Research on Function Block Programmable Technology of Machine Vision

Chongquan Zhong\textsuperscript{1,2*}, Xianglong Dong\textsuperscript{1}, Hongtao Sun\textsuperscript{2} and Zhipeng Xing \textsuperscript{2}

\textsuperscript{1}Dalian University of Technology, Dalian 116000, China
\textsuperscript{2}DUT Computer Control Engineering Co., Ltd, Dalian 116000, China.
\*Corresponding author’s e-mail: zhongcq@dlut.edu.cn

Abstract. Aiming at the problems of low efficiency, difficult debugging and poor reusability of program in developing vision program by using text editing, a graphical program development technology is developed for vision program development. In this paper, the image processing algorithms and functions in the OpenCV vision library are studied and encapsulated according to the IEC61131-3 Function block specification. The interface model and program model of vision Function block are established. According to these two models, the vision Function block interface and the vision Function block program are described by using XML language, and the programming specification and instruction set of the vision Function block are established. The editing and compiling algorithm of the Function block program is researched to realize the configuration development environment and operate environment of the vision Function block program by using C plus plus language. Finally, the function of the software is tested and verified.

1. Introduction
With the development of intelligent manufacturing technology centered on industrial robot technology, the combination of computer vision technology and PLC technology realizes machine substitution and unmanned manufacturing, solving the problem of insufficient labor supply in manufacturing and becoming the mainstream trend of the current manufacturing development[1]. There are currently many cross-platform computer vision libraries such as OpenCV[2] and Halcon[3], which provide many common algorithm functions in the aspect of image processing and computer vision. By calling algorithm functions implemented in the computer vision library, the development of machine vision programs can be simplified. However, the current mainstream development method of vision programs is based on text programming, which requires developers to have sufficient knowledge of the algorithm, functions, and interfaces of the vision library, leading to the cumbersome programming process. It needs to repeatedly adjust vision algorithms and process parameters to adapt to different application environments and needs. The way of text programming makes it difficult for beginners to get started, which greatly restricts the development of machine vision technology. Aiming at the problems of complex programming and poor reusability of traditional text-based vision program development methods, domestic and foreign scholars have made a lot of explorations[4-9]. Zhu and others encapsulated the vision algorithm in a componentized manner based on the Microsoft COM component object model, and achieve the completion of writing image processing programs by calling components in the development environment. This approach improves the reusability of the software, but the invocation of components during the development process and the establishment of relationships between components still need to be implemented with text programming[10]. Ren and others developed
a flowchart-based vision programming software based on nodes, in which vision algorithms are replaced by nodes. The algorithm data dependency relationship is established through the connection between nodes, which can effectively improve the editing efficiency of vision programs, but the programming platform does not support the development of logically complex vision programs[11]. Hu and others encapsulated the vision algorithm into a primitive control. In the editing interface, the vision program development is realized by editing the graphic element control to form a flowchart. This method can effectively reduce the development difficulty of the vision program, but the algorithm parameter display of the graphic element control is not intuitive, which is not conducive to program editing and debugging[12].

IEC61131-3 as a programming language standard for programmable controllers has been widely used in logic and motion control programming since it was proposed in 1993[13]. Function block diagram language is a graphical programming language defined in the IEC61131-3 standard. It replaces algorithm function with Function blocks. The input and output parameters of algorithm are configured through the interfaces on the left and right sides of the Function block. The algorithm execution order and data transfer are controlled by the connection relationship between the Function blocks. Users can realize the development of complex function programs only by completing the connection between instructions and the configuration of instruction parameters by simply dragging and dropping.

2. Vision Function Block Program Model

Referring to the functional block language standard of IEC61131-3, the function block is the smallest execution unit in a functional block program. It can encapsulate the vision algorithms commonly used in vision processing programs and expose only the input and output interfaces to the outside. Combining the characteristics of image processing algorithms, the vision function block structure is designed shown in Figure1.

