The Development of Computer Program for the Welding Interface of the Developed Welding Mini-Robot

D.H. Oladebeye, Ph.D.[1*]; Prof. S.B. Adejuyigbe[2]; and A.A.G. Olorunnishola, Ph.D.[3]

[1]Department of Mechanical Engineering Technology, Federal Polytechnic, Ado-Ekiti, Ekiti State, Nigeria.
[2]Mechatronics Engineering Department, Federal University, Oye-Ekiti, Ekiti State, Nigeria.
[3]Department of Mechanical Engineering Technology, Federal Polytechnic, Ado-Ekiti, Ekiti State, Nigeria.

E-mail: dayobeye@yahoo.com*
solaakim73@gmail.com

ABSTRACT

There have been scanty studies on automation of welding operations in developing countries, especially Nigeria. This phenomenon had brought the need for the development of a computer program for the welding interface of the developed welding mini-robot to enable automation of welding processes. The welding robot, a Cartesian type for its simplicity and cost-effectiveness in production, was constructed and operated by developing computer software with Peripheral Interface Controller (PIC) of specification PIC16F877 for the automation of the welding robot operation. The PIC was connected to stepper motors of specification NEMA 23 of 2 Nm torque and 1000 rpm for X-axis, Y-axis and Z-axis to control the movement of the welding robot in any specified direction. The developed program was validated by comparing results of practical values obtained from developed welding robot with the conventional electric arc welding machine operations. Results have shown that the developed welding robot can weld mild steel plates linearly along the length of guide 470 mm on X-axis, 350 mm on Y-axis and 110 mm on Z-axis at welding time (4.7-32.94s) faster than the conventional Electric Arc Welding Machine's (15-45s).

(Keywords: *computer program, welding interface, welding, mini-robot, manual arc welding*).

INTRODUCTION

According to the American Robotics Institute, a robot is a "reprogrammable, multifunctional manipulator designed to move materials, parts, tools, or specialized devices, to variable programmed motions for performing a variety of tasks". While Joseph Engelburger already developed the first industrial robot in the mid-1950s, robotic arc welding was first used in production until the mid-1970s [1]. Based on the geometry of the workspace, the most commonly used type for industrial robotic arc welding is robot with a revolute (or joined arm) configuration [2]. To this end, it is necessary to develop an intelligence technology to enhance the current method of learning and use for playback programming for welding robots to achieve the high quality and versatility required of welded products [3, 4].

In the past, manufacturers also only looked at their most complicated pieces as candidates for automation and concluded that the projects were not economically justifiable. More flexible systems today minimize the need for dedicated fixing and facilitate the switching of production from one part to another. The 80/20 rule also applies, as eighty percent of a company's production usually accounts for only 20 percent of its part sizes.

The new robotic welding cells make it easier to automate more of these easy, often repetitive jobs and to turn quickly from one to another. The key reason to automate this is that it makes economic sense. Robotic systems are now more affordable than ever, for short to medium run applications. With a lower investment and greater productivity improvements, the return on investment (ROI) is greater than with larger, individually designed welding lines [5]. Consistent consistency is another justification for automating. The less uncontrolled variables in the process, the higher the weld quality and the more homogeneous they are. Increased service rates and quicker processing times also help generate greater value for the customers.

The flexible fixing or workholding incorporated in the newest robotic cells makes it easy to switch back and forth between varieties of different products with little change over time, as opposed to custom fixtures. This also increases productivity and boosts ROI [6]. In addition to competitive unit costs, robotic welding systems provide other advantages, such as increased efficiency, health, weld consistency, flexibility and utilization of the workspace, and reduced labor costs [7, 8].

Ismara and Prianto, [9] have developed that the needs analysis relating to the industrial arm's welding robot arm is used as the basic template for redesigning, developing and compiling the implementation of the robotic arms-assisted welding education laboratory model. It concluded that the use of the robotic arm is expected to make the students who practice as operators feel comfortable, happy, and enjoyable to enhance the motivation to learn. There are several things to consider when building a robotic welding facility.

Robotic welding must be viewed differently from manual welding. Some of the considerations for a robotic welding facility are listed below: start/stop, pre-flushing gas, electrode feed and nozzle flushing included in the selected welding programs. Robots were used for the welding of complete automotive body assembly and sub-assembly components for around 15 years. In general the automatic arc welding equipment is constructed differently from that used for manual arc welding.

Automatic arc welding normally involves high duty cycles and, under those conditions, the welding equipment must be able to operate. In addition, the components of the equipment must have the functions and controls necessary to interface with the control system. The number of items to be welded of any type must be sufficiently high to justify process automation [6]. In view of the importance of robotic welding to the economic advancement of Nigeria, a computer program for the welding interface of a mini welding robot was developed in this research work.

<u>History of Peripheral Interface Controller (PIC)</u>

Peripheral Interface Controller (PIC) is a digital computer used for automation of typically industrial electro-mechanical processes, such as control of machinery on factory assembly lines, amusement rides, or light fixtures. Peripheral Interface Controllers (PICs) are used in many industries and machines.

PICs are designed for multiple analogue and digital inputs and output arrangements, extended temperature ranges, immunity to electrical noise, and resistance to vibration and impact. Programs to control machine operation are typically stored in battery-backed-up or non-volatile memory. PIC is an example of a "hard" real-time system since output results must be produced in response to input conditions within a limited time, otherwise unintended operation will result, [10].

The original PIC was built to be used with General Instrument's new CP1600 16-bit Central Processing Unit (CPU). The CP1600 had poor I/O performance, and the 8-bit PIC was developed in 1975 to improve performance of the overall system by offloading I/O tasks from the CPU. The PIC used simple microcode stored in ROM to perform its tasks, and although the term was not used at the time, it shares some common features with Reduced Instructions Set Computer (RISC) designs.

In 1985, General Instrument spun off their microelectronics division and the new ownership cancelled almost everything which by this time was mostly out-of-date. The PIC, however, was upgraded with an internal EPROM to produce a programmable channel controller. Today, a huge variety of PICs are available with various on-board peripherals (serial communication modules, Universal Asynchronous Receiver/Transmitters (UARTs), motor control kernels, etc.) and program memory from 256 words to 64k words and more (a "word" is one assembly language instruction, varying in length from 8 to 16 bits, depending on the specific PIC micro family) [11].

PIC and PIC micro are registered trademarks of Microchip Technology. It is generally thought that PIC stands for Peripheral Interface Controller, although General Instruments' original acronym for the initial PIC1640 and PIC1650 devices was "Programmable Interface Controller". The acronym was quickly replaced with "Programmable Intelligent Computer". The Microchip 16C84 (PIC16x84), introduced in 1993, was the first Microchip CPU with on-chip EEPROM memory. By 2013, Microchip was shipping over one billion PIC microcontrollers every year [10].

