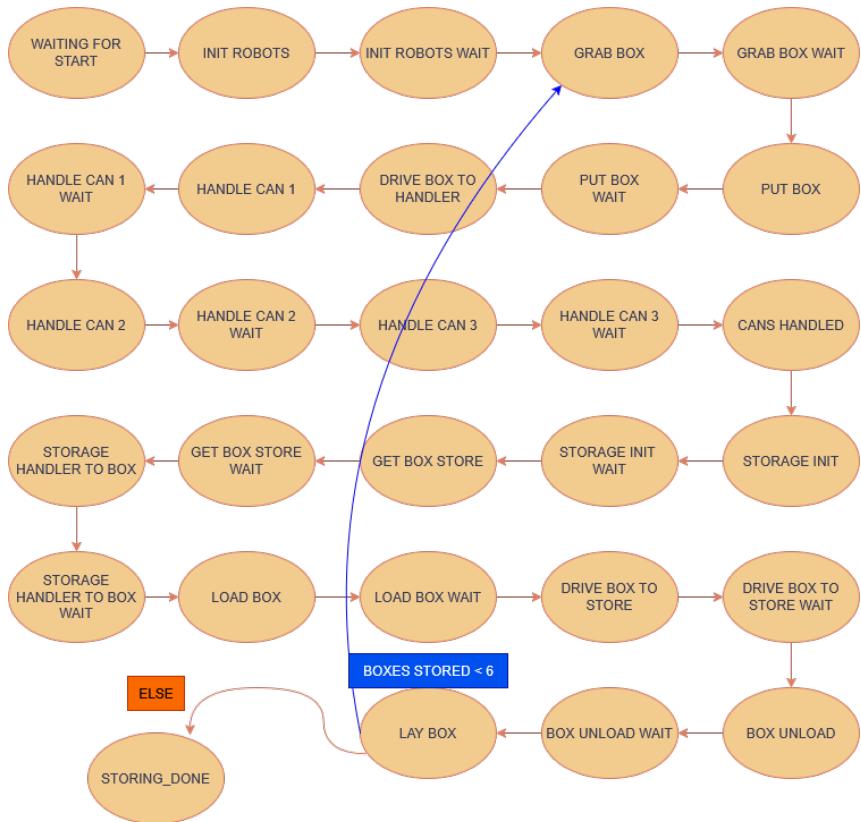


- Zapojení senzorů:



Stavový automat

- Celý níže zpracovaný program pro PLC je realizován jedním stavovým automatem, který představuje automatický režim. Z důvodu časové tísni nebyl manuální režim dopracován.



Obrázek 1: Stavový automat pro zadanou výrobní linku

- K popisu jednotlivých stavů:
 - WAITING FOR START: Čekání na to, než zmáčknut start tlačítko.
 - INIT ROBOTS: Přesuneme robotické rameno a handler plechovek do pozice 0.
 - INIT ROBOTS WAIT: Stav použitý kvůli časovači. Čekáme po danou dobu (u většiny WAITů po dobu 1 s, v případě čekání u zavádění krabice do).
 - GRAB BOX: Přepneme robotické rameno do stavu 1.
 - PUT BOX: Vezmeme robotickým ramenem krabici a přesuneme ji na velký pásový dopravník.
 - DRIVE BOX TO HANDLER: Přesuneme krabici pomocí pásového dopravníku přímo k handleru plechovek.
 - HANDLE CAN1,2,3: Postupně do krabice naskládáme první, druhou a na závěr třetí plechovku.

- CANS HANDLED: Jakmile jsou v krabici všechny tři plechovky, dostaneme krabici na konec zeleného zakládacího dopravníku.
- STORAGE INIT: Nastavíme souřadnice 2osého dopravníku na (0,0).
- GET BOX STORE: Najedu dvouosým dopravníkem na souřadnice přímo pod krabici.
- STORAGE HANDLER TO BOX: Vysunu píst LOAD, zvednu krabici do vzduchu.
- LOAD BOX: Píst s naskladněnou krabicí je zasunut.
- DRIVE BOX TO STORE: Krabice je zavezena na požadovanou pozici + offset (nelze srazit krabicí regál).
- BOX UNLOAD: Od pozice krabice je odečtený offset.
- LAY BOX: Zasuneme píst zpátky. Inkrementujeme proměnnou BoxesStored. Je-li výsledek menší než 6, přejdeme do stavu GRAB BOX. V opačném případě jsme ve finálním stavu STORING DONE.

