

A 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. Meanwhile, their respectively numerous types of the programs and hardwares, lots of communication protocols among them, mixed development methods, and ever-growing high requirements and complexity lead to a cumbersome task of the implementation of the applications for users. However, there are few works researching the integration of these three systems to ease the complexity. Most of them are focusing on their applications individually.

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

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

I. INTRODUCTION

Integration technologies are driving the industrial automation [1]. The continuously integrations of sensors, controllers, robots, tools, etc., bring the concepts of self-aware equipment, intelligent factory, cyber-physical system, etc. [2], [3]. Recent researches [4]–[6] introduce the development of integration technologies. In industrial automation, PLC system, motion control system, and visual system become increasingly important and inseparable which could be seen in various domains [7]–[9].

A. Motivations

Recently, advances in image processing and pattern recognition contribute to the thriving of visual system which has been applied in various fields. By extracting features from the image, the visual system could obtain parameters to replace human visual system to address lots of tasks; specially, the ever-growing requirements in industrial scenarios; e.g., works that should be finished in dangerous environment, tasks that

human vision is difficult to satisfy, or applications in some large scale industrial production that numerous visual systems are needed.

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

Furthermore, PLCs have become a base of automation owing to its high reliability and easy programming [10]. Lots of researches are focusing on it to extremely extend its applied fields; for instance, [11], [12] guarantee the reliability by verifying the program of PLCs, [13], [14] improve the performance of PLCs using advanced algorithms, [15] alleviates the development complexity of PLCs with a special software structure, [16] poses methods to update PLC programs dynamically.

In addition, the visual system, motion control system, and PLC system are becoming increasingly inseparable. [17] is a typical case that describes how the three parts collaborate. The visual system analyzes the context and get error put into the motion control system. Simultaneously, the PLC is analyzing the 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 system, motion control system, and PLC system are developed individually and using the communication protocol combines them. However, this method should be always redesigned the programs in these systems because of the difference of visual algorithms, motion control algorithms, and their organized logic programs. There are also lots of communication protocols (e.g., EtherCat, Modbus, CAN, etc.) using among these systems.
- 2) The visual, logic, and motion control programs are always mixed together developed within numerous visual systems, PLCs, and motion controllers. This increases the unnecessary complexity.
- 3) The requirements of customers are ever growing. Combined the before-mentioned two points, the development of the corresponding applications has become a cumbersome task for developers.

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

B. Our Contributions

Our contributions are introduced as follows:

- 1) We propose a *VCA* protocol for generic data interaction between *ePLC* and *VS*. This benefits the reusing of algorithms and the uniform using of communication protocols.
- 2) A multi-level flexible architecture is posed for individual programming which includes flexible layer, control layer, and algorithm layer. Correspondingly, 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 the further integration of visual system, motion control system, and *PLC* system. Through two implemented cases, it indicates that the proposed architecture could reduce the complexity of one application and from one to another.

In the remaining paper, in Section II, we introduce the related works. In Section III, the hardware and software structure, and memory allocation are introduced. In Section IV, the mechanism of *VCA* protocol is addressed in detail. In Section V, we illustrate the execution process of the proposed system. Then, we implement two cases which are binocular catching robot and winding machine with visual system in Section VI. In the last section, we conclude our works.

II. RELATED WORKS

Visual control system is combined of special motion control system and visual system which has been applied in various fields, such as transport [18], circuit detection [19], sorting system, welding [17], assembling [20], [21], robot [22], [23], unmanned aerial vehicles [24], [25], and sorting [26]. These works address their problems in relevant fields. However, all these solutions are based on special motion control system and visual system.

Additionally, the integration of logic control and motion control has variously deep researches [27]–[30]. [27], [30] realize the motion control directly in PLC. [31]–[33] use motion control module collaborated with PLC to implement their applications. However, owing to the confusion of the development method in these works, PLCopen organization released a related standard [34] which standardizes the motion control in PLC. Based on this standardization, [35] provides an advanced implementation in distributed automation system, and companies, such as 3S [36], provide some tools for users. [15] also poses a customized real-time compilation method to reduce the development complexity.

Above works provide impressive integrations on visual servo control system and PLC with motion control functions, however there are few works researching the integration of visual system, motion control, and PLC to decrease the complexity. Most of applications are focusing on their applications with three individual systems, such as [17]. Hence, an further integration structure of PLC functioned with motion control and visual servo system should be provided to reduce the complexity and expand the application fields.