Table 1: Core Architecture of Peripheral Interface Controller (PIC).

Architecture	Family		Data Width	Instruction Width
8-bit MCU	PIC 10 / PIC 12 / PIC 16	Base line	8 bits	12-bits
	PIC 18	Mid-Range		14-bit
				16-bits
16-bit MCU	PIC 24	Integrated DSP	16 bits	16 bits
	DS-PIC 30			
32-bit MCU			32 bits	32 bits

Source: Peripheral Interface Controller (PIC) Micro Family Tree, 2004.

Core Architecture of Peripheral Interface Controller (PIC)

There is no distinction between memory space and register space because the RAM serves the job of both memory and registers, and the RAM is usually just referred to as the register file or simply as the registers. The PIC architecture is characterized by its multiple attributes and its families as shown in Table 1.

Data spaces of PICs have a set of registers that function as general-purpose RAM. Special-purpose control registers for on-chip hardware resources are also mapped into the data space. The addressability of memory varies depending on device series, and all PIC devices have some banking mechanism to extend addressing to additional memory. Later series of devices feature move instructions, which can cover the whole addressable space, independent of the selected bank. In earlier devices, any register move had to be achieved through the accumulator.

To implement indirect addressing, a "File Select Register" (FSR) and "INDirect register File" (INDF) are used. A register number is written to the FSR, after which reads from or writes to INDF will actually be to or from the register pointed to by FSR. Later devices extended this concept with post- and pre- increment/decrement for greater efficiency in accessing sequentially stored data. This also allows FSR to be treated almost like a Stack Pointer (SP) [11].

The code space is generally implemented as ROM, EPROM or flash RAM. In general, external code memory is not directly addressable due to the lack of an external memory interface. External data memory is not directly addressable except in some PIC18 devices with high pin count. The exceptions are PIC17 and select high pin count PIC18 devices. PICs handle data in 8-bit chunks. However, the unit of addressability of the code space is not generally the same as the data space. For example, PICs in the baseline (PIC12) and mid-range (PIC16) families have program memory addressable in the same word size as the instruction width, i.e. 12 or 14 bits respectively. In contrast, in the PIC18 series, the program memory is addressed in 8-bit increments (bytes), which differ from the instruction width of 16 bits. In order to be clear, the program memory capacity is usually stated in number of (single-word) instructions, rather than in bytes.

PICs have a hardware call stack, which is used to save return addresses. The hardware stack is not software-accessible on earlier devices, but this changed with the 18 series devices. Hardware support for a general-purpose parameter stack was lacking in early series, but this greatly improved in the 18 series, making the 18 series architecture more-friendly to high-level language compilers. PIC's instructions vary from about 35 instructions for the low-end PICs to over 80 instructions for the high-end PICs.

The instruction set includes instructions to perform a variety of operations on registers directly, the accumulator and a literal constant or the accumulator and a register, as well as for conditional execution, and program branching. Some operations, such as bit setting and testing, can be performed on any numbered register, but bi-operand arithmetic operations always involve Windows (W) which is the accumulator), writing the result back to either W or the other operand register. To load a constant, it is necessary to load it into W before it can be moved into another register. On the older cores, all register moves needed to pass through W, but this changed on the "high-end" cores.

PIC cores have skip instructions, which are used for conditional execution and branching. The skip instructions are "skip if bit set" and "skip if bit not set". Because cores before PIC18 had only unconditional branch instructions, conditional jumps are implemented by a conditional skip (with the opposite condition) followed by an unconditional branch. Skips are also of utility for conditional execution of any immediate single following instruction.

It is possible to skip instructions. For example, the instruction sequence "skip if A; skip if B; C" will execute C if A is true or if B is false. The 18 series implemented shadow, registers which save several important registers during an interrupt, providing hardware support for automatically saving processor state when servicing interrupts, [11].

The architectural decisions are directed at the maximization of speed-to-cost ratio. The PIC architecture was among the first scalar CPU designs and is still among the simplest and cheapest. The Harvard architecture, in which instructions and data come from separate sources, simplifies timing and microcircuit design greatly, and this benefits clock speed, price, and power consumption. The PIC instruction set is suited to implementation of fast lookup tables in the program space. Such lookups take one instruction and two instruction cycles. Many functions can be modeled in this way.

Optimization is facilitated by the relatively large program space of the PIC (e.g. 4096 × 14-bit words on the 16F690) and by the design of the instruction set, which allows embedded constants. For example, a branch instruction's target may be indexed by W, and execute a Return with Literal in W/Return Literal to W (RETLW), which does as it is named return with literal in W. Interrupt latency is constant at three instruction cycles.

External interrupts have to be synchronized with the four-clock instruction cycle; otherwise there can be a one instruction cycle jitter. Internal interrupts are already synchronized. The constant interrupt latency allows PICs to achieve interrupt-driven low-jitter timing sequences. An example of this is a video sync pulse generator. This is no longer true in the newest PIC models, because they have a synchronous interrupt latency of three or four cycles.

The advantages of PIC are: small instruction set to learn, Reduced Instruction Set Computer (RISC) architecture, built-in oscillator with selectable speeds, easy entry level, in-circuit programming plus in-circuit debugging, Inexpensive microcontrollers, availability of processors in Dual in-Line (DIL) package make them easy to handle for hobby use, wide range of interfaces including I²C, Serial Peripheral Interface Bus (SPI), USB, Universal Synchronous / Asynchronous Receiver / Transmitter (USART), Analog-to-Digital (A/D), Programmable Comparators, Pulse-width Modulation (PWM), Leisure Information Network (LIN), CAN, Play Station Portable (PSP) and Ethernet [11].

Peripheral Interface Controller (PIC) Control Circuit Panel

Control panel with PIC (grey elements in the center) is shown in Figure 1. The unit consists of separate elements, from left to right; power supply, controller, relay units for in- and output. The main difference from other computers is that PICs are armored for severe conditions (such as dust, moisture, heat, cold) and have the facility for extensive input/output (I/O) arrangements.

These connect the PIC to sensors and actuators. PICs read limit switches, analog process variables (such as temperature and pressure), and the positions of complex positioning systems. Some use machine vision. On the actuator side, PICs operate electric motors, pneumatic or hydraulic cylinders, magnetic relays, solenoids, or analog outputs. The input/output arrangements may be built into a simple PIC, or the PIC may have external I/O modules attached to a computer network that plugs into the PIC Scan time.