Vlastní vypracování

```

PROGRAM MAIN
VAR CONSTANT
    //hlavní stavový automat
    WAITING_FOR_START : USINT := 0;
    INIT_ROBOTS : USINT := 1; // inicializace handlueru plechovek i velkeho robota
    INIT_ROBOTS_WAIT : USINT := 2; //cekaci stav, protoze pouzivam casovac
    GRAB_BOX : USINT := 3;
    GRAB_BOX_WAIT : USINT := 4;
    PUT_BOX : USINT := 5;
    PUT_BOX_WAIT : USINT := 6;
    DRIVE_BOX_TO_HANDLER : USINT := 7;
    HANDLE_CAN1 : USINT := 8;
    HANDLE_CAN1_WAIT : USINT := 9;
    HANDLE_CAN2 : USINT := 10;
    HANDLE_CAN2_WAIT : USINT := 11;
    HANDLE_CAN3 : USINT := 12;
    HANDLE_CAN3_WAIT : USINT := 13;
    CANS_HANDLER : USINT := 14;
    STORAGE_INIT : USINT := 15;
    STORAGE_INIT_WAIT : USINT := 16;
    GET_BOX_STORE : USINT := 17;
    GET_BOX_STORE_WAIT : USINT := 18;
    STORAGE_HANDLER_TO_BOX : USINT := 19;
    STORAGE_HANDLER_TO_BOX_WAIT: USINT := 20;
    LOAD_BOX : USINT := 21;
    LOAD_BOX_WAIT : USINT := 22;
    DRIVE_BOX_TO_STORE : USINT := 23;
    DRIVE_BOX_TO_STORE_WAIT : USINT := 24;
    BOX_UNLOAD : USINT := 25;
    BOX_UNLOAD_WAIT : USINT := 26;
    LAY_BOX : USINT := 27;
    STORING_DONE : USINT := 28;
END_VAR

VAR
    MainState : USINT := WAITING_FOR_START;
    StartProcess : BOOL := FALSE;
    CanSpeed : WORD := 1900;
    ConvSpeed : WORD := 2800;
    BoxAtTheEnd : BOOL := FALSE;
    TON_DelayCycle : TON;
    x : ARRAY [0..10] OF WORD := [25580, 23000, 20420, 17845, 15260, 12700, 10120, 7520, 4940, 2380, 750];
    y : ARRAY [0..5] OF WORD := [23750, 19142, 14535, 9927, 5318, 710];
    //FirstBoxDriven : BOOL := FALSE;
    BoxesStored : USINT := 0;
    ID : ARRAY [0..5] OF USINT := [2,5,6,4,2,1];
    P_RegulatorX : P_regulator;
    P_RegulatorY : P_regulator;
    yPosition : WORD := 0;
    xPosition : WORD := 0;
    RoboticArmStart : BOOL := FALSE;
    CentralStop : BOOL := FALSE;
END_VAR

```

Obrázek 2: Proměnné a konstanty, které jsou používány

```

IO * x
1 //attribute 'qualified_only'
2 VAR_GLOBAL
3   // Analog Inputs
4   StorageSupportPosition : WORD; // Pozice suportu skladovacího manipulátoru [0..27648]
5   StorageCarriagePosition : WORD; // Pozice vozíku skladovacího manipulátoru [0..27648]
6
7   // Digital Inputs
8   SensorBoxStart : BOOL; // Sensor přítomnosti bedny na vstupu velkého dopravníkového pásu [0/1]
9   SensorBoxHandling : BOOL; // Sensor přítomnosti bedny v pracovním prostoru překládacího manipulátoru [0/1]
10  SensorBoxHandled : BOOL; // Sensor přítomnosti bedny na konci velkého dopravníkového pásu [0/1]
11  SensorBoxStoring : BOOL; // Sensor přítomnosti nové plechovky na místě dopravníkovém pásu [0/1]
12  SensorCanNew : BOOL; // Sensor přítomnosti nové plechovky na místě dopravníkovém pásu [0/1]
13  SensorCanMoving : BOOL; // Sensor přítomnosti nové plechovky v přesunovacím prostoru překládacího manipulátoru [0/1]
14  ConvCanMoving : BOOL; // Signálizace pohybu motoru malého dopravníkového pásu [0/1]
15  ConvBoxMoving : BOOL; // Signálizace pohybu motoru základového dopravníkového pásu [0/1]
16  ConvLoadMoving : BOOL; // Signálizace pohybu motoru suportu skladovacího manipulátoru [0/1]
17  StorageSupportMoving : BOOL; // Signálizace pohybu motoru vozíku skladovacího manipulátoru [0/1]
18  StorageCarriageMoving : BOOL; // Signálizace pohybu motoru vozíku skladovacího manipulátoru [0/1]
19  RobotArmMoving : BOOL; // Signálizace vykonání programového cyklu 6-osého robottického manipulátoru [0/1]
20  HandlingMoving : BOOL; // Signálizace vykonání programového cyklu překládacího manipulátoru [0/1]
21
22
23   // Analog Outputs
24  RoboticArmProgram : WORD; // Číslo programu pro vykonání 6-osý manipulátorem [0 .. základní pozice, 1 .. uchopení bedny, 2 .. polcení bedny na pás]
25  HandlingProgram : WORD; // Číslo programu pro vykonání překládacím manipulátorem [0 .. základní pozice, 1 .. přenesení na první pozici, 2 .. přenesení na druhou pozici, 3 .. přenesení na třetí pozici]
26  ConvCanSpeed : WORD; // Rychlosť motoru malého dopravníku [0..27648]
27  ConvBoxSpeed : WORD; // Rychlosť motoru velkého dopravníku [0..27648]
28  ConvLoadSpeed : WORD; // Rychlosť motoru základového dopravníku [0..27648]
29  StorageSupportSpeed : WORD; // Rychlosť motoru suportu skladovacího manipulátoru [0..27648]
30  StorageCarriageSpeed : WORD; // Rychlosť motoru vozíku skladovacího manipulátoru [0..27648]
31
32   // Digital Outputs
33  RoboticArmStartCycle : BOOL; // Start programového cyklu 6-osého manipulátoru [nabídka hran]
34  HandlingStartCycle : BOOL; // Start programového cyklu překládacího manipulátoru [nabídka hran]
35  StorageLoad : BOOL; // Aktivace výlož (ventilu) plodiny skladovacího manipulátoru DO následkovací polohy [0/1]
36  StorageUnload : BOOL; // Aktivace výlož (ventilu) plodiny skladovacího manipulátoru prvyškladovací polohy [0/1]
37  ConvCanEnable : BOOL; // Povolení frekvenciho ménice motoru malého dopravníku [0/1]
38  ConvCanForward : BOOL; // Aktivace chodu motoru malého dopravníku směrem [0/1]
39  ConvCanbackward : BOOL; // Aktivace chodu motoru malého dopravníku směrem [0/1]
40  ConvBoxEnable : BOOL; // Povolení frekvenciho ménice motoru velkého dopravníku standardním směrem [0/1]
41  ConvBoxForward : BOOL; // Aktivace chodu motoru velkého dopravníku standardním směrem [0/1]
42  ConvBoxbackward : BOOL; // Aktivace chodu motoru velkého dopravníku různým směrem [0/1]
43  ConvLoadEnable : BOOL; // Povolení frekvenciho ménice motoru základového dopravníku [0/1]
44  ConvLoadForward : BOOL; // Aktivace chodu motoru základového dopravníku standardním směrem [0/1]
45  ConvLoadbackward : BOOL; // Aktivace chodu motoru základového dopravníku spětným směrem [0/1]
46  StorageSupportEnable : BOOL; // Povolení frekvenciho ménice motoru suportu skladovacího manipulátoru [0/1]
47  StorageSupportForward : BOOL; // Aktivace chodu motoru suportu skladovacího manipulátoru spětným směrem [0/1]
48  StorageSupportBackward : BOOL; // Aktivace chodu motoru suportu skladovacího manipulátoru spětným směrem [0/1]
49  StorageCarriageEnable : BOOL; // Povolení frekvenciho ménice motoru vozíku skladovacího manipulátoru [0/1]
50  StorageCarriageForward : BOOL; // Aktivace chodu motoru vozíku skladovacího manipulátoru standardním směrem [0/1]
51  StorageCarriageBackward : BOOL; // Aktivace chodu motoru vozíku skladovacího manipulátoru spětným směrem [0/1]
52
53 END_VAR

