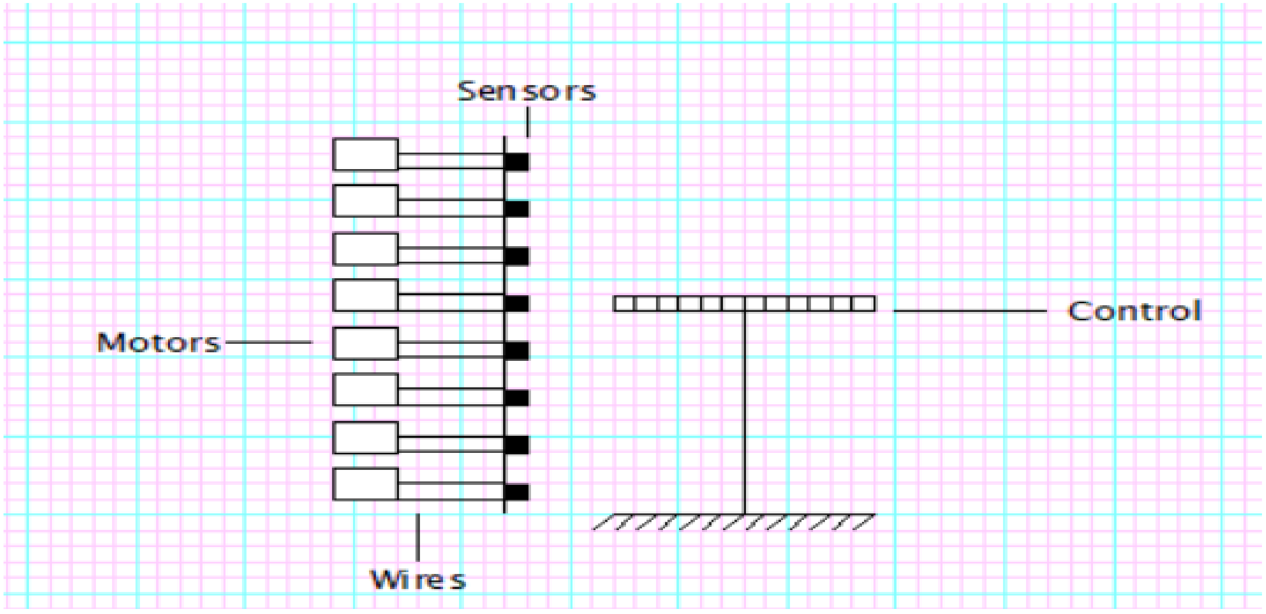


Fig.0 Sensor-Based MUX

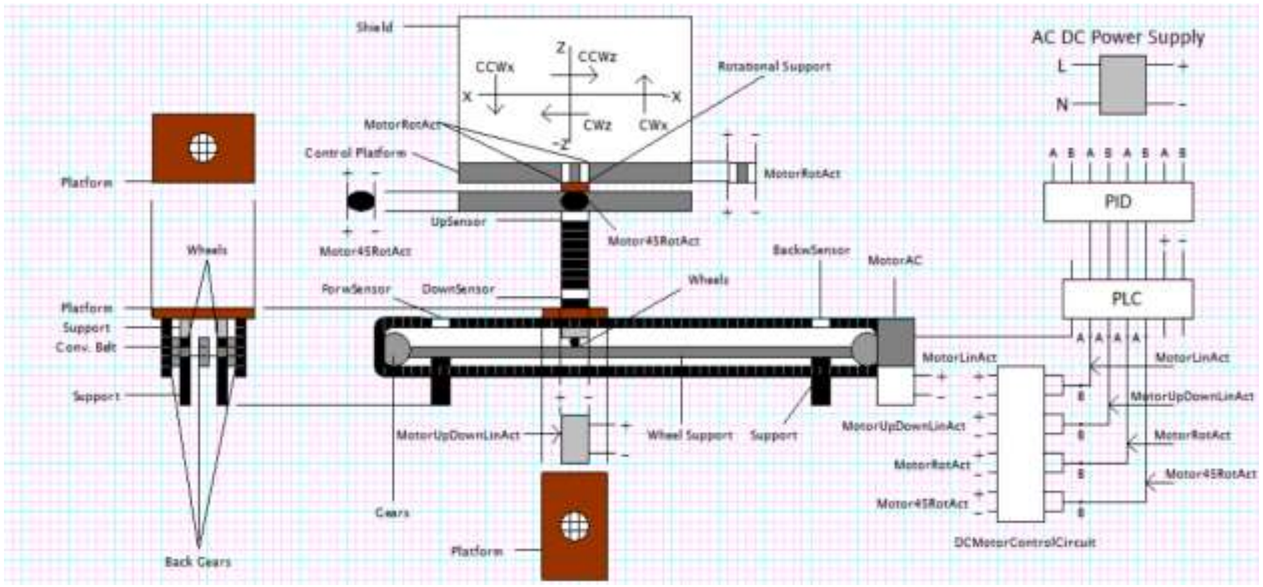


Fig.1 Platform with Multiple Degrees of Freedom

The working principle consists of a control input that activates a set of proximity sensors. The activated sensor triggers any electrically connected components or motors. The goal is to implement a method to control a platform with multiple degrees of freedom (Fig 1) using only one multiplexer or DC Motor Control Circuit that can implement any desired control algorithm (Fig 0).

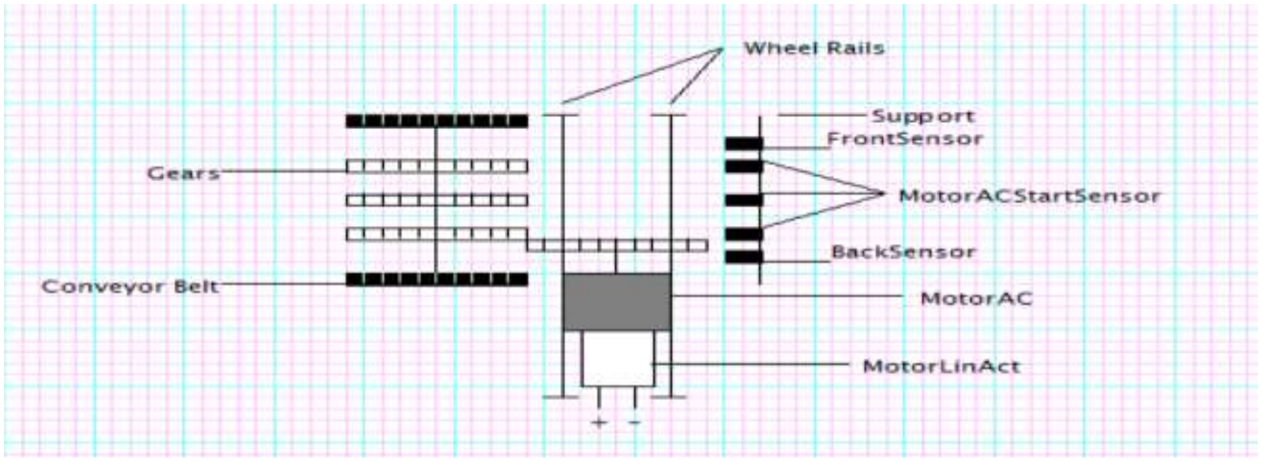


Fig.1a Sensor-Based MUX

The goal behind this implementation is to design an electromechanical like multiplexer that controls several motors simultaneously while simplifying the number of user input commands. To demonstrate, we will consider the example project in (Fig 1). It consists of a conveyor belt; one AC motor that controls the speed and frequency of rotation of the gears; one DC linear actuator that moves the AC motor through a system of gears; one DC linear actuator responsible for moving the vertical shaft holding the control platform along the z axis; one DC rotational linear actuator responsible for CW and CCW platform rotations; one DC rotational linear actuator responsible for 45-degree angular platform rotations. There are a total of eight proximity sensors: ForwSensor, BackwSensor, UpSensor, DownSensor, FortyFiveDegPosSensor, FortyFiveDegNegSensor, FrontSensor, BackSensor. The ForwSensor and BackwSensor control the movement of the platform in the x dimension acting as limits. Similarly, (UpSensor, DownSensor), (PosRotSensor, NegRotSensor) limit motion in the z dimension and CCWx/CWx direction respectively. The MotorAC and MotorLinAct complex move in the y dimension and are controlled by sensors: FrontSensor and BackSensor acting as limits. At each MotorACStartSensor proximity sensor, power is transferred to actuators: MotorLinAct, MotorRotLinActPistonMotionStart (controls platform motion in CCWz/CWz), MotorUpDownLinActMotionStart (controls platform motion along the z axis), MotorFortyFiveDegTiltLinAct (controls platform along the CCWz/CWz direction) and finally to the AC motor itself named as MotorACHorzMotionStart.

Note: When we mention AC Motor we mean a Brushless DC Motor.

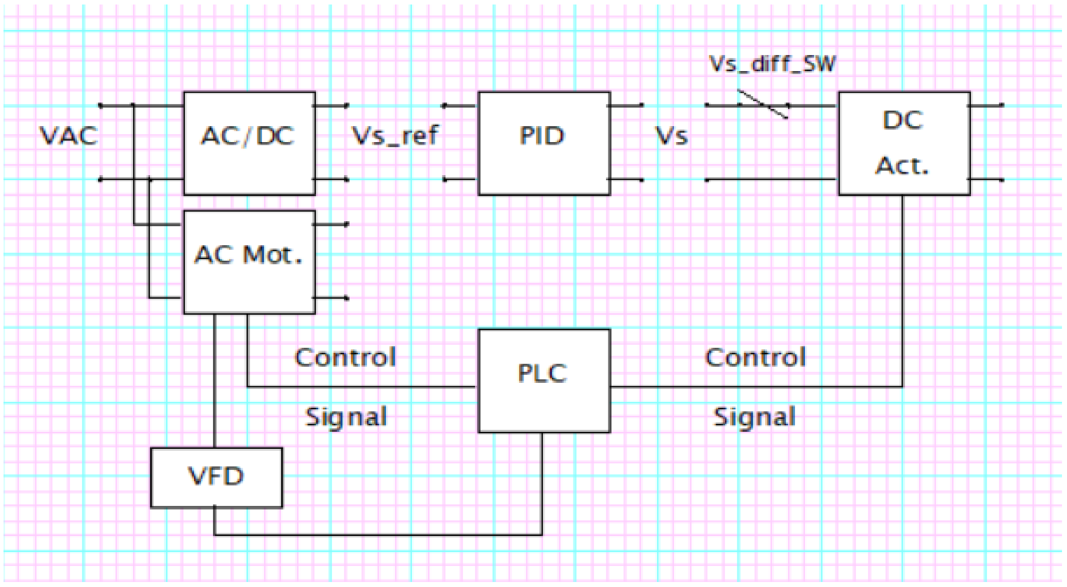


Fig.2 Sensor Based MUX Overview

An AC voltage signal VAC is converted to a DC voltage signal Vs_ref. A PID controller is used to detect noticeable changes in the DC voltage from the AC DC supply to a DC actuator. If Vs differs noticeably from Vs_ref, the switch Vs_diff_SW is activated and breaks the connection. Vs_diff_SW is returned to its normal state when there is close to zero difference between Vs_ref and Vs. Upon successful transmission the DC Linear Actuator is activated with Vs. The AC Motor and DC Actuators are both then ready to receive instructions from the PLC and perform their operations in any desired order. The speed of the AC Motor can be controlled by a Variable Frequency Driver VFD.

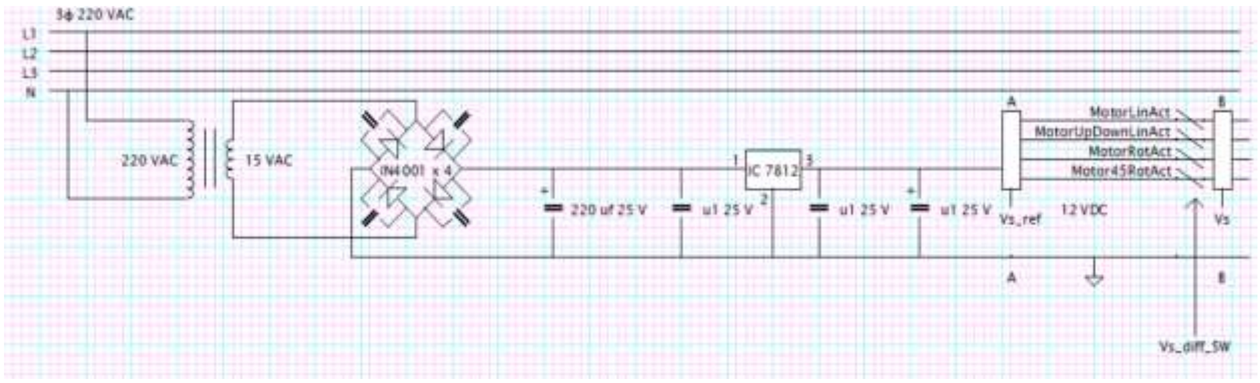


Fig.3 AC/DC Converter

A 220 V AC signal is converted to 15 V AC with a 220 VAC to 15 VAC transformer. The 15 V AC signal then passes through a full wave rectifier. The capacitors in parallel to the IN4001 diodes are responsible for minimizing signal oscillations and vibrations ensuring a smooth rectified signal with minimal disturbances. The rectified signal passes through a polarized 220 mF capacitor which filters out high frequency components and a non-polarized 1 mF capacitor which filters out any remaining ripples in the 15 V AC signal. The linear voltage regulator IC 7812 reduces the signal amplitude to 12 V DC. The two remaining parallel capacitors: non-polarized 1mF and polarized 1mF are used for assisting the regulator supply gross load changes by removing some of the load from the regulator.