![Figure 1. Vision Function block](image)

The structure of the vision function block can be divided into two parts: the head and the body. The name of the Function block is displayed in the head. The input and output elements of the function block are displayed on both sides of the body. The size of a function block is determined by the number of input and output elements. The information flow transmitted by the body can be divided into energy flow and data flow. The energy flow passes the event information of the Function block. For example, EN and ENO are the default element 1 of the Function block interface, which represents the instruction's energy flow control by passing Boolean data. The data flow passes the operation data performed by the encapsulation algorithm. For example, the default No. 1 element of the visual instruction interface is Pic, and the passed data is a string type indicating the index of the image data. The parameter configuration of the Function block interface has three forms, which are immediate value, Variable and port connection. According to the structure of the vision function block, the function block interface model is defined as follows:

Definition 2-1: The function block interface model "Port" can be defined as a 7-tuple Port = <Name, InOrOut, Type, Visible, Editable, Wired, Value>. Among them, "Name" represents the name of the interface element. "InOrOut" indicates whether the interface element is an input or an output. "Type"
represents the data type of the interface element. "Visible" indicates whether the interface element is hidden or not. "Editable" indicates whether the interface element is editable. "Wired" indicates the connection status of the interface element pins. "Value" indicates the default value of the parameter of the interface element.

According to the function block interface model, the shape and interface of all the Function blocks in the instruction set can be described by using XML language. Finally, the description file of the function block is formed, which facilitates the unified design and management of the Function blocks. The specific content of the Function block description file is shown in Figure 2.

```
<?xml version="1.0" encoding="GB2312"?>
<FBDDescription>
    ......
    <Classify ID="Morphological processing instruction">
        ......
        <FBDBase Type="CV_Erode" PS=""/>
            <BasicProperty Row="" Col="" />
        </BasicProperty>
        <FunctionProperty>
            <Property ID="EN" InOrOut="In" PS="" Visible="Yes" Wired="No" Editable="Yes" Type="BOOL">NULL</Property>
            <Property ID="Pic" InOrOut="In" PS="" Visible="Yes" Wired="No" Editable="Yes" Type="STRING">NULL</Property>
            <Property ID="Kernel_Row" InOrOut="In" PS="" Visible="Yes" Wired="No" Editable="Yes" Type="INT">3</Property>
            <Property ID="Kernel_Col" InOrOut="In" PS="" Visible="Yes" Wired="No" Editable="Yes" Type="INT">3</Property>
            <Property ID="Iteration" InOrOut="In" PS="" Visible="Yes" Wired="No" Editable="Yes" Type="INT">1</Property>
            <Property ID="ENO" InOrOut="Out" PS="" Visible="Yes" Wired="No" Editable="Yes" Type="BOOL">NULL</Property>
            <Property ID="Pic" InOrOut="Out" PS="" Visible="Yes" Wired="No" Editable="Yes" Type="STRING">NULL</Property>
        </FunctionProperty>
    </FBDBase>
    ......
    </Classify>
    ......
</FBDDescription>
```

Figure 2. Function block description file

Figure 2 shows the description information of the corrosion function block "CV_Erode". The attribute "type" of the node <FBDBase> records the instruction name displayed in the instruction header. The size of the Function block body is determined by the maximum number of input and output elements of the interface. Each element under the node <FunctionProperty> represents an interface element of the function block, which records attribute information such as interface element name, input / output, data type, and parameter default values. The description information of the Function block CV_Erode in the figure shows that there are five input elements: "EN", "Pic", "Kernel_Row", "Kernel_Col", Iteration, and two output elements: "ENO", "Pic". The maximum number of input and output interface elements is 5. Therefore, the height of the function block "CV_Erode" occupies a 6-row grid, and the default width occupies a 2-column grid. The function block interface description method is versatile and can quickly develop Function block.

The function block program in the network consists of several independent program segments, which are composed of interconnected function blocks. The structure of the vision function block program is shown in Figure 2. The network is shown as a uniformly distributed grid array. Function block programs consist of visual programming elements, including function blocks, connecting lines, and variables. Among them, the function block represents an algorithm function, and the execution order of the Function block instruction is determined by the position of the grid array where the function block is located and the connection relationship between the function blocks. The data transmission between the function blocks is realized through the connecting lines between the variables and the function blocks. The program has three basic structures: sequential structure, branch structure, and cyclic structure, by which arbitrarily complex program structure can be combined. In the vision function block program, the sequence structure is realized by the position relationship and connection relationship between instructions. The branch structure is implemented by the logic variable control instruction power flow, and the loop structure is implemented by the loop variable control program control instruction. The function block program model is defined as follows:

