The Future of Engineering Education
2024 Annual Conference & Exposition
Oregon Convention Center
Portland, OR . June 23 - 26, 2024
ASEE
Paper ID #43520

Design of a PLC System Simulator and Application to Teaching Programmable Logic Controller Course Online

Dr. Wenle Zhang, University of Arkansas, Little Rock

Dr. Wenle Zhang is an Associate Professor of Electrical and Electronics Program in the Dept. of Engineering Technology of University of Arkansas at Little Rock since 2007. He received his B.S. and M.S. from Shandong Univ. of Sci. and Tech. in 1983 and Shanghai Univ. of Sci. and Tech. in 1986, respectively. He received his Ph.D. in Electrical Engineering from Ohio University in 2000. Prior to pursuing his Ph.D. at Ohio University, he was employed with Rockwell Automation, a famous industrial automation supplier, as a control engineer for more than 5 years. He has experience in PLC application, industrial control networking, control software development. From 2001 to 2006, Dr. Zhang was an assistant professor in the School of Electrical Engineering and Computer Science at Ohio University. Dr. Zhang's current research interests include discrete event system, industrial automation, system identification and neural networks.

Design of a PLC System Simulator and Application to Teaching Programmable Logic Controller Course Online

Wenle Zhang
School of Eng. and Eng. Tech., University of Arkansas at Little Rock

A. Introduction

A.1 The Basic Concept of Programmable Logic Controllers

Programmable Logic Controllers (PLC) were first built for automation of the automobile industry to provide flexible, rugged, and easy programmable controllers to replace hard-wired. hardware and software used to perform control functions. Because PLCs can offer high speed logic processing, high reliability, flexibility, easy programming and troubleshooting, they are widely used in industrial automation of both sequential controls (such as factory assembly lines) and process controls (such as waste water treatment) across most industries, such as manufacturing, mining, metal, power and etc. Traditionally, a PLC is considered to consist of three major basic sections: the power supply (high reliability), the central processing unit (CPU) and the Input and Output (I/O) interface system, see Fig. 1.

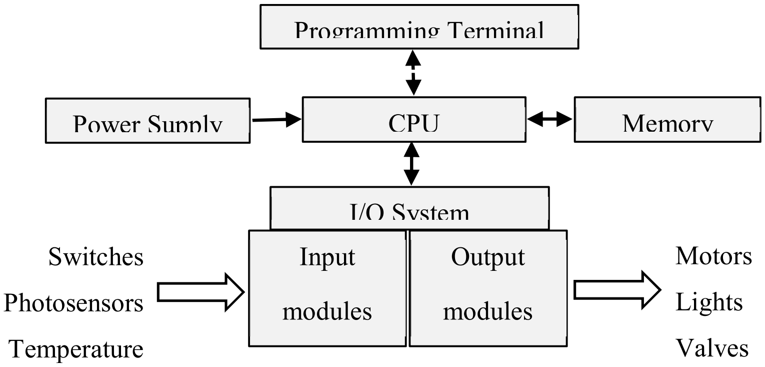


Fig 1. The PLC System Components

A.2 Design Motivation and Goal of the PLC System Simulator

For the past several decades, the programmable logic controller has been one of the major driving forces in industrial automation and many colleges and universities offer Programmable Logic Controllers or Industrial Automation course. Our Engineering Technology program has been offering the Robotics and Programmable Logic Controller (PLC) course for many years. At present, our PLC lab has 10 Allen-Bradley CompactLogix 1769-L33ERM series PLCs each paired with a Windows workstation which has Allen-Bradley programming software Studio 5000 installed. Each workstation also has a custom-built lab console made of Industrial Control Components, shown in Fig 2. As a result of COVID19 since 2020 spring, our students had not been able to access the lab as a result of campus closures and stay-at-home orders. While now our campus is open, per university assessment, our PLC lab capacity is limited to 6 students in a class that typically has 20 students in a full class. Therefore, a virtual PLC lab is more desirable than ever for helping the students to master the knowledge.

