VCA Protocol-based Multi-level Flexible Architecture on Embedded PLCs for Visual Servo Control

***Abstract*—The visual system, motion control system, and programmable logic controller (PLC) system are becoming increasingly inseparable and important. However, their numerous types of programs and hardware, the many communication protocols among them, the mixed development methods, and the ever-growing high requirements and complexity lead to a cumbersome task of application implementation for users. Few works are researching the integration of these three systems to ease complexity. Most of them are focusing on their applications individually.**

We propose herein a flexible multi-level architecture based on a vision control algorithm (VCA) protocol to decrease the complexity of ever-growing integrated applications. This multi-level architecture includes a flexible layer, a control layer, and an algorithm layer. The flexible layer is adopted to seamlessly integrate the visual system and the embedded PLC ($ePLC$). The VCA protocol is designed for data interaction between the layers. Correspondingly, customized hardware, memory allocation, and Petri-Net-based multithreading structure are described to support the proposed flexible architecture. We implement two cases using the proposed VCA protocol-based flexible architecture, in which the generality of the proposed flexible architecture is verified. In the first case, a winding machine with a visual system implements a regular winding effect by correcting θ. In the second case, a binocular catching robot uses cameras to track the trajectory and send to $ePLC$. The robot can successfully catch the ball by adjusting the speed and the position. The cases indicate that the proposed VCA-based architecture could easily be applied between two different cases.

***Index Terms*—embedded PLC, motion control, multi-level architecture, Petri Net, visual servo control**

I. Introduction

Integration technologies are driving industrial automation [1]. The continuous integration of sensors, controllers, robots, tools, etc. has brought the concepts of self-aware equipment, intelligent factory, and cyber-physical system, among others [2], [3]. Recent studies [4]–[6] introduced the development of integration technologies. In industrial automation, the PLC system, motion control system, and visual system have become increasingly important and inseparable, as can be seen in various domains [7], [8].

A. Motivations

Recent advances in image processing and pattern recognition contributed to the thriving of the visual system, which is being applied in various fields. The visual system could obtain parameters to replace the human visual system and address many tasks by extracting features from the image, especially the ever-growing requirements in industrial scenarios (e.g., works that should be finished in a dangerous environment, tasks that the human vision finds difficult to satisfy, and applications in some large-scale industrial productions, in which numerous visual systems are needed).

After visual processing, the motion control system, as the power of automation, is normally needed to drive some actuators to finish severe tasks, which remarkably benefits labor force replacement.

Furthermore, PLCs have become a base of automation because of their high reliability and easy programming [9]. Many studies are focusing on PLCs to extremely extend their applied fields. For instance, [10] guaranteed reliability by verifying the program of PLCs; [11] improved their performance using advanced algorithms; [12] alleviated their development complexity with a special software structure; and [13] presented methods to dynamically update PLC programs.

As mentioned before, the visual, motion control, and PLC systems are becoming increasingly inseparable. [14] described a typical case about how these three collaborate. The visual system analyzes the context and obtains error put into the motion control system. Simultaneously, the PLC analyzes information, such as the position limitation of every axis, to make some logical judgments accordingly. Regarding the normal development of these applications, the problems are met as follows:

1) The visual, motion control, and PLC systems are individually developed and combined using the communication protocol. However, this method should always be redesigned with the programs in these systems because of the difference of the visual and motion control algorithms and their organized logic programs. Moreover, many communication protocols (e.g., EtherCat, Modbus, CAN) are using these systems.
2) The visual, logic, and motion control programs are always mixed together and developed within numerous visual systems, PLCs, and motion controllers, which increases the unnecessary complexity.
3) The requirements from customers are also increasing. Combined with the abovementioned two points, the development of corresponding applications has become a cumbersome task for developers.

Hence, how to pose a flexible structure for the integration of the visual, motion control, and PLC systems, which will ease the complexity, has attracted our interest.

B. Our Contributions

The contributions of our study are as follows:

1) We propose herein a VCA protocol for generic data interaction between $ePLC$ and VS, which benefits the reuse of algorithms and the uniform use of communication protocols.
2) A multi-level flexible architecture is posed for individual programming, which includes the flexible, control, and algorithm layers. Correspondingly, the customized hardware, memory allocation, and Petri-Net-based multi-threading structure are described to support the proposed flexible architecture.
3) The VCA protocol-based multi-level flexible architecture implements further integration of the visual, motion control, and PLC systems. The two implemented cases indicate that the proposed architecture could reduce the complexity of one application and from one to another.

The remainder of this paper is structured as follows: Section II introduces the related works; Section III presents the hardware and software structure and memory allocation; Section IV discusses the mechanism of the VCA protocol in detail; Section V illustrates the execution process of the proposed system; Section VI presents the two implemented cases (i.e., binocular catching robot and winding machine with a visual system); and the last section concludes our work.

II. Related Works

The visual control system combined with the special motion control system and the visual system is applied in various fields, including the transportation [15], circuit detection [16], sorting system, welding [14], assembly [17], robotics [18], [19], unmanned aerial vehicle [20], and sorting [21] fields. Works have addressed problems in relevant fields; however, all solutions are based on the special motion control and visual systems.

The integration of logic control and motion control has also been presented in various deep studies [22]–[24]. [22], [25] directly realized motion control in the PLC. [26], [27] used a motion control module collaborated with the PLC to implement applications. However, it caused confusion as regards the development method in these works; hence, the PLCopen organization released a related standard [28], which standardized the motion control in the PLC. Based on this standardization, [29] provided an advanced implementation in the distributed automation system, and companies, such as 3S [30], provided some tools for users. [12] also presented a customized real-time compilation method to reduce the development complexity.

The abovementioned works have provided impressive integrations on the visual servo control system and the PLC with motion control functions; however, only a few works have investigated the integration of the visual, motion control, and PLC systems to decrease complexity. Most applications focus on their applications with three individual systems (e.g., [14]); hence, a further integration structure of the PLC that functions with motion control and the visual servo system should be provided to reduce the complexity and expand the application fields.