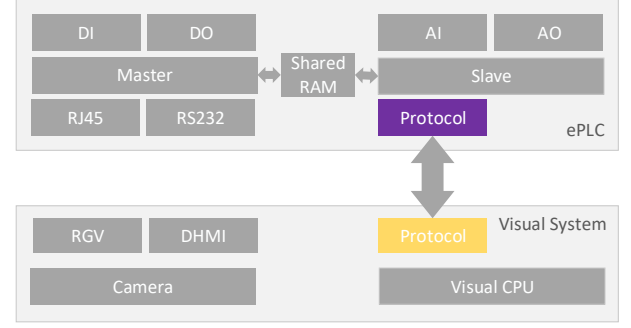


Fig. 1. Hardware architecture of typical *ePLC* and *VS*. The *ePLC* adopts two processor architecture and the *VS* contains one camera and a processor. The communication between *ePLC* and *VS* adopt the CAN Bus.

III. SYSTEM ARCHITECTURE

A. Hardware Structure

The hardware is comprised of *ePLC* and visual system (*VS*) shown in Fig.1. 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 processors of *ePLC* could also be customized in cater to high performance-motion-control applications, and in motion-control-function contained applications more than two processors are always adopted that one processor is mainly used to logic control and the others are used for motion control. Fig. 1 shows a typical *ePLC* which adopts two processor architecture: a master and a slave processor. The shared memory is used to interact data between master and slave processor. The *VS* normally contains one or more cameras, and a processor for visual tasks. The communication between *ePLC* and *VS* could adopt multiple protocols, such as TCP, modbus, and CAN.

B. Software Structure

The software of the proposed multi-level flexible structure is shown in Fig. 2. To our best knowledge, the *VS* commonly contains several visual algorithms and every algorithm could extract some required parameters form pictures or videos. Meanwhile, the *ePLC* is comprised of modules which include logic part and algorithms. The algorithms here mainly donate 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 joint the *VS* and *ePLC*, which consists of *PLC* interface and visual interface. The parameters are framed with *VCA* protocol frame (henceforth, *VCA* protocol frame is abbreviated form of *PF*) interacted between *VS* and *ePLC*. Furthermore, a special protocol template (*PT*) is saved in both systems to illustrate the protocol.

Control layer: this layer is separated form algorithm layer to cope with the logic tasks. This makes it possible to run the logic task and algorithm task in different processors.

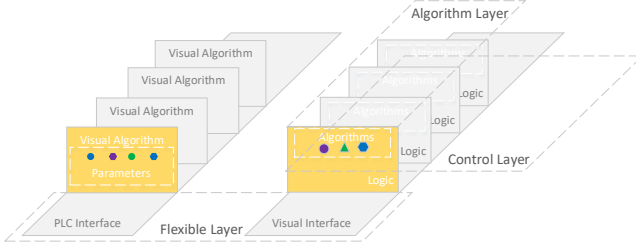


Fig. 2. Multi-level flexible architecture contains three layers: flexible layer, control layer and algorithm layer. The *VS* contains several visual algorithms which are used to extract parameters from picture or videos. The flexible layer consists of PLC interface and visual interface used to combine *ePLC* and *VS*. Control layer is mainly designed for logic program. Algorithm layer is comprised of many algorithms. Every algorithm has several parameters.

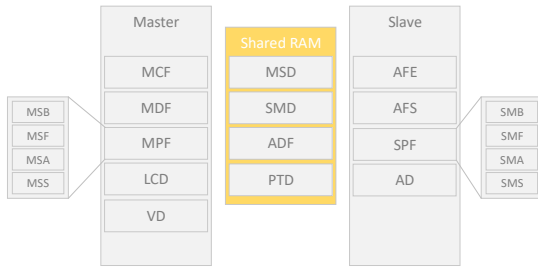


Fig. 3. Memory allocation of master processor, shared ram and slave processor.

Algorithm layer: it is mainly comprised of various types of algorithms. The independent algorithm layer makes the algorithms with high performance requirement that could be built in individual processors.

C. Memory Allocation

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

In master processor, its RAM contains the special areas: 1) Module control flag area (*MCF*) are used to start the modules. 2) Master processor data interaction flag area (*MPF*) contains begin data transfer flag from master to slave (*MSB*), transfer state of master from master to slave (*MSF*), acknowledge flag of master from master to slave (*MSA*), and transfer state of master from slave to master (*MSS*). 3) Module data saved flag area (*MDF*) denotes whether the module data are saved. 4) Logic control data area (*LCD*) is used to store the

control frame. Every element lcd_i is associated with a specified module. 5) *VCA* protocol frame data area (*VD*) stores all the frames received from the *VS*.

In shared RAM, it could be read by both of master and slave processors and consists of the special areas: 1) Master processor data interaction data area (*MSD*) stores the data delivered from slave processors. 2) Slave processor data interaction data area (*SMD*) stores the data delivered from master processor. 3) Algorithm data saved flag area (*ADF*) denotes whether the algorithm data are saved. 4) Protocol template data area (*PTD*) stores the protocol template which contains *PTs* from *VS* to *ePLC* and from *ePLC* to *VS*.