Peripheral Interface Controller (PIC) Basic Parts and System Scale

Peripheral Interface Controllers (PICs) used in larger I/O systems may have peer-to-peer (P2P) communication between processors, as indicated in Figure 2. This allows separate parts of a complex process to have individual control while allowing the subsystems to co-ordinate over the communication link. These communication links are also often used for Human Machine Interface (HMI) devices such as keypads or PC-type workstations.

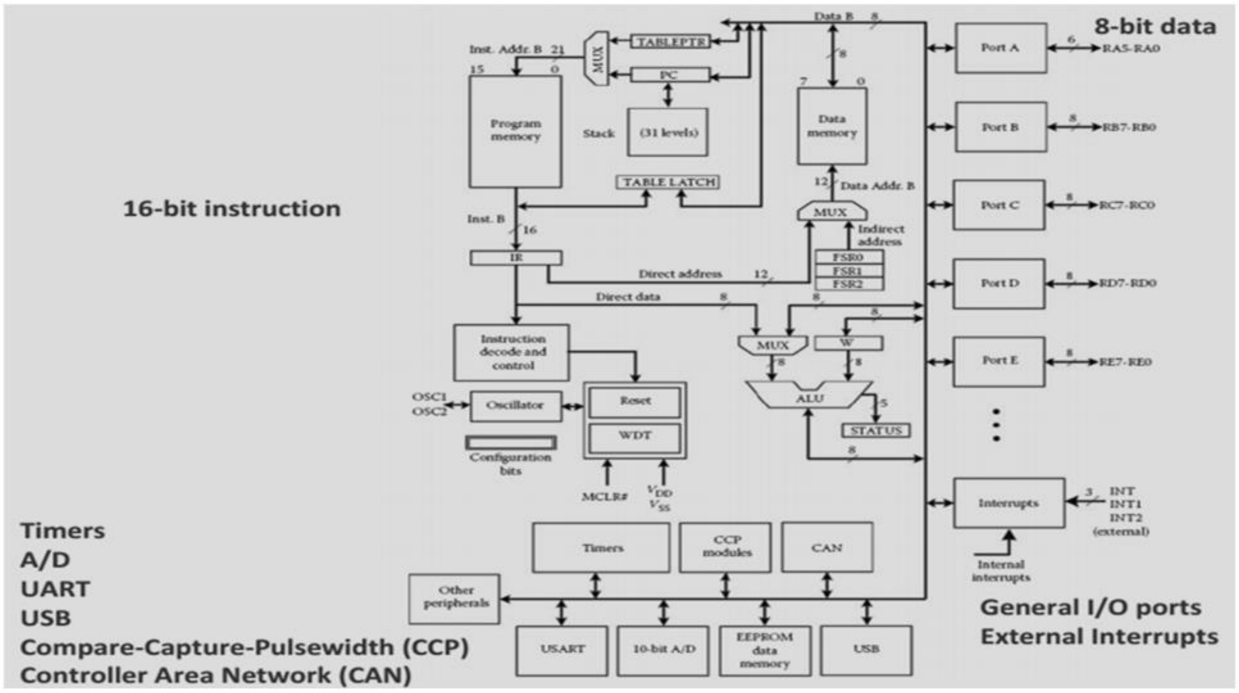

Figure 1: Peripheral Interface Controller (PIC) Control Circuit Panel
Source: Martins, [12].

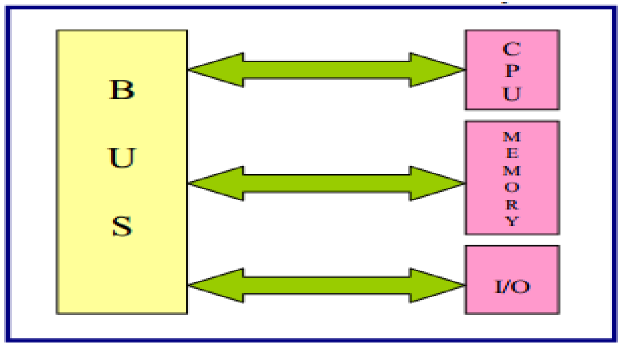

Figure 2: Peripheral Interface Controller (PIC) Basic Parts
Source: Micro Controller Broad, [13].

Peripheral Interface Controller (PIC) Programming

PIC programs are typically written in a special application on a personal computer, and then downloaded by a direct-connection cable or over a network to the PIC, as shown in Figure 3. The program is stored in the PIC either in battery-backed-up RAM or some other non-volatile flash memory. Often, a single PIC can be programmed to replace thousands of relays. PICs can be programmed using standards-based programming languages. A graphical programming notation called Sequential Function Charts is available on certain programmable controllers [14].

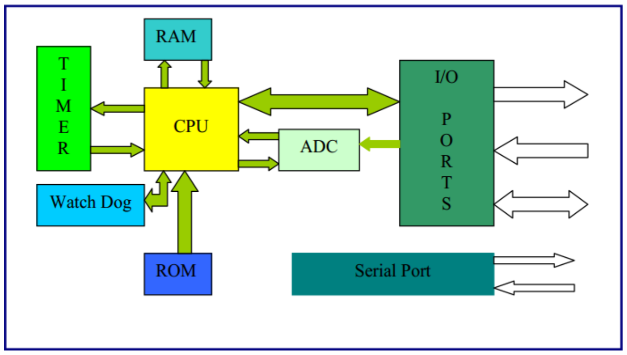

Figure 3: Peripheral Interface Controller (PIC) Block Diagram Structure
Source: Micro Controller Broad, [13].

THE DEVELOPMENT OF COMPUTER PROGRAM FOR THE WELDING INTERFACE OF THE DEVELOPED WELDING MINI-ROBOT

Algorithm and Flowchart Development

This algorithm describes the processes followed in the programming of the welding guide (See Appendix I) used to obtain the welding interface for the welding robot developed in readiness for the welding operations carried out as shown in Figure 4.

