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

***Abstract*—**

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

I. Introduction

Integration technologies is driving the industrial automation [1]. The continuously integrations of sensors, controllers, robots, tools, etc. bring the concepts of self-aware equipment, intelligent factory and CPS, etc. [2], [3]. Recent researches [4]–[6] introduce the development of integration technologies. In industrial automation, logic control system, motion control system and visual system become more and more important and inseparable [7]–[9]. On the other hand, the advances of edge computing, fog computing and edge artificial [10]–[12] put forward new challenges on edge equipment.

A. Motivations

Nowdays, visual system has been applied in various fields. The logic control on PLC has wide applications. Motion control [13] is a typical case that describes how the three parts collaborate. The visual system analyzes the context and get error put into the motion system. Simultaneously the logic program are judging the information, such as the position limitation of the every axis. Hence, how to pose a flexible structure to the integration of logic control system, servo system and visual system, guides us.

B. Related Works

Visual control system is combined of special motion control system and visual system. Such as transport [14], circuit detection [15], sorting system, welding [13], assembling [16], [17], robot [18], [19], unmanned aerial vehicles [20], [21]. These works address their problems in relevant fields. However, all these solutions are based on special motion control system and visual system.

On the other hand. The integration of logic control and motion control has variously deep researches [22]–[25]. [22], [25] realize the motion control directly in PLC. [26]–[28] use motion control module collaborated with PLC to implement their applications. However, the development method in these papers is disordered. Therefore, in 2005, PLCopen organization has released a related standard [29] which standardizes the motion control in PLC. Based on this standardization, [30] provides an advanced implementation in distributed automation system and companies, such as 3S [31], provide some tools. [32] poses a customized real-time compilation method to reduce the development complexity.

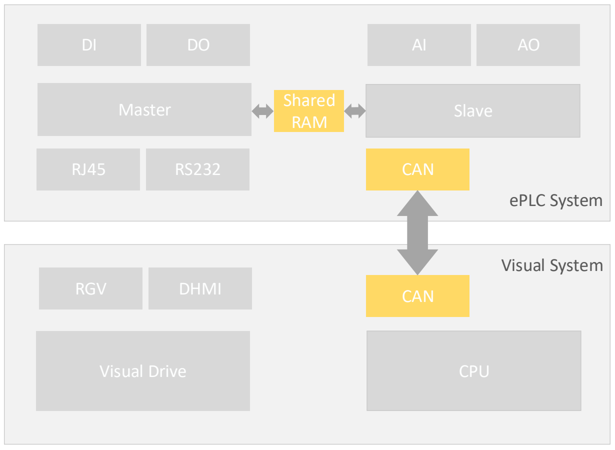


Fig. 1. A typical ES which adopts two processor architecture. The shared memory is used to transfer data between master and slave processors. The communication between ES and VS adopt the CAN Bus.

Above works provide impressive integrations on visual servo control system and PLC with motion control functions, however there are few papers discussing the integration of visual system, motion control and PLC. Most of applications are focusing on its application with three individual system, such as [13]. Hence, an integration structure of logic control system, servo system and visual system should be provided to reduce the complexity and expand the application fields.

C. Our Contributions

.

II. System Architecture

A. Hardware Structure

The hardware is comprised of ePLC system (ES)and visual system (VS). The ES is a customized structure. The number of DI/DO, AI/AO and controlled servo system could be increased according to applications. Fig. 1 shows a typical ES which adopts two processor architecture. The shared memory is used to transfer data between master and slave processors. The communication between ES and VS could adopt multiple protocols, such as TCP, modbus, CAN, etc. In the shown Fig. 1, we adopt the CAN Bus.