In slave processor, the RAM is comprised of the special areas: 1) Algorithm flag area (*AF*) includes algorithm flag of execution (*AFE*) and algorithm flag of state (*AFS*). 2) Slave processor data interaction flag area (*SPF*) includes the begin data transfer flag from slave to master (*SMB*), transfer state of slave from slave to master (*SMF*), acknowledge flag of slave from slave to master (*SMA*), and transfer state of slave from master to slave (*SMS*). 3) Algorithm data area (*AD*) saves data which help specified algorithm executing.

We define the \mathcal{I} to interact data between master processor and slave processor. It contains two parts: transferring data from master to slave (\mathcal{I}_{mts}) and transferring data from slave to master (\mathcal{I}_{stm}), which is defined below:

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

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

IV. INTEGRATION OF *VS* AND *ePLC*

A. *VCA PT*

In cater to most applications, we design a *VCA* protocol to interact data among the layers. As shown in Fig. 4, a protocol template (*PT*) is adopted to support various types of implementations and a *PT* uniquely corresponds to a type of application. In the flexible architecture, users only need to redesign and reload the *PT* and then they can reuse the visual servo system again. The *PT* could be loaded into a stationary address of visual system and *ePLC* system. After restarting both systems, it will be stored into a fixed area of *RAM*. The parsing modules from both systems will read it when parsing the *PF*. The *VCA* protocol could be used to bidirectionally transfer *PF* using the same *PT* and algorithms of framing and deframing.

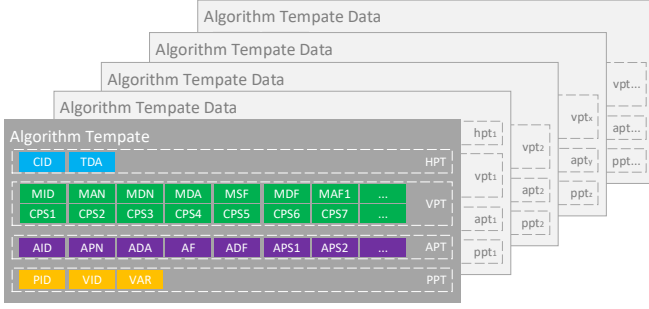


Fig. 4. A *PT* uniquely corresponds to a type of application. It contains four parts: head of protocol template, visual layer template, control layer template and algorithm layer template.

The *PT* is defined below:

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

The *PT* contains four parts: head of protocol template (*HPT*), VCA protocol template (*VPT*), algorithm protocol template (*APT*) and parameter protocol template (*PPT*). Every part explains as follows:

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

B. VCA PF

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

- 1) *PF*: it consists items of *MID*, visual frame length (*VFL*), data from parameter frames (*PFDATA*)

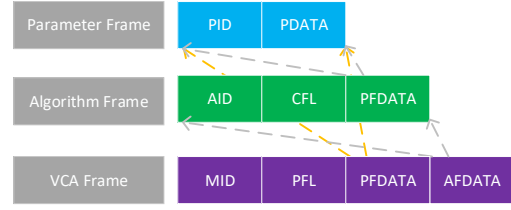


Fig. 5. *VCA* protocol frame contains parameter frames and algorithm frames in its data field and the algorithm protocol frame contains several parameter frames.

and data form algorithm frames (*AFDATA*). The *PFDATA* and *AFDATA* contains several parameter frames and control frames, respectively. The *MIDs* both in *PT* and *PF* need one-to-one correspondence.

- 2) Algorithm frame: it is comprised of *AID*, control frame length (*CFL*) and *PFDATA*. The *AIDs* both in algorithm frame and *PF* need one-to-one correspondence.
- 3) Parameter frame: it contains *PID*, *PDATA*. *PID* is also the address of *PDATA*. The *PIDs* both in parameter frame and *PF* need one-to-one correspondence.

C. Framing of PF

The framing contains three steps: *VCA* framing (*VCAF*), algorithm framing (*AMF*), and parameter framing (*PRF*). Transferring data area (*TD*) saves the data to be transferred and it 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 the Algorithm 1. It searches the *PT* with *mid* to find the relevant *vlt_x*. Through it, we can obtain the *man* and *mdn*. According to *man* and *mdn*, the Algorithm 3 and Algorithm 2 are called to gain the *pfdata* and *afdata*. Then, it is finished after calculating the length (*vfl*).