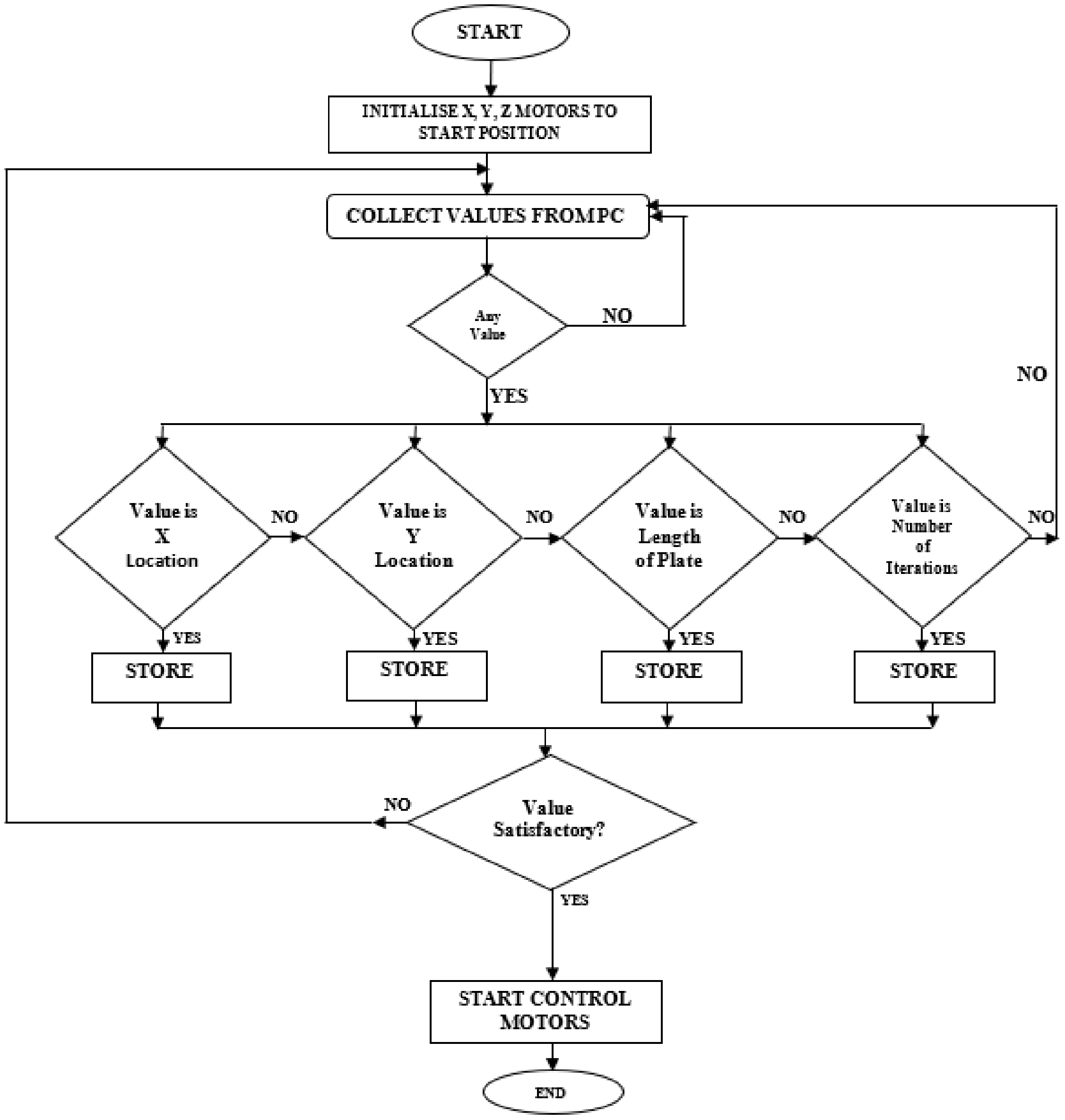

Figure 4: Algorithm for the Peripheral Interface Controller (PIC) Operation.

Modeling of the Developed Welding Robot Operation Interface

Figure 5 is the Graphical user interface using MATLAB Program for the input of parameters, such as length of weld in the X-axis, weld position in the Y-axis and the positioning of the welding tong in the Z-axis, all to control the operation of the welding robot developed as shown in Figure 6.

COMPARING THE OPERATIONAL RESULTS OF THE DEVELOPED WELDING ROBOT WITH THE MANUAL ARC WELDING

Table 2 shows the various times of weld and welding speeds at some set lengths of weld for the welded mild plates of different sizes used as test specimens for the quality of weld of the developed welding robot. While Table 3 shows the various lengths of weld and welding speeds at some set time of weld for welded mild plates of different sizes used as test specimens for the electric arc welding (Manual).

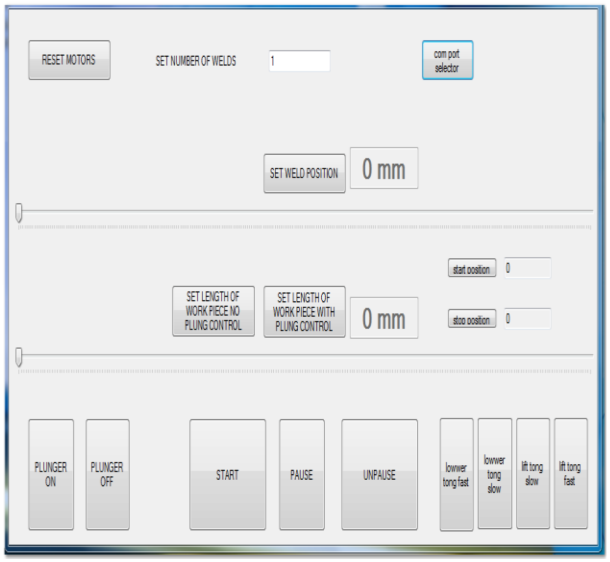

Figure 5: Developed Welding Robot Operation Interface

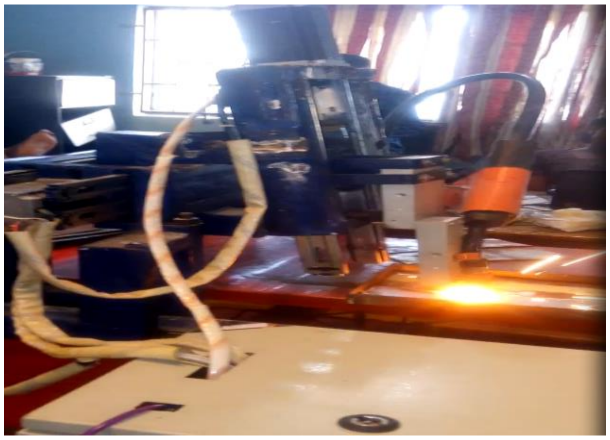

Figure 6: Developed Welding Mini-Robot in Operation.

Table 2: Time, Length of Weld and Welding Speed of Developed Welding Robot.

