Design of remote electro-pneumatic control system using microcontroller

Design of remote electro-pneumatic control system using microcontroller

M Y Stoychitch and B Z Knezevic

University of Banja Luka, Faculty of Mechanical Engineering, Vojvode Stepe Stepanovica 71, 78000 Banja Luka, Republic of Srpska, Bosnia and Herzegovina

E-mail: mihajlo.stojcic@mf.unibl.org

Abstract. In this paper we considered design and realization of remote control system that is used for control of movement of pneumatic cylinders. It is assumed that each cylinder is controlled by electro pneumatic monostable 5/2 (or 4/2) valve. Also, each cylinder has a limit switches in its two final positions. The control system is implemented as a web application, where the server is realized using the ESP8266 or ESP32 microcontroller and Arduino software, while as the client uses some Windows or Android device. The use of this control system is very simple, so (after client-server connection) it is only necessary on the client side (according to the already defined rules) to enter the desired operation sequence of the cylinders. It is possible, also, to control the times between the adjacent movements of the all cylinders. Software and necessary documentation for practical realization of described solution is given in this paper.

1. Introduction

Electro-pneumatic systems (EPS) are very common in industry, especially in industry of food and beverages as well as in industry of furniture. Industrial application of the EPS increases the production productivity and decreases costs. From technical point of view, EPS are integration of pneumatic and electric technologies. So, in EPS actuators are pneumatic elements (cylinders) while sensors and control system (CS) are electric. The interconnection of these technologies is achieved on the main directional control valve of each cylinder. Directional control valve directs the flow of the operating - pneumatic energy through the actuator. In other hand, directional control valves are actuated by electrical signal so, complete system is electro-pneumatic.

According to the control mode, the directional control valve can be bistable or monostable [1], Figure 1 a) and b). Bistable directional control valve has a two stable states and two solenoids which are used for switching between these two states. Solenoids, which are indicated by Z and Y in Figure 1*a*), are activated by electrical signals. For instance, after a short-term control pulse, the directional control valve switches to a stable state in which the output 4 is connected to the source of pressure 1. In this state the directional control valve remains until the signal Y appears and switch it to another stable state in which output 2 is connected to the pressure line 1. Simultaneous activation of both solenoids (Z and Y signals simultaneously) is not allowed, because then the state of the directional control valve is not defined. In this case, the overall system is blocked, and this signal is called a "blocking signal".

The monostable directional control valve, Figure 1*b*) has only one stable state. In this state (when the output 2 is connected to the pressure line) the directional control valve is positioned mechanically

by the spring. To another state (when the output 4 is connected to the pressure line) which is not stable state the directional control valve is positioned by the solenoid activation (signal Z, Figure 1*b*). In this state the directional control valve remains while the Z signal is active. Inactive signal Z means return to a stable state by the mechanical spring. At the Figure 1 τ_1, τ_2 and τ_3 denotes delay times between electrical signals and pneumatic signal at output 4. This delay times are dependent on the directional control valve types but generally they are in range up to ten milliseconds.

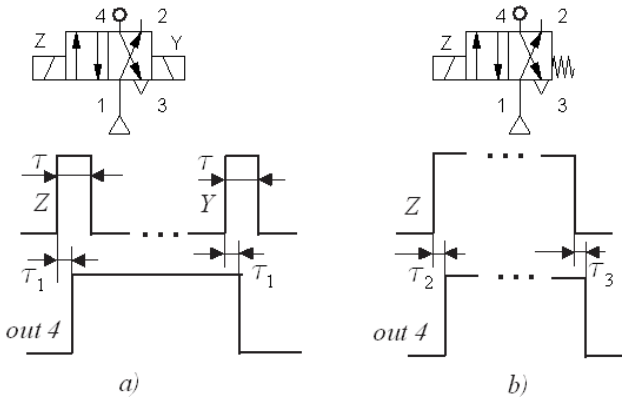

Figure 1. Bistable *a*) and monostable *b*) directional control valves with their inputs and outputs

The main task of the control system of almost all EPS is the realization of the default operation cycle. Operation cycle implies the necessary order of retractions and extractions of the piston rods of all cylinders in order to accomplish the required task. Therefore, changing of the operation task means changing of the operation cycle which make EPS very flexible. Flexibility is especially interesting nowadays, when industry 4.0 is increasingly becoming an issue of interest in underdeveloped and medium-developed countries.

However, flexibility also requires a new approach to designing control systems. Designing of CS must be automated and “designer” must be “smart machine” which, in interaction with environment, uses IoT (Internet of Things) technologies. In this way, it is achieved that one machine - the server can notify the other machine - the client that the task is finished. After that, the client can ask the server to do another task that is in accordance with the current and predetermined state of the overall system.

In this paper, algorithms that can realize the described automatic designing process are proposed. As a computer platform, WiFi microcontrollers (microcontroller with embedded WiFi interface) from the ESP8266 and ESP32 series are used, using Arduino software for their programming, [2-5]. The proposed algorithms have been implemented and experimentally confirmed on ESP8266. In doing so, the following rules and assumptions apply:

1. Double-acting pneumatic cylinders are used as actuators. Cylinders are marked with: A, B, C … H;
2. The required operation cycle is written in the text-string form, where the extraction of the piston rod is indicated by "+" and the retraction by "-", for instance: "A+, B+, B-, A-". At the starting point, the piston rods of all the cylinders in the operation cycle are retracted, so the first movement of a cylinder is always "+". All of the cylinders must have the same number of retractions and extractions (+ and -) in one operation cycle;
3. Each cylinder is operated via a monostable directional control valve 4/2 (5/2). Solenoid actuation causes the "+" movement of the cylinder and actuation by spring causes the "-" movement. The denotes of the directional control valve's solenoids are "Z" plus cylinder. For instance, the directional control valve of the cylinder A, have the marking of the solenoid "ZA" or "zA";

4. If an operation cycle requires the holding of the piston rod of some cylinder in the retracted or extracted position prior to initiating the next movement, that task is denoted after the denote of the cylinder and the type of movement as "/ xS", where the "x" is the holding time in the tenths of a second (maximal value for x is 65535). For instance, "... F + / 65S ..." indicates that after extraction of the cylinder's piston rod F system waits 65 * 0.1 = 6.5sec before initialization movement of some another cylinder.
5. Each cylinder during movements activates two limit switches, one for the fully extraction and the other in the fully retraction position. Only one of these switches can be activated at the same time. Limit switches are designated as: cylinder mark plus "0" for the limit switch that is activated when the cylinder is retracted and cylinder mark plus "1" for the switch that is activated in the extracted position. For example, the limit switches of the cylinder F are designated as F0 (retracted position) and F1 (extracted position). Also, the f_0 and f_1 tags refer to the same limit switches.