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

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

D. Deframing of PF

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

VCAD: As illustrated in Algorithm 4, obtain the *mid* and *pfl* from the start four and following four bytes data of *PDF*, respectively. Search the *PT* to gain the relevant *vpt_x*. Send

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
 $PF.pfdata \leftarrow pfdata$;
 $PF.afdata \leftarrow afdata$;
 $PF.vfl \leftarrow (vpt_x.mdn + vpt_x.man) \times 4 + 8$;

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
 $af.cfl \leftarrow apt_y.apn \times 4 + 8$;
 $af.pfdata \leftarrow pfdata$;
 $afdata \leftarrow afdata + clf$;

the $pfl - 8$ bytes start from the eighth byte of $PDF + 8$ to the address mda of vlt_x .

AMD: According to mid , search 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 the aid to find apt_y and then send the $afdata$ to correlative address of $RD2$. Algorithm 5 is described the process.

PRD: Use aid to get its apt_y . Loop cope with the algorithm frame apn times. Send the parameters to $RD3$ with the correlative pid . The process is shown in Algorithm 6.

E. Process of the Algorithms

Figure 7 is shown the process of the six algorithms and the execution sequence is $PRF, AMF, VCAF, VCAD, AMD$,

Algorithm 3: PRF

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

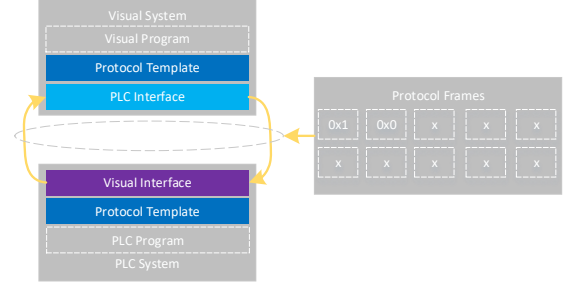


Fig. 6. PF interaction between PLC interface and visual interface using the agreed communication protocol which is defined as CID in HPT of PT . Here, $0x1$ is the CID .

Algorithm 4: VCAD

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

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 form $mda + 4$;
 $mda \leftarrow mda + 8$;
end
for $i = 0; i < vpt_x.man; i++$ **do**
 searchPT($vpt_x.mas[i], apt_y$);
 $cfl \leftarrow 4$ bytes form $mda + 4$;
 $RD2[apt_y.apa] \leftarrow cfl - 8$ bytes form $mda + 8$;
 $mda \leftarrow mda + cfl$;
 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 form $ada + 4$;
 $ada \leftarrow ada + 8$;
end

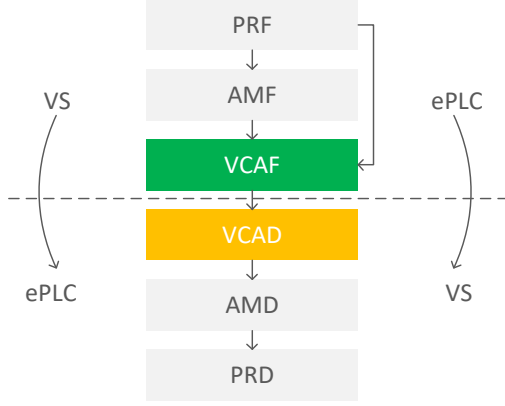


Fig. 7. Process of the six algorithms and the execution sequence is *PRF*, *AMF*, *VCAF*, *VCAD*, *AMD*, and *PRD*; and the *VCAF* will particularly call the *PRF* to frame the control relevant parameters. The direction of transfer *PF*s could be form *VS* to *ePLC* or form *ePLC* to *VS*.

and *PRD*; and the *VCAF* will particularly call the *PRF* to frame the control relevant parameters. The direction of transfer *PF*s could be form *VS* to *ePLC* or form *ePLC* to *VS*. Form *VS* to *ePLC*, *PRF*, *AMF*, and *VCAF* are running in *VS*; and the remaining algorithms are running in *ePLC*. Form *ePLC* to *VS*, *PRF*, *AMF*, and *VCAF* are running in *ePLC*; and the remaining algorithms are running in *VS*.

V. SYSTEM OPERATION MECHANISM

A. Implementation of Flexible Program and Execution

The *FL* contains PLC interface and visual interface, shown in Fig. 6. They interact with each other using the agreed communication protocol which is defined as *CID* in *HPT* of *PT*.

In the *ePLC*, the visual interface includes the following parts.

V1: communicates with *VS*, according to the *CID* in *HPT*. It has three transitions. **V1.1** donates no communication and other operations; **V1.2** donates receiving *PF*s from *VS*, and **V1.3** donates sending *PF*s to *VS*. In *V1.3*, the *PF*s will be saved in the *VD* of *ePLC* and there are two pointers: *PDF* and *PSP*, shown in Fig. 8. *PDF* points to the address of deframing *PF* and *PSP* points to the address of saving *PF*. After saved the *PF*, the *PSP* will point to the next new address according the *vfl* of *PF*. In *V1.3*, visual interface frames *TD* into *PF* with the Algorithm 1, 2 and 3. Here, the *TD* is in *RAM* of *ePLC*.