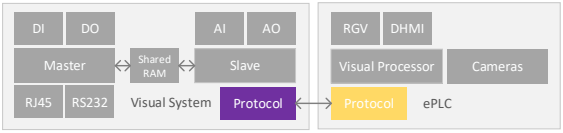

Fig. 1. Hardware architecture of the typical $ePLC$ and VS. The $ePLC$ adopts a two-processor architecture, while the VS contains a camera and a processor. The communication between the $ePLC$ and the VS adopts the CAN Bus.

III. System Architecture

A. Hardware Structure

Fig.1 shows hardware comprising $ePLC$ and the visual system (VS). The $ePLC$ is a customized structure. The number of digital input/output, analog input/output, and controlled servo system could be increased according to requirements. Particularly, the number of $ePLC$ processors could be customized for high-performance motion control applications. In motion control function-contained applications, more than two processors are always adopted, wherein one processor is mainly used for logic control, and the others are used for motion control. Fig. 1 shows a typical $ePLC$, which adopts a two-processor architecture: a master and a slave processor. Shared memory is used for data interaction between the master and slave processors. The VS normally contains one or more cameras and a processor for visual tasks. The communication between the $ePLC$ and the VS could adopt multiple protocols, such as TCP, Modbus, and CAN.

B. Software Structure

Fig. 2 shows software of the proposed multi-level flexible structure. To the best of our knowledge, the VS commonly contains several visual algorithms, and every algorithm could extract some required parameters from pictures or videos. Meanwhile, the $ePLC$ comprises modules, which include logic part and algorithms. The algorithms here mainly denote the motion control algorithms. The structure contains three layers: flexible layer (FL), control layer (CL), and algorithm layer (AL).

Flexible layer: This layer is responsible for joining the VS and the $ePLC$ consisting of the PLC interface and the visual interface. The parameters are framed with the VCA protocol frame (henceforth, the VCA protocol frame is the abbreviated form of PF) that interacted between the VS and the $ePLC$. A special protocol template (PT) is also saved in both systems to illustrate the protocol.

Control layer: This layer is separated from the algorithm layer to cope with the logic tasks, making it possible to run the logic task and the algorithm task in different processors.

Algorithm layer: This layer mainly comprises various types of algorithms. The independent algorithm layer allows the algorithms to achieve a high-performance requirement that could be built in individual processors.

C. Memory Allocation

The dedicated storage area of the $ePLC$ in the memory is made up of the bit data area (M area) and the byte data area

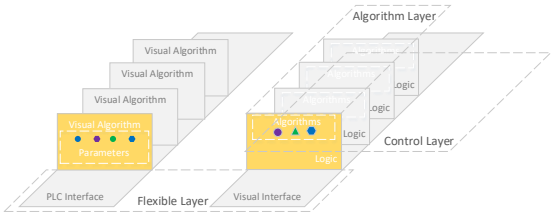

Fig. 2. The multi-level flexible architecture contains three layers: flexible, control, and algorithm layers. The VS contains several visual algorithms used to extract parameters from pictures or videos. The flexible layer consists of the PLC interface and the visual interface used to combine the $ePLC$ and the VS. The control layer is mainly designed for the logic program. The algorithm layer comprises many algorithms, and every algorithm has several parameters.

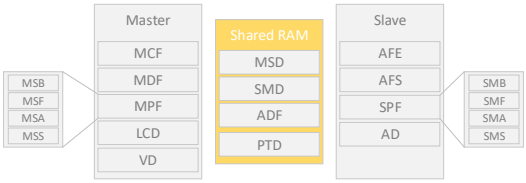

Fig. 3. Memory allocation of the master processor, shared RAM, and slave processor.

(D area). We regard the M and D areas as the set M of bit and the set D of byte, respectively. Henceforth, if set S $\exists$, we define its subscripted lowercase letter s_i as an element of S, and the subscripted i is used to distinguish the elements. Fig. 3 shows the memory allocation of the master processor, shared RAM, and slave processor according to hardware of Fig. 1. The shared RAM is a special structure for the fast data interaction between the master and slave processors. In more general cases, the MSD is located in the master processor; SMD and ADF are located in the slave processor; and PT is in both the master and slave processors.

The RAM in the master processor contains special areas: 1) the module control flag area (MCF) is used to start the modules; 2) the master processor data interaction flag area (MPF) contains the begin data transfer flag from master to slave (MSB), transferring the state of master from master to slave (MSF), acknowledging flag of master from master to slave (MSA), and transferring the state of master from slave to master (MSS); 3) the module data saved flag area (MDF) denotes whether the module data are saved; 4) the logic control data area (LCD) is used to store the control frame, where every element lcd_i is associated with a specified module; and 5) the VCA protocol frame data area (VD) stores all the frames received from the VS.

In the shared RAM, it could be read by both the master and slave processors and consists of special areas: 1) the master processor data interaction data area (MSD) stores the data delivered from the slave processors; 2) the slave processor data interaction data area (SMD) stores the data delivered from the master processor; 3) the algorithm data saved flag area (ADF) denotes whether the algorithm data are saved; and 4) the protocol template data area (PTD) stores the protocol

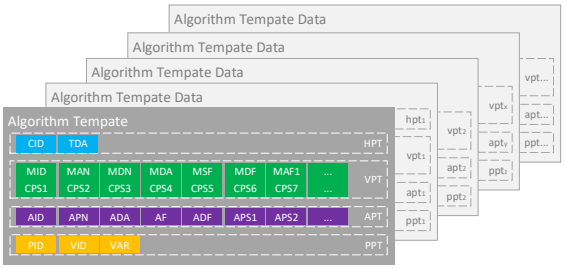

Fig. 4. A PT uniquely corresponds to a type of application and contains four parts: head of the protocol template, visual layer template, control layer template, and algorithm layer template.

template containing the PTs from the VS to the $ePLC$ and from the $ePLC$ to the VS.

In the slave processor, the RAM comprises special areas: 1) the algorithm flag area (AF) includes the algorithm flag of execution (AFE) and the algorithm flag of state (AFS); 2) the slave processor data interaction flag area (SPF) includes the begin data transfer flag from slave to master (SMB), transferring the state of slave from slave to master (SMF), acknowledging flag of slave from slave to master (SMA), and transferring the state of slave from master to slave (SMS); and 3) the algorithm data area (AD) saves data that help in the specified algorithm execution.