While there are some examples of PLC simulation software available on the market, they are based on older or legacy PLC products developed over a decade ago, and what is needed to have something matching the state-of-the-art modern PLCs such as the Allen Bradley ControlLogix 5000 series PLCs equipped in our PLC lab. So around late spring of 2020, the project of building our own virtual PLC lab – Converting the mainly hardware lab in Fig 2(a) to a software only virtual lab in Fig 2(b) – was underway with the initial aim of having an integrated Soft PLC and I/O components lab console simulator (termed the PLC System Simulator) and to have a working prototype in the shortest period of time possible to meet the immediate needs of the coming semester. The plan was to build the software with four major components:

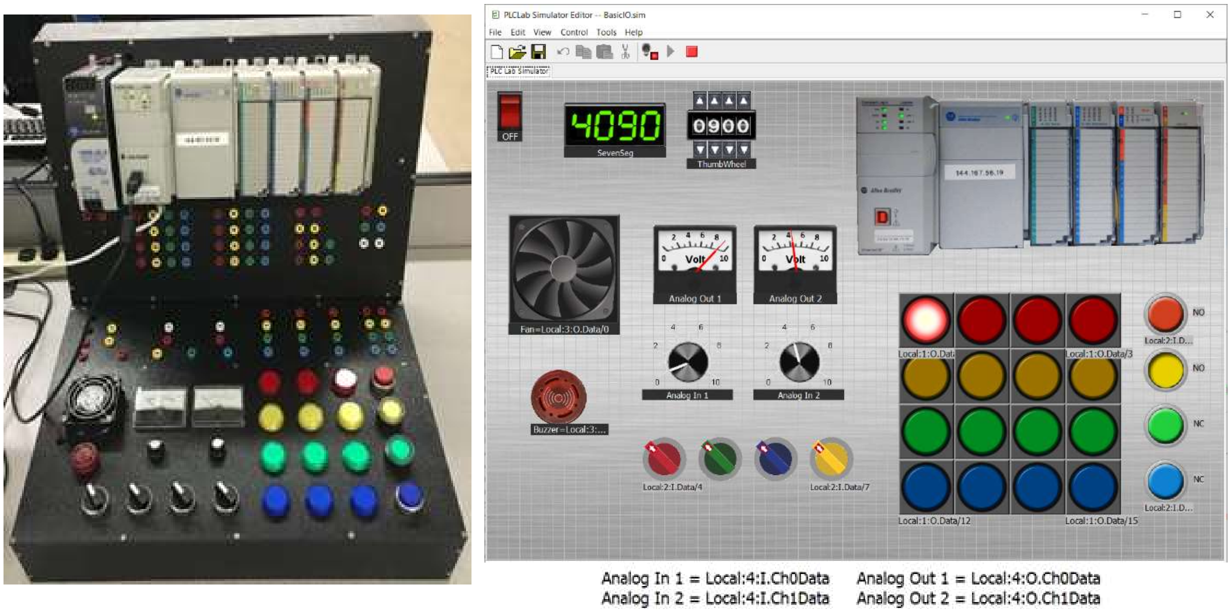


Fig 2. (a) The Lab PLC and Components

(b) The Lab Console Simulator

i) The Soft PLC should be programmed in Ladder Diagram language (should support a subset of most commonly used instructions, but include advanced instructions available in modern PLCs, such as, log, exponential and trigonometric instructions) and should be able to execute the program.
ii) The ladder diagram program editor should have a look and feel similar to Allen-Bradley's RSLogix 5000, which can also be used to monitor the ladder logic program with animations if the Soft PLC is in Run Mode.
iii) The data tag editor and monitor should support data as objects partially conformant to IEC 61131 [7], just as all Allen-Bradley 5000 series PLCs are based on Object-Oriented Programming (OOP) where data objects are defined as tags.
iv) The 3D I/O simulator should be similar to the lab console as shown in Fig 2 (a) vs (b). It should have: 4 animated switches (double click to turn ON/OFF); 4 animated push buttons (click to push); 16 lights (4 x 4 light matrix) that can turn ON (illuminate) or OFF; 1 fan that can turn ON (animated turning) or OFF; 1 buzzer that can turn ON (with sound) or OFF; 2 animated analog dials that can turn; 2 animated analog meters whose needle turns; 1 animated 4-digit BCD 7-segment display; and 1 animated 4-digit BCD thumbwheel switch.

Plus, the Soft PLC and the I/O Simulator should be integrated such that all 3D animated components on the simulator are configured to have matching virtual I/O addresses which can be controlled by the ladder logic program running in the Soft PLC.