A and B are six-way and nine-way wire connectors respectively. Connector A captures the initial voltage or the reference voltage Vs_ref, while B captures the voltage Vs near the DC Actuators. Vs_ref is compared with Vs and if the difference Vs_diff >> 0, the four-pole relay is energized. At energized state the connection between AC/DC converter and DC Motor is broken.

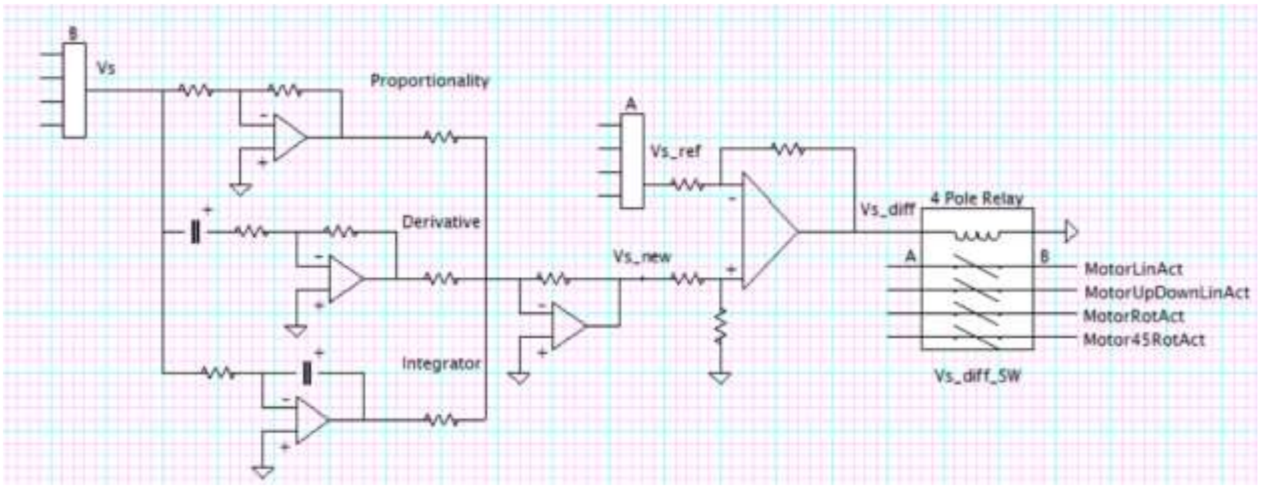


Fig.4 Proportionality Integrator Derivative (PID) Controller

The PID controller corrects and fixes voltage drops between the AC/DC converter (Vs_ref) and the DC Actuators (Vs). When Vs_ref and Vs differ by a large amount and are not approximately equal the Vs_diff_SW is energized and immediately disconnects DC actuators from power supply. The parameters within the PID controller are designed to be small such that Vs is approximately equal to Vs_new. And as such the PID controller is only useful for very small corrections in voltage.

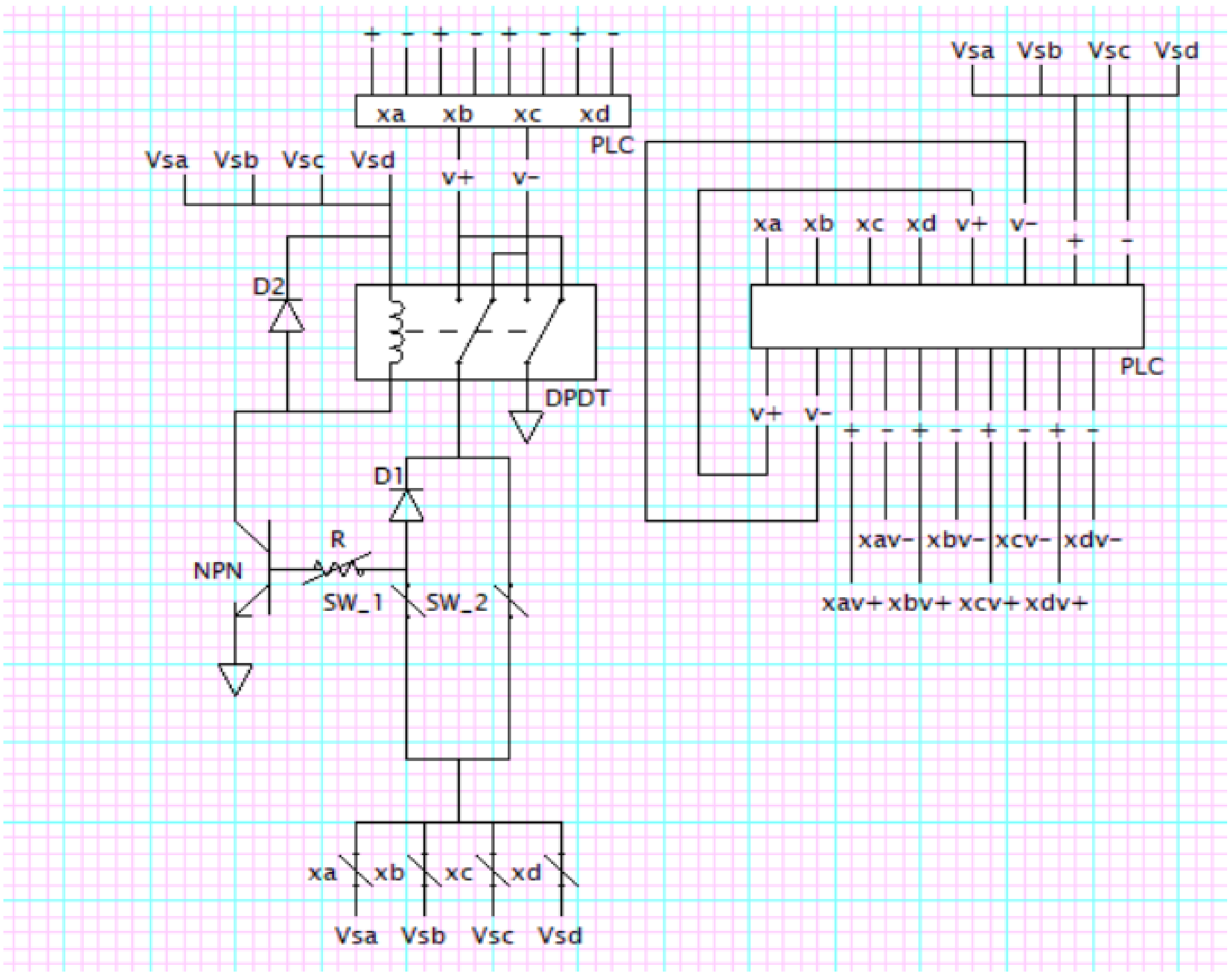
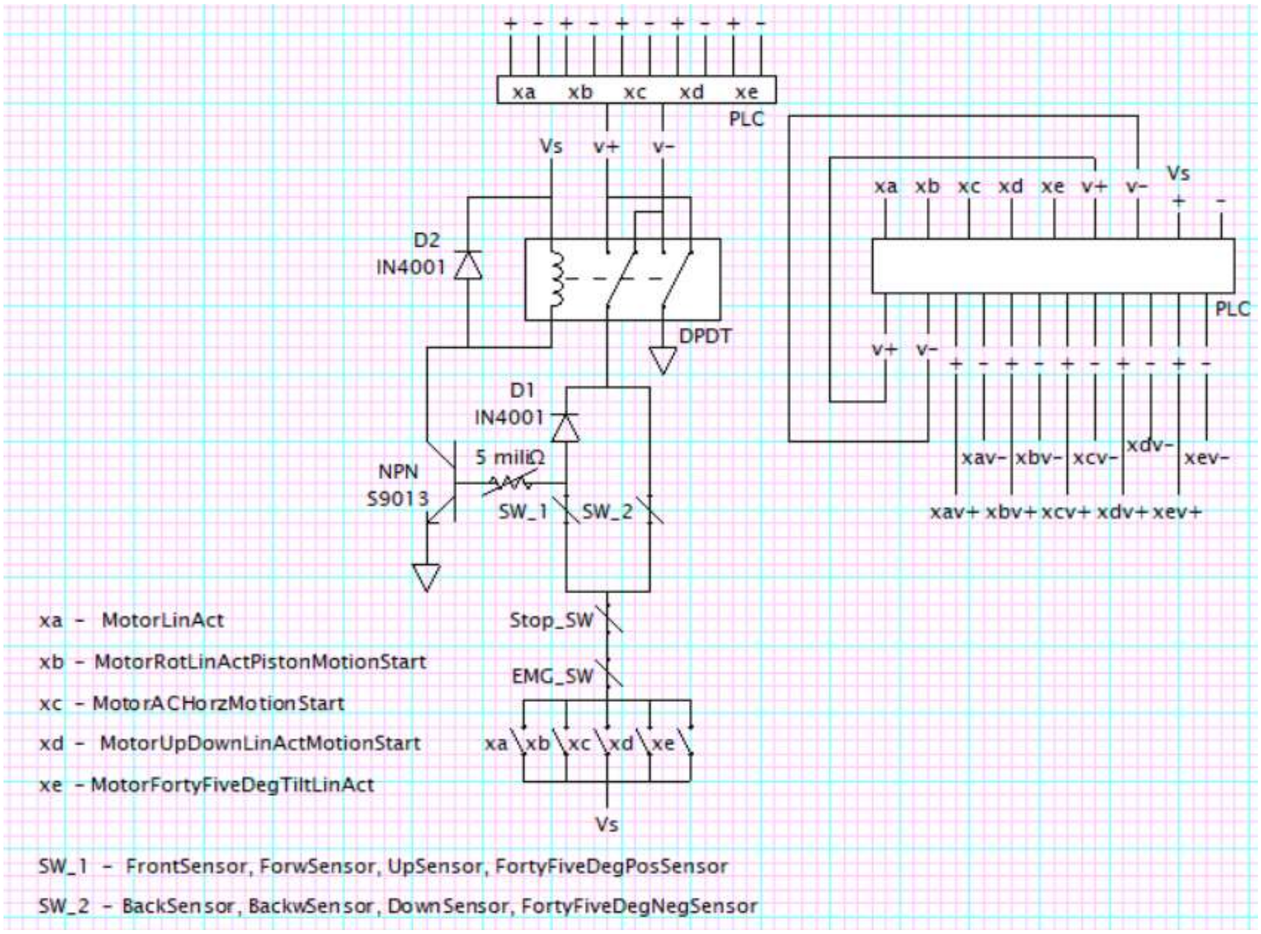


Fig.5 DC Motor Control Circuit (Controller)

All DC Motor actuators will be controlled by a single DC Motor Control Circuit. Controls SW_1 and SW_2 control the forward and reverse motions of the linear or rotational actuators. Inputs xa, xb, xc, xd determine which DC actuator terminals v+ and v- will be distributed to by feeding them back into the PLC. This will result in multiple 2 terminal pairs: (xav+, xav-) ... (xdv+, xdv-) each supplying individual actuators with power of varying levels. The voltages supplied: Vsa, Vsb, Vsc, Vsd can either be equal or different depending on the characteristics of the inductor, NPN transistor, variable resistor and diodes.



xa xb xc xd xe
PLC
Vs v+ v-
D2
IN4001
DPDT
D1
IN4001
5 miliΩ
NPN
S9013
SW_1 SW_2
Stop_SW
EMG_SW
xa xb xc xd xe
Vs
xa xb xc xd xe v+ v- Vs
PLC
v+ v-
xav- xbv- xcv- xdv- xev-
xav+ xbv+ xcv+ xdv+ xev+
xa - MotorLinAct
xb - MotorRotLinActPistonMotionStart
xc - MotorACHorzMotionStart
xd - MotorUpDownLinActMotionStart
xe - MotorFortyFiveDegTiltLinAct
SW_1 - FrontSensor, ForwSensor, UpSensor, FortyFiveDegPosSensor
SW_2 - BackSensor, BackwSensor, DownSensor, FortyFiveDegNegSensor

Fig.5a DC Motor Controller

For this project switches SW_1, SW_2 each represent the multiple switch sensor states of each DC motor, effectively reducing eight different switches into two by sharing the same memory address. Switches xa, xb, xc, xd, xe represent the states in when each motor will operate upon activation.