Welding Operation using the Developed Welding Robot						
Length of Weld (mm)	25	50	75	100	125	150
0.5 mm Mild Steel Plate						
Time of Weld (s)	5.37	10.94	17.01	21.82	26.53	32.94
Welding Speed (mm/s)	4.66	4.57	4.41	4.58	4.71	4.55
0.6 mm Mild Steel Plate						
Time of Weld (s)	5.25	10.89	16.99	21.74	26.45	32.82
Welding Speed (mm/s)	4.76	4.59	4.41	4.60	4.73	4.57
0.7 mm Mild Steel Plate						
Time of Weld (s)	5.12	10.77	16.92	21.69	26.33	32.44
Welding Speed (mm/s)	4.88	4.64	4.43	4.61	4.75	4.62
0.8 mm Mild Steel Plate						
Time of Weld (s)	5	10.66	16.84	21.64	26.3	32.43
Welding Speed (mm/s)	5.00	4.69	4.45	4.62	4.75	4.63
0.9 mm Mild Steel Plate						
Time of Weld (s)	4.88	10.58	16.75	21.42	26.02	32.42
Welding Speed (mm/s)	5.12	4.73	4.48	4.67	4.80	4.63
1.0 mm Mild Steel Plate						
Time of Weld (s)	4.7	10.13	16.51	21.3	25.74	31.52
Welding Speed (mm/s)	5.32	4.94	4.54	4.69	4.86	4.76

Table 3: Time, Length of Weld and Welding Speed of Electric Arc Welding Machine.

Welding Operation using Electric Arc Welding							
Time of Weld (s)	15	20	25	30	35	40	45
0.5 mm Mild Steel Plate							
Length of Weld (mm)	35	45	56	75	88	101	107
Welding Speed (mm/s)	2.33	2.25	2.24	2.50	2.51	2.53	2.38
0.6 mm Mild Steel Plate							
Length of Weld (mm)	45	53	61	78	92	107	135
Welding Speed (mm/s)	3.00	2.65	2.44	2.60	2.63	2.68	3.00
0.7 mm Mild Steel Plate							
Length of Weld (mm)	46	56	67	95	106	117	148
Welding Speed (mm/s)	3.07	2.80	2.68	3.17	3.03	2.93	3.29
0.8 mm Mild Steel Plate							
Length of Weld (mm)	62	88	114	123	144	165	174
Welding Speed (mm/s)	4.13	4.40	4.56	4.10	4.11	4.13	3.87
0.9 mm Mild Steel Plate							
Length of Weld (mm)	65	97	129	158	175	184	192
Welding Speed (mm/s)	4.33	4.85	5.16	5.27	5.00	4.60	4.27
1.0 mm Mild Steel Plate							
Length of Weld (mm)	70	102	134	165	180	187	195
Welding Speed (mm/s)	4.67	5.10	5.36	5.50	5.14	4.68	4.33

Table 2 indicates, for developed welding robot, welding speed range for 0.5 mm thick plate, (4.409-4.655) mms-1; 0.6 mm mild steel plate (4.414-4.762) mms-1; 0.7 mm plate, (4.433-4.883) mms-1 ; 0.8 mm plate (4.454-5.000) mms-1; 0.9 mm mild steel plate, (4.478-5.123) mms-1 and for 1.0 mm mild steel plate, (4.543-5.319) mms-1. While for manual arc welding, Table 3 indicates, welding speed range of (2.330-2.517) mms-1 for 0.5 mm mild steel; (2.440-3.000) mms-1 for 0.6 mm mild steel plate; (2.680-3.289) mms-1 for 0.7 mm plate but for thicker mild steel plates 0.8 mm - 1.0 mm, the welding speeds appreciated. For 0.8 mm plate, the welding speed ranges from (3.867-4.560) mms-1; for 0.9 mm plate, (4.267-5.267) mms-1 and for 1.0 mm plate, (4.333-5.500) mms-1.

Table 2 also indicates, for developed welding robot, welding time range for 0.5 mm thick plate, (5.37-32.94) s; 0.6 mm mild steel plate (5.25-32.82) s; 0.7 mm plate, (5.12-32.44) s; 0.8 mm plate (5-32.43) s; 0.9 mm mild steel plate, (4.88-32.42) s and for 1.0 mm mild steel plate, (4.7-31.52) s. while for manual arc welding, Table 3 also indicates, for electric arc welding (manual), welding time range of (15-45) s for 0.5 mm thick plate to 1.0 mm mild steel plate.

CONCLUSION

In conclusion, the results of the developed welding program for the welding interface of welding mini-robot when compared with the manual electric arc welding indicated that welding with the developed welding mini-robot will reduce the production cost and increase the quality as well as the reliability of weld during welding processes.

REFERENCES

1. Kah, P., M Shrestha, E. Hiltunen, and J. Martikainen. 2015. "Robotic Arc Welding Sensors and Programming in Industrial Applications". *International Journal of Mechanical and Materials Engineering*. 10(13): 1-16.

2. Ross, L.T., S.W. Fardo, J.W. Masterson, and R.L. Towers. 2010. *Robotics: Theory and Industrial Applications*. p. 47 Goodheart Willocx Company: Chicago, IL.

3. Chen, S.B, and J. Wu. 2008. "Intelligentized Technology for Arc Welding Dynamic Process". *LNEE*. 29. Springer: Heidelberg, Germany.

4. Chen, S.B. 2007. “On the Key Intelligentized Technologies of Welding Robot”. *LNCIS*. 362:105–116.

5. Cary, H.B. and S.C. Helzer. 2011. *Modern Welding Technology*. Prentice Hall: Saddle Hill, NJ.

6. Hong, T.S., M. Ghobakhloo, and W. Khaksar. 2014. “Robotic Welding Technology in Access”. https://www.researchgate.net/publication/28595126. June 20, 2020.

7. Robot Welding Company. 2013. “Benefits of Robotic Welding”. [Online]. http://www.robotwelding.co.uk/benefits-of-robot-welding.html. Accessed 20 June, 2020.

8. Robert, G. 2013. “Top 5 Advantages of Robotic Welding”. Robotiq. [Online]. http://blog.robotiq.com/bid/63115/Top-5-Advantages-of-Robotic-Welding. 20 June, 2020.

9. Ismara, K.I. and E. Prianto. 2020. “Safety Education Management in Welding Robotic Laboratory”. J. Phys.: Conf. Ser. 1446 012061: 1-6.

10. Cliton, A. 1984. “Programmable Logical Controller”. University of Coimbra, Thesis.

11. Crispin, A. 1997. *Programmable Logic Controller and their Engineering Application, 2nd Edition.* McGraw-Hill Publishing: London, UK.

12. Martins, L. 2012. *PIC Microcontroller Embedded Systems*. Hadassah College: Jerusalem, Israel.

13. Micro Controller Broad. 2015. “Peripheral Interface Controller (PIC) Basic Parts and PIC Block Diagram Structure. www.ranksays.com.

14. Pires, J.N. and J.M.G. Sá da Costa. 1999. “Programming Robotic Manufacturing Cells”. University of Reading: London, UK.