2. Design of the control system

2.1. Design of the control system using PLC

In nowadays, the control systems for EPS are mostly realized using PLC (Programmable Logic Controller), [6-7], while for their simulation the software package FluidSim-P [8] is mainly used. It is known that the work of the PLC takes place through several so-called scan cycles: reading inputs, executing program, updating outputs and internal update. For the same application the total duration of all these scan cycles is constant and it is designated as T_{sc}. Therefore, PLC periodically with period T_{sc}, reads all inputs and updates all outputs. If there are some changes within the system that are shorter than one period T_{sc}, then these changes will not be registered, i.e. the control system behaves as like these changes did not occur. In EPS all changes are much longer than the period T_{sc}, so the period T_{sc} is negligible compared to other changes (for example, the reaction of the limit switch lasts dozens of T_{sc} periods).

Although PLC is very fast to control EPS, there are some problems with its implementation. The basic problems are in determining the logical functions that are realized within the control system. Let's explain this in one example. Let the EPS contain three cylinders A, B, and C and next requested operation cycle: A +, B +, B-, C +, C-, A-. This cycle is shown in Figure 2. Using the displacement-step diagram shown in a dark full line. The inputs to the system are information from the limit switches of each cylinder: a0, a1, b0, b1, c0 and c1 as well as ON and OFF signals, while the outputs are signals: *zA*, *zB* and *zC* which activates the solenoids of directional control valves of cylinders A, B and C, respectively. Inputs and outputs make the vectors: inputs $\boldsymbol{x}=[ON, OFF, a_0, a_1, b_0, b_1, c_0, c_1]^T$ and outputs $\boldsymbol{y}=[zA, zB, zC]^T$. From Figure 2 is obvious that the same combination of inputs in steps: 2 (B+), 4 (C+) and 6(A-) does not generate the same outputs. Thus, for the input vector $\boldsymbol{x}=[1,0,0,1,1,0,1,0]^T$ in the step 2 output vector is $\boldsymbol{y}_2=[1,1,0]^T$, in step 4 output vector is $\boldsymbol{y}_4=[1,0,1]^T$ and in step 6 output vector is $\boldsymbol{y}_6=[0,0,0]^T$. Therefore, new variables are introduced, state variables M_1 and M_2, which are together with signal $S = START$ makes state vector $\boldsymbol{m}=[S, M_1, M_2]^T$.

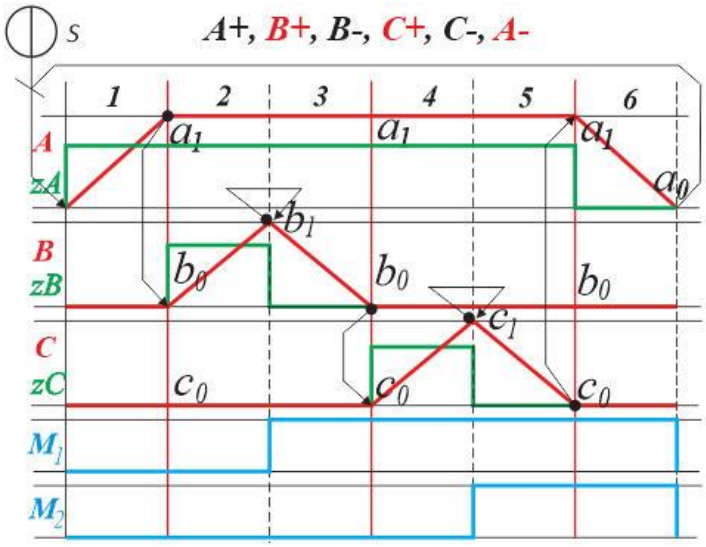

Figure 2. Displacement – step diagram with directional control valves *zA, zB* and *zC* and states M_1 and M_2

State variables M_1 and M_2 are selected in that way that together with inputs make unique combination for each generated output. Value of the state variables in some future instant $t+$, i.e. states $\boldsymbol{m}(t+)$, depends on inputs $\boldsymbol{x}(t)$ and previous states $\boldsymbol{m}(t)$ in instant t. This means that in each period T_{sc} inside PLC must be realized two logical functions: one which change states, function $\boldsymbol{g}(\square)$ and another which change outputs, function $\boldsymbol{f}(\square)$:

$$\boldsymbol{m}(t+)=\boldsymbol{g}\left(\boldsymbol{x}(t),\boldsymbol{m}(t)\right), \qquad \boldsymbol{y}(t)=\boldsymbol{f}\left(\boldsymbol{x}(t),\boldsymbol{m}(t)\right). \tag{1}$$

After one period T_{sc}, current states $\boldsymbol{m}(t)$ have the same value as previous calculated states $\boldsymbol{m}(t+)$, i.e. $\boldsymbol{m}(t)=\boldsymbol{m}(t+)$.

For the observed operation cycle, vector function $\boldsymbol{g}(\square)$ and $\boldsymbol{f}(\square)$ from (1) becomes:

$$\boldsymbol{m}(t+)=\begin{bmatrix} S(t+)=(S(t)+ON)\overline{OFF} \\ M_1(t+)=\left(M_1(t)+b_1\right)\bar{a}_0 \\ M_2(t+)=\left(M_2(t)+c_1\right)\bar{a}_0 \end{bmatrix}, \quad \boldsymbol{y}(t)=\begin{bmatrix} zA=S\overline{M}_2+\bar{c}_0 \\ zB=a_1\overline{M}_1\overline{M}_2 \\ zC=b_0M_1\overline{M}_2 \end{bmatrix}. \tag{2}$$

At Figure 3 is shown ladder diagram for PLC for realization logical function from (2).