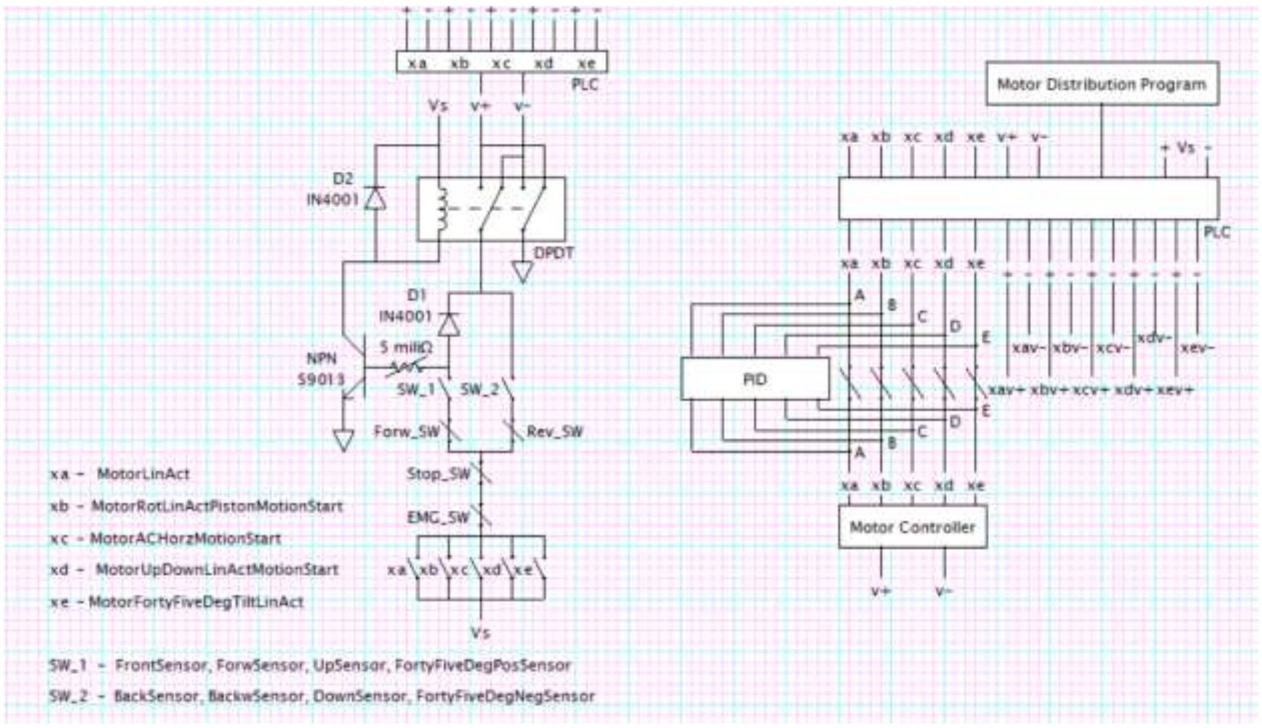

xa xb xc xd xe
PLC
Vs v+ v-
D2
IN4001
DPDT
D1
IN4001
5 milliΩ
NPN
S9013
SW_1
SW_2
Forw_SW
Rev_SW
Stop_SW
EMG_SW
xa xb xc xd xe
Vs
xa - MotorLinAct
xb - MotorRotLinActPistonMotionStart
xc - MotorACHorzMotionStart
xd - MotorUpDownLinActMotionStart
xe - MotorFortyFiveDegTiltLinAct
SW_1 - FrontSensor, ForwSensor, UpSensor, FortyFiveDegPosSensor
SW_2 - BackSensor, BackwSensor, DownSensor, FortyFiveDegNegSensor
Motor Distribution Program
xa xb xc xd xe v+ v-
+ Vs -
PLC
A B C D E
PID
xav- xbv- xcv- xdv- xev-
xav+ xbv+ xcv+ xdv+ xev+
Motor Controller
v+ v-

Fig.5b DC Motor Controller

Overview of how this controller will operate on this given project. Additional switches Forw_SW and Rev_SW are added for additional protection. A program will also be added to the PLC to assist it in locating the correct motors for proper power distribution.

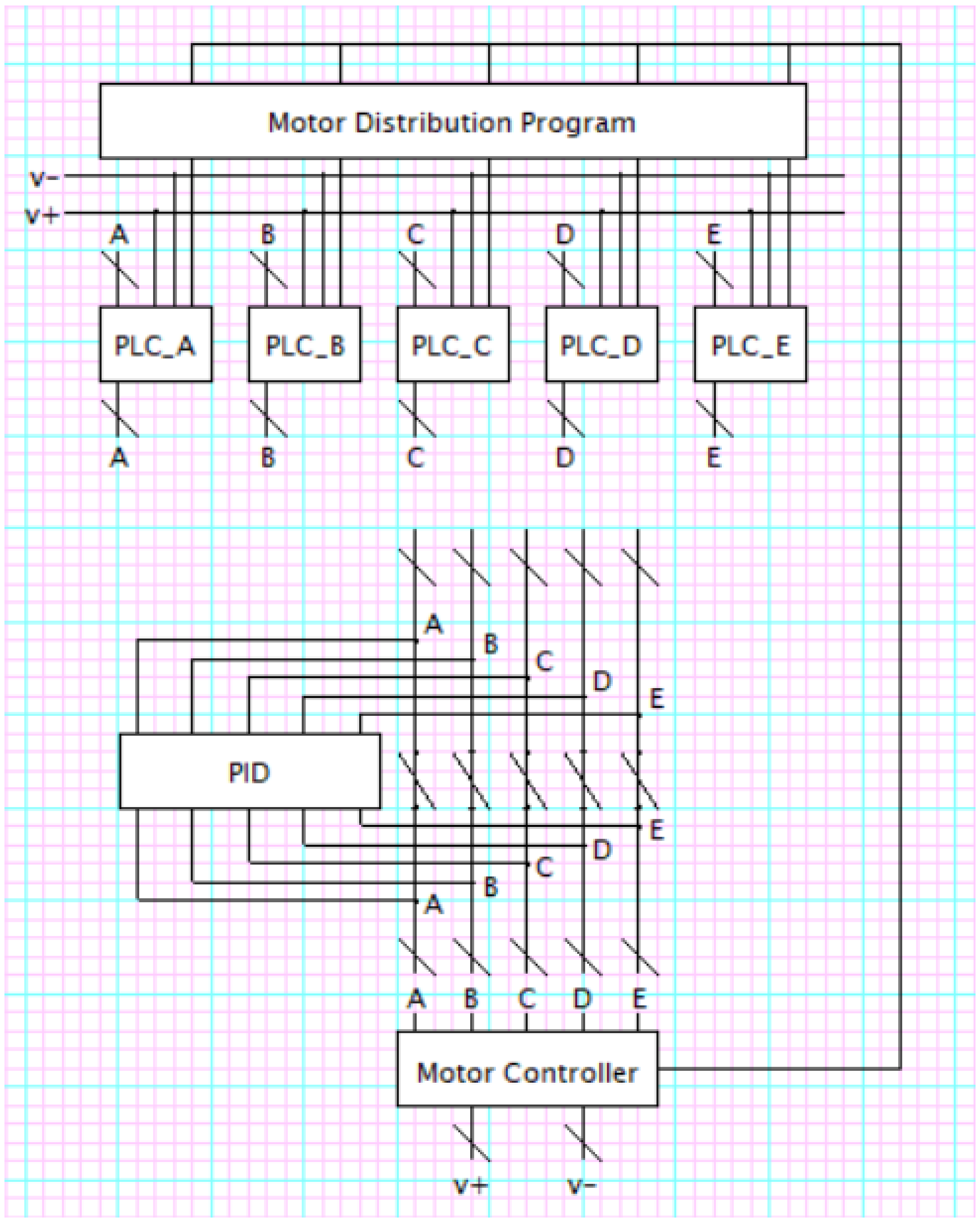


Fig.5c DC Motor Controller

The controller can also be used on multiple PLC's that have similar voltage requirements. For safety this application can be used only for low voltage applications. For this controller to function safely and efficiently for any system voltage low or high, we would need a system of virtual infinite inductors. The advantage here is that multiple processes can be synchronized and controlled concurrently with only one motor controller, PID and PLC motor distribution program (which can be connected to the PLC's using either USB or ethernet cables or can be written using structured text if the PLC has no programming support for C programs).

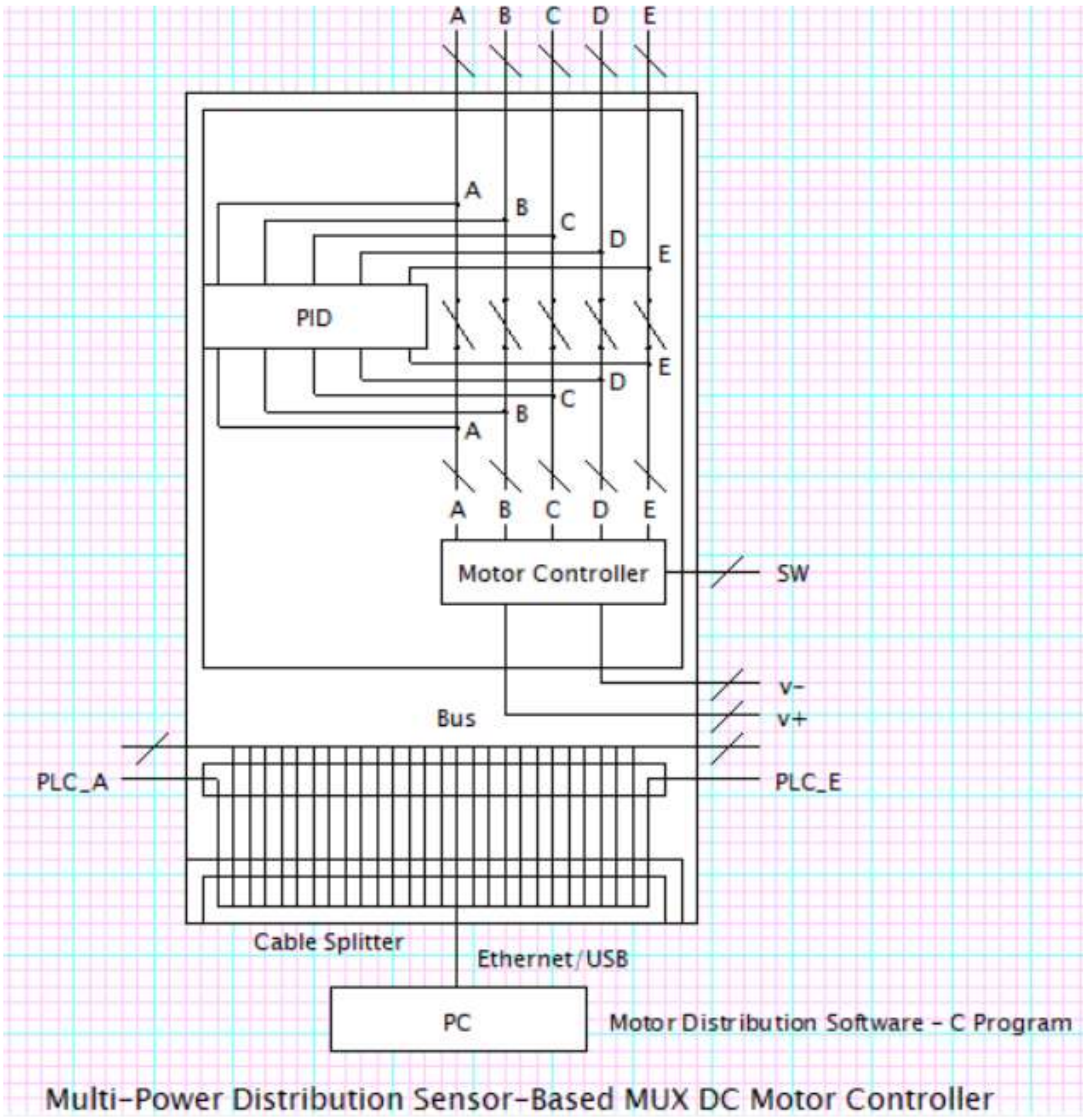

A
B
C
D
E
PID
Motor Controller
SW
v-
v+
Bus
PLC_A
PLC_E
Cable Splitter
Ethernet/USB
PC
Motor Distribution Software - C Program
Multi-Power Distribution Sensor-Based MUX DC Motor Controller

Fig.5d Multi-Power Distribution Sensor-Based MUX DC Motor Controller

This shows a compact form of the MUX. It can be placed inside a control box for more complex motor control systems.