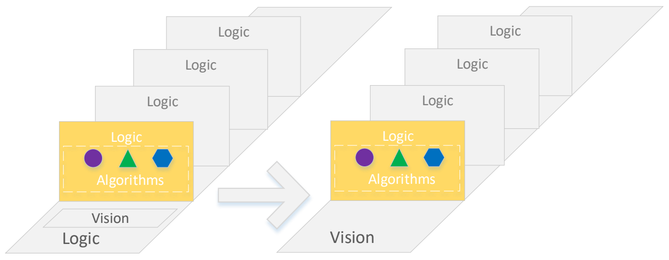


Fig. 2. The three-layer architecture of ePLC

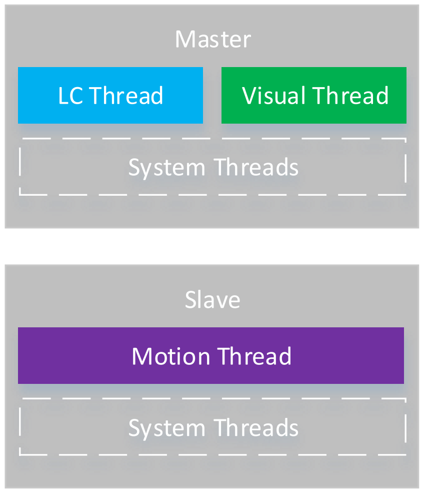


Fig. 3. The process of the three type threads

B. Software Structure

In normal visual control system, the typical software structure is seen as in the left of Fig. 2.

C. Thread Structure

We adopt the analogous thread structure in [32]. There special threads introduced below:

1) Visual Thread. This thread is responsible for interaction with VS. It will receive the protocol frame from the VS and save it to relevant address.
2) LC Thread. Logic control thread mainly execute logic program, deframe the protocol frame and interaction data with algorithm thread.
3) Algorithm Thread. Algorithm thread runs in slave processors. It is used to implement data interaction between processors and execution of algorithms.

D. Memory Allocation

The dedicated storage area of PLC 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, we have three definitions below.

Definition 1 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.

Definition 2 If $\exists S \subseteq M$ and $(\forall s_i \in S) \in \{0,1\}$. Meanwhile, each element s_i has four operators: $\mathcal{S}_0(s_i)$ assigns 0 to s_i, $\mathcal{S}_1(s_i)$ assigns 1 to s_i, $\mathcal{J}_0(s_i)$ represents that the value of s_i is 0, $\mathcal{J}_1(s_i)$ means that the value of s_i is 1. Then we define the set S has $\mathcal{B}$ attribute.

Definition 3 If $\exists S \subseteq D$ and $\forall s_i \in S$ has 4 byte. We define the set S has $\mathcal{D}$ attribute:

Fig. **??** shows the memory allocation of master and slave processors. All slave processors have the same storage structure.

LCF (Logic Control Flag Area): the flag are used to start the modules. It has $\mathcal{B}$ attribute.

LCD (Logic Control Data Area): these data will be used to delivery to algorithm. It has $\mathcal{D}$ attribute.

AF (Algorithm Flag Area): it includes algorithm flag of execution (AFE) and algorithm flag of state (AFS). Both of them have $\mathcal{B}$ attribute.

AD (Algorithm Data Area): these data help specified algorithm executing. It has $\mathcal{D}$ attribute.

MF (Message Flag Area): it includes defined message flag (DMF) and user customized message flag (UMF). DMF is the neccesary message for system execution including start module flag, alarm flag, etc. UMF could be defined by users. Both of them have $\mathcal{B}$ attribute.

MD (Message Data Area): it is used to transfer message information which includes system message data area (DMD) and usr message data area (UMD). It is defined in D area.

$MPDIF$ (Master Processor Data Interaction Flag Area): it 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). All of them have $\mathcal{B}$ attribute.

MSD (Master Processor Data Interaction Data Area): an area stores the data delivered from slave processors and it has $\mathcal{D}$ attribute.

$SPDIF$ (Slave Processor Data Interaction Flag Area): this area 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). All of them have $\mathcal{B}$ attribute.

SMD (Slave Processor Data Interaction Data Area): an area stores the data delivered from master processor and it has $\mathcal{D}$ attribute.