2.2. Design of the control system using microcontroller – new approach

Designing of control system of EPS using PLC in some cases can be very complex. Mainly, this is due to the PLC operational system i.e. synchronous executions of the scan cycles. In addition, for each specific operational cycle, it is necessary to re-define the logic functions and program the PLC. So, automatic programming based on a given operational cycle is either impossible or very difficult to implement.

In this section is given a proposal for the automated EPS design solution using a microcontroller. EPS control system which using the microcontroller is also given in [9], but microcontroller programming is performed for each specific cycle. Similarly, in [10] and [11], the authors deal with EPS control using a microcontroller. In [11] authors use the MATLAB/SimuLink software for designing control systems. As a link between controlled object and programs in MATLAB they use

the Arduino microcontroller platform. The proposed solution is partly use the procedure given in the FluidSim-P software in the section dealing with GRAFCET programming.

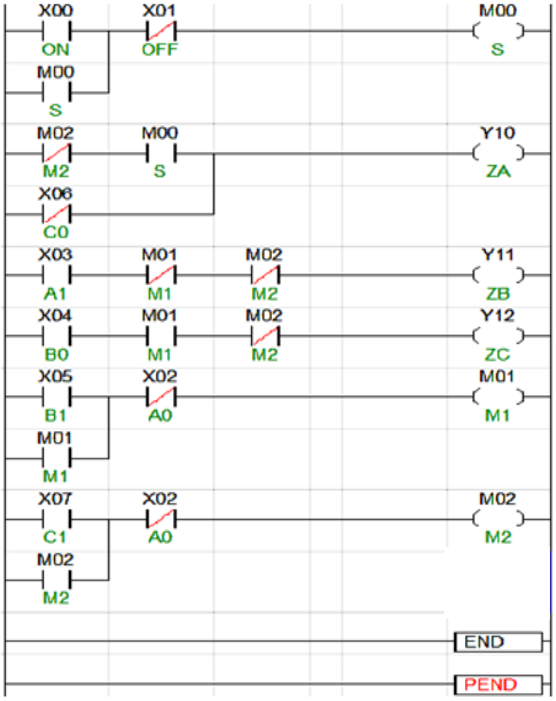

Figure 3. Ladder diagram, program for the cycle: A+,B+,B-,C+,C-,A-

The operation of almost all EPS is carried out so that the completion of the movement of one actuator-cylinder activates the next movement. This procedure continues until the last movement in the operation cycle when the start conditions are checked again. If start conditions are fulfilled the operation cycle is repeated. Therefore, the operation of these systems is asynchronous and can be divided into several steps. In doing so, the realization of the next step (movement) is conditioned by the completion of all operations (actions) in the previous step and by fulfilling certain conditions for the transition to the next step, Figure 4.

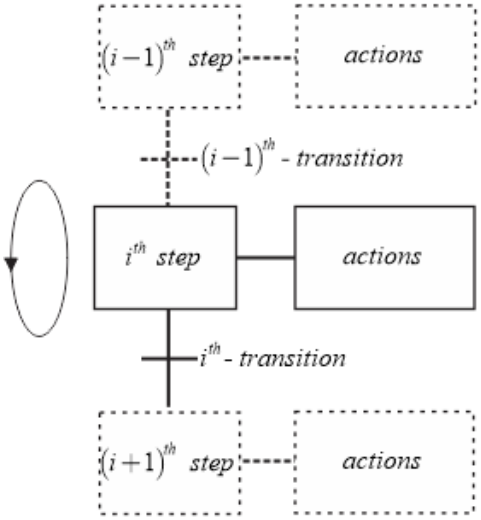

Figure 4. Division of the operation cycle into the steps with their actions and transitions

Each step is described by actions within the steps and transition conditions to the next step. Under actions in the step we consider the activation of directional control valves of all cylinders that are required in this step by default. Also, actions are considered to be the realization of the required holding time in that step. The passage conditions between two steps are determined based on the state of the limit switches of each cylinder from the operational cycle. This condition is fulfilled if all the cylinders have completed the movements required in the actions, which is signaled by the activation of the appropriate limit switches - sensors.

Actions and transitions conditions can be written in the form of three arrays, two for actions and one for transitions conditions. Let these arrays be *NIZR* and *NIZV* for actions and *NIZG* arrays for transitions conditions. In the *NIZR* array, some bits indicate the solenoids of the monostable directional control valves of each cylinder in an operational cycle. If this bit is set, then this directional control valve is activated and the cylinder piston rod is extracted (movement "+"). The *NIZV* array contains holding times in each step. The array of transitions conditions, *NIZG* is similar to the *NIZR* array, only the bits refer to the cylinder limit switches. The formation of these arrays will be explained on the previous example, i.e. for the operational cycle: A +, B +, B-, C +, C-, A-.

Operational cycle doesn't have holding times, so the all of elements of the NIZV array are zero and they will not be considered. Since the operational cycle contains three cylinders, then three bits for each solenoid are enough for the *NIZR* array. The *NIZG* array takes 6 bits, because for the three cylinders 6 limit switches are needed. Therefore, for the required operation cycle, it is necessary to form the arrays given in Table 1. The number of elements of each of the formed arrays is the same as the number of steps in the operational cycle, i.e. for each step one element of the array.

Table 1. The arrays of actions (*NIZR [6]*) and transition conditions (*NIZG [6]*) for the operational cycle: A+, B+, B-, C+, C-, A-

	actions			steps	transitions					
	NIZR[6]				*NIZG[6]*					
	zA	*zB*	*zC*		a_0	a_1	b_0	b_1	c_0	c_1
0	1	0	0	A+,	0	1	1	0	1	0
1	1	1	0	B+,	0	1	0	1	1	0
2	1	0	0	B-,	0	1	1	0	1	0
3	1	0	1	C+,	0	1	1	0	0	1
4	1	0	0	C-,	0	1	1	0	1	0
5	0	0	0	A-,	1	0	1	0	1	0

When the arrays are formed and stored in the memory of the microcontroller, then it is very easy to realize the required operating cycle. After the START signal, the first element of the action array, the *NIZR[0]* element, is sent to the output. Since in that element $zA = 1$ only the solenoid of the directional control valve of the cylinder A is activated, which further causes the extracting of the piston rod of this cylinder. Immediately after that, the microcontroller reads the state of the limit switches, i.e. reads the input status. This reading is repeated continuously until the actual state of the input is identical to the first element of the transition array, the *NIZG* element *[0]*. This state will be identical when the limit switch a_1 is activated, i.e. $a_0 = 0$ and $a_1 = 1$. Since in this step there were no movement of the cylinders B and C, the state of their limit switches are remain unchanged, i.e. $b_0 = 1$, $b_1 = 0$, $c_0 = 1$ and $c_1 = 0$ (the piston rods of both cylinders are retracted). When the actual state of the inputs identical to the

element of the transition array, the step is completed. Due to the conditions for transition to the next (second) step are fulfilled.