```

Obrázek 3: Globální proměnné, se kterými v programu pracujeme v programu

```

1
2  IOMap(); //namapuj vstupy a vystupy
3 CASE MainState OF
4   WAITING_FOR_START:
5     IF StartProcess THEN
6       MainState := INIT_ROBOTS;
7     END_IF
8
9   INIT_ROBOTS:
10    IO.HandlingProgram := IO.RoboticArmProgram := 0;
11    IO.HandlingStartCycle := IO.RoboticArmStartCycle := 1;
12    TON_DelayCycle(IN := TRUE, PT := T#1S);
13    IF TON_DelayCycle.Q THEN
14      TON_DelayCycle(IN := 0);
15      IF NOT IO.RoboticArmMoving AND NOT IO.HandlingMoving THEN
16        IO.HandlingStartCycle := IO.RoboticArmStartCycle := 0;
17        MainState := INIT_ROBOTS_WAIT;
18      END_IF
19    END_IF
20
21   INIT_ROBOTS_WAIT:
22    TON_DelayCycle(IN := TRUE, PT := T#1S);
23    IF TON_DelayCycle.Q THEN
24      TON_DelayCycle(IN := 0);
25      MainState := GRAB_BOX;
26    END_IF
27
28   GRAB_BOX:
29    IO.RoboticArmProgram := 1;
30    IO.RoboticArmStartCycle := 1;
31    TON_DelayCycle(IN := TRUE, PT := T#1S);
32    IF TON_DelayCycle.Q THEN
33      TON_DelayCycle(IN := 0);
34      IF NOT IO.RoboticArmMoving THEN
35        IO.RoboticArmStartCycle := 0;
36        MainState := GRAB_BOX_WAIT;
37      END_IF
38    END_IF
39
40   GRAB_BOX_WAIT:
41    TON_DelayCycle(IN := TRUE, PT := T#1S);
42    IF TON_DelayCycle.Q THEN
43      TON_DelayCycle(IN := 0);
44      MainState := PUT_BOX;
45    END_IF
46