ABOUT THE AUTHORS

Dr. Oladebeye Dayo Hephzibah is a Lecturer in the Department of Mechanical Engineering, Federal Polytechnic, Ado-Ekiti. He holds a Ph.D. degree in Mechanical Engineering from the Federal University of Technology Akure. His research interests are in robotic and mechanical engineering.

Prof. Adejuyigbe Samuel Babatope is a Professor in the Mechatronics Engineering Department, Federal University, Oye-Ekiti. He holds a Ph.D. in Production Engineering. His research interests are in Computer Aided Engineering/Manufacture, Artificial Intelligence and Engineering Management.

Dr. Olorunnishola Akim Abayomi Gideon, is a Lecturer in the Department of Mechanical Engineering, Federal Polytechnic, Ado-Ekiti. He holds a Ph.D. degree in Mechanical Engineering (Applied thermo-fluid option) from the Federal University of Technology Akure. His research interests are in mechanical engineering, renewable energy and energy efficiency and automotive engineering.

APPENDIX I

WELDING INTERFACE G-CODE

```
Public Class Form1
    Dim com3 As IO.Ports.SerialPort
    Dim portnum As Integer = 0
    Dim strvalue As Integer
    Dim value As Short
    Dim valuet As Integer
    Dim valuet2 As Integer
    Dim utimer1 As Short
    Dim saveweldp As Integer = 0
    Dim saveweldp2 As Integer = 0
    Dim tempv As Integer
    Dim tempv2 As Integer
    Dim done As Integer = 0
    Dim realvalue As Integer
    Dim valuep As Integer = 0
    Dim valueup As Integer = 0

    Private Sub Form1_Load(ByVal sender As
System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
        Show()
        Button1.Enabled = False
        Button2.Enabled = False
        Button3.Enabled = False
        Button6.Enabled = False
        Button7.Enabled = False
        Button8.Enabled = False
        Button10.Enabled = False
        Button11.Enabled = False
        Button12.Enabled = False
        Button13.Enabled = False
        Button14.Enabled = False
        Button15.Enabled = False
        Button16.Enabled = False
        TrackBar1.Enabled = False
        TrackBar2.Enabled = False
        TextBox1.Enabled = False
redo:

System.Windows.Forms.Application.DoEvents()
        If portnum = 0 Then GoTo redo
        Button1.Enabled = True
        Button2.Enabled = True
        Button3.Enabled = True
        Button6.Enabled = True
        Button7.Enabled = True
        Button8.Enabled = True
        Button10.Enabled = True
        Button11.Enabled = True
        Button12.Enabled = True
        Button13.Enabled = True
        Button14.Enabled = True
        Button15.Enabled = True
         Button16.Enabled = True

        TrackBar1.Enabled = True
        TrackBar2.Enabled = True
        TextBox1.Enabled = True

check:

System.Windows.Forms.Application.DoEvents()
        If com3.BytesToRead >= 1 Then

System.Windows.Forms.Application.DoEvents()
            strvalue = com3.ReadByte()

System.Windows.Forms.Application.DoEvents()
            TextBox4.Text = TextBox4.Text &
strvalue & " "

System.Windows.Forms.Application.DoEvents()
        End If

System.Windows.Forms.Application.DoEvents()

        GoTo check

    End Sub

    Private Sub TrackBar1_ValueChanged(ByVal
sender As Object, ByVal e As System.EventArgs)
Handles TrackBar1.ValueChanged
        TextBox2.Text = TrackBar1.Value & "
mm"
        Button2.Enabled = True
    End Sub
    Private Sub TrackBar2_ValueChanged(ByVal
sender As Object, ByVal e As System.EventArgs)
Handles TrackBar2.ValueChanged
        TextBox3.Text = (TrackBar2.Value * 2)
& " mm"
        done = 0
        Button3.Enabled = True
        Button16.Enabled = True
    End Sub
    Private Sub Button6_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs)
Handles Button6.Click
        If (TextBox5.Text = "0") And
(TextBox6.Text = "0") Then Exit Sub
        If TextBox5.Text = TextBox6.Text Then
Exit Sub
        If TextBox5.Text > TextBox6.Text Then
Exit Sub
        Dim holder As Integer
        holder = TextBox1.Text
        Do While holder <> 0
            Call process3()
            Call process3()
            Call process1()
            Call process2()
            holder = holder - 1
        Loop
    End Sub
    Private Sub Button7_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs)
Handles Button7.Click
        Button7.Enabled = False
        valueup = 1
        Button7.Enabled = True
    End Sub
    Private Sub Button9_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs)
Handles Button9.Click
        On Error GoTo errorcode
        portnum = InputBox("enter comport
number", , 1, )
        com3 = _
    My.Computer.Ports.OpenSerialPort("COM" &
portnum, 2400)
        com3.Encoding =
System.Text.Encoding.Default
        Exit Sub
errorcode:
        If portnum = "0" Then End
        MsgBox("invalid port number")
        End
    End Sub
    Private Sub Button2_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs)
Handles Button2.Click
        Button2.Enabled = False
        valuet = TrackBar1.Value
        valuet = valuet * 32768
        valuet = valuet / 290
        tempv = valuet
        If valuet >= saveweldp Then
            valuet = valuet - saveweldp
            com3.Write(Chr(3))
            utimer1 = 0
            Timer1.Enabled = True
            Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
            Loop
            Timer1.Enabled = False
check:

System.Windows.Forms.Application.DoEvents()
            If com3.BytesToRead >= 1 Then
                strvalue = com3.ReadByte()
                TextBox4.Text = TextBox4.Text
& strvalue & " "
                GoTo check2
            End If
            GoTo check
            Loop
            Timer1.Enabled = False
checkb:

System.Windows.Forms.Application.DoEvents()
            If com3.BytesToRead >= 1 Then
                strvalue = com3.ReadByte()
                TextBox4.Text = TextBox4.Text
& strvalue & " "
                GoTo check2b
            End If

            GoTo checkb
check2b:

            If strvalue = 66 Then
                MsgBox(" busy")
                Exit Sub
            End If
            saveweldp = tempv
            GoTo proceed
        End If
```

```
proceed:
        valuet2 = 0
recheck:
        If valuet >= 256 Then
            valuet2 = valuet2 + 1
            valuet = valuet - 256
            GoTo recheck
        End If
        com3.Write(Chr(valuet))
        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False
        com3.Write(Chr(valuet2))
        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False

    End Sub
    Private Sub Button3_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs)
Handles Button3.Click
        Button3.Enabled = False
        Call process3()
        Call process3()
        valuet = TrackBar2.Value * 2
        valuet = valuet * 50176
        valuet = valuet / 480
        tempv2 = valuet
        If valuet >= saveweldp2 Then
            valuet = valuet - saveweldp2
            com3.Write(Chr(2))
            utimer1 = 0
            Timer1.Enabled = True
            Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
            Loop
           Timer1.Enabled = False
check:

System.Windows.Forms.Application.DoEvents()
            If com3.BytesToRead >= 1 Then
                strvalue = com3.ReadByte()