In second step, in the array of actions (*NIZR [1]*), *zA* = 1 and *zB* = 1, so the solenoid of the directional control valve of the cylinder B is active and the cylinder B piston rod is extracted. After that, the actual state of the inputs is continuously read until the actual state of the input is identical to the other element of the transition array, the *NIZG* element *[1]*.

This procedure is repeated all the way to the last element of the action array, the *NIZR* element *[5]*. When this element is forwarded to the output, the solenoid of all directional control valves is switched off and the piston rods of the all cylinders are retracted. System waiting that all limit switches with the index "0" be activated (the corresponding bit is 1) and with the index "1" be deactivated (goes to 0). This condition is identical to the initial state of the observed EPS. After that the start state is checked. If it is active, the process is repeated and if not, the system stops in the initial position. This ensures that the system always stops in the starting position regardless of the moment when the START signal has been reset.

The problem of designing the control system in manner which is proposed is significantly simplified. Instead of control the operational cycle (like PLC), it is control of each individual step from that operating cycle. Therefore, it cannot be a problem that the same combination of inputs gives different outputs, because each step is considered separately. Since the activities in each step are the same, then the same control system can control all steps (i.e. the required operation cycle), only in certain steps the output and input values are changed, that is, the elements of arrays of the actions and conditions of transition..

In the previous section, the actions arrays and the transition condition array were formed manually based on the required operational cycle. In this chapter is described an algorithm that creates these arrays automatically. Input for proposed algorithm is operational cycle being given in a text format - a string that is written by specific syntax and rules. There are some limitations that must be respected and they are:

1. The maximum number of cylinders is 8 so that the elements of the *NIZR* array are 1-byte length = 8 bits (one bit for each directional control valve). As the 8 cylinders require 16 limit switches, then the length of the elements of the *NIZG* is 2-byte = 16 bits (one bit for each limit switch);
2. The holding time array *NIZV* is 2 bytes long. This means that it is possible to set a maximum of 65535 tenths of a second or a maximum holding time of 65535 * 0.1 / 3600 = 1.82 hours. If the holding time are not required in the operational cycle at any step, then all the elements of the *NIZV* array are zero, and the logical variable *vremC* = *false* (0). If at least one holding time is required, then *vremC* = *true* (1);
3. The length of the string in which the requested operational cycle is written is a maximum of 256 bytes. The maximum number of steps in one operational cycle is 80.

The limitations are primarily related to the number of cylinders and limit switches (outputs and inputs), which is limited by the selected microcontroller. It is possible to use 8 cylinders for the ESP 32 microcontroller, while only 3 cylinders can be used for the ESP8266 microcontroller. Limitations related to the number of steps and the length of the string in which the operational cycle is written can be much higher. These limitations are related to the size of the operating and EEPROM memory of the microcontroller.

At Figure 5, one part of the algorithm for creating arrays is given and the function of the *praviNizove* is in Appendix. The required operating cycle of cylinders is written in the string "*sc*", which is a parameter for this function. All letters in this string were previously converted to capital letters. In the variable "*gpE*" is written the initial state of the limit switches, i.e. the state when the piston rods of all the cylinders are retracted and when the limit switches with the index "0" are activated. In the variable "*abcE*" is the initial state of the solenoids of all directional control valves, while in the variable "*abc*" the cylinder marks that can occur in any cycle.

Other variables used in this function are: "*i*" - serves as a character counter in the string "*sc*", "*masGP*" - a 2-byte length number, which serves as a mask for elimination of limit switches influence of all cylinders which could be occur but which are not be in operation cycle, "*onB*" - a logical variable that turns on when a "/" character appears in the string "*sc*" (the indication for the start of the holding time), and turns off when the "S" symbol appears (the indication for the end of the holding time). All characters between "/" and "S" are numbers that are convert to a variable "broj" that represents the holding time value, "*iR*" - the step counter in the operational cycle and "*p*" - the number which indicating the position of the cylinder from the cycle "*sc*" in the string "*abc*". For instance, the position of the cylinder A is 0, the cylinder F is 5, the cylinder H is 7, and so on.