As shown in Fig. **??**, we define the LPM data interaction with three parts: $\mathcal{L}$ (layer data interaction), $\mathcal{P}$ (processor data interaction) and $\mathcal{M}$ (module data interaction). $\mathcal{L}$ seen in [**?**] is the process to exchange the data between application customized layer and control layer.

$\mathcal{P}$ is used to interact data between master processor and slave processors, hence it has transferring data from master to slave ($\mathcal{P}_{mts}$) and transferring data from slave to master ($\mathcal{P}_{stm}$) and it is defined below:

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

Where $\mathcal{U}$ is the function to implement the process of data interaction between master and slave processors. $\mathcal{P}_{mts}$ and $\mathcal{P}_{mts}$ use the same function $\mathcal{U}$.

The process of $\mathcal{P}_{mts}$ is seen below: $\mathcal{S}_1(msb_i) \rightarrow \mathcal{S}_1(msf_i) \rightarrow send(smd_i) \rightarrow \mathcal{S}_0(msb_i) \rightarrow \mathcal{S}_1(sms_i) \rightarrow check(smd_i) \rightarrow \mathcal{S}_1(sma_i) \rightarrow \mathcal{S}_0(sms_i) \rightarrow \mathcal{S}_0(sma_i) \rightarrow \mathcal{S}_0(msf_i)$.

Where $send(msd_i)$ means sending data to msd_i in slave processor. $check(msd_i)$ means to check data of msd_i.

$\mathcal{M}$ is used to interact data among modules. It includes two types message: system defined message interaction $\mathcal{M}_d$ and user message interaction $\mathcal{M}_u$. The process is defined below:

$$\begin{cases} \mathcal{M}_d = \mathcal{V}(dmf_i, dmd_i, \mathcal{E}) \\ \mathcal{M}_u = \mathcal{V}(umf_i, umd_i\, \mathcal{E}) \end{cases} \quad (2)$$

Where $\mathcal{V}$ is the function to broadcast message and transfer data. $\mathcal{E}$ is the collection of all execution functions after getting related message, $\mathcal{M}_s$ and $\mathcal{M}_u$ have the same function $\mathcal{V}$.

One module receives a system defined message shown below: $\mathcal{J}_1(smf_i) \rightarrow GetMessage_j(dmf_i) \rightarrow GetData(dmd_i) \rightarrow \mathcal{E}_j$.

Where $GetMessage_j(dmf_i)$ represents the ith module getting a message dmf_i and $GetData(dmd_i)$ represents the ith module getting the message information.

III. Integration of VS and ES

A. VCA Protocol Template

In cater to most applications, we propose a VCA-based flexible architecture. As shown in Fig. , 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 put into a fixed area of RAM. The parsing modules form both systems will read it when parsing the protocol data.

The PT is defined below:

$$\begin{cases} PT = \{HPT, VLT, CLT, ALT\} \\ HPT = \{CID, TDA\} \\ VLT = \{MID, MAN, MSF, MDA, MAS, CDATA\} \\ CLT = \{AID, APN, AF, ADA, APS, ADATA\} \\ ALT = \{PID, PDATA\} \end{cases} \quad (3)$$

The PT contains four parts: head of protocol template (HPT), visual layer template (VLT), control layer template (CLT) and algorithm layer template (ALT). 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) VLT: it consists of module unique ID (MID), module contained algorithm number (MAN), module start flag (MSF), module data start address (MDA), module contained algorithm IDs (MAS) . VLT is not $\emptyset$. Every MAS include MAN algorithm IDs.
3) CLT: it is comprised of algorithm unique ID (AID), algorithm contained parameter number (APN), AF, algorithm data start address ADA, algorithm contained parameter IDs (APS). CLT is not $\emptyset$. Every APS include APN parameter IDs.
4) ALT: it contains parameter unique ID (PID). ALT is not $\emptyset$.