                TextBox4.Text = TextBox4.Text
& strvalue & " "
                GoTo check2
            End If
            GoTo check
            Loop
            Timer1.Enabled = False
checkb:

System.Windows.Forms.Application.DoEvents()
            If com3.BytesToRead >= 1 Then
                strvalue = com3.ReadByte()
                TextBox4.Text = TextBox4.Text
& strvalue & " "
                GoTo check2b
            End If
            GoTo checkb
check2b:

            If strvalue = 66 Then
                MsgBox(" busy")
                Exit Sub
            End If
            saveweldp2 = tempv2
            GoTo proceed
        End If
proceed:
        done = 1
        realvalue = saveweldp2
        valuet2 = 0
recheck:
        If valuet >= 256 Then
            valuet2 = valuet2 + 1
            valuet = valuet - 256
            GoTo recheck
        End If
        com3.Write(Chr(valuet))
        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False
        com3.Write(Chr(valuet2))
        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False

    End Sub

    Private Sub Button1_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
        Button1.Enabled = False
        valuet = 1

        com3.Write(Chr(valuet))
        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False

check:

System.Windows.Forms.Application.DoEvents()
        If com3.BytesToRead >= 1 Then
            strvalue = com3.ReadByte()
            TextBox4.Text = TextBox4.Text &
strvalue & " "
            GoTo check2
        End If

        GoTo check
check2:

        If strvalue = 42 Then
            MsgBox(" busy")
            Button1.Enabled = True
            Exit Sub
        End If
        saveweldp = 0
        saveweldp2 = 0
        Button1.Enabled = True
    End Sub

    Private Sub Timer1_Tick(ByVal sender As
System.Object, ByVal e As System.EventArgs)
Handles Timer1.Tick
        utimer1 = 1
    End Sub

    Private Sub Button8_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs)
Handles Button8.Click
        Button8.Enabled = False
        valuep = 1
        Button8.Enabled = True

    End Sub

    Private Sub Button4_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs)
Handles Button4.Click
        If done = 1 Then
            TextBox5.Text = realvalue
        End If
    End Sub

    Private Sub Button5_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs)
Handles Button5.Click
        If done = 1 Then
            TextBox6.Text = realvalue
        End If
    End Sub

    Private Sub Button10_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs)
Handles Button10.Click
        Button10.Enabled = False

        valuet = 4
        com3.Write(Chr(valuet))
        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False
check:

System.Windows.Forms.Application.DoEvents()

        If com3.BytesToRead >= 1 Then
            strvalue = com3.ReadByte()
            TextBox4.Text = TextBox4.Text &
strvalue & " "
            GoTo check2
        End If

        GoTo check
check2:

        If strvalue = 66 Then
            MsgBox(" busy")
            Button10.Enabled = True
            Exit Sub
        End If

        valuet = 250
        valuet2 = 0
recheck:
        If valuet >= 256 Then
            valuet2 = valuet2 + 1
            valuet = valuet - 256
            GoTo recheck
        End If

        com3.Write(Chr(valuet))
        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False
        com3.Write(Chr(valuet2))
        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False
        Button10.Enabled = True
    End Sub

    Private Sub Button11_Click(ByVal sender
As System.Object, ByVal e As
System.EventArgs) Handles Button11.Click
        Button11.Enabled = False
        valuet = 7

        com3.Write(Chr(valuet))
        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False

check:

System.Windows.Forms.Application.DoEvents()
        If com3.BytesToRead >= 1 Then
            strvalue = com3.ReadByte()
            TextBox4.Text = TextBox4.Text &
strvalue & " "
            GoTo check2
        End If

        GoTo check
check2:

        If strvalue = 66 Then
            MsgBox(" busy")
            Button11.Enabled = True
            Exit Sub
        End If

        valuet = 250
        valuet2 = 0
        Loop
        Timer1.Enabled = False

check:

System.Windows.Forms.Application.DoEvents()

        If com3.BytesToRead >= 1 Then
            strvalue = com3.ReadByte()
            TextBox4.Text = TextBox4.Text &
strvalue & " "
            GoTo check2
        End If

        GoTo check

check2:

        If strvalue = 66 Then GoTo xyz
        valuet = TextBox6.Text - saveweldp2
        valuet2 = 0
recheck:

        If valuet >= 256 Then
            valuet2 = valuet2 + 1
            valuet = valuet - 256
            GoTo recheck
        End If

        com3.Write(Chr(valuet))
        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False
        com3.Write(Chr(valuet2))
        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False
    End Sub

    Public Sub process2()
xyz:
        If valuep = 1 Then Call processpause()
        If valueup = 1 Then Call
processunpause()

        com3.Write(Chr(5))

        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False

check:

System.Windows.Forms.Application.DoEvents()

        If com3.BytesToRead >= 1 Then
            strvalue = com3.ReadByte()
            TextBox4.Text = TextBox4.Text &
strvalue & " "
            GoTo check2
        End If

        GoTo check
check2:

        If strvalue = 66 Then GoTo xyz

        valuet = TextBox6.Text - saveweldp2
```

```
        valuet2 = 0
recheck:

        If valuet >= 256 Then
            valuet2 = valuet2 + 1
            valuet = valuet - 256
            GoTo recheck
        End If

        com3.Write(Chr(valuet))

        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False
        com3.Write(Chr(valuet2))
        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False
    End Sub
    Public Sub process3()
        com3.Write(Chr(8))

        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False

check:

System.Windows.Forms.Application.DoEvents()

        If com3.BytesToRead >= 1 Then
            strvalue = com3.ReadByte()
            TextBox4.Text = TextBox4.Text &
strvalue & " "
            Exit Sub
        End If

        GoTo check
    End Sub

    Public Sub process4()
        com3.Write(Chr(9))

        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False
check:

System.Windows.Forms.Application.DoEvents()
        If com3.BytesToRead >= 1 Then
            strvalue = com3.ReadByte()
            TextBox4.Text = TextBox4.Text &
strvalue & " "
            Exit Sub
        End If
        GoTo check
    End Sub

    Private Sub Button12_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs)
Handles Button12.Click
        Button12.Enabled = False
        Call process3()
        Button12.Enabled = True

    End Sub

    Private Sub Button13_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs)
Handles Button13.Click
        Button13.Enabled = False
        Call process4()
        Button13.Enabled = True
    End Sub
    Public Sub process5()
xyz:
        com3.Write(Chr(4))

        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False