```

Obrázek 4: První část kódu

- Jak je na obrázku 4 uvedeno, používáme v první části kódu funkci IOmap. Tato funkce mapuje vstupy a výstupy pomocí globálních proměnných, abychom je potom v mainu mohli snadno volat. Je uvedená níže.

```

1 // Analog Inputs
2 IO_StorageSupportPosition := GVL_mb_Output_Registers[1]; // Pojize supportskladovaciho manipulatoru [0..27648]
3 IO_StorageCarriagePosition := GVL_mb_Output_Registers[2]; // Pojize vosiku skladovaciho manipulatoru [0..27648]
4
5 // Digital Inputs
6 IO_SensorEndState := GVL_mb_Output_Registers[0].0; // Sensor pítomnosti bedy na výstupu velkého dopravníkova písma [0/1]
7 IO_SensorEndHandling := GVL_mb_Output_Registers[0].1; // Sensor pítomnosti bedy v procesním posuvnici pěšilidelského manipulátoru [0/1]
8 IO_SensorBoxHandled := GVL_mb_Output_Registers[0].2; // Sensor pítomnosti bedy na konci velkého dopravníkova písma [0/1]
9 IO_SensorBoxString := GVL_mb_Output_Registers[0].3; // Sensor pítomnosti nové plechovky na nazkladovaci pozici pěšilidelského dopravníku [0/1]
10 IO_SensorCanNew := GVL_mb_Output_Registers[0].4; // Sensor pítomnosti plechovky uprostřed malého dopravníkova písma [0/1]
11 IO_SensorCanPrepare := GVL_mb_Output_Registers[0].5; // Sensor pítomnosti plechovky na malém dopravníkova písma [0/1]
12 IO_SensorCanHandling := GVL_mb_Output_Registers[0].6; // Sensor pítomnosti plechovky v procesním prostoru pěšilidelského manipulátoru [0/1]
13 IO_ConvBoxMoving := GVL_mb_Output_Registers[0].7; // Signalaice pohybu motoru velkého dopravníkova písma [0/1]
14 IO_ConvLoadMoving := GVL_mb_Output_Registers[0].8; // Signalaice pohybu motoru nazkladovacího dopravníkova písma [0/1]
15 IO_StorageSupportMoving := GVL_mb_Output_Registers[0].9; // Signalaice pohybu motoru supportu skladovacieho manipulátoru [0/1]
16 IO_StorageCarriageMoving := GVL_mb_Output_Registers[0].10; // Signalaice pohybu motoru vosiku skladovacieho manipulátoru [0/1]
17 IO_RoboticsArmMoving := GVL_mb_Output_Registers[0].11; // Signalaice výkonání programového cyklu čtvrtého robotického manipulátoru [0/1]
18 IO_RoboticsArmNoMoving := GVL_mb_Output_Registers[0].12; // Signalaice výkonání programového cyklu pěšilidelského manipulátoru [0/1]
19 IO_HandlingMoving := GVL_mb_Output_Registers[0].13; // Signalaice výkonání programového cyklu pěšilidelského manipulátoru [0/1]
20
21 // Analog Outputs
22 GVL_mb_Input_Registers[2] := IO_RoboticsArmProgram; // Číslo programu pro rychouně 6-osým manipulátoru (0 ... sakklesní pozice, 1 ... uchopení bedny, 2 ... polození bedny na pís)
23 GVL_mb_Input_Registers[2] := IO_HandlingStartCycle; // Číslo programu pro rychouně pěšilidelském manipulátoru (nabízíme hranu)
24 GVL_mb_Input_Registers[4] := IO_ConvCanSpeed; // Rychlosť motoru malého dopravníku [0..27648]
25 GVL_mb_Input_Registers[5] := IO_ConvBoxSpeed; // Rychlosť motoru velkého dopravníku [0..27648]
26 GVL_mb_Input_Registers[6] := IO_ConvLoadSpeed; // Rychlosť motoru nazkladovacího dopravníku [0..27648]
27 GVL_mb_Input_Registers[7] := IO_StorageSupportSpeed; // Rychlosť motoru supportu skladovacieho manipulátoru [0..27648]
28 GVL_mb_Input_Registers[8] := IO_StorageCarriageSpeed; // Rychlosť motoru vosiku skladovacieho manipulátoru [0..27648]
29
30 // Digital Outputs
31 GVL_mb_Input_Registers[0].0 := IO_RoboticsArmStartCycle; // Start programového cyklu čtvrtého manipulátoru (nabízíme hranu)
32 GVL_mb_Input_Registers[0].1 := IO_HandlingStartCycle; // Start programového cyklu pěšilidelského manipulátoru (nabízíme hranu)
33 GVL_mb_Input_Registers[0].2 := IO_StorageLoad; // Aktivace výložky (rentílu) plôšiny skladovacieho manipulátoru DO nazkladovaci polohy [0/1]
34 GVL_mb_Input_Registers[0].3 := IO_StorageUnload; // Aktivace výložky (rentílu) plôšiny skladovacieho manipulátoru prvyklaďovaci polohy [0/1]
35 GVL_mb_Input_Registers[0].4 := IO_ConvCanForward; // Aktivace chodu motoru malého dopravníku standardním směrem [0/1]
36 GVL_mb_Input_Registers[0].5 := IO_ConvCanBackward; // Aktivace chodu motoru malého dopravníku opačným směrem [0/1]
37 GVL_mb_Input_Registers[0].6 := IO_ConvBoxForward; // Aktivace chodu motoru nazkladovacího dopravníku standardním směrem [0/1]
38 GVL_mb_Input_Registers[0].7 := IO_ConvBoxBackward; // Aktivace chodu motoru nazkladovacího dopravníku opačným směrem [0/1]
39 GVL_mb_Input_Registers[0].8 := IO_ConvBoxForward; // Aktivace chodu motoru velkého dopravníku standardním směrem [0/1]
40 GVL_mb_Input_Registers[0].9 := IO_ConvBoxBackward; // Aktivace chodu motoru velkého dopravníku opačným směrem [0/1]
41 GVL_mb_Input_Registers[0].10 := IO_ConvLoadEnable; // Aktivace chodu motoru nazkladovacího manipulátoru standardním směrem [0/1]
42 GVL_mb_Input_Registers[0].11 := IO_ConvLoadDisable; // Aktivace chodu motoru nazkladovacího manipulátoru opačným směrem [0/1]
43 GVL_mb_Input_Registers[0].12 := IO_ConvLoadForward; // Aktivace chodu motoru nazkladovacího manipulátoru standardním směrem [0/1]
44 GVL_mb_Input_Registers[0].13 := IO_ConvLoadBackward; // Aktivace chodu motoru nazkladovacího manipulátoru opačným směrem [0/1]
45 GVL_mb_Input_Registers[0].14 := IO_StorageSupportEnable; // Aktivace chodu motoru supportu skladovacieho manipulátoru standardním směrem [0/1]
46 GVL_mb_Input_Registers[0].15 := IO_StorageSupportForward; // Aktivace chodu motoru supportu skladovacieho manipulátoru standardním směrem [0/1]
47 GVL_mb_Input_Registers[0].16 := IO_StorageCarriageEnable; // Aktivace chodu motoru vosiku skladovacieho manipulátoru [0/1]
48 GVL_mb_Input_Registers[0].17 := IO_StorageCarriageForward; // Aktivace chodu motoru vosiku skladovacieho manipulátoru standardním směrem [0/1]
49 GVL_mb_Input_Registers[0].18 := IO_StorageCarriageBackward; // Aktivace chodu motoru vosiku skladovacieho manipulátoru opačným směrem [0/1]

```

Obrázek 5: Namapování vstupů a výstupů pomocí registů - globálních proměnných

```

1 VAR_GLOBAL
2   mb_Input_Coils : ARRAY [0..20] OF BOOL;
3   mb_Output_Coils : ARRAY [0..20] OF BOOL;
4   mb_Input_Registers : ARRAY [0..20] OF WORD;
5   mb_Output_Registers : ARRAY [0..20] OF WORD;
6 END_VAR