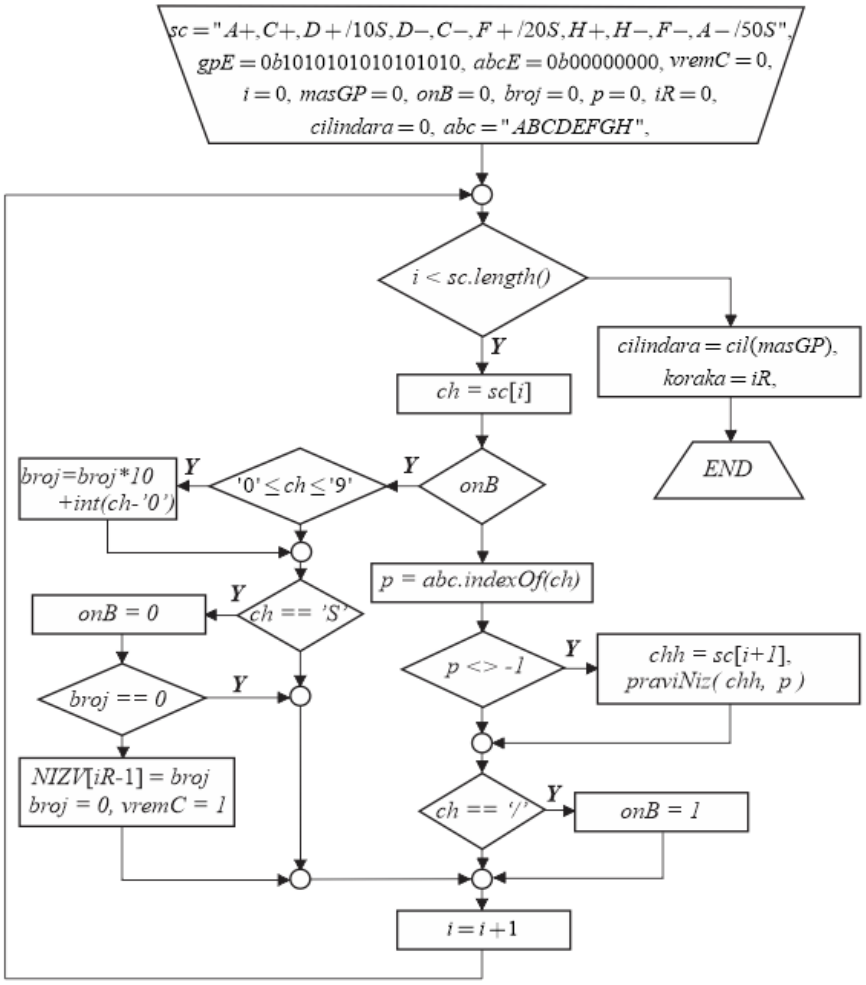

Figure 5. General algorithm for creating the arrays, function *praviNizove*

In the main program loop the characters of the operational cycle-string "*sc*" are read in sequence and are placed in the variable "*ch*". If this character is contained in the variable "*abc*" (it is checked by the command *p = abc.indexOf (ch)*), then "*p*" is the ordinal number in the string "*abc*". If p <> - 1 the character "*ch*" is contained in "*abc*", which means that this sign is a cylinder mark. This sign must be followed by a sign either "+" or "-" (i.e. extraction or retraction of the piston rod) which is read as a variable "*chh*".

The values of the character "*chh*" (sign "+" or "-") and the number "*p*" (position "*ch*" in the string "*abc*") are passed to the function *praviNiz* whose algorithm is shown in Figure 6. The function

praviNiz creates one element of the arrays *NIZR* and *NIZG*, and the element of the array *NIZV* sets to 0. The function *praviNiz* will be explained later.

If the variable "*ch*" is character "/", then the next number is the number that represents the value of the required holding time. After that, the logical variable "*onB*" is turned on, digits in a sequence are read and the total value of the holding time are calculated. This goes all the way to the sign "*S*" when the variable "*onB*" is turned off, and the current element of the string *NIZV* is set up to value of holding time, see Figure 5.

As we have said before, Figure 6 shows the algorithm of the function (the function *praviNiz*) that creating one element of the action array *NIZR* and the transition condition array *NIZG*. To this function are assigned two parameters, the first parameter "*pm* = *chh*" is the "+" or "-" character, and the second parameter "*a* = *p*" is the ordinal number of the cylinder mark in the string "*abc*" (*abc* = ABCDEFGH). For instance, let "D+" movement from operational cycle "*sc*". Then it is *pm* = "+", and the parameter *a* = 3 (because 3 is the position of the letter D in the string "*abc*").

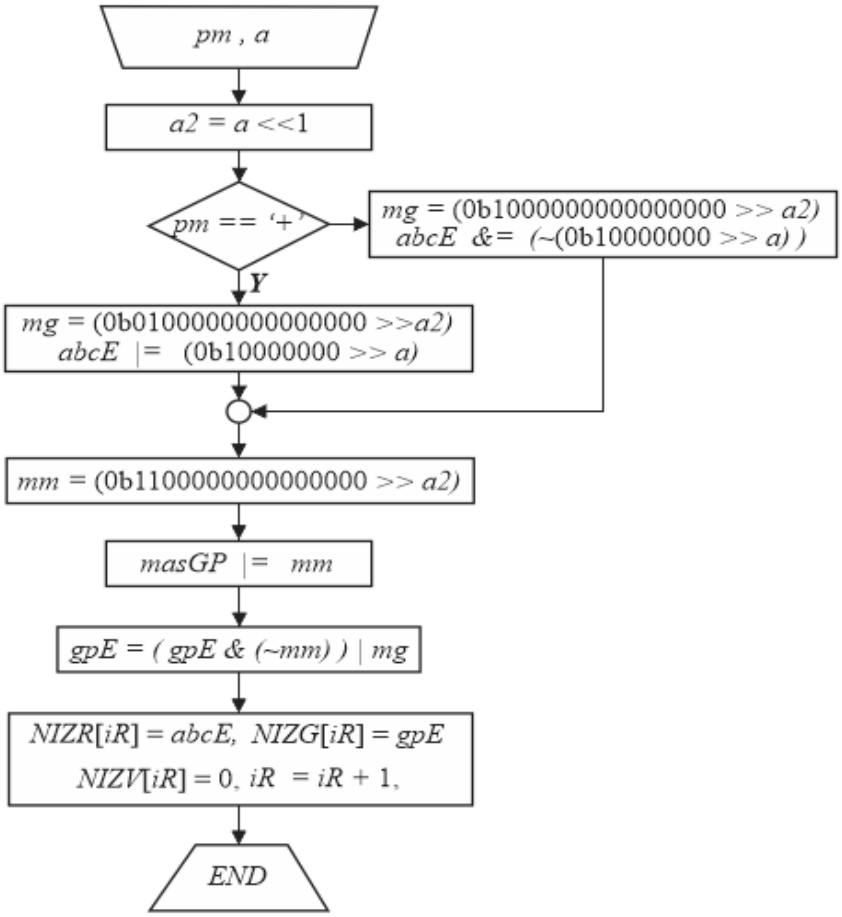

Figure 6. General algorithm for creating the arrays, function *praviNiz*

The variable *a2* is obtained as 2**a* = 6 or, which is equivalent, by moving the bits from a to one position to the left. The following masks are further defined: *m* = 0b1000 0000, *mm* = 0b1100 0000 0000 0000 and a "*mg*" mask whose value depends on the parameter "*pm*". If *pm* = "+", then *mg* = 0b0100 0000 0000 0000, and if *pm* = "-" then *mg* = 0b1000 0000 0000 0000. Using the mask *m*, via the variable *abcE* and depending on the "*pm*", is calculated the new value of the array *NIZR* element in the following way. If *pm* = "+", then *abcE* = *abcE* | (*m* >> *a*), which is the same as setting

the bits *7-a* in a variable *abcE*. If *pm* = "-", then *abcE* = *abcE* & (~ (*m* >> *a*)), which is equivalent to resetting the bits *7-a* in the variable *abcE*.