A few months of the project development, including dedicated time and effort to the analysis and design of the software, going over a large number of Allen-Bradley's manuals and references for the ContolLogix series PLCs [1-5], IEC 61131 document [7] and coding and debugging, resulted in a working Virtual PLC Lab (called **zPLC Lab**) prototype. The first version of this software was experimentally used in our summer 2020 PLC online class. During the term, by using the software, our students conducted 12 normal labs on various ladder logic instructions and 2 programming projects: The traffic light control (used 2 sets of Red, Yellow, Green lights on the Basic I/O Simulator) and the elevator project (assigned to control up to 3 floors to reduce complexity on the preliminary Elevator Simulator). It was quite successful, with everyone enjoying the software, without which the class would not have even been possible. The zPLC Lab has since been used in our PLC classes.

B. Design of Architecture and Functions

The architecture of the zPLC Lab is summarized in the diagram shown in Fig. 3, which consists of six major blocks. Four blocks correspond to the aforementioned major components: The Soft PLC Run-Time Engine, the Ladder Diagram Editor and Monitor, the Data/Tag Editor and Monitor, and the I/O Simulator. There is also a GUI component overlaying each of the four blocks (not shown in the diagram). In addition, there are two more blocks corresponding to the ladder logic program and data tags database of a user project. These two blocks are the two major data objects, which can be more appropriately considered as data containers in Object Oriented Programming (OOP) terms.

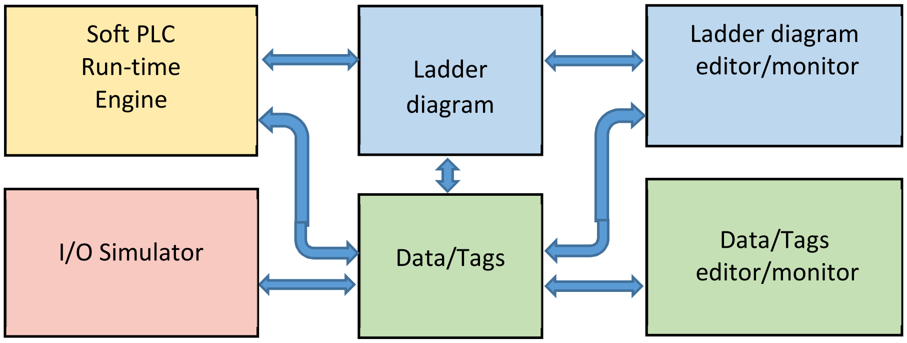


Fig 3. Architecture of the zPLC Lab

i) The Soft PLC Run-Time Engine component implements the runtime engine of the zPLC Lab software. Virtual I/O modules can be added to the PLC with built-in I/O addresses for the user to program and use. It interacts with the user developed ladder diagram program and its associated data and tags defined by the user, as indicated by the double arrows in Fig 3. It works in two modes: (i) Edit mode – when the ladder logic diagram is being edited. This is the default mode or by pressing the "Stop" button. (ii) Run mode – when a working ladder logic diagram program is completed, it can be put into run mode by pressing the "Play" button.

ii) The Ladder Diagram Editor and Monitor (LDEM): As the ladder diagram editor, a ladder diagram routine (a program unit) can be entered, modified and saved. As a common GUI practice to allow for ease of program editing, a windows menu and toolbars along with copy, cut and paste functionality has been implemented. A ladder diagram program consists of one or more ladder logic routines. A routine consists of one or more rungs, and a rung consists of one or more instructions. The GUI presentation of a routine consists of two vertical rails in the main panel, with rungs shown in-between as horizontal lines on which instructions are populated across. Routines can be added to the program. Rungs can be added to a routine being edited using the new rung button from the toolbar (see Fig 4.). An instruction can be selected and added to the rung being edited by using the instructions' button on the toolbar.

As the ladder diagram monitor, once the PLC is put in run mode, the ladder program can no longer be edited. LDEM acts as a live ladder diagram monitor and animated with live data fed from the I/O simulator. The two vertical rails will glow in green along with the active instructions on a rung. In run mode, the program can be monitored for its behavior to assist in debugging and troubleshooting of the program.