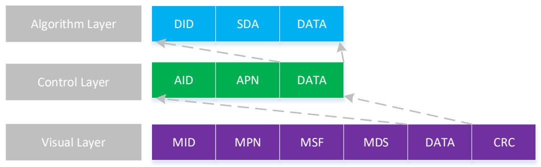


Fig. 4. VCA Protocol.

B. VCA Protocol Frame

The VCA protocol frame (PF) contains control frame and algorithm frame as shown in 4.

1) PF: it consists items of MID, visual frame length VFL, $CDATA$ and cyclic redundancy check CRC.The $CDATA$ contains several control frames. The MIDs both in PT and PF need one-to-one correspondence.
2) Control frame: it is comprised of AID, control frame length (CFL) and $ADATA$. The $ADATA$ includes several algorithm frames. The AIDs both in control frame and PF need one-to-one correspondence.
3) Algorithm frame: it contains PID, $PDATA$. PID is also the address of $PDATA$. The PIDs both in algorithm frame and PF need one-to-one correspondence.

C. Framing of PF

In VS, the PLC interface is designed to transfer PF to PLC . It has five steps: create data, VL framing, CL framing, AL framing and transmission.

Create data: after image processing, the VS extracts the useful data and storages into visual extracted data (VED) in which the parameters could be indexed by the PID.

***VL* Framing**: the process is realized as the Algorthm 1. It searches the PT with mid to find the relevant vlt_x. Through it, the VS can obtain the msf and mda. According to man, call the next step (VL Framing) to gain the $cdata$. Then calculate the length (vfl) and crc to finish the VL framing.

***CL* Framing**: Algorthm 2 illustrates the process. The VS seeks the FT to gain the clt_y which contains the af and ada. According to apn, call the next step (AL Framing) to obtain the $cdata$ and then calculate the length (cfl) to finish the CL framing.

***AL* Framing**: Algorthm 3 shows the process. The VS obtains the alt_z frome FT and then combines the pid and $pdata$.

Transmission: transfer the VCA protocol frame to $ePLC$ with the relevant communication protocol in CID.

Algorithm 1: $VLFraming$

```
Input: $mid$, $VED$
Output: $PF$
SearchPT ($mid$, $vlt_x$);
$PF.mid \leftarrow mid$;
for $i = 0$;$i < vlt_x.man$;$i++$ do
    ALFraming ($vlt_x.mas$[i], $VED$,$cdata$);
end
$PF.cdata \leftarrow cdata$;
$PF.vfl \leftarrow$ Length ($cdata$) + 8;
$PF.crc \leftarrow$ CRC16 ($PF$);
```

Algorithm 2: $CLFraming$

```
Input: $aid$, $VED$, $cdata$
Output: $cdata$
SearchPT ($aid$, $clt_y$);
Create $clf$;
$clf.aid \leftarrow aid$;
for $i = 0$;$i < clt_y.apn$;$i++$ do
    ALFraming($clt_y.aps$[i], $visualData$, $adata$);
end
$clf.cfl \leftarrow$ Length($adata$) + 8;
$clf.adata \leftarrow adata$;
$cdata \leftarrow cdata$ + $clf$;
```

Algorithm 3: $ALFraming$

```
Input: $pid$, $VED$,$pdata$
Output: $pdata$
Search the $PT$ with $pid$ and get $alt_z$;
Create $alf$;
$alf.pid \leftarrow pid$;
$alf.pdata \leftarrow VED[pid]$;
$pdata \leftarrow pdata$ + $alf$;
```

D. Deframing of PF

VL deframing, CL deframing and AL deframing.

Receiving *PF*: according to the CID in PT, the ES chooses the communication protocol receives the PF form VS and then check the CRC. If correct, prepare to save.

Saving *PF*: there exists a pointer pointing to the address where could save the next PF. After saved the PF, the pointer point to the next new address according the vfl of PF.

***VL* Deframing**: there exits a pointer PDF pointing to the PF which needs to be parsed.

IV. System Operation Mechanism

A. Implementation of Flexible Program and Execution

The FL contains two parts: PLC interface and visual interface. They interact with each other with the agreed communication protocol.