Similarly, using the masks *mm* and *mg*, over the variable *gpE*, the elements of the transition conditions array are calculated (i.e. determining the state of the limit switches for which the condition of the transition to the next step is fulfilled). The variable *gpE* is calculated as: *gpE* = (*gpE* & (~ (*mm* >> *a2*))) | (*mg* >> *a2*). For a given case (*a* = 3 Þ *a2* = 6), moving the mask *mm* to 6 in the right leads to *mm* >> *a2* = 0b0000 0011 0000 0000, so that the expression (*gpE* & (~ (*mm* >> *a2*))) in the variable *gpE* resets the bits 9 and 8 (bits 15-*a2* and 15-*a2*-1 in the word *gpE*) while the other bits do not change. After that, via the OR function, i.e. the terms "| (*mg* >> *a2*) ", are replaced by bits 01 (for *pm* = "+") or by 10 (for *pm* = "-").

The variable *masGP* is also used as a mask. In it bits that are corresponding to the position of the limit switches of cylinders which are used in the required operation cycle are set to the value 1, while the other bits are zero.. In this way, the *masGP* in the logic AND function with the element of the array *NIZG* or the actual state of the limit switches distinguishes only the limit switches of cylinders which are used. For instance, let's have 6 cylinders A, B, C, D, F, and H on a machine. In one cycle we use all the cylinders, while in the second cycle we do not use, say, cylinder B, but this cylinder is physically presence of the machine and its limit switches are connected to the microcontroller. In the first case, the mask is *masGP* = 0b1111 1111 0011 0011, and in the second case, the states of the limit switches of cylinder B are ignored so that the mask has a value of *masGP* = 0b1100 1111 0011 0011.

For the operational cycle *sc* = "*A*+,*C*+,*D*+ /10*S*,*D*- ,*C*- ,*F* + /20*S*,*H*+,*H*- ,*F*- ,*A*- /50*S*" from Figure 5, using the algorithms from Figure 5 and Figure 6, the following arrays are generated:

```
CYCLE : $A+/30S,B+,C+,D+/50S,D-,C-,B-,A-/20S,H+,H-/50S#
    Valves    Limit Switches
    ABCDEFGH  A B C D E F G H   Cy. T[s]
 0. 10000000, 0110101010101010, A+, 3.0,
 1. 11000000, 0101101010101010, B+, 0.0,
 2. 11100000, 0101011010101010, C+, 0.0,
 3. 11110000, 0101010110101010, D+, 5.0,
 4. 11100000, 0101011010101010, D-, 0.0,
 5. 11000000, 0101101010101010, C-, 0.0,
 6. 10000000, 0110101010101010, B-, 0.0,
 7. 00000000, 1010101010101010, A-, 2.0,
 8. 00000001, 1010101010101001, H+, 0.0,
 9. 00000000, 1010101010101010, H-, 5.0,
Steps = 10, Mask = 1111111100000011, Cyl. = 5
```

Figure 7. The arrays: *NIZR*, *NIZG* and *NIZV* that are generated using algorithms which are given on the Figure 5 and Figure 6 and for the given cycle

3. Remote control system

In Industry 4.0 everything is networked. In doing so, Internet technology is mainly used for the exchange and transmission of information in that network. Each unit in that network has its own unique IP address. The exchange of information takes place through the client-server system. The client sends requests, the server processes these requests and sends the answers. In this case, the client is a remote computer or mobile phone, and the server is a microcontroller ESP32 or ESP8266. All the inputs and outputs of the observed EPS are connected to the microcontroller. Besides, on this microcontroller is installed software which has the role of the web server and the EPS controller. The web server is at the same time the HTTP server, because HyperText Transfer Protocol is used to exchange information.

Operation within this system takes place as follows. After the client connects to the server, the server from the EEPROM reads the operational cycle which it is last executed. If it is reading for the first time, then the initial operational cycle is taken (the starting cycle is predefined as: "A +, B +, B-, A-"), which is then treated as the last cycle also. After that, the server offers the client a choice of two options, to enter a new operational cycle or a start / stop operational cycle. If a start / stop operational cycle is selected, then the last operational cycle on the server is taken as the operational cycle. On the basis of this cycle, arrays are created and the program continues its work as a control system.

In case the client chooses to enter a new cycle, then a window with the necessary instructions and a last operation cycle offered is opened. In the offered operational cycle, it is possible to make some changes or input a completely new cycle instead. In the case that the required cycle is not identical to the last cycle, a syntax check is performed, as well as a check that the cycle is entered under Rules 1, 2 and 4 specified in the introduction and whether the limitations given in section 2.3 of this paper have been complied with. If there is a problem, then the server informs the client about it. For instance, if the new operational cycle is entered as: "A, +, b +. , / 20s r B- qx a - ", then after checking returns the operational cycle as" A +, B + / 20S, B-, A- " without any notification of any problem, because only syntax errors are here. But if, for example, the entered operational cycle is "A, +, b +, / 20s B-qx", then the client receives a message that there is "A +", but there is no "A-" and changes are required in the entered operating cycle.

If the cycle is correct after the described checks, then the server again offers the client a new cycle or a start / stop cycle. In the case that the start / stop cycles are selected, the cycle is remembered in the server's EEPROM memory, the required arrays are created and the server continues to operate as a control system. At the same time as the server control the operation of the observed EPS, it is still ready to respond to client requests.

4. Implementation of the control system

The above suggested algorithms are realized in laboratory conditions. The Arduino Uno R3 compatible development board based on WiFi microcontroller ESP-8266EX was used. As the software environment, the Arduino IDE is used, so programming is identical to the programming of the Arduino UNO. The board includes integrated 3.3V and 5V voltage regulators, 2.4GHz antenna and required circuits that enable USB serial communication with CH340 conversion. The power supply of the board is via a USB port or an external power supply with voltages from 7V to 24V. The board is connected to the connectors: all 9 I/O ports (from D0 to D8 with GPIO addresses 0,4,5,12,13,14,15 and 16), Rx and Tx, A0 (AD converter) and RESET connectors, Figure 8. Nine I/O connectors allow operation with 3 pneumatic actuators. In addition, 6 I/O are configured as PULLUP inputs (these are inputs for limit switches) and 3 are configured as outputs (solenoids of directional control valves). To activate the solenoid, the output relay module is used (Figure 9), where the microcontroller (using the LOW logic level) activates relays with one normally open and one normally closed contact via the optocoupler and the transistor.