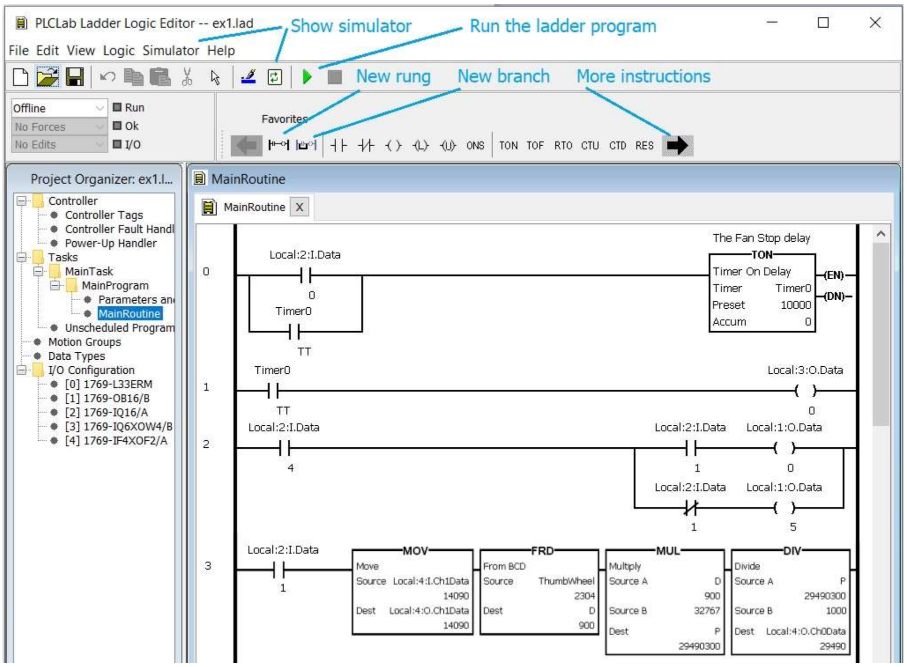


Fig 4. The ladder diagram editor and program for example 1

iii) The Ladder Diagram Program is the target data object of the LDEM. A user project for an Allen-Bradley CompactLogix Controller allows for multiple tasks (multi-tasking), where each task can have multiple programs and each program can have its own set of local data tags and set of routines. It's not our desire to implement all the functions, but rather to have the basic functions needed to train our students, so this component was made to allow for only one task per project and only one program per task. The program consists of one or more data tags and one or more ladder logic routines, with each routine consisting of one or more rungs, each rung consisting of one or more instructions, and each instruction consisting of zero, one, or more data tags as its operand(s). The set of instructions implemented is a subset of most commonly used instructions as summarized in the following. No instruction which is hardware-dependent is included. For a complete list of implemented ladder logic instructions by the zPLC, refer to the user's manual.

a) **Relay Logic**: This type of instructions takes a Logic tag as its operand, such as, Examine if closed (XIC), Examine if open (XIO), Output energize (OTE).
b) **Timer and Counter**: This type of instruction takes a Timer or Counter tag as its operand, whose status bits, .TT, .DN and .EN are accessible, such as, Timer on delay (TON), Timer off delay (TOF), Counter up/down (CTU/CTD).

c) **Data Manipulation**: This type of instruction takes one or more Numeric tags, including math, compare, move/logic, conversion and advanced instructions, and are further categorized as unary operands, such as Clear (CLR); binary operands, such as, Move (MOV), Convert to BCD (TOD), Less than or equal to (LEQ); and ternary operands, such as Add (ADD), Multiply (MUL), Masked Equal (MEQ)

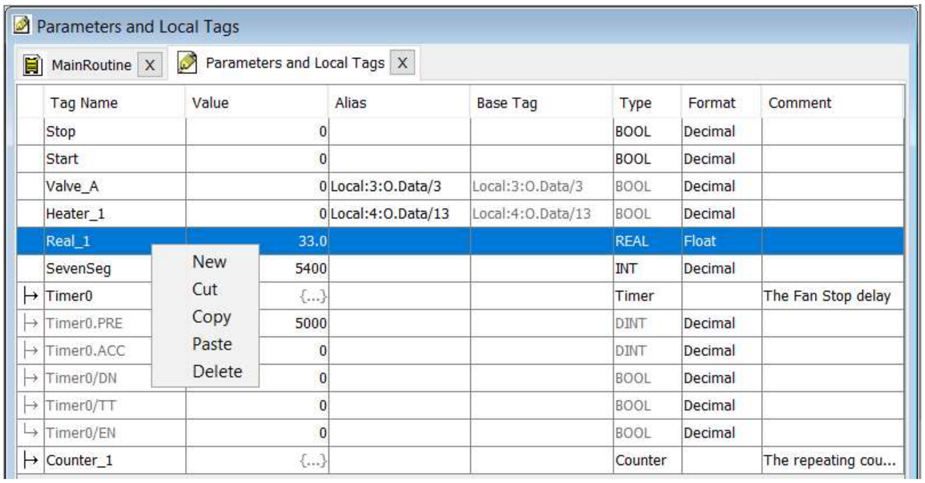
d) **Program Control**: This type of instruction takes one or more operands, or no operand (such as MCR and RET); the operand is unique in that it can be a Label for JMP/LBL instruction, or a routine name for JSR.