```

Obrázek 6: Deklarace registrů jakožto úložiště dat (prostředník mezi simulací a virtuálním PLC)

```

47     PUT_BOX:
48         IO.RoboticArmProgram := 2;
49         IO.RoboticArmStartCycle := 1;
50         TON_DelayCycle(IN := TRUE, PT := T#1S);
51         IF TON_DelayCycle.Q THEN
52             TON_DelayCycle(IN := 0);
53             IF NOT IO.RoboticArmMoving THEN
54                 IO.RoboticArmStartCycle := 0;
55                 MainState := PUT_BOX_WAIT;
56             END_IF
57         END_IF
58
59     PUT_BOX_WAIT:
60         TON_DelayCycle(IN := TRUE, PT := T#1S);
61         IF TON_DelayCycle.Q THEN
62             TON_DelayCycle(IN := 0);
63             MainState := DRIVE_BOX_TO_HANDLER;
64         END_IF
65
66     DRIVE_BOX_TO_HANDLER:
67
68         IF IO.SensorBoxHandling THEN
69             IO.ConvBoxEnable := 0;
70             MainState := HANDLE_CAN1;
71         ELSE
72             IO.ConvBoxEnable := 1;
73         END_IF
74
75
76     HANDLE_CAN1:
77         IO.HandlingProgram := 1;
78         IO.HandlingStartCycle := 1;
79         TON_DelayCycle(IN := TRUE, PT := T#1S);
80         IF TON_DelayCycle.Q THEN
81             TON_DelayCycle(IN := 0);
82             IF NOT IO.HandlingMoving THEN
83                 IO.HandlingStartCycle := 0;
84                 MainState := HANDLE_CAN1_WAIT;
85             END_IF
86         END_IF
87
88     HANDLE_CAN1_WAIT:
89         TON_DelayCycle(IN := TRUE, PT := T#1S);
90         IF TON_DelayCycle.Q THEN
91             TON_DelayCycle(IN := 0);
92             MainState := HANDLE_CAN2;
93         END_IF
94
95

```

Obrázek 7: Druhá část kódu

- Jak již bylo výše uvedeno, tak v každém z výše uvedených stavů pracujeme s časovačem. Jakmile se na výstupu časovače objeví impulz, ověřujeme, zda se daná komponenta ještě pohybuje, či nikoliv. Například pokud již uběhne daný čas 1 s a robot se nehýbe, pustíme sestupnou hranu.

```

  96      HANDLE_CAN2:
  97          TON_DelayCycle(IN := TRUE, PT := T#1S);
  98          IO.HandlingProgram := 2;
  99          IO.HandlingStartCycle := 1;
 100         IF TON_DelayCycle.Q THEN
 101             TON_DelayCycle(IN := 0);
 102             IF NOT IO.HandlingMoving THEN
 103                 IO.HandlingStartCycle := 0;
 104                 MainState := HANDLE_CAN2_WAIT;
 105             END_IF
 106         END_IF
 107
 108         HANDLE_CAN2_WAIT:
 109             TON_DelayCycle(IN := TRUE, PT := T#1S);
 110             IF TON_DelayCycle.Q THEN
 111                 TON_DelayCycle(IN := 0);
 112                 MainState := HANDLE_CAN3;
 113             END_IF
 114
 115         HANDLE_CAN3:
 116             TON_DelayCycle(IN := TRUE, PT := T#1S);
 117             IO.HandlingProgram := 3;
 118             IO.HandlingStartCycle := 1;
 119             IF TON_DelayCycle.Q THEN
 120                 TON_DelayCycle(IN := 0);
 121                 IF NOT IO.HandlingMoving THEN
 122                     IO.HandlingStartCycle := 0;
 123                     MainState := HANDLE_CAN3_WAIT;
 124                 END_IF
 125             END_IF
 126
 127         HANDLE_CAN3_WAIT:
 128             TON_DelayCycle(IN := TRUE, PT := T#1S);
 129             IF TON_DelayCycle.Q THEN
 130                 TON_DelayCycle(IN := 0);
 131                 MainState := CANS_HANDLED;
 132             END_IF
 133
 134         CANS_HANDLED:
 135             MainState := GET_BOX_STORE;
 136             IO.ConvBoxEnable := 1;
 137

```

Obrázek 8: Třetí část kódu

```

145
146     STORAGE_INIT:
147         TON_DelayCycle(IN := TRUE, PT := T#1S);
148
149         P_RegulatorX.DesiredPos := 750;
150         P_RegulatorY.DesiredPos := 1100;
151
152     IF TON_DelayCycle.Q THEN
153         TON_DelayCycle(IN := 0);
154         IF NOT IO.StorageCarriageMoving AND NOT IO.StorageSupportMoving THEN
155
156             MainState := STORAGE_INIT_WAIT;
157             END_IF
158
159     END_IF
160
161     STORAGE_INIT_WAIT:
162         TON_DelayCycle(IN := TRUE, PT := T#1S);
163         IF TON_DelayCycle.Q THEN
164             TON_DelayCycle(IN := 0);
165             MainState := GET_BOX_STORE;
166             END_IF
167

```

Obrázek 9: Čtvrtá část kódu

- Jak můžeme ve čtvrté části kódu vidět, pro inicializaci dvouosého dopravníku krabic do skladu voláme funkční blok. Žlutý, dvouosý skladovací manipulátor je typickou aplikací pro regulátor. Známe-li pozici v pixelech, kam se chceme dostat, regulujeme rychlosť manipulátoru v každé ose. Implementace tohoto "P regulátoru" je uvedena níže.

```
1  FUNCTION_BLOCK P_regulator
2  VAR CONSTANT
3      //FSM states
4      NOT_MOVING : USINT := 0;
5      MOVING : USINT := 1;
6
7      OUTER_POS_TOL : REAL := 10;
8      INNER_POS_TOL : REAL := 5;
9
10 END_VAR
11
12 VAR
13     MoveState : USINT := NOT_MOVING;
14
15     DesiredSpeed : REAL;
16     DecelerationRatio : REAL;
17
18 END_VAR
19
20 VAR_INPUT
21     CurrentPos : REAL;
22     SetpointPos : REAL;
23     MaxSpeed : REAL := 13000;
24     DecelerationLength : REAL := 18000;
25 END_VAR
26
27 VAR_OUTPUT
28     Speed : REAL;
29     MoveForward : BOOL;
30     MoveBackward : BOOL;
31 END_VAR
```