Figure 8. Arduino Uno R3 Development board based on ESP-8266EX microcontroller

Figure 9. Output relay module with optocouplers and four relay outputs

In the experimental setup the PULLUP inputs of microcontroller are directly connected to the limit switches, so the activation of the limit switches is directly achieved by connecting the PULLUP input to the ground, which is common to the microcontroller and inputs. The control circuits of solenoids are at one end connected to a 24V DC supply, and at the other end they are connected to the ground via the output contacts of the output relay. In Figure 10 and Figure 11, the practical realization is demonstrated, where FESTO electro-pneumatic didactic equipment was used. The pneumatic scheme of this EPS is shown in Figure 12.

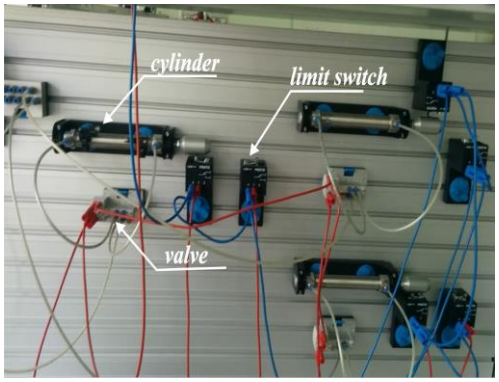

Figure 10. Three cylinders with limit switches and valves

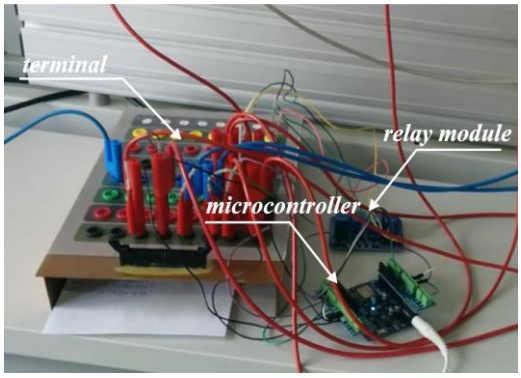

Figure 11. Microcontroller with output relay module and terminals

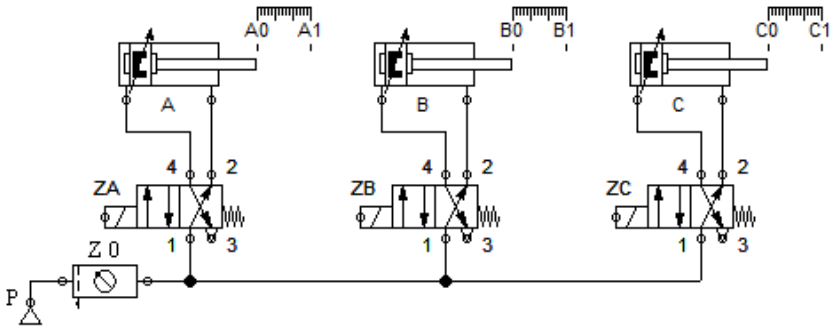

Figure 12. Pneumatic part of the considered electro-pneumatic system

In order to realize the observed EPS in industrial conditions, it is necessary galvanic separation of input module from the microcontroller as shown in Figure 13.

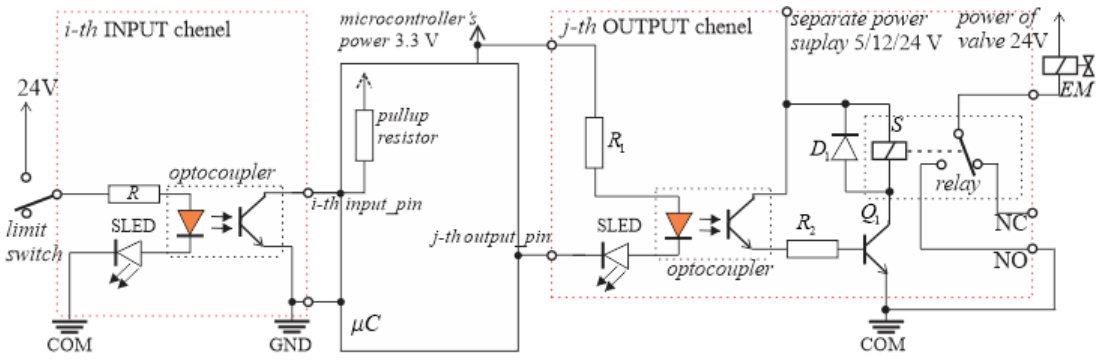

Figure 13. Input module, control module and output module of the considered electro-pneumatic system

The figure shows the galvanic separation using the optocoupler, where the voltage of the limit switch is used to activate the inputs (it is most often 24V DC). For the 8-cylinder EPS, 16 identical input channels and 8 identical output channels from Figure 13 should be used. The figure shows the relay output module. In the case of transistor outputs, in the diagram from the Figure 13, instead of the solenoid S, the solenoid EM should be connected. Power supply for the output transistors and the EM is common. In this case the relay at Figure 13 is unnecessary. The ESP32 WiFi microcontroller can be used as the control module (both microcontrollers of Chinese manufacturer Espressif Systems are) in nodeMCU or Arduino versions and with the Arduino IDE application software.

As an input module, it is possible to use ready-made modules from Figure 14 and Figure 15 as follows: two modules from Figure 14 as input and one relay module from Figure 15 as output.