e) **Shift Register**: This type of instruction takes an array with its initial element, a Control tag, a Numeric source or destination tag or a Logic source for bit shift, Such as, BSL and BSR.

f) **Sequencers**: This type of instruction takes an array with its initial element, a Control tag, a Numeric source or destination tag for word shift, and an optional mask, a Numeric tag. Such as, SQI and SQO.

iv) The Data/Tag Editor and Monitor (DTEM): The data/tag editor provides a table-like editor where all data and/or tags can be defined, modified and saved, as shown in Fig 5. In addition to all basic data types, structured data types, including Timer, Counter, Control, and Arrays of all basic data types are supported. Each tag can have properties which are presented in the table's columns, including Icon, Name, Value, Alias/Base, Type, Format and Comment.

DTEM acts as the data/tag monitor once the PLC is in run mode. While the ladder logic program is being executed, data changes dynamically by the ladder logic program or by the I/O simulator. These dynamic data changes will in turn affect program execution. Data values can also be changed by the user in run-time for the purposes of debugging and troubleshooting.



	Tag Name	Value	Alias	Base Tag	Type	Format	Comment
	Stop	0			BOOL	Decimal	
	Start	0			BOOL	Decimal	
	Valve_A	0	Local:3:O.Data/3	Local:3:O.Data/3	BOOL	Decimal	
	Heater_1	0	Local:4:O.Data/13	Local:4:O.Data/13	BOOL	Decimal	
	Real_1	33.0			REAL	Float	
	SevenSeg	5400			INT	Decimal	
↦	Timer0	{...}			Timer		The Fan Stop delay
↦	Timer0.PRE	5000			DINT	Decimal	
↦	Timer0.ACC	0			DINT	Decimal	
↦	Timer0/DN	0			BOOL	Decimal	
↦	Timer0/TT	0			BOOL	Decimal	
↳	Timer0/EN	0			BOOL	Decimal	
↦	Counter_1	{...}			Counter		The repeating cou...

Fig 5. The data tag editor/monitor

v) The Data Tags Database is the target data object of the DTEM. Allen-Bradley 5000 series PLCs are based on Object-Oriented Programming where data are treated as objects called tags. A user project always has a set of controller tags predefined based on the virtual I/O modules defined. It also has a set of user created data tags for the program. There are several data types or tag types:

a) Basic data types:

BOOL ---- Boolean, 1 bit; SINT ----- Short integer, 8 bit; INT ------- Integer, 16 bit; DINT ----- Double integer, 32 bit; and REAL ----- Floating point, 32 bit

b) Structured data types:
Timer, Counter, Control (TCC) ---- Each consists of 3 DINT members: Statuses, Preset value (or Length) and Accumulated value (or Position).
c) Array types, where an array can be defined for basic data types, e.g., INT[8] --------- 8 elements of type INT, index 0 – 7

vi) The Lab Console Basic I/O Simulator is essential for teaching and learning PLC online, because no physical devices will be available for the Soft PLC to connect to. The zPLC Lab integrates the Soft PLC and the I/O Simulator by data sharing. The Soft PLC maintains all the data/tags and shares some of the data/tags (which can be configured). The I/O Simulator is made to be configurable and expandable through the SDK for developers. The I/O Simulator now has a built-in Basic IO Simulator (shown in Fig 2 (b)), and more simulators are being built. The Basic IO simulator has the following simulated I/O components:

a) 4 animated switches, double click turn ON/OFF, a logic 1/0 will be set to its I/O address.
b) 4 animated push buttons, click to press, a logic 1/0 will be sent to its I/O address.
c) 16 (4 x 4 light matrix) lights which turn ON/OFF (or illuminate/dim) once they receive logic 1/0 at their I/O addresses.
d) 1 fan which turns ON (animated turning)/OFF once it gets a logic 1/0 at its I/O address.
e) 1 buzzer, it turns ON (sounding an alarm)/OFF once it gets a logic 1/0 at its I/O address.
f) 2 animated analog dials (drag mouse to turn dial), providing INT data ranges 0 – 32767 which can be scaled. The data will be sent to its I/O address.
g) 2 animated analog meters (needle turns as source data changes), displaying INT data ranges 0 – 32767 which can be scaled. The data will be received from its I/O address.
h) 1 animated 4-digit BCD 7-segment display, displaying source data.
i) 1 animated 4-digit BCD thumbwheel switch, providing source data.