We define $\mathcal{I}$ for the data interaction between the master and slave processors, and it contains two parts: transferring data from master to slave ($\mathcal{I}_{mts}$) and transferring data from slave to master ($\mathcal{I}_{stm}$):

$$\begin{cases} \mathcal{I}_{mts} = \mathcal{L}(msb_i, msf_i, sma_i, sms_i, smd_i), \\ \mathcal{I}_{stm} = \mathcal{L}(smb_i, smf_i, msa_i, mss_i, msd_i), \end{cases} \tag{1}$$

where $\mathcal{L}$ is the function to implement the data interaction process between the master and slave processors. $\mathcal{I}_{mts}$ and $\mathcal{I}_{mts}$ use the same function $\mathcal{L}$. The $\mathcal{I}_{mts}$ process is described as follows: the master processor sets msb_i to one, sets msf_i to one, sends data to smd_i, recovers msb_i to zero, and informs the slave processor to obtain the data; the slave processor sets sms_i to one, checks the data of msd_i, sets sma_i to one, feeds back the master processor to end the transferring, recovers sms_i to zero, and recovers sma_i to zero; and the master processor recovers msf_i to zero.

IV. Integration of VS and $ePLC$

A. VCA PT

For most applications, we design the VCA protocol for data interaction among the layers. As shown in Fig. 4, a protocol template (PT) was adopted to support various types of implementations, and the PT uniquely corresponded to a type of application. In the flexible architecture, the users only needed to redesign and reload the PT. They can then reuse the visual servo system. The PT could be loaded into a stationary address of the visual and $ePLC$ systems. After restarting both systems, it will be stored into a fixed area of the RAM. The parsing modules from both systems will read it when parsing

the PF. The VCA protocol could be used to bidirectionally transfer the PF using the same PT and algorithms of framing and deframing.

PT is defined below:

$$\begin{cases} PT = \{HPT, VPT, APT, PPT\} \\ HPT = \{CID, TDA\} \\ VPT = \{MID, MAN, MDN, MDA, MSF, MDF, \\ \qquad \bigcup_{x=1}^{i} MAF_x, \bigcup_{x=1}^{j} CPS_x\} \\ APT = \{AID, APN, ADA, AF, ADF, \bigcup_{y=1}^{i} APS_y\} \\ PPT = \{PID, VID, VAR\} \end{cases} \tag{2}$$

The PT contains four parts: head of the protocol template (HPT), VCA protocol template (VPT), algorithm protocol template (APT), and parameter protocol template (PPT). Every part is explained below:

1) HPT: This part includes a communication unique ID (CID) and a template data storage address (TDA). Every PT only has one HPT.
2) VPT: This part consists of a module unique ID (MID), a module contained algorithm number (MAN), a module contained data number (MDN), a module data start address (MDA), a module start flag (MSF), a module data saved flag (MDF), and module contained algorithm IDs (MAS_x). VLT is not $\emptyset$. Every VPT includes MAN algorithm IDs.
3) APT: This comprises an algorithm unique ID (AID), an algorithm contained parameter number (APN), an algorithm data start address ADA, an algorithm data saved flag (ADF), and an algorithm contained parameter IDs (APS_y). CLT is not $\emptyset$. Every APT includes APN parameter IDs.
4) PPT: This contains a parameter unique ID (PID), its relevant visual algorithm ID (VID), and the conversion ratio of the visual and motion algorithm parameters (VAR).

B. VCA PF

The VCA protocol frame (PF) contains parameter and algorithm frames in its data field. The algorithm protocol frame contains several parameter frames, as shown in 5 (a).

1) PF: This comprises items of MID, visual frame length (VFL), data from the parameter frames ($PFDATA$), and data from the algorithm frames ($AFDATA$). $PFDATA$ and $AFDATA$ contain several parameter and control frames, respectively. The MIDs both in the PT and the PF need a one-to-one correspondence.
2) Algorithm frame: This comprises AID, control frame length (CFL), and $PFDATA$. The AIDs in both the algorithm frame and the PF need a one-to-one correspondence.
3) Parameter frame: This contains PID and $PDATA$. The PID is also the address of the $PDATA$. The PIDs both in the parameter frame and PF need a one-to-one correspondence.

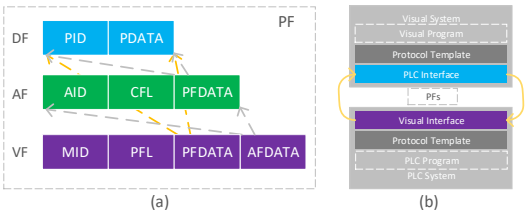

Fig. 5. (a) The VCA protocol frame contains the parameter and algorithm frames in its data field. The algorithm protocol frame contains several parameter frames. (b) PFs interaction between $ePLC$ and VS.

C. Framing of PF

The framing contains three steps: VCA framing ($VCAF$), algorithm framing (AMF), and parameter framing (PRF). The transferring data area (TD) saves the data to be transferred, and is indexed by the PID. In the VS, it represents the parameters obtained from the visual algorithms. In the $ePLC$, the data come from the RAM.

VCAF: The process is realized as Algorithm 1. It searches for the PT with mid to find the relevant vlt_x. We can obtain man and mdn through it. According to man and mdn, algorithms 3 and 2 are called to gain $pfdata$ and $afdata$, respectively. This is then finished after calculating the length (vfl).

AMF: Algorithm 2 illustrates the process and seeks the PT to gain alt_y, which contains apn. According to apn, call Algorithm 3 to obtain $pfdata$, then calculate the length (cfl) to finish the AL framing.

PRF: Algorithm 3 shows the process. The VS obtains alt_z from FT, then combines pid and $pdata$.