Obrázek 10: Deklarace funkčního bloku P regulátoru

```

1 CASE MoveState OF
2   NOT_MOVING:
3     IF SetpointPos - CurrentPos > OUTER_POS_TOL THEN
4       MoveForward := TRUE;
5       MoveBackward := FALSE;
6       MoveState := MOVING;
7     ELSIF CurrentPos - SetpointPos > OUTER_POS_TOL THEN
8       MoveForward := FALSE;
9       MoveBackward := TRUE;
10      MoveState := MOVING;
11    END_IF
12
13  MOVING:
14    IF NOT (MoveForward AND SetpointPos - CurrentPos < INNER_POS_TOL OR MoveBackward AND CurrentPos - SetpointPos < INNER_POS_TOL) THEN
15      IF DecelerationLength > 0 THEN
16        DecelerationRatio := ABS(SetpointPos - CurrentPos) / DecelerationLength;
17        IF DecelerationRatio > 1 THEN
18          DecelerationRatio := 1;
19        END_IF
20      ELSE
21        DecelerationRatio := 1;
22      END_IF
23      DesiredSpeed := MaxSpeed * DecelerationRatio;
24    ELSE
25      DesiredSpeed := 0;
26      MoveForward := MoveBackward := FALSE;
27      MoveState := NOT_MOVING;
28    END_IF
29
30  END_CASE
31  Speed := DesiredSpeed;
32
33

```

Obrázek 11: Implementace funkčního bloku P regulátoru

- Následující kapitola kódu ukazuje najetí 2osého skladovacího manipulátoru přímo pod krabici.

```

GET_BOX_STORE:
  IO.StorageCarriageEnable := IO.StorageSupportEnable := 1;
  IF IO.SensorBoxStoring THEN
    TON_DelayCycle(IN := TRUE, PT := T#1S);
    P_RegulatorX.SetpointPos := 750;
    P_RegulatorY.SetpointPos := 1100;
    IF TON_DelayCycle.Q THEN
      IF NOT IO.StorageSupportMoving AND NOT IO.StorageCarriageMoving THEN
        MainState := GET_BOX_STORE_WAIT;
      END_IF
    END_IF
  END_IF

GET_BOX_STORE_WAIT:
  TON_DelayCycle(IN := TRUE, PT := T#1S);
  IF TON_DelayCycle.Q THEN
    TON_DelayCycle(IN := 0);

  MainState := STORAGE_HANDLER_TO_BOX;
  END_IF

STORAGE_HANDLER_TO_BOX:
  TON_DelayCycle(IN := TRUE, PT := T#25);
  IF TON_DelayCycle.Q THEN
    P_RegulatorY.SetpointPos := 2100;
    IO.StorageLoad := 1;
    MainState := STORAGE_HANDLER_TO_BOX_WAIT;
  END_IF

```

Obrázek 12: Pátá část kódu

- V šesté části kódu byl zaveden určitý jednoduchý "algoritmus" pro vyzdvihnutí té správné souřadnice z polí x a y.

```

198     STORAGE_HANDLER_TO_BOX_WAIT:
199         TON_DelayCycle(IN := TRUE, PT := T#2S);
200         IF TON_DelayCycle.Q THEN
201
202             TON_DelayCycle(IN := FALSE);
203             MainState := LOAD_BOX;
204         END_IF
205
206     LOAD_BOX:
207         TON_DelayCycle(IN := TRUE, PT := T#2S);
208         IF TON_DelayCycle.Q THEN
209             TON_DelayCycle(IN := FALSE);
210             IO.StorageLoad := 0;
211             MainState := LOAD_BOX_WAIT;
212         END_IF
213
214     LOAD_BOX_WAIT:
215         TON_DelayCycle(IN := TRUE, PT := T#2S);
216         IF TON_DelayCycle.Q THEN
217             TON_DelayCycle(IN := FALSE);
218             MainState := DRIVE_BOX_TO_STORE;
219         END_IF
220
221
222     DRIVE_BOX_TO_STORE:
223         TON_DelayCycle(IN := TRUE, PT := T#1S);
224         IF TON_DelayCycle.Q THEN
225             TON_DelayCycle(IN := FALSE);
226             P_RegulatorX.SetpointPos := x[ID[BoxesStored]];
227             P_RegulatorY.SetpointPos := y[BoxesStored] + 400;
228             MainState := DRIVE_BOX_TO_STORE_WAIT;
229         END_IF
230
231     DRIVE_BOX_TO_STORE_WAIT:
232         TON_DelayCycle(IN := TRUE, PT := T#1S);
233         IF TON_DelayCycle.Q THEN
234             TON_DelayCycle(IN := FALSE);
235             MainState := BOX_UNLOAD;
236         END_IF
237

```

Obrázek 13: Šestá část kódu

- Jak již výše bylo uvedeno, v předposledním stavu automatu inkrementují proměnnou BoxGrabbed. Indexujeme od nuly, proto je-li BoxGrabbed = 5, jdu zpět do stavu, kdy vezmu krabici. Při dosažení 5 dojdeme do konečného stavu.