I/O addresses for all I/O components on the Basic I/O Simulator are configured as I/O tags defined in the Soft PLC. They can also be configured as any other type of tags. For example, the ThumbWheel and SevenSeg are internal tags of INT for the BCD thumbwheel switch and the BCD 7-segment display.

vii) Implementation in Java: Because of the object modeling adopted in the design of the software whose major components are heavily GUI-based, the object oriented programming language Java is the perfect candidate for the implementation of the software. With the model based design, coding and debugging is a straightforward and smooth process which helped to achieve the completion of the 1st version in the shortest time possible. The resulting zPLC Lab software has a GUI with a look and feel familiar to users of most modern Windows-based software, as shown in Fig 4.

C. Application Examples

Example 1 – Fan and Light Control: configured as shown in Fig. 2, a simple ladder diagram program will be created to run the Fan for a given amount of time, that is, 10 seconds every time the red push button is pressed. Start the zPLC Lab software, and the GUI window should show up as in Fig 4. Double click on MainRoutine in the project organizer to open the LDEM which should have one empty rung and the End rung. Following Fig 4., add an Examine If Closed instruction to the empty rung 0 by clicking the ┤├ button and double-clicking on the added instruction and entering tag Local:2:I.Data/0, and add a Timer On Delay instruction by clicking

TON button with tag Timer0 of Timer type with a preset value 10000 in milliseconds (for 10 seconds). Add a new rung by clicking on the New Rung button. Put in the rest of all instructions for rung1 and 2, as shown by Fig 4. Click the refresh button to show the Basic I/O Simulator, as shown in Fig 2., and run both the simulator and the ladder diagram program by clicking the green play button. Each time the red push button is pressed, the Fan will run for 10 seconds. Turn on the red switch, repeatedly press and release the yellow push button, the red and yellow lights will be lit alternately.

Example 2 – Adding Analog and BCD Control. First, add the rung 3, as shown in Fig 4, to the program. The analog data for both the dial and meter is of 16-bit INT type with a range 0 – 32767. Second, proper scaling to engineering unit is desirable, which can be accomplished through a linear function, say the first meter (Tag M1 alias to Local:4:O.Ch0Data) is to represent a desired temperature with a range 0 – 1000°C as tag D taken from the thumb wheel switch and converted from BCD, then M1 = D*32767/1000, the result from the Divide instruction is sent to M1, where the desired temperature is set by pushing the yellow push button. The second dial (Tag A2 alias to Local:4:I.Ch1Data) is to mimic the measured temperature as tag T sent to the second meter. Finally, run the simulator and set a desired number on the thumb wheel switch, for example 900°C, and press the yellow push button The first meter will reflect this setting by turning the needle to 9. Turn the 2nd dial by dragging the mouse on it to set it to 4, and the value will be reflected on the 2nd meter, as shown in Fig 2(b).

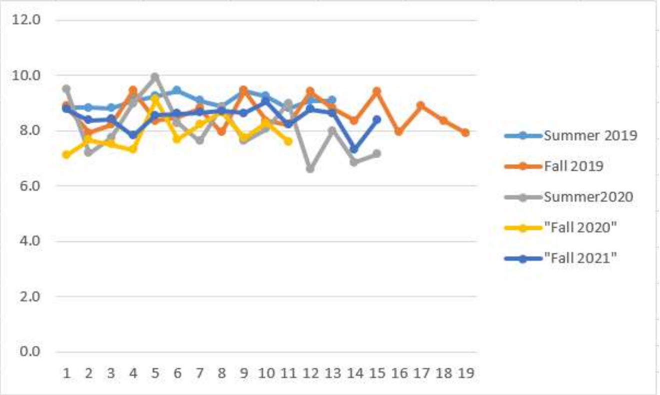


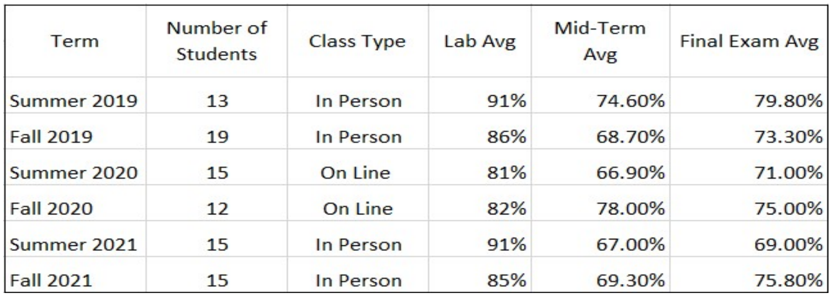
Fig 6. Individual Lab scores over three terms