Algorithm 1: $VCAF$

Input: mid, TD
Output: PF
SearchPT (mid, vpt_x);
$PF.mid \leftarrow mid$;
for $i = 0; i < vpt_x.mdn; i++$ **do**
  PRF ($vpt_x.cps$[i], TD,$pfdata$);
end
for $i = 0; i < vpt_x.man; i++$ **do**
  AMF ($vpt_x.mas$[i], TD,$afdata$);
end
$PF.pfdata \leftarrow pfdata$;
$PF.afdata \leftarrow afdata$;
$PF.vfl \leftarrow (vpt_x.mdn + vpt_x.man) \times 4 + 8$;

D. Deframing of PF

The process of PF deframing is divided into VCA deframing ($VCAD$), CL deframing (AMD), and AL deframing (PRD). The receiving data (RD) area is used to save the deframing parameters. $RD1$, $RD2$, and $RD3$ are the different temporarily stored data area. In VS, $RD3$ is the data fed back to the visual algorithms. In the $ePLC$, $RD1$, $RD2$, and $RD3$ denote LCD, MSD, and AD, respectively.

Algorithm 2: AMF

Input: aid, TD, $afdata$
Output: $afdata$
SearchPT (aid, apt_y);
Create clf;
$af.aid \leftarrow aid$;
for $i = 0; i < apt_y.apn; i++$ **do**
  PRF($apt_y.aps$[i], TD, $pfdata$);
end
$af.cfl \leftarrow apt_y.apn \times 4 + 8$;
$af.pfdata \leftarrow pfdata$;
$afdata \leftarrow afdata + clf$;

Algorithm 3: PRF

Input: pid, TD,$pfdata$
Output: $pfdata$
SearchPT (pid, ppt_z);
Create pf;
$pf.pid \leftarrow pid$;
$pf.pdata \leftarrow TD[pid] \times ppt_z.var$;
$pfdata \leftarrow pfdata + alf$;

VCAD: As illustrated in Algorithm 4, obtain mid and pfl from the starting four and following 4 bytes of data of PDF, respectively. Search for PT to gain the relevant vpt_x. Send the $pfl - 8$ bytes starting from the eighth byte of $PDF + 8$ to the address mda of vlt_x.

AMD: According to mid, search for the PT to find the relevant vpt_x. Loop deal with the mda times of the parameter frame. In each loop, the data are sent to $RD3$ with the relevant pid. Loop deal with the man times of the algorithm frame. In each loop, use aid to find apt_y and send the $afdata$ to the correlative address of $RD2$. Algorithm 5 describes the process.

PRD: Use aid to obtain its apt_y. Loop cope with the algorithm frame apn times. Send the parameters to $RD3$ with the correlative pid. Algorithm 6 shows the process.

Algorithm 4: $VCAD$

Input: PDF
$mid \leftarrow$ 4 bytes from PDF;
searchPT (mid, vpt_x);
$pfl \leftarrow$ 4 bytes from $PDF + 4$;
$RD1[vpt_x.mda] \leftarrow pfl - 8$ bytes from $PDF + 8$;
$PDF \leftarrow PDF + pfl$;
value of $vpt_x.mdf \leftarrow 1$;

E. Process of the Algorithms

The execution sequence of the six algorithms is PRF, AMF, $VCAF$, $VCAD$, AMD, and PRD. $VCAF$ will particularly call the PRF to frame the control relevant parameters. The direction of transfer PFs could be from VS to $ePLC$ or from $ePLC$ to VS. From VS to $ePLC$, PRF, AMF, and $VCAF$ are running in VS, and the remaining algorithms are running in $ePLC$. From $ePLC$ to VS, PRF, AMF, and $VCAF$ are running in $ePLC$, and the remaining algorithms are running in VS.

Algorithm 5: AMD

Input: mid
searchPT(mid, vpt_x);
$mda \leftarrow vpt_x.mda$;
for $i = 0; i < vpt_x.mdn; i++$ **do**
  $RD3[RD1[mda]] \leftarrow$ 4 bytes from $mda + 4$;
  $mda \leftarrow mda + 8$;
end
for $i = 0; i < vpt_x.man; i++$ **do**
  searchPT($vpt_x.mas[i]$, apt_y);
  $cfl \leftarrow$ 4 bytes from $mda + 4$;
  $RD2[apt_y.apa] \leftarrow cfl - 8$ bytes from $mda + 8$;
  $mda \leftarrow mda + cfl$;
  value of $apt_y.adf \leftarrow 1$;
end

Algorithm 6: PRD

Input: aid
searchPT(aid, apt_y);
$ada \leftarrow apt_y.ada$;
for $i = 0; i < apt_y.apn; i++$ **do**
  $RD3[RD2[ada]] \leftarrow$ 4 bytes from $ada + 4$;
  $ada \leftarrow ada + 8$;
end

V. System Operation Mechanism

A. Implementation of Flexible Program and Execution

FL contains the PLC interface and the visual interface shown in Fig. 5 (b). They interact with each other using the agreed communication protocol defined as the CID in the HPT of the PT.

In the $ePLC$, the visual interface includes the following parts:

V1: communicates with the VS according to the CID in the HPT, and has three transitions: **V1.1** denotes no communication and other operations; **V1.2** denotes receiving PFs from the VS; and **V1.3** denotes sending PFs to the VS. In $V1.3$, the PFs will be saved in the VD of the $ePLC$, and two pointers are found, PDF and PSP (Fig. 6). The PDF points to the address of deframing PF, while the PSP points to the address of saving PF. After saving the PF, the PSP will point to the next new address according to the vfl of the PF. In $V1.3$, the visual interface frames the TD into the PF with algorithms 1, 2 and 3. Here, the TD is in the RAM of the $ePLC$.

V2: Algorithm 4 will be called. The PDF will point to the next PF if a PF is deframed.

The following parts are in the VS:

V3: visual algorithms extract the useful data and stores them into the array of data required to be transferred (TD), in which the parameters could be indexed by the PID.

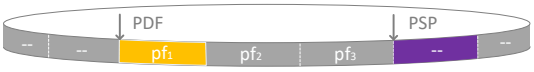

Fig. 6. Process of PDF and PSP.

V4: communicates with the $ePLC$ and contains three transitions: **V4.1** has no communication and other operations; **V4.2** frames the TD into the PFs with algorithms 1, 2, and 3 and sends the PFs to the $ePLC$ (here, the TD is an array in the VS); and **V4.3** receives the PFs from the $ePLC$ and deframes the PFs with algorithms 4, 5, and 6.