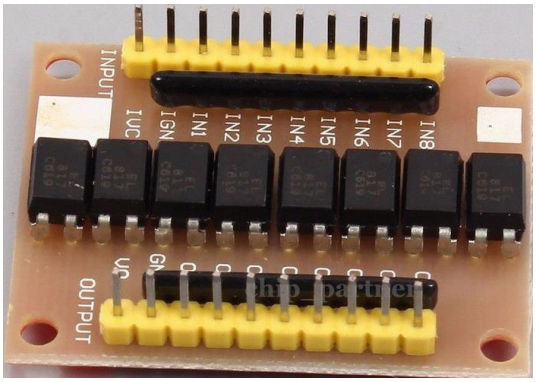

Figure 14. Eight channel input module with optocouplers, high/low 3-5V output and 24V input

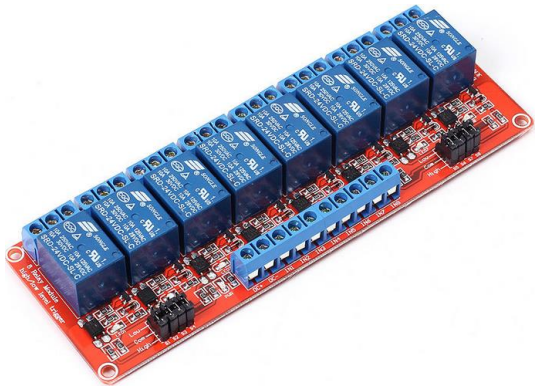

Figure 15. Eight channel relay output module with optocouplers, low activation and 24 V power supply

5. Conclusion

The paper deals with the design of remote control of the electro-pneumatic system, whereby the control is solved using the microcontroller ESP8266 and ESP32 with WiFi communication. The control part of the system has two functions: first, to communicate with the client in order to get tasks from it and fulfill its tasks, and second, to achieve the control function according to the assigned task in real time. Therefore, the microcontroller is configured as a web server, and the client is a computer that provides HTTP protocol and internet connection. With very small changes in the software it is possible to achieve that the client is another microcontroller, which creates all the prerequisites for direct communication between two machines (machine to machine communication), which is one of the basic requirements in the Industry 4.0.

The client sends an operation task in the form of a text string in which, according to certain rules and in a specific syntax, the operational cycle is written. Based on this operational cycle, the arrays of actions and transition conditions are generated, with one element of these arrays being able to realize all the necessary activities for one step in the operational cycle. In this way, with the realization of tasks in each step, the overall task set in the operational cycle is realized.

The fulfillment of requirements which are set for the control system was also experimentally verified. The ESP8266 microcontroller and the three-cylinder electro-pneumatic system were used. The results of the experiment confirm all the theoretical considerations which are noted and explained in the paper.

Appendix

Because of the limited scope of the paper, only some parts of the control software are provided here. Parts of the control software which are given in this paper are the functions *praviNizove* and *praviNiz*.

Flowcharts of these functions are shown at Figure 5 and Figure 6, respectively. Arduino IDE software was used to program the ESP32 and / or ESP8266 microcontroller.

```
byte praviNizove(String sc){//string sc is desired cycle
  char ch,chh;
  boolean onB = false;
 int p;
 uint16_t broj = 0;
 gpE = 0b1010101010101010; masGP = 0; cilindara = 0; iR = 0; abcE = 0; vremC = false;
 for (int i = 0; i < sc.length(); i++){// sc.length() - it is length of the string sc
   ch = sc[i];
   if (onB){
     if (ch >= '0' && ch <= '9') broj = broj * 10 + int(ch - '0');// int(ch - '0') -conversion of the character ch to int
                                                                  //numeric and string between '/' and 'S' to broj
     if (ch == 'S'){
       if (broj != 0) { vremC = true; NIZV[iR - 1] = broj; }
       onB = false; broj = 0;
     }
   }
   p  = abc.indexOf(ch);
   if (p != -1) { chh = sc[i+1];  praviNiz(chh,p); }
   if (ch == '/')  onB = true;
 }
 cilindara = cil(masGP); //function cil() calculate the number of cylinders which are exist in the cycle sc
 return (iR);                // iR-number of steps
}
//
void praviNiz(char pm, byte a){//pm - it may be only '+' or '-', a-position of ch into string abc
 byte a2 = a << 1;   //multiply by 2
 uint16_t mg =  (pm == '+')?(0b0100000000000000 >> a2):(0b1000000000000000 >> a2);
 uint16_t mm = ( 0b1100000000000000 >> a2 ) ;
 masGP |= mm;
 if (pm == '+')  abcE |= (  0b10000000 >> a) ;    //the same as bitSet  (abcE,7-a);
 else             abcE &= (~(0b10000000 >> a));  // the same as bitClear(abcE,7-a);
 gpE = (gpE & (~mm)) | mg;
 NIZR[iR]   = abcE;   NIZG[iR]   = gpE ;    NIZV[iR++]  = 0;
}
```

References

[1] *** FESTO Didactic (2000), *Fundamental of Electro-pneumatic*
[2] Javed A 2016 *Building Arduino Projects for the Internet of Things: Experiments with Real-World Applications*, Lake Zurich, Illinois, USA
[3] Schwartz M 2016 *Internet of Things with ESP8266*, Packt Publishing Ltd, Birmingham B3 2PB, UK
[4] Kolban N 2016 *Kolbon's Book on ESP32 & ESP8266*, October 2016
[5] Kolban N 2017 *Kolban's Book on ESP32*, May 2017
[6] Suyetno A, Kustono D and Sudjimat D A 2016 *Pneumatic Control System Simulation Based a PLC Micro in Mechatronics Courses,* Proceedings of the International Mechanical Engineering and Engineering Education Conferences (IMEEEC), doi: 10.1063/1.4965769,
[7] Singh R and Verma H K 2017 Development of PLC Based Controller for Pneumatic Pressing Machine in Engine-Bearing Manufacturing Plant, *Procedia Computer Science* **125** 449-458
[8] Kumar R and Rosario R *Design and Simulation of Electro-Pneumatic Motion Sequence Control*

using FluidSim, De La Salle University, Taft, Manila 1004 Philippines
[9] Swider J, Wszolek G and Carvalho W 2005 Programmable controller designed for electro-pneumatic systems, *Journal of Materials Processing Technology* **164-165** 1459-1465
[10] Kadam S P, Gaikwad V R, Bhavsar H D and Kulkarni S V 2016 Electro-Pneumatic Lift and Carry Conveying System, *International Journal of Science Technology & Engineering* **2**(10)
[11] Parikh P, Vasani R, Sheth S and Gohil J 2016 Actuation of Electro-Pneumatics System using MATLAB Simulink and Arduino Controller- A case of a Mechatronics systems Lab, *Advances in Intelligent System Research* **137** 59-64