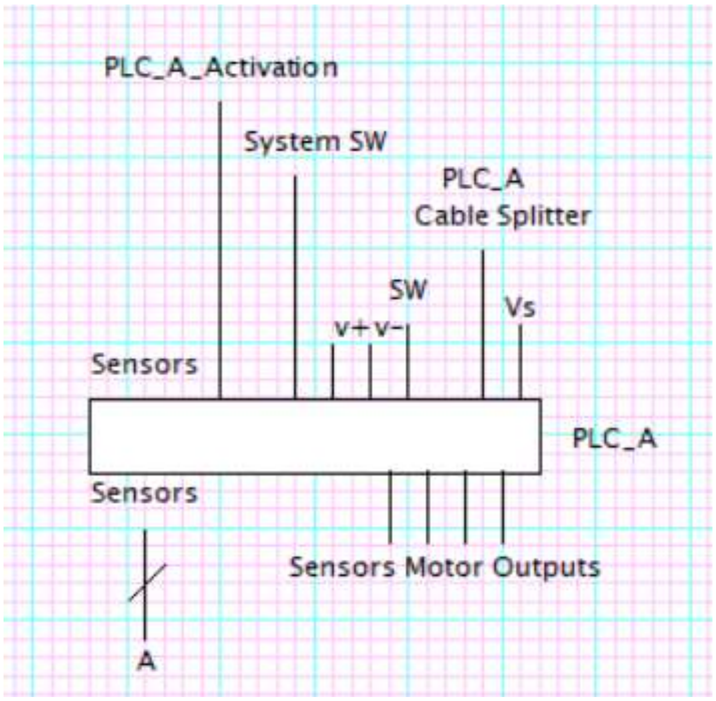


Fig.5e Multi-Power Distribution Sensor-Based MUX DC Motor Controller Action on a PLC

This describes the working principle behind the MUX. A PLC_A cable from the cable splitter is connected to a PLC. Assuming the PLC has an Ethernet/IP module the cable downloads a program from the PC to instruct the PLC which motors can be activated at a given time. The PLC_A_Activation contains the address of the PLC port at the PLC_A Cable Splitter input. In this way the program can identify which PLC it needs to program. The System_SW are a set of inputs containing all the switches in the system the PLC controls that are all memory mapped one at a time to the motor controller's switches SW depending on the state of the system. Ports V+ and V- are the two end terminals of the motor controller. Depending on which sensors are activated at a given time the program copies the inputs V+, V- through memory and transfers them to the appropriate terminals to activate the appropriate motor. This method can easily be scaled to more advanced systems with multiple controls.

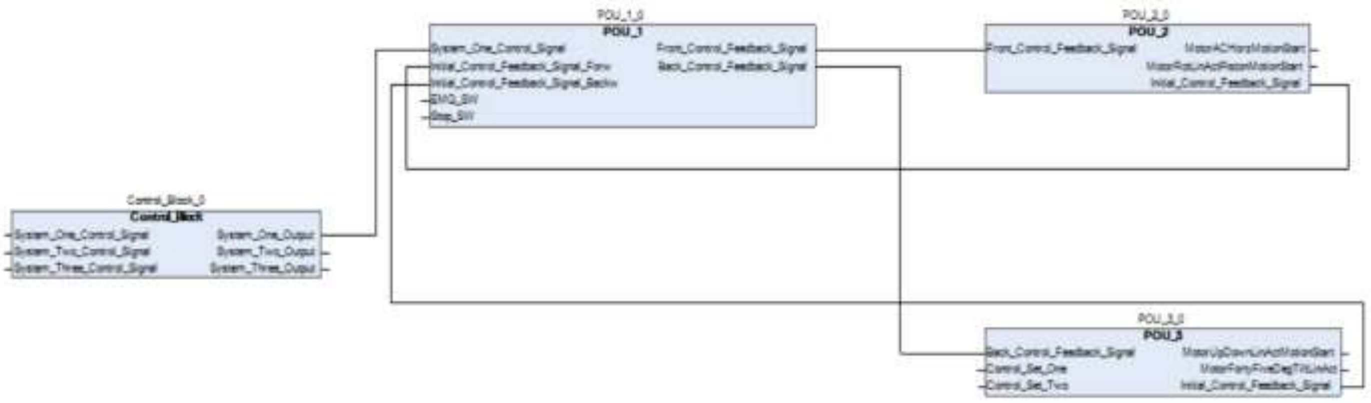

POU_1_0
POU_1
System_One_Control_Signal
Initial_Control_Feedback_Signal_Forw
Initial_Control_Feedback_Signal_Backw
EMQ_SW
Stop_SW
Front_Control_Feedback_Signal
Back_Control_Feedback_Signal
POU_2_0
POU_2
Front_Control_Feedback_Signal
MotorACHorzMotionStart
MotorRotLinActPistonMotionStart
Initial_Control_Feedback_Signal
Control_Block_0
Control_Block
System_One_Control_Signal
System_Two_Control_Signal
System_Three_Control_Signal
System_One_Output
System_Two_Output
System_Three_Output
POU_3_0
POU_3
Back_Control_Feedback_Signal
Control_Set_One
Control_Set_Two
MotorUpDownLinActMotionStart
MotorFortyFiveDegTiltLinAct
Initial_Control_Feedback_Signal

Fig.6 Overview of Example Project

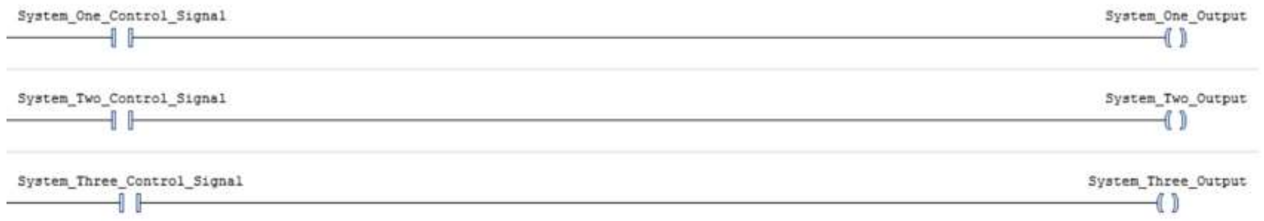

System_One_Control_Signal
System_One_Output
System_Two_Control_Signal
System_Two_Output
System_Three_Control_Signal
System_Three_Output

Fig.6a Control Block

For multiple such systems we can use a control block to activate a particular system of interest.

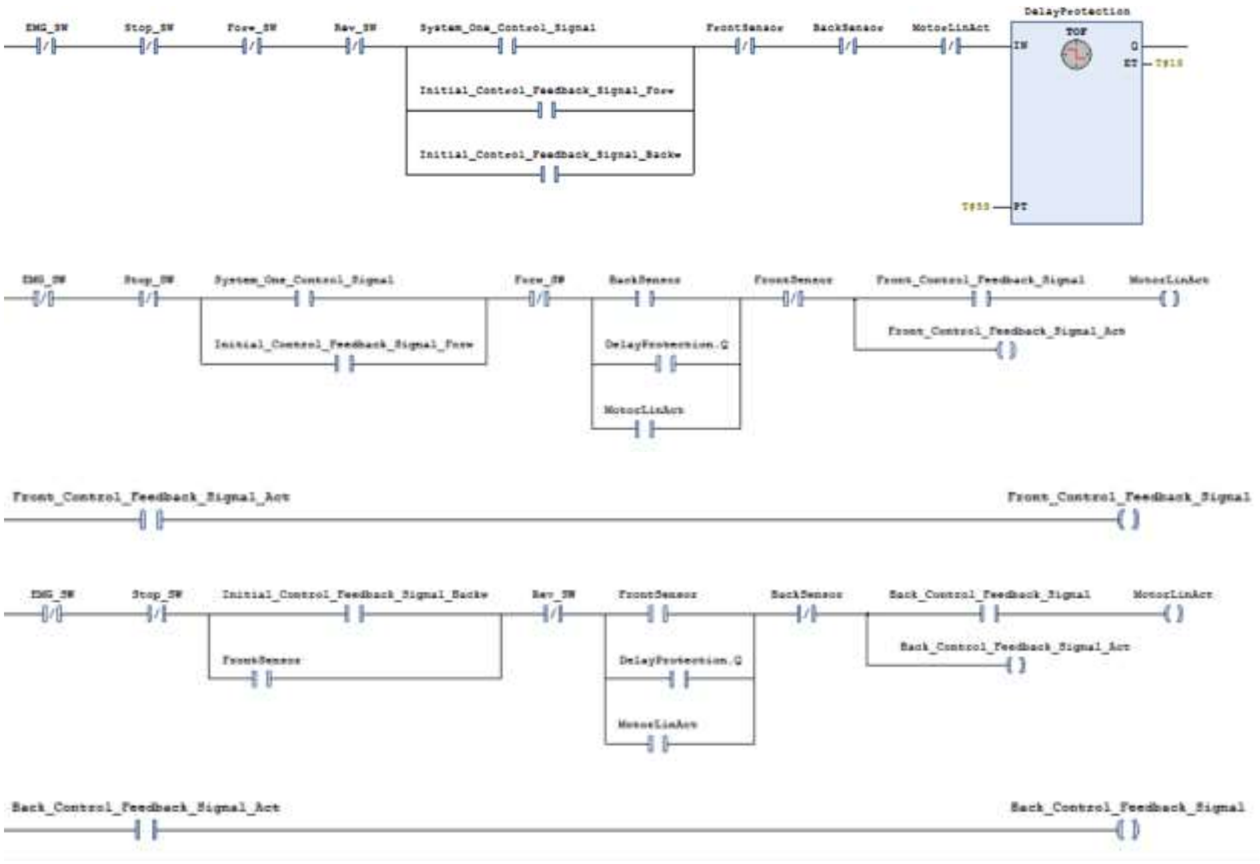

EMG_SW
Stop_SW
Forw_SW
Rev_SW
System_One_Control_Signal
Initial_Control_Feedback_Signal_Forw
Initial_Control_Feedback_Signal_Backw
FrontSensor
BackSensor
MotorLinAct
DelayProtection
TOF
IN
Q
ET
PT
T#5S
DelayProtection.Q
Front_Control_Feedback_Signal
Front_Control_Feedback_Signal_Act
Back_Control_Feedback_Signal
Back_Control_Feedback_Signal_Act

Fig.6b POU_1

Controls the MotorLinAct relative to the FrontSensor and BackSensor. The linear actuator will start at the back sensor upon activation and proceed in the forward phase activating Front_Control_Feedback_Signal. When the front sensor is reached the linear actuator will then retract until it reaches the back sensor again. This will be the back phase where Back_Control_Feedback_Signal is activated. The process will repeat back and forth until the emergency switch, stop switch, or manual Forw_SW and Rev_SW are activated.