B. Implementation of the Control Program and Execution

The control program (CP) is responsible for organizing the modules, executing the logic program, deframing the PF, and handling data interaction with the algorithm program. The implementation of the control program is described below:

C1: traverses the MCF and the MDF and contains three transitions: **C1.1** means that every $mcf \in MCF$ and $mdf \in MDF$ is recovered to zero; **C1.2** means that both mcf_i and mdf_i are checked as one; and **C1.3** means that mcf_i is checked as one, while mdf_i is zero.

C2: executes Algorithm 5.

C3: executes the logic program and checks whether there are any data required to transfer.

C4: $\mathcal{P}_{mts}$ is used to send data and inform the slave processor to receive parameters and start algorithms.

C5: checks the end of this module. **C5.1** states that this module executes once. **C5.2** states that the logic program is still required to run.

C. Implementation and Execution of the Algorithm Program

The algorithm program (AP) is in EL. $AP = \{\mathcal{P}_{stm}, AS, ALDP\}$, where $\mathcal{P}_{stm}$ feeds back data to the master processor; AS contains all algorithms; and $ALDP$ deframes the AL frame. The execution of the algorithm program is introduced below:

A1: traverses AFS, ADF, and SMB. **A1.1** is not a program required to execute. **A1.2** denotes that PFs that needed to be deframed are checked. **A1.3** denotes that data need to be fed back to the master processor. **A1.4** denotes that an algorithm that needs to be executed is checked. **A1.5** denotes that the parameters of an algorithm that needed to be updated are checked.

A2: $ALDP$ is executed to call Algorithm 5.

A3: the $\mathcal{P}_{stm}$ is used to feed data back to the master processor.

A4: starts an algorithm.

A5: executes the algorithm until the end.

D. Petri-Net-based Execution of Threads

We adopt the analogous thread structure in [12]. Three special threads are illustrated: 1) the visual thread is responsible for the interaction with VS and drives the flexible layer; 2) the control thread is mainly responsible for organizing the whole program and driving the control layer; and 3) the algorithm thread is used to drive the algorithm layer.

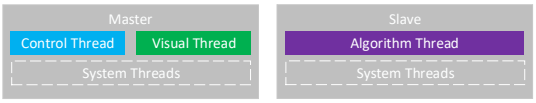

Fig. 7. Thread structure with three special threads: visual, control, and algorithm threads. The control and visual threads run in the master slave, while the algorithm thread runs in the slave thread.

In our works, Petri Net is adopted to describe the execution of the three threads. Every thread is a Petri-Net, and we define the Petri Net as a three tuple:

$$PN = \{P, T, F\}, \tag{3}$$

where P is the finite set of places; T is the finite set of transitions; and F is the set of arcs. $F \subset P \times T \cup T \times P$ denotes that the arcs are from P to T and from T to P.

We can then define the execution of threads (ET) as follows:

$$ET = \{PN_V, PN_C, PN_A\}, \tag{4}$$

where PN_V, PN_C, PN_A are the Petri Net of the visual, control, and algorithm threads, respectively. ET is described in Fig. 8.

PN_V is presented as follows:

$$\begin{cases} PN_V = \{P_V, T_V, F_V\}, \\ P_V = \{P_{V1}, P_{V2}, P_{V3}\}, \\ T_V = \{V_1, V_2\}, \end{cases} \tag{5}$$

where P_{V1} is idle; P_{V2} is $PDF! = PSP$; and P_{V3} denotes that a relevant mdf is set.

PN_C is denoted as follows:

$$\begin{cases} PN_C = \{P_C, T_C, F_C\}, \\ P_C = \{P_{C1}, P_{C2}, P_{C3}, P_{C4}, P_{C5}, P_{C6}\}, \\ T_C = \{C_1, C_2, C_3, C_4, C_5\}, \end{cases} \tag{6}$$

where P_{C1} is idle; P_{C2} denotes that a model is started, and mdf_i is one; P_{C3} denotes that a model is started, and mdf_i is zero; P_{C4} denotes that any msb_i is one; P_{C5} means the end of the data interaction, in which the correlative msf is zero; and P_{C6} denotes that a module finishes a one-time execution.

PN_V is illustrated as follows:

$$\begin{cases} PN_A = \{P_A, T_A, F_A\}, \\ P_A = \{P_{A1}, P_{A2}, P_{A3}, P_{A4}, P_{A5}, P_{A6}, P_{A7}, P_{A8}\}, \\ T_A = \{A_1, A_2, A_3, A_4, A_5\}, \end{cases} \tag{7}$$

where P_{A1} is idle; P_{A2} denotes that adf_i is one; P_{A3} denotes that any adf_i is zero; P_{A4} denotes that smb_i is one; P_{A5} denotes that any smf_i is zero; P_{A6} denotes that afe_i is one; P_{A7} presents that afs_i is one; and P_{A8} denotes that any afe_i is zero.

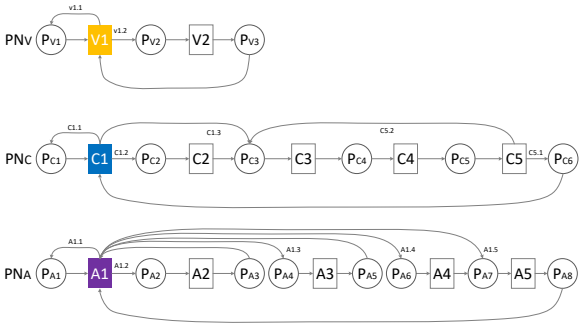

Fig. 8. Process of ET and its contained Petri Nets of the visual, control, and algorithm threads.

VI. Case Analysis

This section introduces two scenarios of the proposed VCA protocol-based integration method in $ePLC$. With the VCA protocol, the users only need to code additional algorithms and change the PT when the scenario of the visual servo system is changed.