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

In the *VS*, there are following parts.

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

V4: communicates with *ePLC*. It also contains three transitions. **V4.1** is no communication and other operations; **V4.2** frames *TD* into *PF*s with the Algorithm 1, 2, and 3 and sends the *PF*s to *ePLC*. Here, the *TD* is an array in *VS*;

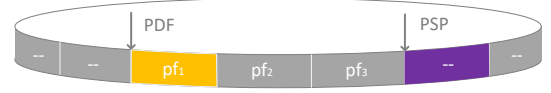


Fig. 8. *PDF* and *PSP*. *PDF* points to the address of deframing *PF* and *PSP* points to the address of saving *PF*. The *PDF* will point to the next address of *PF* if a *PF* is deframed. After saved the *PF*, the *PSP* will point to the next new address according the *vfl* of *PF*.

V4.3 receives the *PF*s form *ePLC* and deframes the *PF*s with the Algorithm 4, 5, and 6.

B. Implementation of Control Program and Execution

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

C1: traverse the *MCF* and *MDF*. Its contains three transitions. **C1.1** donates 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 one. **C1.3** means that mcf_i are checked one while mdf_i is zero.

C2: executes Algorithm 5.

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

C4: the \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** donates that this module executes one time. **C5.2** donates that the logic program still requires to run.

C. Implementation and Execution of Algorithm Program

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

A1: traverse the *AFS*, *ADF*, and *SMB*. **A1.1** is no program required to execute. **A1.2** donates that *PF*s needed to be deframed is checked. **A1.3** donates that data needed to feed back to master processor. **A1.4** donates that a algorithm needed to be executed is checked. **A1.5** donates that parameters of a algorithm needed to be updated is checked.

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

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

A4: starts a algorithm.

A5: executes the algorithm until the end.

D. Petri-Net Based Execution of Threads

We adopt the analogous thread structure in [15]. Three special threads are illustrated as follows: 1) visual thread is responsible for interaction with *VS* and drives the flexible layer; 2) control thread is mainly responsible for organizing the whole program and drives the control layer; 3) algorithm thread is used to drive the algorithm layer.



Fig. 9. Thread structure with three special threads: visual thread, Control thread, and algorithm thread. Control thread and visual thread are running in the master slave, and the algorithm thread is running 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. We define the Petri Net as three tuple:

$$PN = \{P, T, F\}, \quad (3)$$

where P is the finite set of places, T is the finite set of transitions, and F is the set of arcs. Here, $F \subset P \times T \cup T \times P$ which donates that the arcs contain form P to T and from T to P .

Then, we can define the execution of threads (ET) as follows.

$$ET = \{PN_V, PN_C, PN_A\}, \quad (4)$$

where PN_V, PN_C, PN_A are the Petri Net of visual thread, control thread, and algorithm thread, respectively.

The PN_V is donated as follows:

$$\begin{cases} PN_V = \{P_V, T_V, F_V\}, \\ P_V = \{P_{V1}, P_{V2}, P_{V3}\}, \\ T_V = \{V1, V2\}, \end{cases} \quad (5)$$

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

The PN_C is donated as follows:

$$\begin{cases} PN_C = \{P_C, T_C, F_C\}, \\ P_C = \{P_{C1}, P_{C2}, P_{C3}, P_{C4}, P_{C5}, P_{C6}\}, \\ T_C = \{C1, C2, C3, C4, C5\}, \end{cases} \quad (6)$$

where P_{C1} is idle; P_{C2} donates that a model is started and the mdf_i is one; P_{C3} donates that a model is started and mdf_i is zero; P_{C4} donates 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} donates that a module finishes one time execution.

The PN_V is illustrated as follows:

$$\begin{cases} PN_A = \{P_A, T_A, F_A\}, \\ P_A = \{P_{A1}, P_{A2}, P_{A3}, P_{A4}, P_{A5}, P_{A6}, P_{A7}, P_{A8}\}, \\ T_A = \{A1, A2, A3, A4, A5\}, \end{cases} \quad (7)$$

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

The ET is described in Fig. 10. In the Petri Net of visual thread, if $V1.2$ executes and receives a PF from VS , it transits from the initial position of P_{V1} to P_{V2} . After finished deframing of a PF , the state transits to P_{V3} and then visual thread back to execute $V1$. If there is any FS which should be sent to VS , it will execute $V1.2$. Apart from that, it will execute $V1.1$ and back to the position of P_{V1} .