check:

System.Windows.Forms.Application.DoEvents()

        If com3.BytesToRead >= 1 Then
            strvalue = com3.ReadByte()
            TextBox4.Text = TextBox4.Text &
strvalue & " "
            GoTo check2
        End If

        GoTo check
check2:

        If strvalue = 66 Then GoTo xyz
        valuet = 250

        valuet2 = 0
recheck:

        If valuet >= 256 Then
            valuet2 = valuet2 + 1
            valuet = valuet - 256
            GoTo recheck
        End If

        com3.Write(Chr(valuet))

        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False

        com3.Write(Chr(valuet2))

        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False
    End Sub
    Public Sub process6()
xyz:
        com3.Write(Chr(7))

        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False
check:

System.Windows.Forms.Application.DoEvents()
        If com3.BytesToRead >= 1 Then
            strvalue = com3.ReadByte()
            TextBox4.Text = TextBox4.Text &
strvalue & " "
            GoTo check2
        End If

        GoTo check
check2:

        If strvalue = 66 Then GoTo xyz
        valuet = 250

        valuet2 = 0
recheck:

        If valuet >= 256 Then
            valuet2 = valuet2 + 1
            valuet = valuet - 256
            GoTo recheck
        End If

        com3.Write(Chr(valuet))
        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False

        com3.Write(Chr(valuet2))

        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False
    End Sub

    Private Sub Button14_Click(ByVal sender
As System.Object, ByVal e As
System.EventArgs) Handles Button14.Click
        Button14.Enabled = False
        valuet = 4

        com3.Write(Chr(valuet))
        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False
check:

System.Windows.Forms.Application.DoEvents()

        If com3.BytesToRead >= 1 Then
            strvalue = com3.ReadByte()
            TextBox4.Text = TextBox4.Text &
strvalue & " "
            GoTo check2
        End If

        GoTo check
check2:

        If strvalue = 66 Then
            MsgBox(" busy")
            Button14.Enabled = True
            Exit Sub
        End If

        valuet = 6000
        valuet2 = 0
recheck:

        If valuet >= 256 Then
            valuet2 = valuet2 + 1
            valuet = valuet - 256
            GoTo recheck
        End If

        com3.Write(Chr(valuet))
        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False

        com3.Write(Chr(valuet2))
        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False
        Button14.Enabled = True
    End Sub
    Public Sub processpause()
        com3.Write(Chr(10))
        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False

check:

System.Windows.Forms.Application.DoEvents()
        If com3.BytesToRead >= 1 Then
            strvalue = com3.ReadByte()
            TextBox4.Text = TextBox4.Text &
strvalue & " "
            Exit Sub
        End If

        GoTo check
        valuep = 0
        Call process4()
        Call process4()

    End Sub
    Public Sub processunpause()
        com3.Write(Chr(11))
        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False
check:

System.Windows.Forms.Application.DoEvents()

        If com3.BytesToRead >= 1 Then
            strvalue = com3.ReadByte()
            TextBox4.Text = TextBox4.Text &
strvalue & " "
            Exit Sub
        End If

        GoTo check
        valueup = 0
        Call process3()
        Call process3()

    End Sub

    Private Sub Button16_Click(ByVal sender
As System.Object, ByVal e As
System.EventArgs) Handles Button16.Click
        Button16.Enabled = False

        valuet = TrackBar2.Value * 2
        valuet = valuet * 50176
        valuet = valuet / 480
        tempv2 = valuet

        If valuet >= saveweldp2 Then
            valuet = valuet - saveweldp2
            com3.Write(Chr(2))
            utimer1 = 0
            Timer1.Enabled = True
            Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
            Loop
            Timer1.Enabled = False
check:

System.Windows.Forms.Application.DoEvents()

            If com3.BytesToRead >= 1 Then
                strvalue = com3.ReadByte()
                TextBox4.Text = TextBox4.Text
& strvalue & " "
                GoTo check2
            End If

            GoTo check
check2:
```

```
            If strvalue = 66 Then
                MsgBox(" busy")
                Exit Sub
            End If
            saveweldp2 = tempv2
            GoTo proceed
        End If
        If valuet < saveweldp2 Then
            valuet = saveweldp2 - valuet

            com3.Write(Chr(5))

            utimer1 = 0
            Timer1.Enabled = True
            Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
            Loop
            Timer1.Enabled = False
checkb:

System.Windows.Forms.Application.DoEvents()

            If com3.BytesToRead >= 1 Then
                strvalue = com3.ReadByte()
                TextBox4.Text = TextBox4.Text
& strvalue & " "
                GoTo check2b
            End If

            GoTo checkb
check2b:
            If strvalue = 66 Then
                MsgBox(" busy")
                Exit Sub
            End If

            saveweldp2 = tempv2
            GoTo proceed
        End If

proceed:
        done = 1
        realvalue = saveweldp2
        valuet2 = 0
recheck:

        If valuet >= 256 Then
            valuet2 = valuet2 + 1
            valuet = valuet - 256
            GoTo recheck
        End If

        com3.Write(Chr(valuet))

        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False

        com3.Write(Chr(valuet2))
        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False

    End Sub

    Private Sub Button15_Click(ByVal sender
As System.Object, ByVal e As
System.EventArgs) Handles Button15.Click
        Button15.Enabled = False
        valuet = 7
        com3.Write(Chr(valuet))

        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False
check:

System.Windows.Forms.Application.DoEvents()

        If com3.BytesToRead >= 1 Then
            strvalue = com3.ReadByte()
            TextBox4.Text = TextBox4.Text &
strvalue & " "
            GoTo check2
        End If

        GoTo check
check2:

        If strvalue = 66 Then
            MsgBox(" busy")
            Button15.Enabled = True
            Exit Sub
        End If

        valuet = 3000
        valuet2 = 0
recheck:

        If valuet >= 256 Then
            valuet2 = valuet2 + 1
            valuet = valuet - 256
            GoTo recheck
        End If

        com3.Write(Chr(valuet))
        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False
        com3.Write(Chr(valuet2))

        utimer1 = 0
        Timer1.Enabled = True
        Do While utimer1 = 0

System.Windows.Forms.Application.DoEvents()
        Loop
        Timer1.Enabled = False
        Button15.Enabled = True
    End Sub
End Class
```

SUGGESTED CITATION

Oladebeye, D.H., S.B. Adejuyigbe, and A.A.G. Olorunnishola. 2020. "The Development of Computer Program for the Welding Interface of the Developed Welding Mini-Robot". *Pacific Journal of Science and Technology*. 21(2):12-28.

Pacific Journal of Science and Technology