A. Case one: Winding Machine with a Visual System

As shown in Fig. 9, (a) is the visual system, whose processor is the ARM9-based S3C2440 and uses the CAN interface to communicate with the OV9650 CMOS camera; (b) is the $ePLC$, CASS-PLCA149B; (c) is the winding machine with a visual system that contains two axes: the U-axis and the Q-axis; (d) shows the winding effect; and (e) shows the angle of θ, Q-axis, and U-axis. In the winding process, the tension of the copper wire will irregularly change, especially for the thick ones. However, we used the same speed ratio of the U-axis and the Q-axis, which always leads to the irregularity of each coil layer. Hence, we can use the visual system to decrease or increase the speed of the Q-axis to control the wire angle.

In the structure of the winding machine system, the ePLC has two chips: a master chip and a slave chip. It uses RS232 to receive PFs, and could control six servo systems at most. The winding machine has two axes: the U-axis and the Q-axis. VS uses the CMOS camera to gain winding images and transfer the digital information to the ARM CUP. The ARM CUP will extract parameters from the digital information, frame PFs, and transfer PFs to $ePLC$ continuously.

1) Design of PT: In [12], we propose the customized winding machine language, which only contains 13 instructions. Here, we can use the second and third instructions to control the U-axis and the Q-axis, respectively. In the algorithm layer, both use the same motion control algorithm, which controls one axis with speed and position. The winding process mainly aims to control the speed ratio of the U-axis and the Q-axis; hence, we use the VS to influence the speed of the Q-axis only. Table I presents the PT of the winding machine, in which pid_1 is the speed of the Q-axis, and 1000 of var_1 is used to multiply by θ.

2) Process of PF: Table II shows the interacted PF between VS and $ePLC$. First, VS detects that θ is 2 degrees. It then sends the PF to inform the $ePLC$ to increase the Q-axis speed of 2000 pulses per millisecond (p/ms). The increment of the Q-axis speed is changed to 1000 p/ms when the θ decreases to 1 degree. The θ has recovered to 0; hence, the $ePLC$ is informed to recover the initial speed of the Q-axis.

TABLE I
PT of Winding Machine

cid_1: 0x01*	tda_1 : 0x00	-	-	-	-	-
mid_1: 0x01	man_1: 1	mdn_1: 0	mda_1: 0X200	msf_1: 0X400	mdf_1: 0X600	mas_1: 0x01
aid_1: 0x01	apn_1: 0X1	af_1: 0X14A	ada_1: 0x400	$aps1_1$: 0X0	-	-
pid_1: 0X0	vid_1: 0X1	var_1: 0x200	-	-	-	-

* This number is hexadecimal, and the item means that the value of cid_1 is 0x01.

TABLE II
PF of Winding Machine

mid_1	pfl_1	aid_1	cfl_1	pid_1	$pdata_1$
0x01	0x14	0x01	0xE	0x01	0x400
0x01	0x14	0x01	0xE	0x01	0x200
0x01	0x14	0x01	0xE	0x01	0x000

B. Results of Case One

The advantages from case one indicate that: 1) The software architecture is clearly divided into the flexible, control, and algorithm layers. The same control program and algorithms could be reused. The separated control and algorithm layer programs could be developed by different-level programmers when a new application has some customized requirements. 2) The parameters that interacted between the VS and the $ePLC$ could be associated using the PT. Meanwhile, the PFs contained the parameters that could be carried to every layer. Hence, when more parameters need to be interacted, only the PT should be changed without reprogramming. 3) The cus-

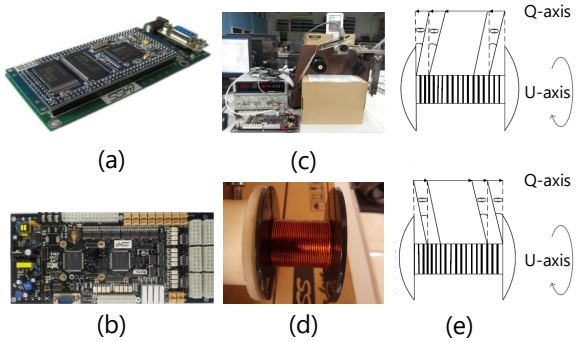

Fig. 9. (a) is the visual system, whose processor is the ARM9-based S3C2440 and uses the CAN interface to communicate with the OV9650 CMOS camera; (b) is the $ePLC$, CASS-PLCA149B; (c) is the winding machine with a visual system that contains two axes: the U-axis and the Q-axis; (d) shows the winding effect; and (e) shows the angle of θ, Q-axis, and U-axis.

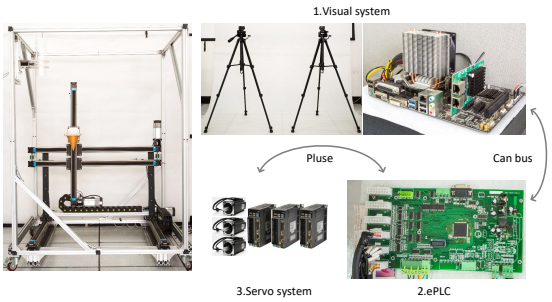

Fig. 10. The binocular catching robot consists of the visual system, $ePLC$, and servo system.

tomized structure of the $ePLC$, favorable memory allocation, and Petri-Net-based multi-threading structure make it possible for high-performance algorithms to run in a single processor. From the view of applications, the proposed architecture can also be easily applied to very different cases.

C. Case Two: Binocular Catching Robot

Fig. 10 shows that the binocular catching robot adopts two cameras to judge the ball position in the space. By continuously sending parameters to the $ePLC$, the robot runs to the position to catch the ball and finally catches it. The cameras and the lens used are GE31GC and FA0401C, respectively. The visual system is GIGABYTE's GA-B85M-D3V-A. The $ePLC$ uses a TI F28M35 chip with two cores, namely ARM Cortex M3 and TI C28x, which contain a shared RAM. The adopted servo system is ASDA-B2.