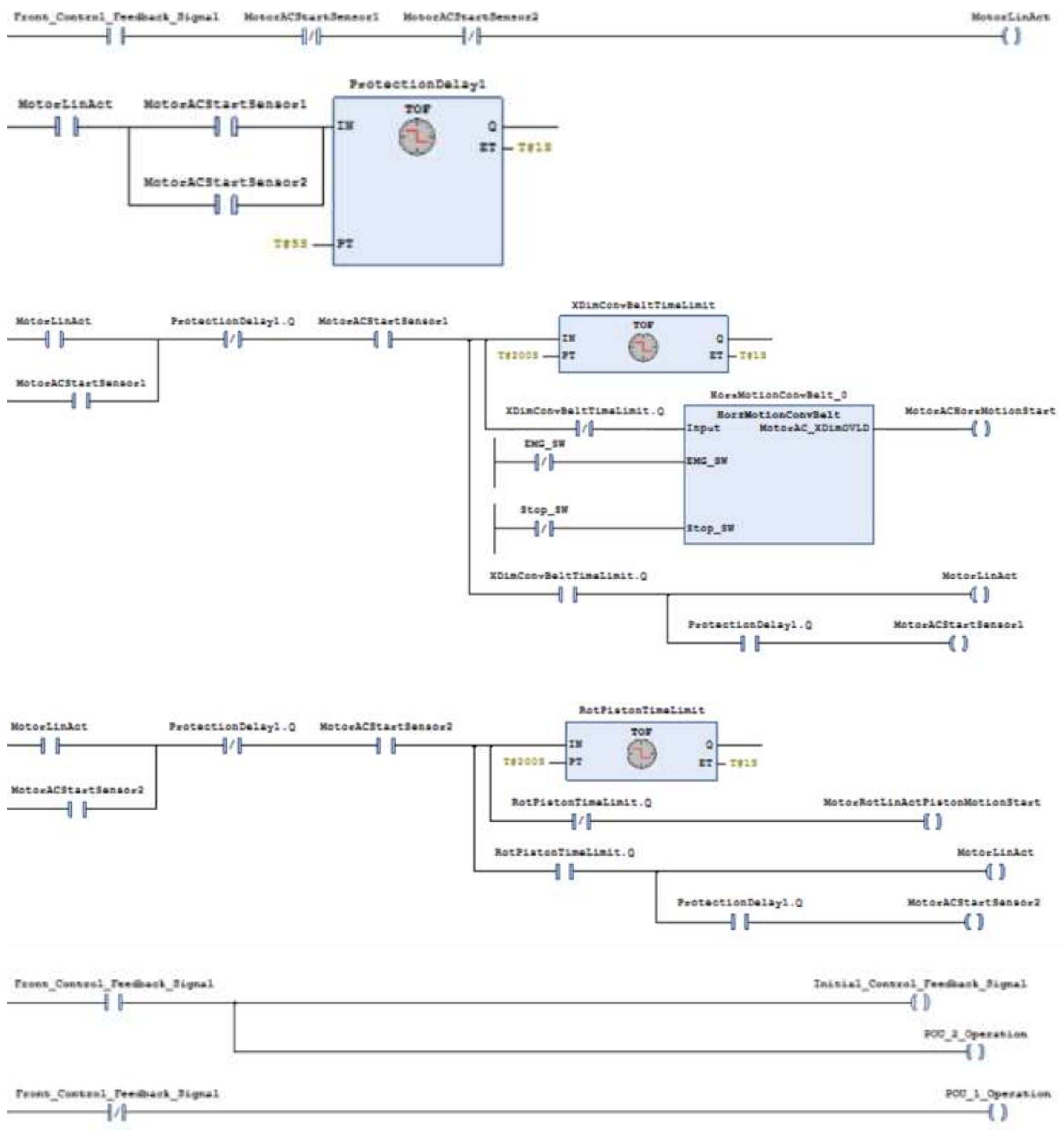
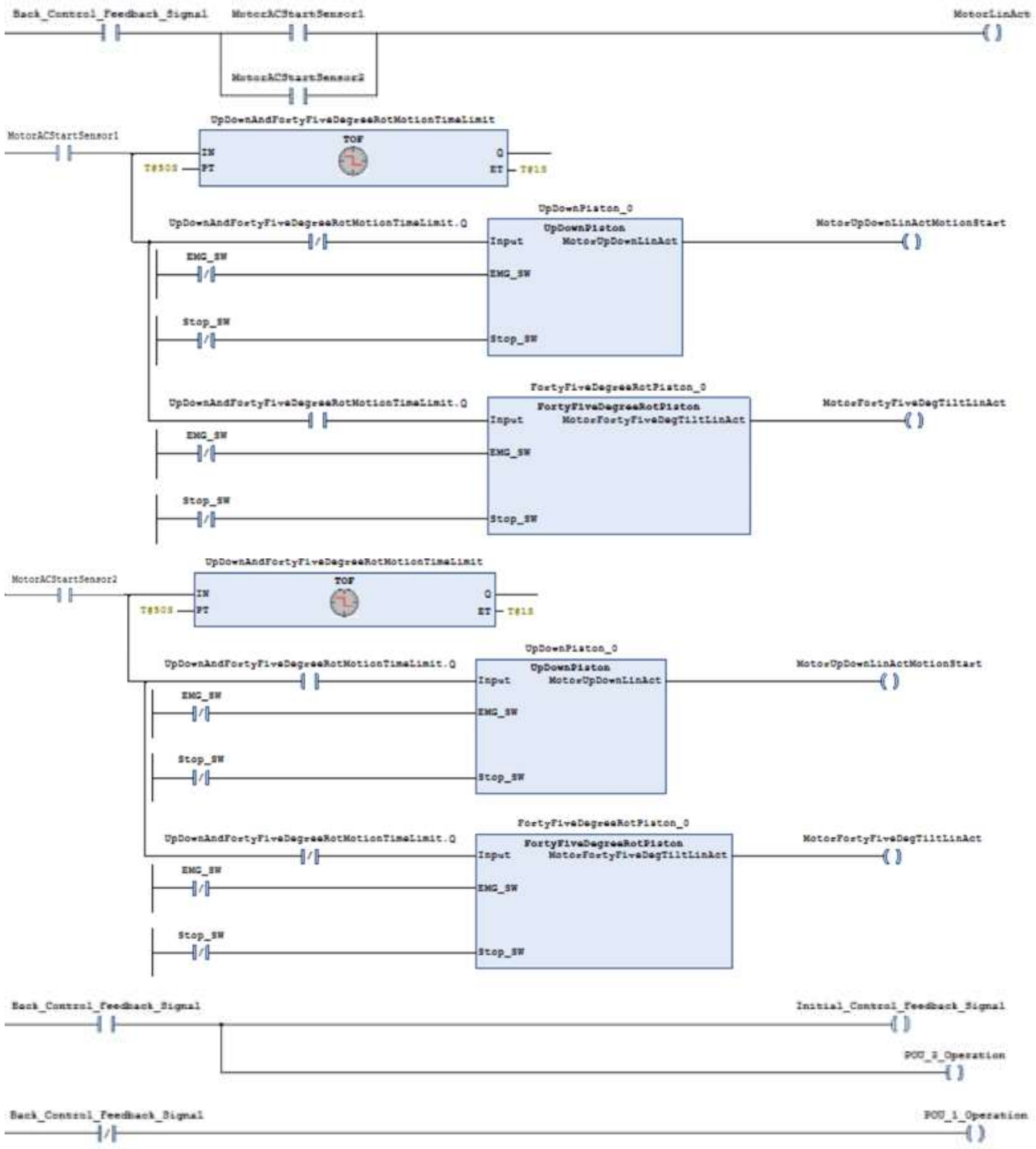


Fig.6c POU_2

Activates the AC motor (MotorACHorzMotionStart), linear actuator responsible for driving the motor (MotorLinAct), and the rotational motor responsible for rotating the platform (MotorRotLinActPistonMotionStart) with respect to proximity sensors (MotorACStartSensor1, MotorACStartSensor2). When the AC motor reaches the first AC sensor the linear actuator will stop and the AC motor will activate a set of gears connected to the conveyor belt. After a period of time the motor will stop, MotorLinAct will activate. Upon reaching the second AC sensor the rotational actuator will activate in a similar way. Upon completion Initial_Control_Feedback_Signal is activated to inform the POU_1 module that the linear actuator is still in the forward phase.



Back_Control_Feedback_Signal
MotorACStartSensor1
MotorLinAct
MotorACStartSensor2
UpDownAndFortyFiveDegreeRotMotionTimeLimit
MotorACStartSensor1
TOF
IN
PT
Q
ET
T#50S
T#1S
UpDownPiston_0
UpDownPiston
UpDownAndFortyFiveDegreeRotMotionTimeLimit.Q
Input
MotorUpDownLinAct
MotorUpDownLinActMotionStart
EMG_SW
Stop_SW
FortyFiveDegreeRotPiston_0
FortyFiveDegreeRotPiston
UpDownAndFortyFiveDegreeRotMotionTimeLimit.Q
MotorFortyFiveDegTiltLinAct
MotorFortyFiveDegTiltLinAct
EMG_SW
Stop_SW
UpDownAndFortyFiveDegreeRotMotionTimeLimit
MotorACStartSensor2
TOF
IN
PT
Q
ET
T#50S
T#1S
UpDownPiston_0
UpDownPiston
UpDownAndFortyFiveDegreeRotMotionTimeLimit.Q
Input
MotorUpDownLinAct
MotorUpDownLinActMotionStart
EMG_SW
Stop_SW
FortyFiveDegreeRotPiston_0
FortyFiveDegreeRotPiston
UpDownAndFortyFiveDegreeRotMotionTimeLimit.Q
MotorFortyFiveDegTiltLinAct
MotorFortyFiveDegTiltLinAct
EMG_SW
Stop_SW
Back_Control_Feedback_Signal
Initial_Control_Feedback_Signal
POU_2_Operation
Back_Control_Feedback_Signal
POU_1_Operation

Fig.6d POU_3

Upon activation of the FrontSensor the linear actuator MotorLinAct is now in reverse mode. Upon the detection of sensor MotorACStartSensor1 power is distributed first to MotorUpDownLinActMotionStart actuator and then to MotorFortyFiveDegTiltLinAct. While if it detects sensor MotorACStartSensor2 power is distributed the opposite way. Upon completion Initial_Control_Feedback_Signal is activate to inform the POU_1 module that the linear actuator is still in the reverse phase.