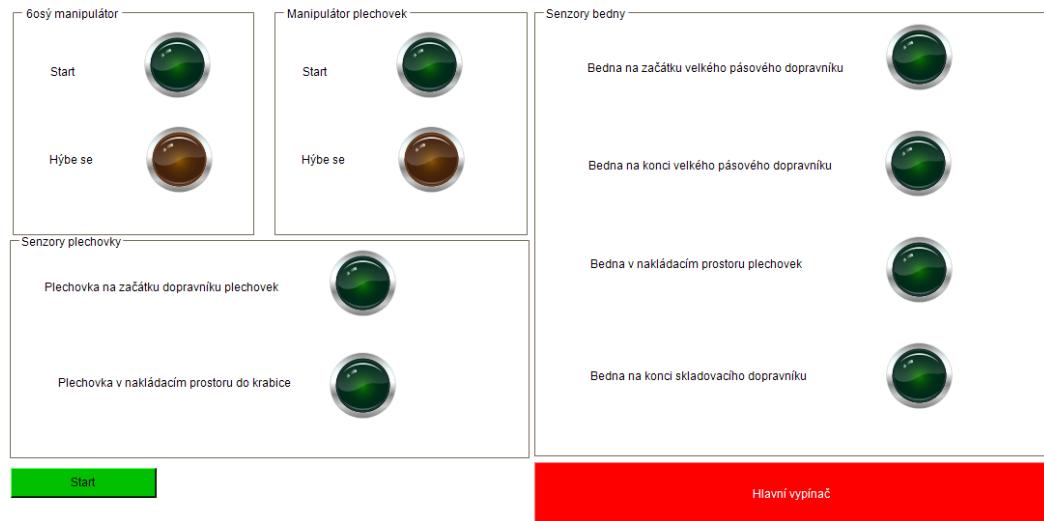
```

237
238     BOX_UNLOAD:
239         TON_DelayCycle(IN := TRUE, PT := T#1S);
240         IF TON_DelayCycle.Q THEN
241             IF NOT IO.StorageCarriageMoving AND NOT IO.StorageSupportMoving THEN
242                 TON_DelayCycle(IN := FALSE);
243                 IO.StorageUnload := 1;
244                 MainState := BOX_UNLOAD_WAIT;
245             END_IF
246         END_IF
247     BOX_UNLOAD_WAIT:
248         TON_DelayCycle(IN := TRUE, PT := T#1S);
249         IF TON_DelayCycle.Q THEN
250             TON_DelayCycle(IN := FALSE);
251             MainState := LAY_BOX;
252             P_RegulatorY.SetpointPos := P_RegulatorY.SetpointPos - 400;
253         END_IF
254
255     LAY_BOX:
256         TON_DelayCycle(IN := TRUE, PT := T#1S);
257         IF TON_DelayCycle.Q THEN
258             IF NOT IO.StorageCarriageMoving AND NOT IO.StorageSupportMoving THEN
259                 TON_DelayCycle(IN := FALSE);
260                 IO.StorageUnload := 0;
261                 IF BoxesStored < 5 THEN
262                     MainState := GRAB_BOX;
263                     BoxesStored := BoxesStored + 1;
264                 ELSE
265                     MainState := STORING_DONE;
266                 END_IF
267             END_IF
268
269         END_IF
270
271 END_CASE

```

Obrázek 14: Sedmá část kódu

Vizualizace



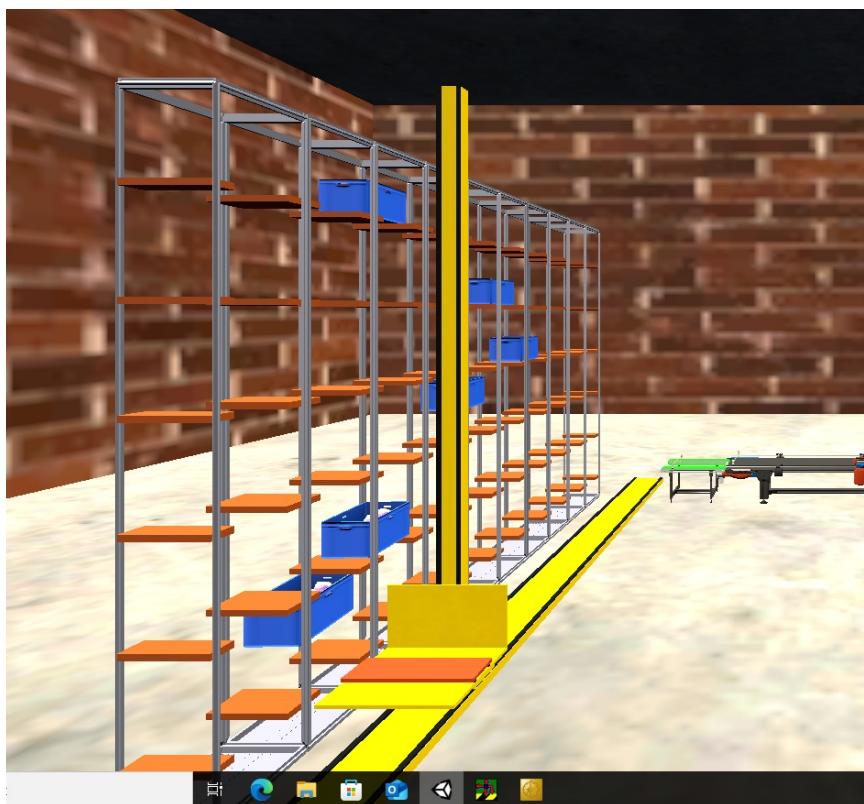
Obrázek 15: Vytvořená vizualizace automatického skladu

- Po stisknutí tlačítka START se automatický sklad rozeběhne.