1) Design of PT: The $ePLC$ is designed to control six parallel axes. We adopt the same algorithm of the Q-axis and the U-axis in case one, while we name the three axes of the Cartesian robot as the X-axis, Y-axis, and Z-axis. In this case, we have the similar module with case one, but it contains three motion algorithms. pid_1 and pid_2 are the position and the speed of the X-axis, respectively. pid_3 and pid_4 are the position and the speed of the Y-axis, respectively. pid_5 and pid_6 are the position and the speed of the Z-axis, respectively. The VS uses meter and meter per second (m/s) to measure the distance and the speed, respectively. Meanwhile, in the $ePLC$, driving every 1 mm needs to output 100 pluses. Hence, the VAR of the position and the speed are both 100.

2) Process of PF: Table IV shows the PFs of the time spent catching the ball. Fig. 11 depicts the ball trajectory, which is the trajectory captured by the cameras. The red point denotes the position where the ball is thrown. The blue points are the positions where the catching points are predicted. The yellow points are the predictive catch points. The VS sends the PF to $ePLC$ to adjust the destination and the speed of every axis.

D. Results of Case Two

The analysis of the two cases showed that the proposed VCA-based flexible structure provides a generic method to address the visual servo control problem in $ePLC$. Based on the VCA protocol, the algorithms in the VS and the

TABLE III
PT OF BINOCULAR CATCHING ROBOT

cid_1: 0x01	tda_1: 0x00	-	-	-	-	-	-	-
mid_1: 0x01	man_1: 0x1	mdn_1: 0	mda_1: 0X200	msf_1: 0X400	mdf_1: 0X600	mas_1: 0x01	mas_1: 0x02	mas_1: 0x03
aid_1: 0x01	apn_1: 0X2	af_1: 0X14A	ada_1: 0x400	$aps1_1$: 0X0	$aps2_1$: 0x4	-	-	-
aid_2: 0x02	apn_2: 0X2	af_2: 0X14B	ada_2: 0x410	$aps1_2$: 0X8	$aps2_2$: 0xA	-	-	-
aid_3: 0x03	apn_3: 0X2	af_3: 0X14C	ada_3: 0x420	$aps1_3$: 0XE	$aps2_3$: 0X10	-	-	-
pid_1: 0X0	vid_1: 0X1	var_1: 0x40	pid_2: 0X4	vid_2: 0X1	var_2: 0x40	pid_3: 0X8	vid_3: 0X1	var_3: 0x40
pid_4	vid_4	var_4	pid_5	vid_5	var_5	pid_6	vid_6	var_6
0XA	0X1	0x40	0XE	0X1	0x40	0X10	0X1	0x40

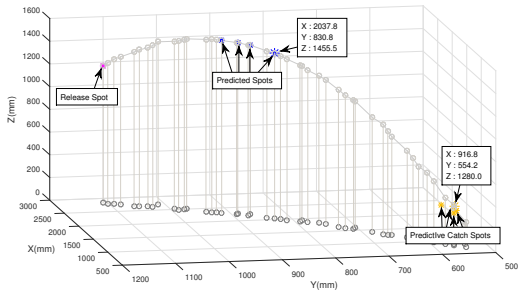

Fig. 11. Trajectory captured by the cameras. The red point denotes the position where the ball is thrown. The blue points are the positions where the catching points are predicted. The yellow points are the predictive catch points.

$ePLC$ have a kind of normative correspondence, and the existing modules and motion control algorithms could be reused without additional programming.

VII. Conclusion

We proposed herein a flexible multi-level architecture based on the VCA protocol to integrate the PLC, motion control, and visual systems. This architecture provides users an easy development method to ease the ever-growing complexity of applications. The multi-level architecture includes the flexible, control, and algorithm layers. The VCA protocol is posed for data interaction among the layers. Correspondingly, customized hardware, memory allocation, and Petri-Net-based multithreading structure were described to support the proposed flexible software architecture. We implemented two cases, namely the winding machine with a visual system and the binocular catching robot, to illustrate the proposed system that provides a generic method to develop different types of applications of the visual servo system in $ePLC$. As a result, the proposed VCA-based architecture could easily be applied between two quite different cases.

In a further research, we will implement a uniform development method of the visual servo control in the ePLC.

TABLE IV
PF OF BINOCULAR CATCHING ROBOT

mid_1	pfl_1	aid_1	cfl_1	pid_1	$pdata_1$	pid_2	$pdata_2$	aid_2	cfl_2	pid_3	$pdata_3$	pid_4	$pdata_4$	aid_3	cfl_3	pid_5	$pdata_5$	pid_6	$pdata_6$
0x01	0x14	0x01	0xE	0x00	0x1C6	0x04	0x10A5E	0x02	0xE	0x08	0x17b	0x0A	0xDE27	0x03	0xE	0x0E	0x140	0x10	0xBB80
0x01	0x14	0x01	0xE	0x00	0x1F6	0x04	0x126AA	0x03	0xE	0x08	0x175	0x0A	0xDA94	0x04	0xE	0x0E	0x140	0x10	0xBB80
0x01	0x14	0x01	0xE	0x00	0x28F	0x04	0x17FDB	0x02	0xE	0x08	0x17A	0x0A	0xE0D8	0x03	0xE	0x0E	0x140	0x10	0xBB80
0x01	0x14	0x01	0xE	0x00	0x263	0x04	0x16624	0x02	0xE	0x08	0x171	0x0A	0xD87D	0x03	0xE	0x0E	0x140	0x10	0xBB80

REFERENCES

[1] M. P. Kazmierkowski, "Integration technologies for industrial automated systems (zurawski, r., ed.; 2006) [book reviews]," *IEEE Industrial Electronics Magazine*, vol. 1, no. 1, pp. 51–52, 2007.

[2] J. Wan, B. Yin, D. Li, A. Celesti, F. Tao, and Q. Hua, "An ontology-based resource reconfiguration method for manufacturing cyber-physical systems," *IEEE/ASME Transactions on Mechatronics*, vol. PP, no. 99, pp. 1–1, 2018.

[3] D. A. Chekired, L. Khoukhi, and H. T. Mouftah, "Industrial iot data scheduling based on hierarchical fog computing: A key for enabling smart factory," *IEEE Transactions on Industrial Informatics*, vol. PP, no. 99, pp. 1–1, 2018.

[4] A. W. Colombo, R. Schoop, and R. Neubert, "An agent-based intelligent control platform for industrial holonic manufacturing systems," *IEEE Transactions on Industrial Electronics*, vol. 53, no. 1, pp. 322–337, 2006.