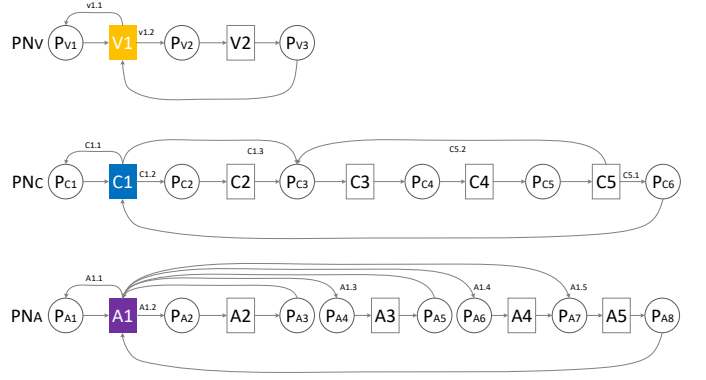


Fig. 10. Process of ET and its contained Petri Nets of visual thread, control thread, and algorithm thread.

In the Petri Net of control thread, the control thread executes the $C1$. The difference between $P_{C1.2}$ and $P_{C1.3}$ is that after $P_{1.2}$, the control thread transits to P_{C2} and executes $C2$. Except that, it will back to P_{C1} . Then, the control thread runs into a module and $C3$ is run. If find any data needed to transfer to slave processor, it transits to P_{C4} and executes $C4$. When finish transfer, it transits to P_{C5} . After execution of $C5$, according to $C5.1$ and $C5.2$, control thread transits to P_{C3} and P_{C6} . The previous one is executed owing to that the logic program of this module is not finished.

In the Petri Net of algorithm thread, the $A1$ is executed; if any AF needed to deframe, algorithm thread transits to P_{A2} and after the execution of $A2$, it transits to P_{A2} ; if any data required to feed back to master processor, algorithm thread transits to P_{A4} , then transfers the data, and transits to P_{A5} ; if any algorithm should be started, algorithm thread transits to P_{A6} , executes the $A4$, transits to P_{A7} , executes $A5$ to update the parameters, and then transits to P_{A8} ; if there are algorithms running and needed to updates parameters, algorithm thread transits to P_{A7} directly and after the execution of $A5$ transits to P_{A8} ; and above-mentioned transitions of $V1$ are executed in sequence.

VI. CASE ANALYSIS

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

A. Case one: Winding Machine with Visual System

As shown in Fig. 11, (a) is the visual system, whose CUP is ARM9 based S3C2440 and which uses the CANIF interface to communicate with OV9650 CMOS Camera; (b) is the $ePLC$, CASS-PLCA149B; (c) is the winding machine with visual system which contains two axes: the U-axis and Q-axis; (d) is shown the Winding effect, the angle of the wire, Q-axis and U-axis. In winding process, the tension of the copper wire will change irregularly, especially for the thick ones. However, we use the same speed ratio of U-axis and Q-axis which always

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	aps_1 : 0X0	-	-
pid_1 : 0X0	vid_1 : 0X1	var_1 : 0x200	-	-	-	-

* this number is hexadecimal, and the item means that the 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

leads to irregularity of each layer of the coil. Hence, we can use the visual system to decrease or increase the speed of Q-axis to control the angle of wire.

Figure 12 is the structure of the winding machine system. The ePLC has two chips: a master chip and a slave chip. It uses the RS232 to receive *PFs* and could control six servo systems at most. The winding machine has two axes: the U-axis and Q-axis. The *VS* use the CMOS camera to gain winding images and transfer the digital information to 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 [15], we propose the customized winding machine language which only contains 13 instructions. Here, we can use the second and third instructions to control U-axis and Q-axis, respectively. In the algorithm layer, both of them use the same motion control algorithm which controls one axis with speed and position. The process of winding is mainly to control the speed ratio of U-axis and Q-axis, hence we use the *VS* to influence the speed of Q-axis only. The *PT* of winding machine is illustrated in Table I in which the pid_1 is the speed of Q-axis and the 1000 of var_1 is used to multiply by the θ .

2) *Process of PF*: Table II is shown the interacted *PF* between the *VS* and ePLC. At first, the *VS* detects that the θ is 2 degrees. Then, it sends the *PF* to inform the ePLC to increase the Q-axis speed of 2000 pulses per millisecond (p/ms). When the θ decreases to 1 degree, the increment of the Q-axis speed is changed to 1000 p/ms. Since the θ has recovered to 0, the ePLC is informed to recover the initial speed of the Q-axis.