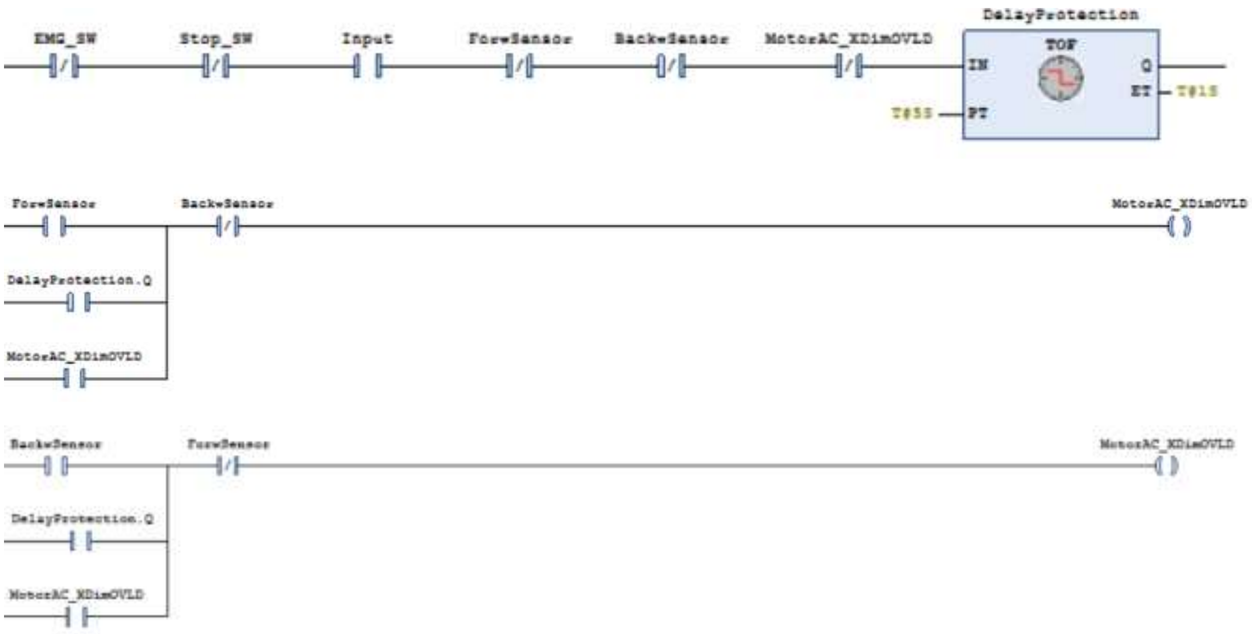


Fig.6e HorzMotionConvBelt_0

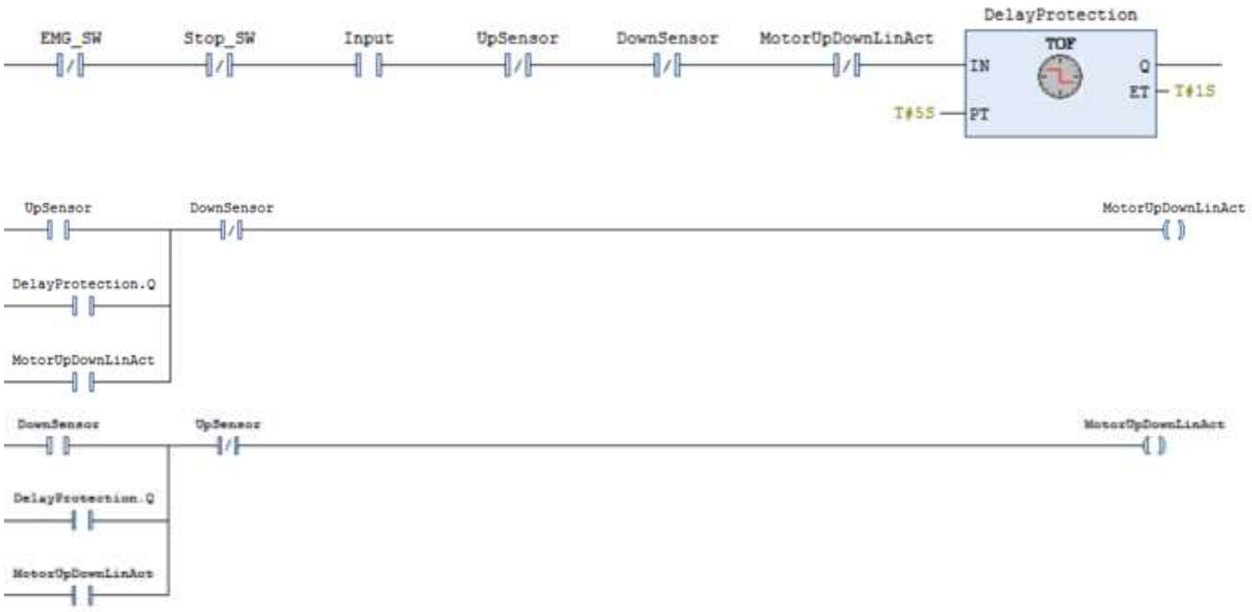


Fig.6f UpDownPiston_0

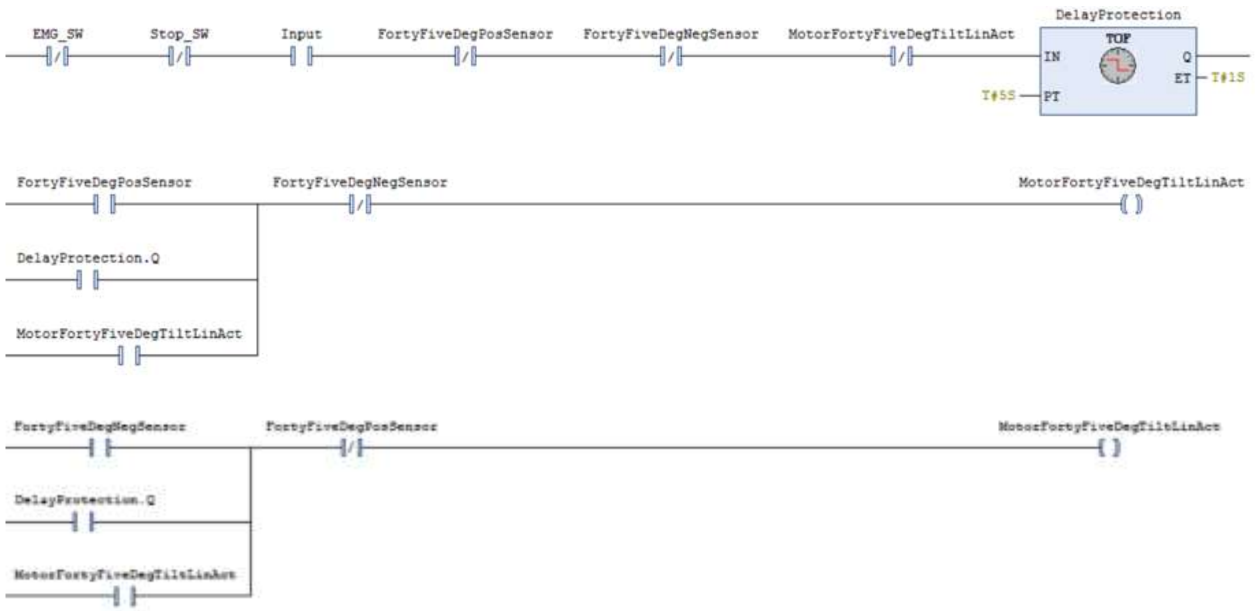
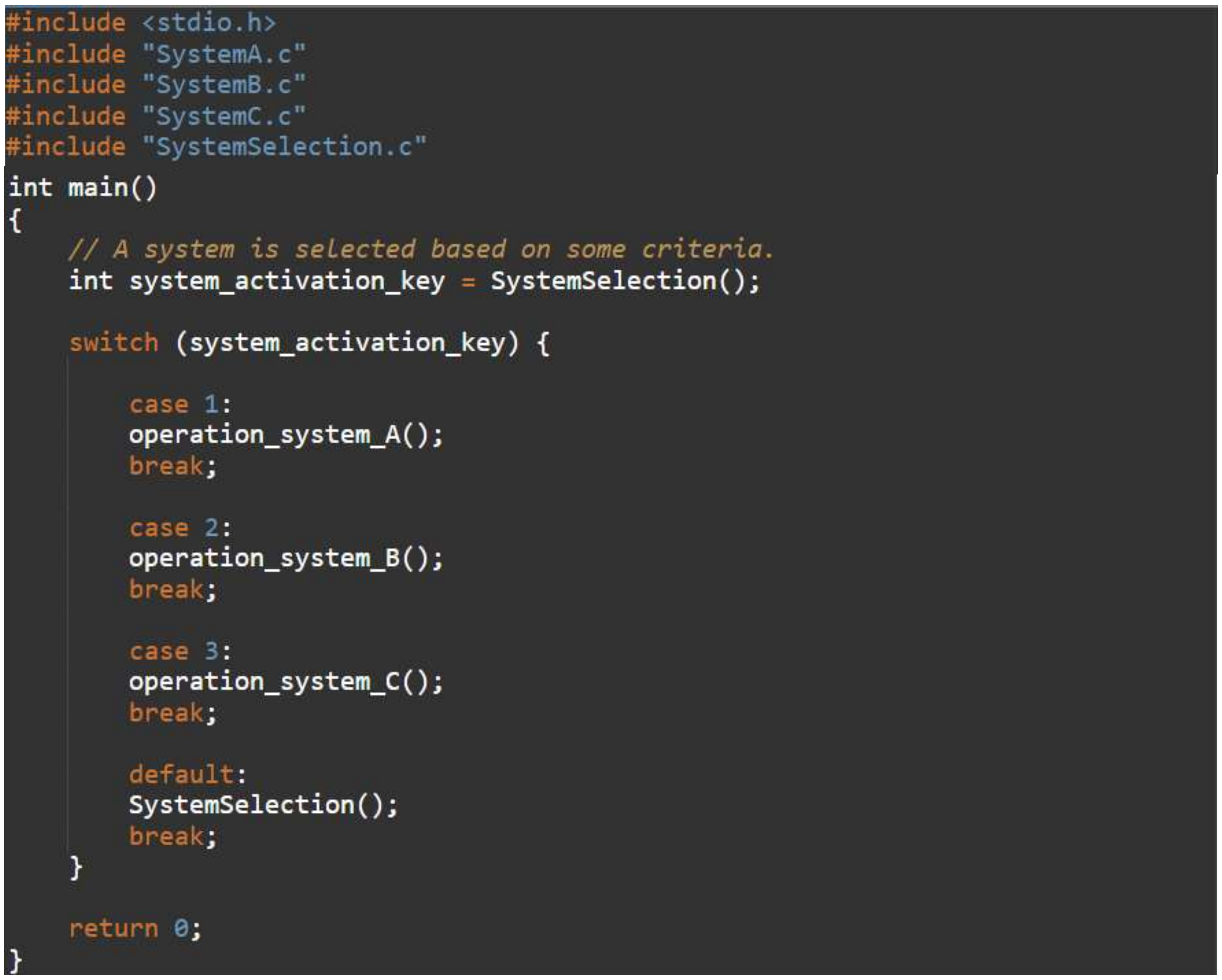


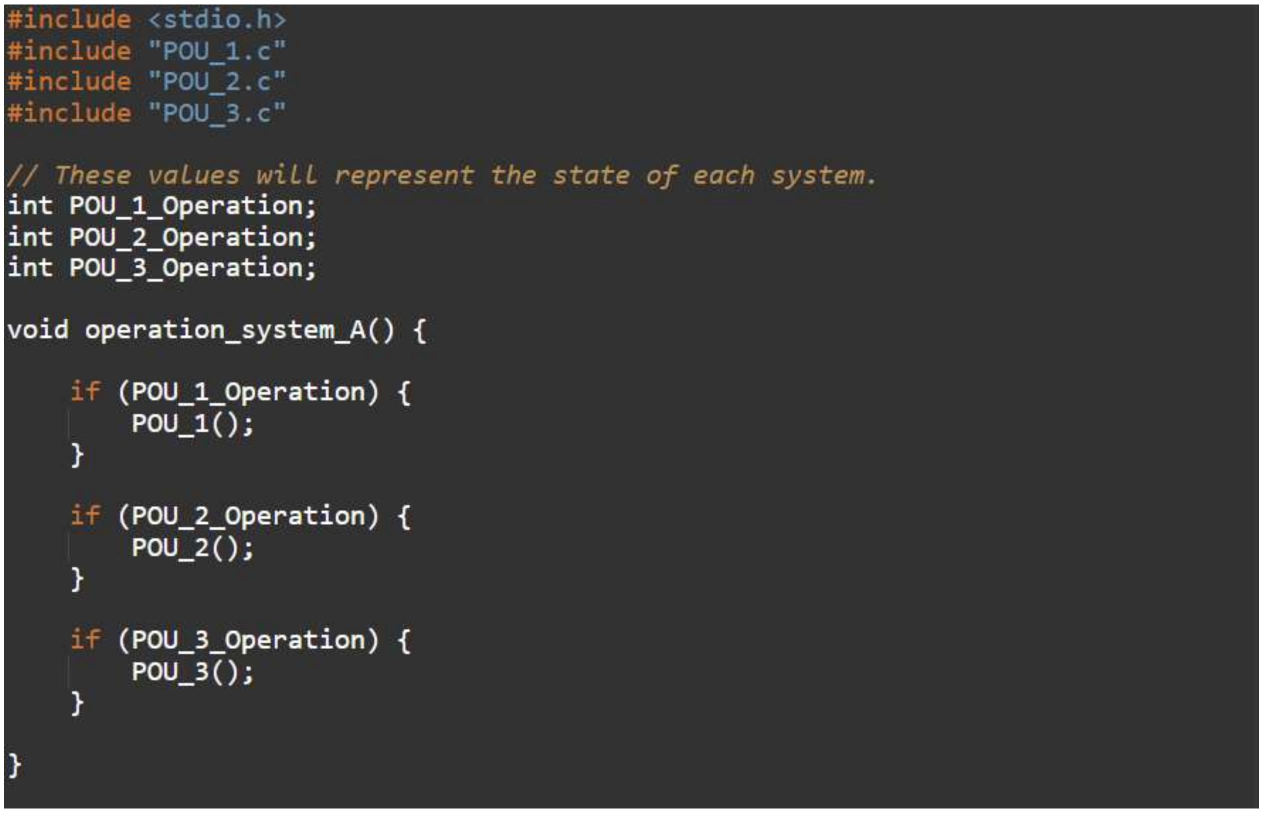
Fig.6e FortyFiveDegreeRotPiston_0



```c
#include <stdio.h>
#include "SystemA.c"
#include "SystemB.c"
#include "SystemC.c"
#include "SystemSelection.c"
int main()
{
    // A system is selected based on some criteria.
    int system_activation_key = SystemSelection();

    switch (system_activation_key) {

        case 1:
        operation_system_A();
        break;

        case 2:
        operation_system_B();
        break;

        case 3:
        operation_system_C();
        break;

        default:
        SystemSelection();
        break;
    }

    return 0;
}
```