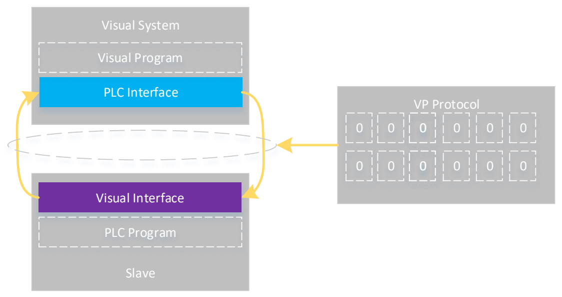


Fig. 5. Data Interaction Based on VP Protocol.

Algorithm 4: $VLDeframing$

```
Input: $PDF$
$mid \leftarrow$ four bytes start form $PDF$;
$msn \leftarrow$ four bytes start form $PDF + 4$;
searchPT ($mid$, $vlt_x$); $cdata \leftarrow msn - 8$ bytes start
  form $PDF + 8$;
$CassMemM[vlt_x.mda] \leftarrow cdata$;
$PDF \leftarrow PDF + msn$;
```

Algorithm 5: $CLDeframing$

```
Input: $mid$
searchPT($mid$, $vlt_x$);
$mda \leftarrow vlt_x.mda$;
for $i = 0$;$i < vlt_x.man$;$i++$ do
    searchPT($vlt_x.mas[i]$, $clt_y$);
    $CassMemS[clt_y.apa] \leftarrow clt_y.vfl - 8$ bytes start
      form $mda + 8$;
    $mda \leftarrow mda + clt_y.vfl$;
end
clean;
```

Algorithm 6: $ALDeframing$

```
Input: $aid$
searchPT($aid$, $clt_y$);
$ada \leftarrow vlt_x.ada$;
for $i = 0$;$i < clt_y.apn$;$i++$ do
    $CassMemA[CassMemA[ada]] \leftarrow$
      $CassMemA[ada + 4]$;
    $ada \leftarrow ada + 8$;
end
clean;
```

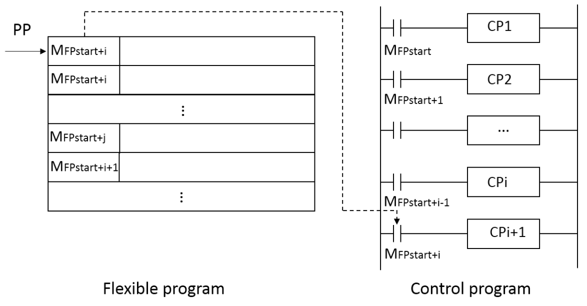


Fig. 6. The execution OF flexible program

The PLC interface is responsible for get parameters of motion control form VS and interact with ES. It includes three steps blew.

Step 1: after image processing, the VS extracts the useful data and storages into visual extracted data (VED) in which the parameters could be indexed by the PID.

Step 2: frame them into PF with the Algorithm 1, 2 and 3.

Step 3: transfer the VCA protocol frame to $ePLC$ with the relevant communication protocol in CID.

Visual interface is in the $ePLC$ designed to interact with VS and it contains two steps below.

Step 1: receive PFs from VS according the CID in HPT.

Step 2: save them in the ES. There are two pointers: 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.

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. $CP = \{VLDP, CLDP, IP, LP, PIP\}$, where CLP is the control layer deframe program. IP is the initial program which is used to initialize the data and state. LP is the logic program. PIP is the processor interaction program implemented to transfer and receive data between processors. The implementation of control program is described below.

Step 1: if $PDF != PSP$, execute the $VLDP$ which means to call Algorithm 4

Step 2: traverse the LCF to check whether there is a module needed to execute. If $\exists\ \forall\ lcf_i \in LCF$ is 1, go to step 2.

Step 3: execute the IP to initialize the data and state and then execute the LP to find the called algorithm. If $\exists\ \forall\ af_i \in AF$ is 1, go to step 4.