B. Results of Case One

From case one, advantages indicate that:

- 1) the software architecture is divided clearly into flexible layer, control layer, and algorithm layer. The same control program and algorithms could be reused. The

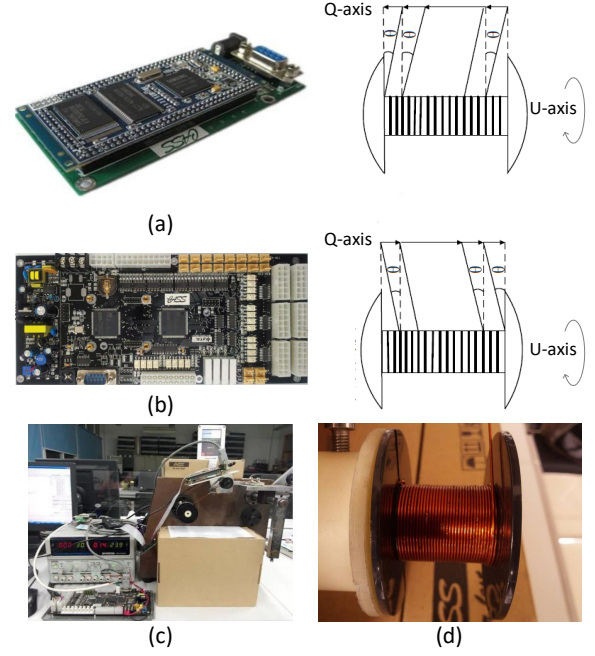


Fig. 11. (a) is the visual system, whose CUP is ARM9 based S3C2440 and which uses the CANIF interface to communicate with OV9650 CMOS Camera; (b) is the ePLC, CASS-PLCA149B; (c) is the winding machine with visual system which contains two axes: the U-axis and Q-axis; (d) is shown the Winding effect, the angle of the wire, Q-axis and U-axis.

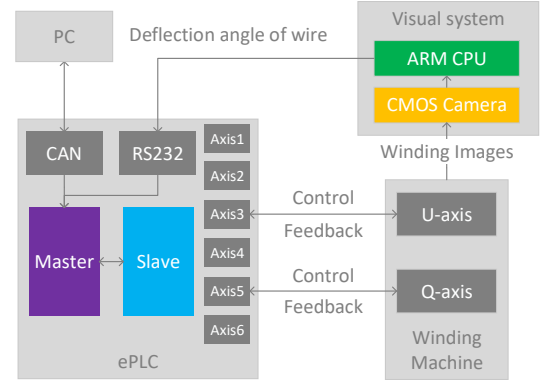


Fig. 12. Structure of the winding machine system. The ePLC has two chips: a master chip and a slave chip. It uses the RS232 to receive *PFs* and could control six servo systems at most. The winding machine has two axes: the U-axis and Q-axis. The *VS* use the CMOS camera to gain winding images and transfer the digital information to ARM CUP. The ARM CUP will extract parameters, frame *PFs* and transfer them to ePLC continuously.

separated control and algorithm layer programs could be developed by different level programmer when a new application has some customized requirements.

- 2) the parameters interacted between *VS* and ePLC could be associated using the *PT*. Meanwhile, the *PFs* contained the parameters could be carried to every layers. This makes it that when more parameters need to be interacted, only the *PT* should be changed without reprogramming.
- 3) the customized structure of ePLC, favorable memory

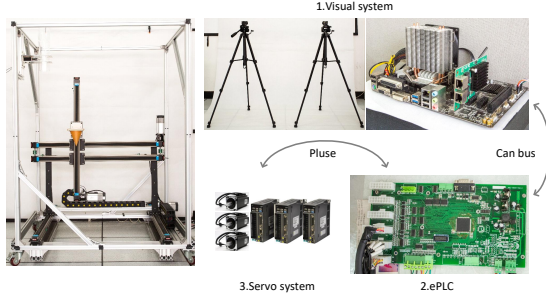


Fig. 13. Binocular Catching Robot consists of visual system, *ePLC* and servo system.

allocation, and Petri-Net based multi-threading structure make it possible that high-performance algorithms could run in single processor.

From the view of the applications, the proposed architecture is also easily applied to very different another case.

C. Case Two: Binocular Catching Robot

As shown in Fig. 13, the binocular catching robot adopts two cameras to judge the position of the ball in the space. Through sending the continuously parameters to *ePLC*, the robot runs to the position to catch the ball. Finally, the robot will catch the ball. The types of cameras and lens are GE31GC and FA0401C, respectively; the visual system is the GIGABYTE's GA-B85M-D3V-A. *ePLC* uses the TI F28M35 chip with two cores: ARM Cortex M3 and TI C28x, which contains a shared RAM. The servo system is adopted ASDA-B2.