Fig.7 System Selection

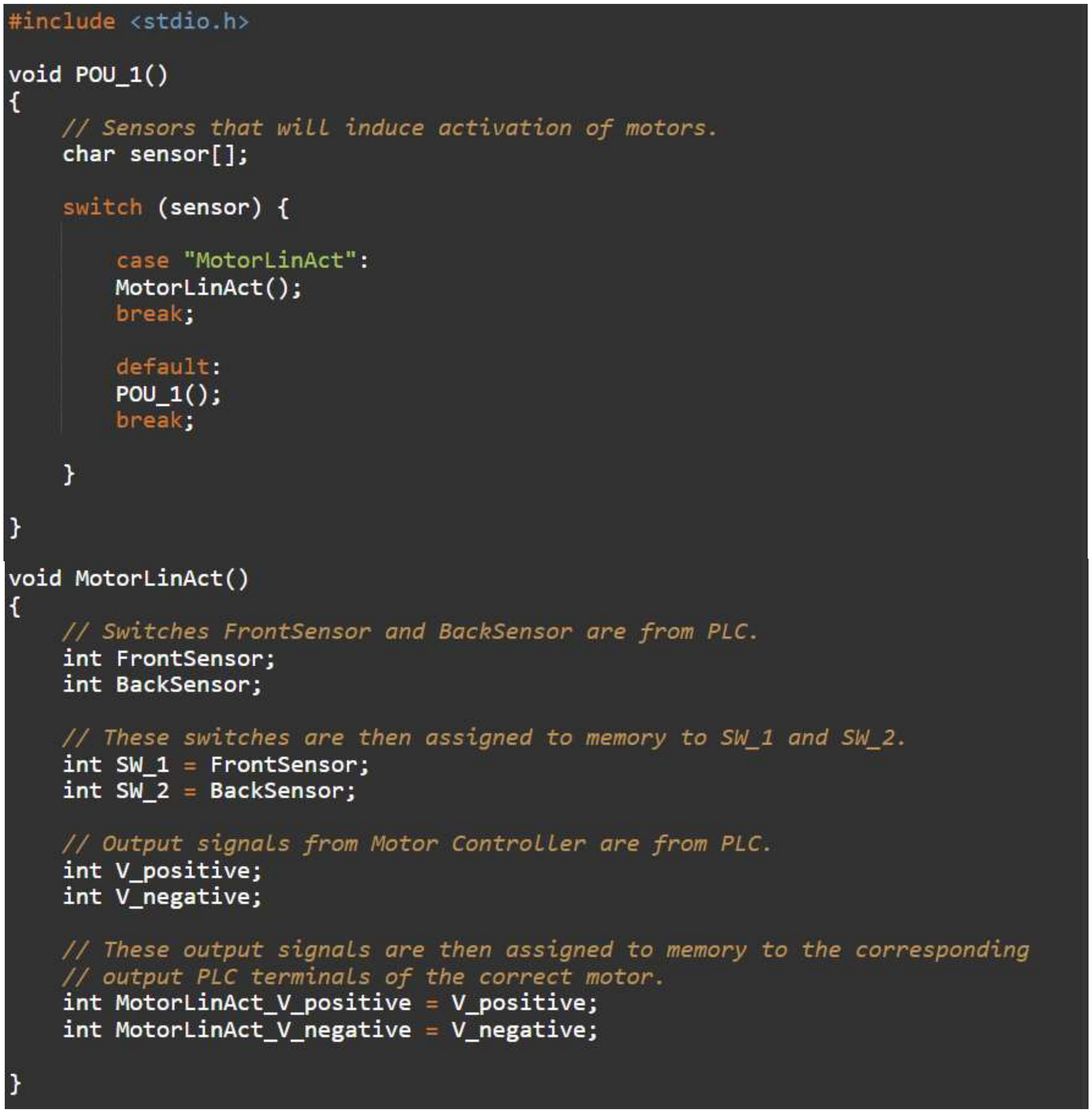
Either manually or through some algorithm a system is selected. The names operation_system A, operation_system B, operation_system C describe the address of the system we would like to program.



```
#include <stdio.h>
#include "POU_1.c"
#include "POU_2.c"
#include "POU_3.c"

// These values will represent the state of each system.
int POU_1_Operation;
int POU_2_Operation;
int POU_3_Operation;

void operation_system_A() {

    if (POU_1_Operation) {
        POU_1();
    }

    if (POU_2_Operation) {
        POU_2();
    }

    if (POU_3_Operation) {
        POU_3();
    }

}
```

Fig.7a System State Selection

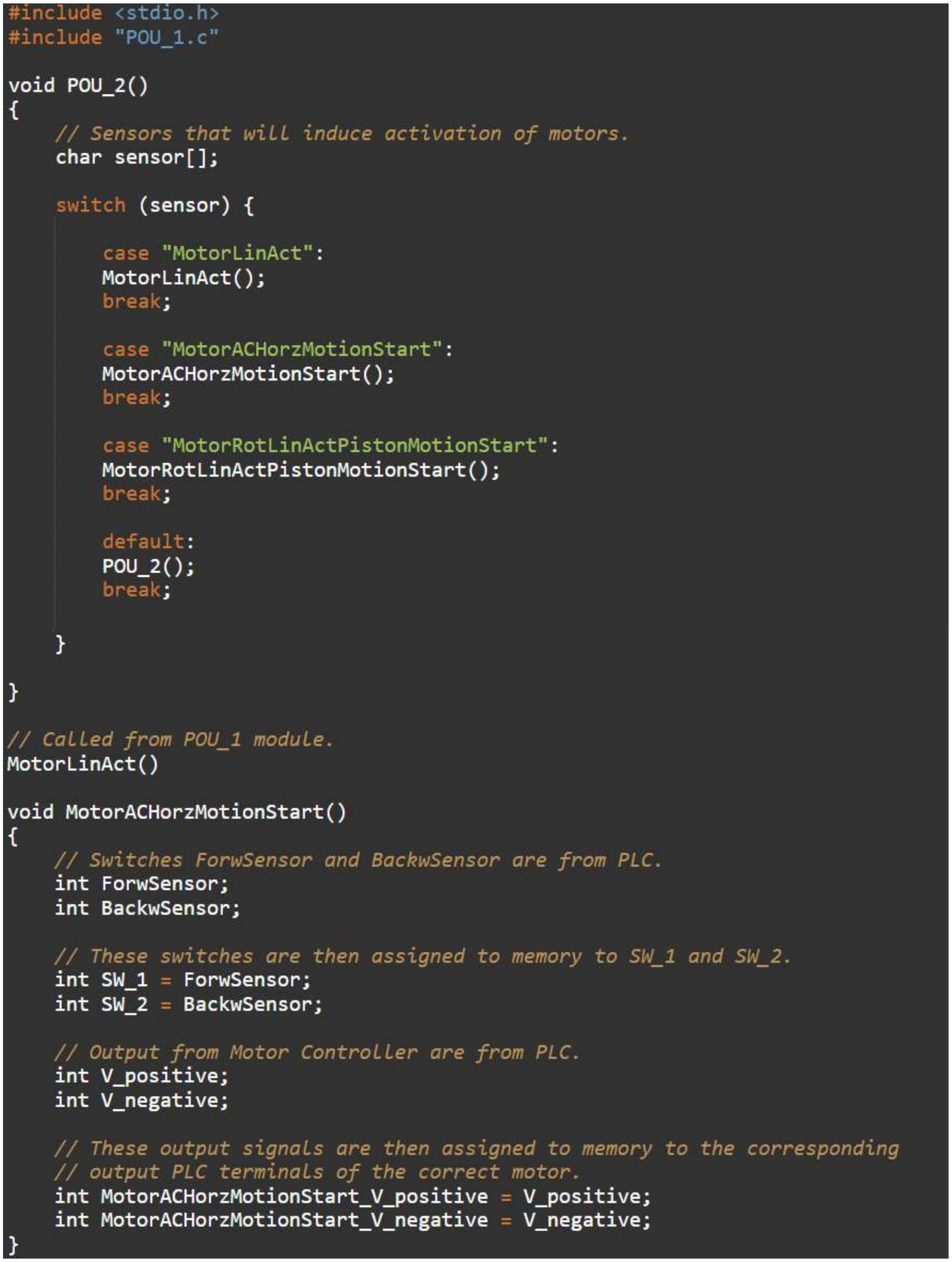
After a system is selected we now have to define the states or phases within the system. For this example project they are identified through the signals POU_1_Operation, POU_2_Operation and POU_3_Operation.

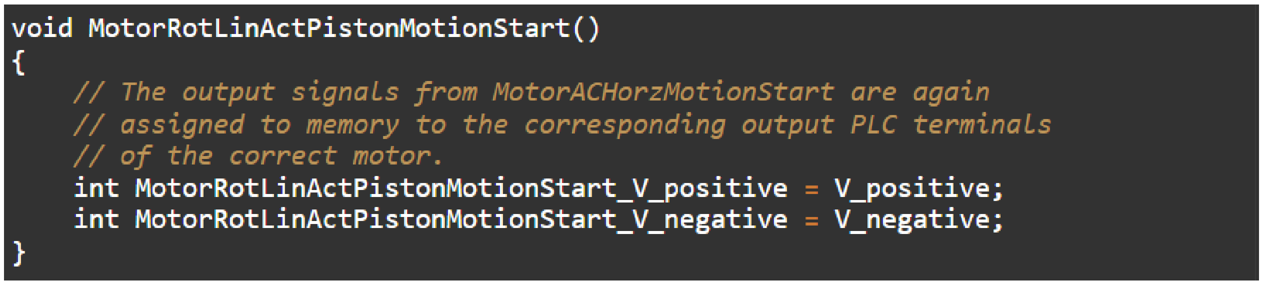


```
#include <stdio.h>

void POU_1()
{
    // Sensors that will induce activation of motors.
    char sensor[];

    switch (sensor) {

        case "MotorLinAct":
        MotorLinAct();
        break;

        default:
        POU_1();
        break;

    }

}
void MotorLinAct()
{
    // Switches FrontSensor and BackSensor are from PLC.
    int FrontSensor;
    int BackSensor;

    // These switches are then assigned to memory to SW_1 and SW_2.
    int SW_1 = FrontSensor;
    int SW_2 = BackSensor;

    // Output signals from Motor Controller are from PLC.
    int V_positive;
    int V_negative;

    // These output signals are then assigned to memory to the corresponding
    // output PLC terminals of the correct motor.
    int MotorLinAct_V_positive = V_positive;
    int MotorLinAct_V_negative = V_negative;

}
```

Fig.7b POU_1 Module

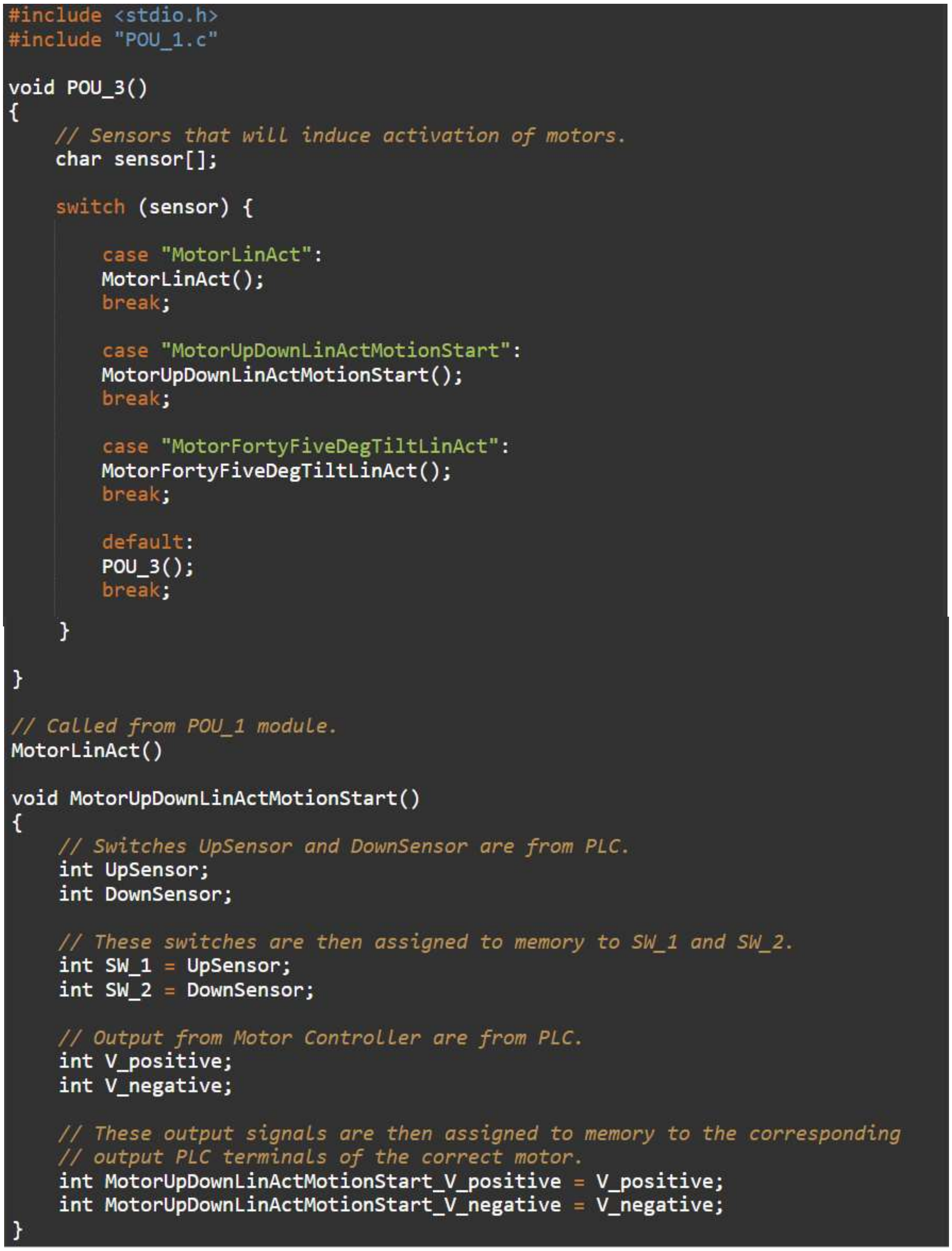
Within each instance, state or module of the system the motor controller terminals V+, V- or V_positive, V_negative are transferred to the proper terminals of the correct motor with respect to the sensor signals in the POU function and the switches from the PLC.


```
#include <stdio.h>
#include "POU_1.c"

void POU_2()
{
    // Sensors that will induce activation of motors.
    char sensor[];

    switch (sensor) {

        case "MotorLinAct":
        MotorLinAct();
        break;

        case "MotorACHorzMotionStart":
        MotorACHorzMotionStart();
        break;

        case "MotorRotLinActPistonMotionStart":
        MotorRotLinActPistonMotionStart();
        break;

        default:
        POU_2();
        break;

    }

}

// Called from POU_1 module.
MotorLinAct()

void MotorACHorzMotionStart()
{
    // Switches ForwSensor and BackwSensor are from PLC.
    int ForwSensor;
    int BackwSensor;

    // These switches are then assigned to memory to SW_1 and SW_2.
    int SW_1 = ForwSensor;
    int SW_2 = BackwSensor;

    // Output from Motor Controller are from PLC.
    int V_positive;
    int V_negative;

    // These output signals are then assigned to memory to the corresponding
    // output PLC terminals of the correct motor.
    int MotorACHorzMotionStart_V_positive = V_positive;
    int MotorACHorzMotionStart_V_negative = V_negative;
}
```