Definition 2-2: The vision function block program model "CVFBD" is a 2-layer structure model, which can be defined as a 4-tuple CVFBD = <NetWork, CVFB, Line, Variable>. Among them, "NetWork" represents the data information of the grid array in the program network. "CVFB" represents
Definition 2-3: The vision function block "CVFB" can be defined as a 6-tuple CVFB = <Name, Type, Position, Port, Dialog, Fun>. Among them, "Name" represents the function block name. "Type" indicates the category to which the function block belongs. "Position" indicates basic attribute information of the function block, such as position and size. "Port" represents the input and output interface information of the instruction. "Dialog" indicates the debug dialog resource of the function block. Most input and output data of vision algorithms are complex. Through the debug dialog, the visual configuration of input parameters and the visual display of execution results can be realized. "Fun" represents an algorithm function encapsulated by the function block.

Definition 2-4: The graphic structure of the connecting line has one input end and multiple output ends, which can be defined as a 5-tuple Line = <InCVFB, OutCVFB, Type, Position, Direction>. Among them, "InCVFB" represents a set of starting function blocks, and records the index of the function block object connected to the input end of the connecting line and the pin number. "OutCVFB" represents a set of ending function blocks. "Type" indicates the type of data passed. "Position" represents the set of coordinate points in the grid array. A maximum of six turning points (five line segments) can be recorded to complete the connection from the output pin of one instruction to any input pin of any other instruction. "Direction" indicates the connection direction information at the turning point, records the direction of data transmission, and provides help for dynamically establishing the turning point when connecting line.

Definition 2-5: Variables can be defined as a 3-tuple Variables = <Name, Type, Value>. Among them, "Name" represents the name of the variable. "Type" represents the data type of the variable. "Value" represents the value of the variable.

Table 1. NetWork class structure

| Property members of the class | Data type | Attribute description                                      |
|-----------------------------|-----------|-----------------------------------------------------------|
| Rows                        | int       | Number of rows in the network cell                         |
| Cols                        | int       | Number of columns in the network cell                      |
| GridArray                   | Array     | Data of network cells stored as a two-dimensional array    |
| BlockList                   | CTypedPtrList | A collection of Function block objects in a program          |
| LineList                    | CTypedPtrList | stored as a linked list                                    |
| ValueList                   | CTypedPtrList | A collection of Variable objects in a program stored as a    |
|                             |           | linked list                                                |

According to the model definition of the vision function block program, a complete function block program mainly includes 4 types of objects: program network, function block, connecting line and variable. The program network consists of a cell array of n * m, and is distributed with several program segments. Each program segment is composed of several vision function blocks connected to each other. Therefore, the attribute members of the class of the program network are shown in Table 1, which mainly includes information such as the size of the network, grid cell data, function block collection, connecting line collection, and variable collection. The data recorded by the grid unit is mainly index information of the function block object or the connecting line object displayed by the overlay grid unit, and is used to directly index to the function block object or the connecting line object displayed by the overlay through the coordinate information.

Function block class is a general basic class, suitable for storing all attribute information of any function block instantiated object, and its attribute members are shown in Table 2. The input and output interface elements are stored separately in the form of a linked list, and each linked list node corresponds to an interface element, which mainly contains information such as element name, input / output, data type, element visibility, parameter editing, connection information, and parameter value.
The members of the connecting line class mainly include input and output port linked lists and data types. The connecting line has one input end and multiple output ends, which are stored in the input port linked list and output port linked list of the connecting line class. The storage form of the linked list is convenient to add the output port of the Connecting line. The data structure of the linked list node includes function block object pointers, function block pin numbers, and visual path information.