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

2) *Process of PF*: The *PF*s of a time catching the ball are shown in Table IV and the trajectory of the ball is shown in Fig. 14 which is shown trajectory captured by the cameras. The red point is the position where the ball is thrown, the blue points are the positions where predict the catching points, and the yellow points are the predictive catch points. The *VS* sends the *PF* to *ePLC* to adjust the destination and speed of every axis.

D. Results of Case Two

Through the analysis of two cases, the proposed *VCA* based flexible structure provides a generic method to address the visual servo control problem in *ePLC*. Based on 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 : 0xA	vid_4 : 0x1	var_4 : 0x40	pid_5 : 0xE	vid_5 : 0x1	var_5 : 0x40	pid_6 : 0x10	vid_6 : 0x1	var_6 : 0x40

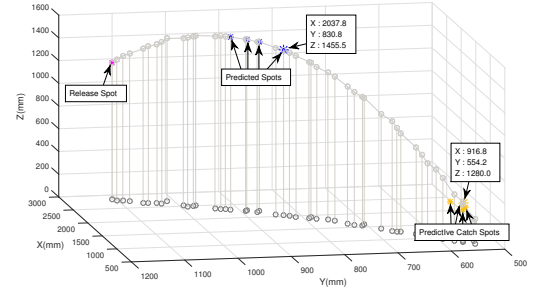


Fig. 14. Trajectory captured by the cameras. The red point is the position where the ball is thrown, the blue points are the positions where to predict the catching points, and the yellow points are the predictive catch points.

VCA protocol, the algorithms in *VS* and *ePLC* have a kind of normative correspondence, and the existing modules and motion control algorithms could be reused without additional programming.

VII. CONCLUSION

We propose a flexible multi-level architecture which is based on a *VCA* protocol to integrate the PLC system, motion control system, and visual system. This architecture provides users an easy development method to ease the ever-growing complexity of the applications. The multi-level architecture included flexible layer, control layer, and algorithm layer are introduced and then *VCA* protocol is posed for data interaction among the layers. Correspondingly, a customized hardware, memory allocation, and Petri-Net based multithreading structure are described to support the proposed flexible software architecture. We implement two cases, the winding machine with visual system and the binocular catching robot, to illustrate the proposed system which provides a generic method to develop different type applications of visual servo system in *ePLC*. This indicates that the proposed *VCA* based architecture could be easily applied between two quite different cases.

In the further research, we will implement a uniform development method of the visual servo control in *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. Kazmierowski, "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. Khokhi, 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] X. Feng, S. A. Velinsky, and D. Hong, "Integrating embedded pc and internet technologies for real-time control and imaging," *IEEE/ASME Transactions on Mechatronics*, vol. 7, no. 1, pp. 52–60, 2002.
- [8] 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.
- [9] 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.
- [10] 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.
- [11] 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.
- [12] Y. Jiang, H. Zhang, X. Song, X. Jiao, W. N. N. Hung, M. Gu, and J. Sun, "Bayesian-network-based reliability analysis of plc systems," *IEEE Transactions on Industrial Electronics*, vol. 60, no. 11, pp. 5325–5336, 2013.
- [13] S. Gerškšič, G. Dolanc, D. Vrančić, J. Kocijan, S. Strmčnik, S. Blažič, I. Škrjanc, Z. Marinšek, M. Božiček, and A. Stathaki, "Advanced control algorithms embedded in a programmable logic controller," *Control Engineering Practice*, vol. 14, no. 8, pp. 935–948, 2006.
- [14] 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.
- [15] 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.
- [16] 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.
- [17] 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.
- [18] 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.
- [19] 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.
- [20] Y. Wang, H. Lang, and C. W. D. Silva, "Visual servo control and parameter calibration for mobile multi-robot cooperative assembly tasks," in *IEEE International Conference on Automation and Logistics*, 2008, pp. 635–639.
- [21] 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.
- [22] 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.
- [23] 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.
- [24] N. Guenard, T. Hamel, and R. Mahony, "A practical visual servo control for an unmanned aerial vehicle," *IEEE Transactions on Robotics*, vol. 24, no. 2, pp. 331–340, 2010.
- [25] 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.
- [26] 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.
- [27] 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.
- [28] 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.
- [29] 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.
- [30] 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.
- [31] W. F. Peng, G. H. Li, P. Wu, and G. Y. Tan, "Linear motor velocity and acceleration motion control study based on pid+velocity and acceleration feedforward parameters adjustment," *Materials Science Forum*, vol. 697–698, pp. 239–243, 2011.
- [32] 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.
- [33] O. Co.Ltd, "Cs1w-mc221(-v1)/mc421(-v1) motion control units." *Operation Manual*, 2004.
- [34] P. T. C. 2., *Function blocks for motion control version 1.1*, 2005.
- [35] C. Sünder, A. Zötl, 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.
- [36] 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.