[5] A. Vaccaro, M. Popov, D. Villacci, and V. Terzija, "An integrated framework for smart microgrids modeling, monitoring, control, communication, and verification," *Proceedings of the IEEE*, vol. 99, no. 1, pp. 119–132, 2010.

[6] E. Dean, K. R. Amaro, F. Bergner, I. Dianov, and G. Cheng, "Integration of robotic technologies for rapidly deployable robots," *IEEE Transactions on Industrial Informatics*, vol. PP, no. 99, pp. 1–1, 2017.

[7] T. N. Chang, B. Cheng, and P. Sriwilaijaroen, "Motion control firmware for high-speed robotic systems," *IEEE Transactions on Industrial Electronics*, vol. 53, no. 5, pp. 1713–1722, 2006.

[8] X. Feng, R. Mathurin, and S. A. Velinsky, "Practical, interactive, and object-oriented machine vision for highway crack sealing," *Journal of Transportation Engineering*, vol. 131, no. 6, pp. 451–459, 2005.

[9] S. Hossain, M. A. Hussain, and R. B. Omar, "Advanced control software framework for process control applications," *International Journal of Computational Intelligence Systems*, vol. 7, no. 1, pp. 37–49, 2014.

[10] Y. Jiang, H. Zhang, H. Liu, X. Song, W. N. Hung, M. Gu, and J. Sun, "System reliability calculation based on the run-time analysis of ladder program," in *Proceedings of the 2013 9th joint meeting on foundations of software engineering*. ACM, 2013, pp. 695–698.

[11] S. Dominic, Y. Lohr, A. Schwung, and S. X. Ding, "Plc-based real-time realization of flatness-based feedforward control for industrial compression systems," *IEEE Transactions on Industrial Electronics*, vol. PP, no. 99, pp. 1–1, 2016.

[12] H. Wu, Y. Yan, D. Sun, and S. Rene, "A customized real-time compilation for motion control in embedded plcs," *IEEE Transactions on Industrial Informatics*, 2018.

[13] D. Schütz, A. Wannagat, C. Legat, and B. Vogel-Heuser, "Development of plc-based software for increasing the dependability of production automation systems," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 4, pp. 2397–2406, 2013.

[14] H. Chen, K. Liu, G. Xing, Y. Dong, H. Sun, and W. Lin, "A robust visual servo control system for narrow seam double head welding robot," *International Journal of Advanced Manufacturing Technology*, vol. 71, no. 9-12, pp. 1849–1860, 2014.

[15] W. Xing, P. Lou, J. Yu, X. Qian, and D. Tang, "Intersection recognition and guide-path selection for a vision-based agv in a bidirectional flow network," *International Journal of Advanced Robotic Systems*, vol. 11, no. 1, p. 1, 2014.

[16] C. Y. Nian and Y. S. Tarng, "An auto-alignment vision system with three-axis motion control mechanism," *International Journal of Advanced Manufacturing Technology*, vol. 26, no. 9-10, pp. 1121–1131, 2005.

[17] S. Xiao and Y. Li, "Visual servo feedback control of a novel large working range micro manipulation system for microassembly," *Journal of Microelectromechanical Systems*, vol. 23, no. 1, pp. 181–190, 2014.

[18] H. Wu, L. Lou, C. C. Chen, S. Hirche, and K. Kuhnlenz, "Cloud-based networked visual servo control," *IEEE Transactions on Industrial Electronics*, vol. 60, no. 2, pp. 554–566, 2013.

[19] C. Y. Tsai, C. C. Wong, C. J. Yu, C. C. Liu, and T. Y. Liu, "A hybrid switched reactive-based visual servo control of 5-dof robot manipulators for pick-and-place tasks," *IEEE Systems Journal*, vol. 9, no. 1, pp. 119–130, 2017.

[20] P. Serra, R. Cunha, T. Hamel, D. Cabecinhas, and C. Silvestre, "Landing of a quadrotor on a moving target using dynamic image-based visual servo control," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1524–1535, 2016.

[21] L. Y. Sun, G. M. Yuan, J. Zhang, Z. Liu, and L. X. Sun, "Automatic button cell sorting manipulator system research based on visual servo control," *Advanced Materials Research*, vol. 712-715, pp. 2285–2289, 2013.

[22] M. G. Ioannides, "Design and implementation of plc-based monitoring control system for induction motor," *IEEE Transactions on Energy Conversion*, vol. 19, no. 3, pp. 469–476, 2004.

[23] X. M. Shi, W. J. Fei, and S. P. Deng, "The research of circular interpolation motion control based on rectangular coordinate robot," *Key Engineering Materials*, vol. 693, pp. 1792–1798, 2016.

[24] N. Fang, "Design and research of multi axis motion control system based on plc," *Academic Journal of Manufacturing Engineering*, vol. 15, no. 1, pp. 17–23, 2017.

[25] A. Syaichu-Rohman and R. Sirius, "Model predictive control implementation on a programmable logic controller for dc motor speed control," in *Electrical Engineering and Informatics (ICEEI), 2011 International Conference on*. IEEE, 2011, pp. 1–4.

[26] J. Qian, H. B. Zhu, S. W. Wang, and Y. S. Zeng, "A 5-dof combined robot platform for automatic 3d measurement," *Key Engineering Materials*, vol. 579-580, pp. 641–644, 2014.

[27] O. Co.Ltd, "Cs1w-mc221(-v1)/mc421(-v1) motion control units." *Operation Mannual*, 2004.

[28] P. T. C. 2., *Function blocks for motion control version 1.1*, 2005.

[29] C. Sünder, A. Zoitl, F. Mehofer, and B. Favre-Bulle, "Advanced use of plcopen motion control library for autonomous servo drives in iec 61499 based automation and control systems," *E & I Elektrotechnik Und Informationstechnik*, vol. 123, no. 5, pp. 191–196, 2006.

[30] S. S. S. GmbH, "Logic and motion control integrated in one iec 61131-3 system:development kit for convenient engineering of motion, cnc and robot applications." 2017.