D. Comparison of Student Performance in In-Person and On-Line Classes

With the use of zPLC Lab, we were able to make our summer 2020 Robotics and Programmable Logic Controllers course offering a complete online class, in which each student got an installation of the zPLC software on their home computer. For this term, all labs were performed by students individually on the zPLC software and the collected lab scores average for each individual were shown by the gray line in Fig 6. For summer 2019 and fall 2019, the course were offered as in-person. For both the two terms, labs were conducted in our PLC lab in which two students worked together as a group on one workstation. Lab average scores were tracked as shown by the blue line for summer 2019 and orange line for fall 2019, respectively in Fig 6. Statistics of labs, the mid-term test and final exam scores are presented in Table 1.

From Table 1, one can observe that student lab performance of summer 2020 is down by 10% comparing to that of summer 2019 and down by 5% against to that of fall 2019. That’s an average of 7.5% performance decrease from in-person to on-line classes. The observation is two-fold here: i) in the in-person lab, students worked in groups of two whereas in the on-line lab, students worked individually, and ii) in the in-person lab, students worked under supervision while in the on-line lab, students were not monitored and some even walked away during the lab time. So, this performance decrease was somewhat expected. Note, the mid-term test score average is down by 5.9% from summer 2019 to fall 2019, and only down by 1.8% from fall 2019 to summer 2020 on-line class. Also, the final exam score average is down by 6.5% from summer 2019 to fall 2019, but only down by 2.3% from fall 2019 to summer 2020 on-line class. Overall, on-line class tends to indicate a few percentage points lower in performance than in-person class.

Table 1. Average scores for Lab, Mid-Term and Final over three terms



Term	Number of Students	Class Type	Lab Avg	Mid-Term Avg	Final Exam Avg
Summer 2019	13	In Person	91%	74.60%	79.80%
Fall 2019	19	In Person	86%	68.70%	73.30%
Summer 2020	15	On Line	81%	66.90%	71.00%
Fall 2020	12	On Line	82%	78.00%	75.00%
Summer 2021	15	In Person	91%	67.00%	69.00%
Fall 2021	15	In Person	85%	69.30%	75.80%

E. Conclusions and Future Work

Since the COVID19 worldwide pandemic started early spring of 2020, our PLC class had to transition to online teaching, labs had to be done in a virtual environment using simulation. To aid our own students and students elsewhere, the proposed PLC simulation software, zPLC Lab, was designed and implemented in Java. The first version of the software was used in our 2020 summer Robotics and PLC course and was evaluated. The zPLC Lab software can be used for online PLC course offerings and for students' home lab assignments. Future improvement may include: (i) on the PLC side, add undo function and add other logic programing language; (ii) on the Simulator side, in addition to the Basic I/O Simulator, the 4-Floor Elevator Simulator needs to be improved and other simulators are to be developed.

References

[1] ControlLogix System User's Manual, Rockwell Automation Publication 1756-UM001P-EN-P - May 2017

[2] Logix5000 Controllers General Instructions Reference Manual, Rockwell Automation Publication 1756-RM003T-EN-P - November2018

[3] Logix 5000 Controllers I/O and Tag Data, Rockwell Automation Publication 1756-PM004H-EN-P - Feb 2018

[4] Logix 5000 Controllers Ladder Diagram, Rockwell Automation Publication 1756-PM008G-EN-P - Feb 2018

[5] Logix 5000 Controllers Tasks, Programs, and Routines, Rockwell Automation Publication 1756-PM005H-EN-P - February 2018

[6] Grady Booch, James Rumbaugh, Ivar Jacobson. The Unified Modeling Language User's Guide. Addison-Wesley, Reading, Mass., 1999.

[7] IEC 61131

[8] Java reference: https://docs.oracle.com/javase/8/docs/api/overview-summary.html