Step 4: execute the $CLDP$ to call Algorithm 5.

Step 5: the $\mathcal{P}_{mts}$ is used in PIP to inform the slave processor to receive parameters and start algorithms and clear the relevant af_i.

Step 6: read and process the feedback data form the slave processor.

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.

Step 1: traverse the LCF to check whether there is a algorithm needed to execute. If $\exists\ \forall\ af_i \in AF$ is 1, go to step 2.

Step 2: ALDP is executed to call Algorithm 5. The parameters are transfered to correlative as_i.

Step 3: as_i is executed and uses the $\mathcal{P}_{stm}$ to feed back data to the master processor.

D. Execution of Threads

Commonly, threads in master processor and in slave processors execute separately according to their priority and the interaction between control thread and algorithm threads occurs when using $\mathcal{P}_{mts}$ and $\mathcal{P}_{mts}$. Basic execution units of control thread are shown as follows:

C_1 Start the module.

C_2 Transfer data to motion thread by $\mathcal{P}_{mts}$.

C_3 Seal with the feedback data.

C_4 Broadcast a message.

C_5 Handle the message.

Motion thread contains the following basic execution units:

M_1 Start the algorithm.

M_2 Execute the algorithm.

M_3 Feedback the data to control thread by $\mathcal{P}_{stm}$.

M_4 End algorithm.

Two cases shown in Fig. 7 are explained below:

Case one: execution of control thread and two algorithm threads among three processors. The CT (Control Thread) traverses LCF, finds ms_i to be executed , runs lcp_i, finds ap_j, executes C_1 unit, executes C_2 unit and then transfers data from LCD to SMD of processor 1. AT (Algorithm Thread) 1 executes M_1 unit, executes M_2 after transferring data from SMD to AD, runs M_3 unit, feedbacks data to CT. When the ap_j finishes, AT 1 executes M_4 and informs CT the end of ap_j. CT executes C_3 to end the process and then finds ap_{j+1}, executes C_1 and C_2, transfers the data from LCD to SMD of processor 2, AT 2 executes M_1 unit, executes M_2 after transferring data from SMD to AD. When the ap_{j+1} finishes, AT 2 executes M_4 unit and informs CT the end of ap_{j+1}. CT executes C_3 to finish the process.

V. 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.

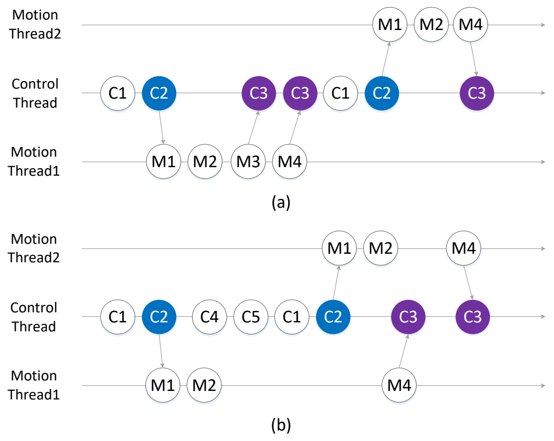


Fig. 7. Visual servo control system

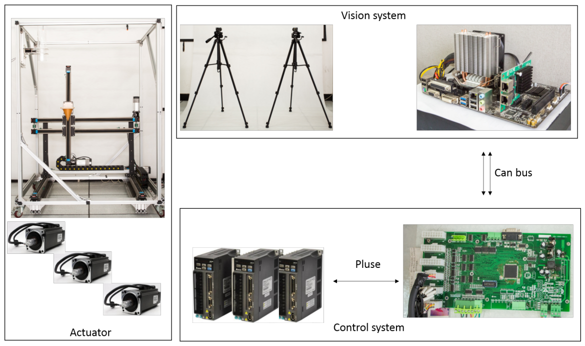


Fig. 8. Visual servo control system

A. Case 1 Binocular Catching Robot

As shown in Fig. 8, the binocular catching robot adopts two cameras to judge the position of the ball. Through sending the continuously parameters to $ePLC$ to adjust the position of the robot. Finally, the robot will catch the ball. The cameras is xxx, the visual system is xxx, and the $ePLC$ is XXX. $ePLC$ uses tht TI F28M35 chip with two cores: ARM Cortex M3 and TI C28x, which contains a shared RAM. The servo driver and motor are xxx and xxx respectively.