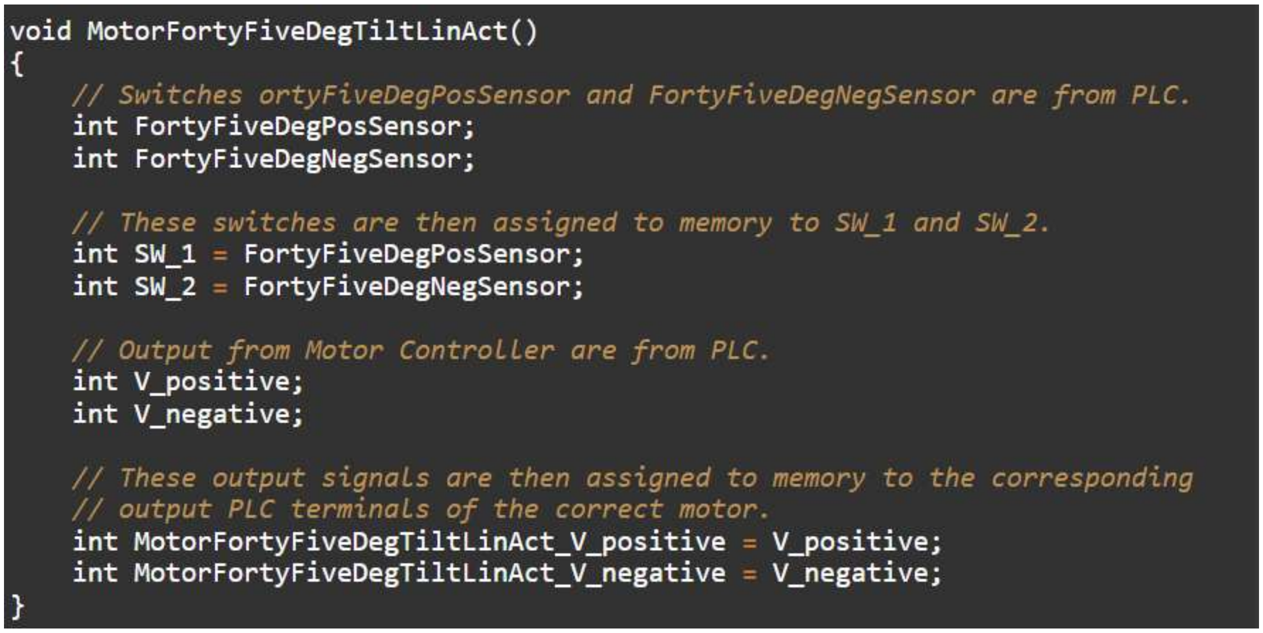


```
void MotorRotLinActPistonMotionStart()
{
    // The output signals from MotorACHorzMotionStart are again
    // assigned to memory to the corresponding output PLC terminals
    // of the correct motor.
    int MotorRotLinActPistonMotionStart_V_positive = V_positive;
    int MotorRotLinActPistonMotionStart_V_negative = V_negative;
}
```

Fig.7c POU_2 Module

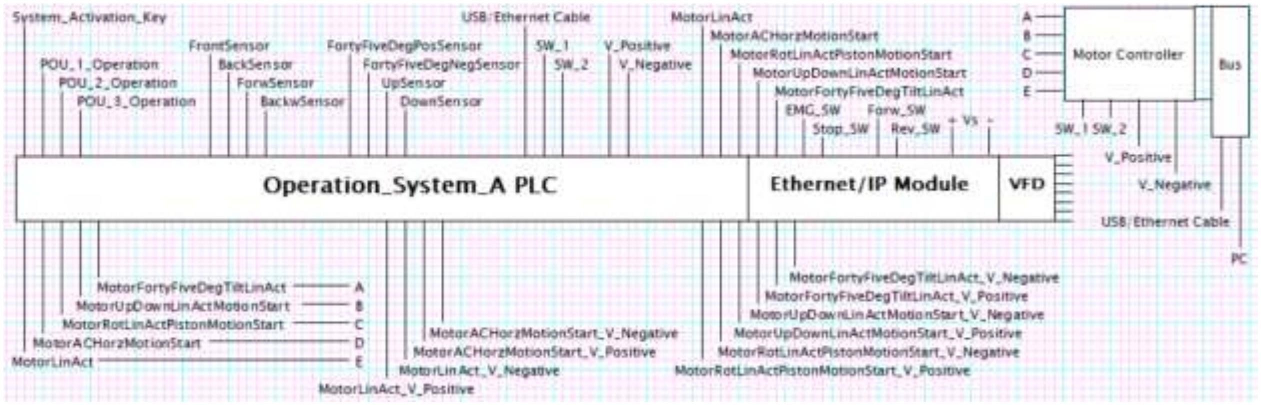


```c
#include <stdio.h>
#include "POU_1.c"

void POU_3()
{
    // Sensors that will induce activation of motors.
    char sensor[];

    switch (sensor) {

        case "MotorLinAct":
        MotorLinAct();
        break;

        case "MotorUpDownLinActMotionStart":
        MotorUpDownLinActMotionStart();
        break;

        case "MotorFortyFiveDegTiltLinAct":
        MotorFortyFiveDegTiltLinAct();
        break;

        default:
        POU_3();
        break;
    }

}

// Called from POU_1 module.
MotorLinAct()

void MotorUpDownLinActMotionStart()
{
    // Switches UpSensor and DownSensor are from PLC.
    int UpSensor;
    int DownSensor;

    // These switches are then assigned to memory to SW_1 and SW_2.
    int SW_1 = UpSensor;
    int SW_2 = DownSensor;

    // Output from Motor Controller are from PLC.
    int V_positive;
    int V_negative;

    // These output signals are then assigned to memory to the corresponding
    // output PLC terminals of the correct motor.
    int MotorUpDownLinActMotionStart_V_positive = V_positive;
    int MotorUpDownLinActMotionStart_V_negative = V_negative;
}
```



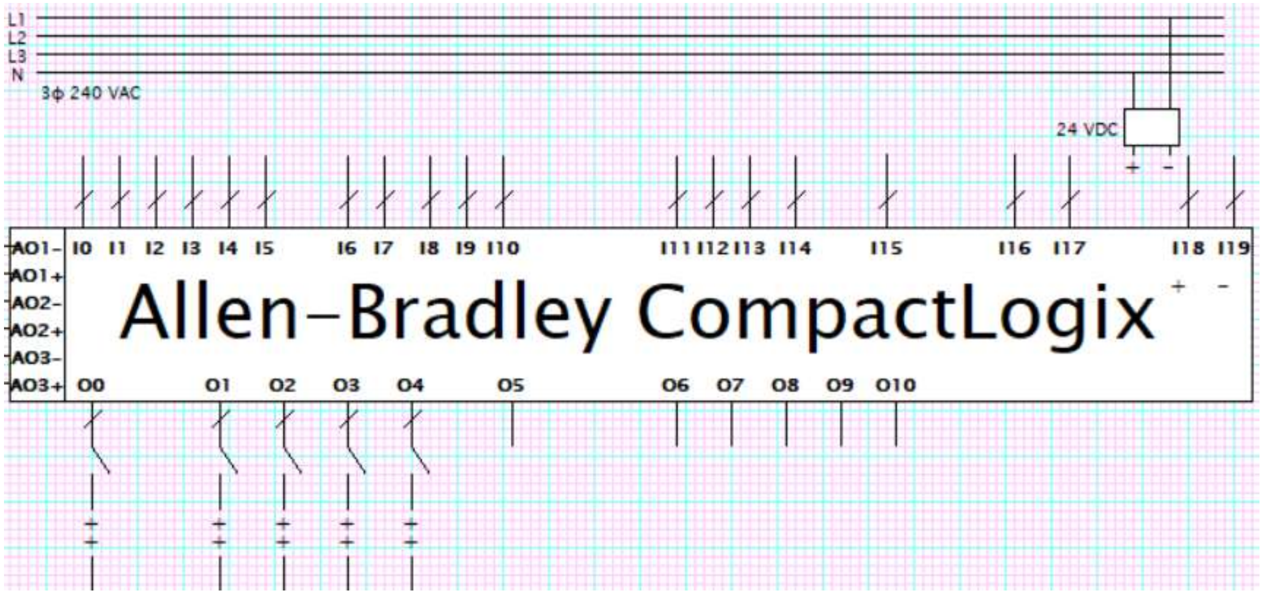
```
void MotorFortyFiveDegTiltLinAct()
{
    // Switches ortyFiveDegPosSensor and FortyFiveDegNegSensor are from PLC.
    int FortyFiveDegPosSensor;
    int FortyFiveDegNegSensor;

    // These switches are then assigned to memory to SW_1 and SW_2.
    int SW_1 = FortyFiveDegPosSensor;
    int SW_2 = FortyFiveDegNegSensor;

    // Output from Motor Controller are from PLC.
    int V_positive;
    int V_negative;

    // These output signals are then assigned to memory to the corresponding
    // output PLC terminals of the correct motor.
    int MotorFortyFiveDegTiltLinAct_V_positive = V_positive;
    int MotorFortyFiveDegTiltLinAct_V_negative = V_negative;
}
```

Fig.7d POU_3 Module



System_Activation_Key
POU_1_Operation
POU_2_Operation
POU_3_Operation
FrontSensor
BackSensor
ForwSensor
BackwSensor
FortyFiveDegPosSensor
FortyFiveDegNegSensor
UpSensor
DownSensor
USB/Ethernet Cable
SW_1
SW_2
V_Positive
V_Negative
MotorLinAct
MotorACHorzMotionStart
MotorRotLinActPistonMotionStart
MotorUpDownLinActMotionStart
MotorFortyFiveDegTiltLinAct
EMG_SW
Forw_SW
Stop_SW
Rev_SW
+ Vs -
Motor Controller
Bus
Operation_System_A PLC
Ethernet/IP Module
VFD
PC
MotorFortyFiveDegTiltLinAct A
MotorUpDownLinActMotionStart B
MotorRotLinActPistonMotionStart C
MotorACHorzMotionStart D
MotorLinAct E
MotorACHorzMotionStart_V_Negative
MotorACHorzMotionStart_V_Positive
MotorLinAct_V_Negative
MotorLinAct_V_Positive
MotorFortyFiveDegTiltLinAct_V_Negative
MotorFortyFiveDegTiltLinAct_V_Positive
MotorUpDownLinActMotionStart_V_Negative
MotorUpDownLinActMotionStart_V_Positive
MotorRotLinActPistonMotionStart_V_Negative
MotorRotLinActPistonMotionStart_V_Positive

Fig.8 Overview of PLC Controlled by MUX Motor Controller for this Example Project



L1
L2
L3
N
3ϕ 240 VAC
24 VDC
I0 I1 I2 I3 I4 I5 I6 I7 I8 I9 I10 I11 I12 I13 I14 I15 I16 I17 I18 I19
AO1- AO1+ AO2- AO2+ AO3- AO3+
Allen-Bradley CompactLogix
O0 O1 O2 O3 O4 O5 O6 O7 O8 O9 O10

Fig.9 Allen-Bradley CompactLogix can be used for this Implementation