- Stiskem tlačítka STOP (Hlavní vypínač) se veškeré Enable vstupy simulace nastaví do logické nuly. Na roboty se toto bohužel nevztahuje, jelikož reagují na hranu, nikoliv na hladinu.
- Na zelené, informativní žárovce Start od 6osého manipulátoru je připojen signál "IO.RoboticArmStartCycle". Po započetí cyklu se žárovka rozsvítí, signalizujíc nastartovaný manipulátor. Jakmile se začne hýbat, rozsvítí se také žlutá žárovka signalizující stav nebezpečných pohybujících se částí. U "handlera" plechovek je to analogické.
- Kromě toho je vizualizace vybavena informativními žárovkami, které nám pomohou sledovat, ve které části skladu se jak bedna na velkém pásovém dopravníku, tak plechovka na malém pásovém dopravníku nachází.

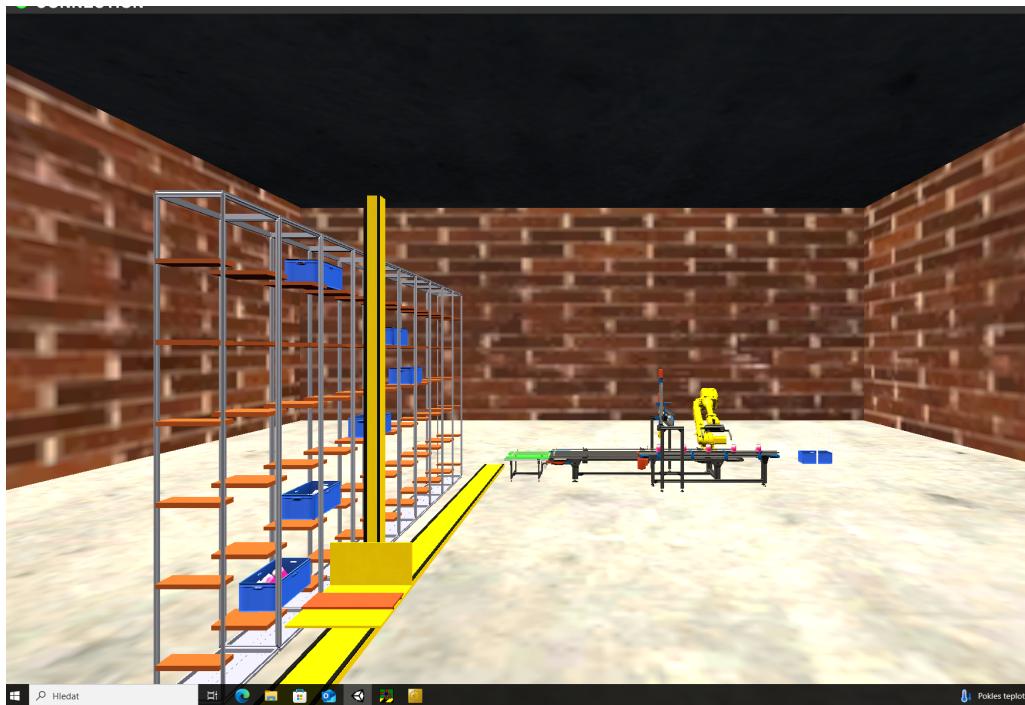
Dokumentace skladu

- Problémem při dokumentaci byla práce ve virtuálním stroji v tom, že je zde podstatně obtížnější integrace myši. V samotném programu bylo nutné nastavit přesné souřadnice. Při nepřesnosti (příliš velké souřadnice Y) došlo k "levitující bedně" - nedosedla do regálu a vlastně se přilepila k horní straně regálu, jak je vidět u prvního pokusu (před optimalizací souřadnic). Ačkoliv se mně nepovedlo provést rotaci perspektivy, mělo by z daného pohledu být patrné, že se jedná o moje ID 256421. Při vyladění ID, jak je v současnosti v dokumentaci kódu, mně manipulátor naskládá veškeré krabice do regálu přesně tak, jak mají být.



Obrázek 16: Ukázka téměř levitujících krabic

– Po změně ID:



Obrázek 17: Hotové ID se spravenými souřadnicemi (aktuální zadané v proměnných)

Závěr projektu

Cílem projektu bylo vyzkoušet si programování automatizované linky. Automatický režim se skládá z jednoho stavového automatu o bezmála 30 stavech. Pozdější zjištění ukázalo, že toto není zcela ideálním řešením z hlediska optimalizace (sdílení úloh), celý proces uskladnění šesti beden byl tak daleko delší, než kdyby jednotlivé součásti průmyslové automatizace pracovaly nezávisle na sobě – v době, kdy už je jedna krabice odbavena 2osým manipulátorem, by bylo možné nakládat novou krabici na pás. Na místo toho musíme ještě čekat, než je jedna krabice zavezena na konkrétní pozici, než teprve šestiosý manipulátor uchopí novou krabici. Při příští implementaci (jedná se o mou první zkušenosť s programováním PLC) podobného řešení bych postupoval jinak. Jak již bylo výše uvedeno, z důvodu časové tísni nebyl proveden manuální režim. V rámci vizualizace vidíme, zda je spuštěn 6osý manipulátor, handler plechovek. Zároveň vidíme, zda v jejich blízkosti dbát zvýšené opatrnosti, pokud se hýbou. Signalizují to žluté žárovky. Celou simulaci (s výjimkou robotů, kteří musí ukončit svůj cyklus) je možné vypnout červeným tlačítkem HLAVNÍ VYPÍNAČ. Zbývající zelené signální žárovky představují signalizaci pohybu bedny a žárovky. Pro první implementaci tuto vizualizaci považuji za dostačující, dále by však bylo možné ještě přidat tlačítko reset, které by vrátilo roboty do poloh 0 (stav INIT ROBOTS), stejně tak by bylo možné v automatickém režimu implementovat zadání libovolného ID, nebo vizualici toho, kolik beden je právě naskladněno.