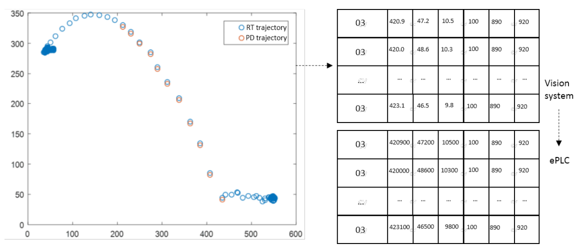
1) Design of PT:

2) Design of Program:

3) Result:

B. Case 2 Winding Machine with Visual System

1) Design of PT:



03	420.9	47.2	10.5	100	890	920
03	420.0	48.6	10.3	100	890	920
...	...	...	...	...	...	...
03	423.1	46.5	9.8	100	890	920

03	420900	47200	10500	100	890	920
03	420000	48600	10300	100	890	920
...	...	...	...	...	...	...
03	423100	46500	9800	100	890	920

Fig. 9. The execution OF flexible program

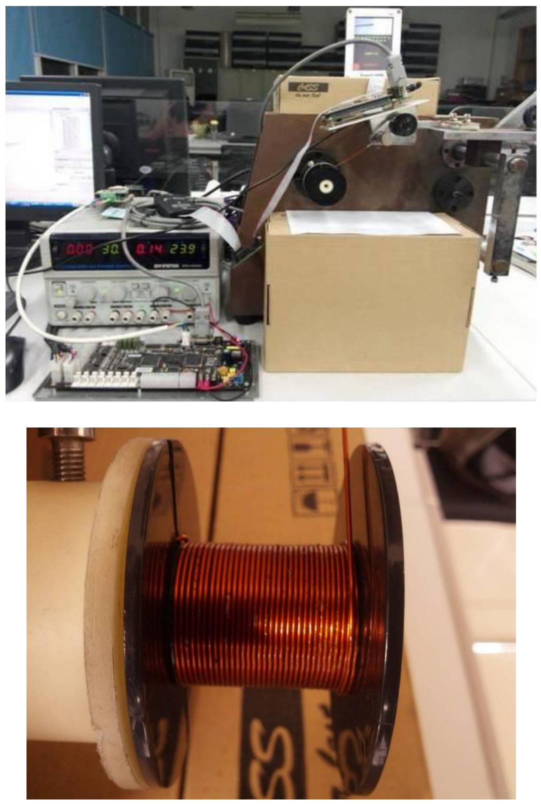


Fig. 10. Winding machine Based on Visual System

2) Design of Program:

3) Result:

VI. Conclusion

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

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] 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] P. Hu, H. Ning, T. Qiu, Y. Zhang, and X. Luo, “Fog computing based face identification and resolution scheme in internet of things,” *IEEE Transactions on Industrial Informatics*, vol. 13, no. 4, pp. 1910–1920, 2017.

[11] W. Hou, Z. Ning, and L. Guo, “Green survivable collaborative edge computing in smart cities,” *IEEE Transactions on Industrial Informatics*, vol. PP, no. 99, pp. 1–1, 2018.

[12] P. Pace, G. Aloi, R. Gravina, G. Caliciuri, G. Fortino, and A. Liotta, “An edge-based architecture to support efficient applications for healthcare industry 4.0,” *IEEE Transactions on Industrial Informatics*, vol. PP, no. 99, pp. 1–1.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[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] 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.

[21] 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.

[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] 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.

[27] 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.

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

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

[30] 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.

[31] 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.

[32] 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.