### Table 2. Function block class structure

| Property members of the class | Data type      | Attribute description                      |
|-------------------------------|----------------|--------------------------------------------|
| FBName                        | CString        | Function block name                       |
| Row                           | int            | Row position of Function block             |
| Col                           | int            | Column position of Function block          |
| Cols                          | int            | Function block width                      |
| Rows                          | int            | Function block height                     |
| FBTType                       | CString        | Function block type                       |
| Inputs                        | CTypedPtrList  | Input interface of the Function block      |
| Outputs                       | CTypedPtrList  | Output interface of the Function block     |
| Fun                           | int (*Fun)(CTypedPtrList,) | Function of the Function block |
| Dialog                        | CCVBaseDialog* | Debug dialog                              |

The attribute members of the variable class mainly include the name, value, and data type of the variable. The basic data types supported by the programming platform are BOOL, INT, FLOAT, DOUBLE, and STRING. Data of complex types can be passed through the STRING type as an intermediary. By serializing data of complex types into strings or passing dynamically generated Variable addresses to transfer data. For example, the image data storage structure refers to the IPlImage structure in the OpenCV vision library. By passing the address of the IPlImage object, data sharing of image resources between instructions can be achieved.

The graphic interface of the visual function block program can be stored in serialized files through XML language, and finally generate the function block program description file which mainly saves the information of the function block network, Function block, function block connection and variables.

## 3. Compiling and running of Function block programs

The purpose of compilation is mainly to determine the execution order of Function blocks in the function block program. According to the distribution of Function blocks in the network, the execution order of function blocks conforms to the following scheduling strategy:

1. The Function block program is executed in units of program segment. The head function block in each program segment is the function block closest to the upper left without a predecessor function block. The execution order between program segments is determined by the position of the head function block: From top to bottom, from left to right.

2. The program segment uses the function block as the basic execution unit, and executes from the head function block. The connection relationship between function blocks determines that the predecessor function block should be executed before the subsequent function block. If there is no direct connection between the two function blocks without unexecuted predecessor function block, the position relationship between the two function blocks determines who executes first: from top to bottom and from left to right.

There are three steps to determining the execution order of function blocks in a program. The first step is to map the vision function block program to an AOV (Activity On Vertex Network) directed graph containing coordinate information. The second step is to convert the AOV graph into an adjacency list for storage. The third step is to perform a topological sort on the adjacency list to obtain the execution order of function blocks.

AOV graphs are directed graphs in which vertices represent activities and directed edges represent priority execution relationships among activities[14]. The vertices of the AOV graph are used to
represent the function blocks, and the directed edges are used to represent the connection relationship between the function blocks, so as to realize the mapping of the function block programs to the AOV graph. The problem of determining the execution order of the function blocks in the vision function block program is transformed into solving the topological ordering of the AOV diagram. Figure 3 shows the AOV map of a function block program. The numbers shown on the vertices of the AOV diagram in the figure indicate the coordinate position of the corresponding function block in the network view.

![Figure 3. AOV diagram of Function block program and adjacency list storage](image)

This paper uses adjacency list to store information in AOV graph. The adjacency list is composed of a vertex table and an edge table. The data structure of the vertex table node is composed of the in-degree field of the vertex, the edge table pointer, and the vertex field (the index of the function block object). The data structure of the edge table node is composed of the position information in the vertex table and the pointer of the edge table node. Figure 3 shows the storage structure of adjacency list of AOV graph. The vertex table of the adjacency list will be stored in the order of ascending value of the vertex table nodes transformed from the vertices of the AOV graph with row coordinates as the primary key and column coordinates as the secondary key.

The topology sorting process of the adjacency list requires the use of two stacks "Stack1" and "Stack2" for storing vertex table nodes and a one-dimensional vertex domain array "Array" storing the final execution order. Topological sorting steps are as follows:

Step1: Iterate through the nodes of the vertex table in the order from top to bottom, and take out the vertex table nodes with the degree domain 0 in order and push them on the "Stack2".

Step2: Pop the top element of "Stack2" out of the stack and push it on "Stack1" until there are no more elements in "Stack2".

Step3: Determine whether there are elements in "Stack1". If there are no elements in "Stack1", skip to Step4. If there are elements in "Stack1", pop the top element from the "Stack1", store the vertex field of the vertex table node at the end of the "Array", and reduce the in-degree field of the vertex table node corresponding to each edge table node in its side table, and then return to Step1.

Step4: Determine whether the vertex table is empty at this time. If it is not empty, it means that the compilation fails, and the wiring in the function block program forms a closed loop. If the vertex table is empty, it means that the compilation is successful and the topological sort is completed.
After the topology sorting is completed, the index of each function block object is stored in the "Array" in accordance with the execution order of the Function blocks, and an executable code file can be generated based on this data. The code file is composed of code instructions in a specific format, where the sequentially stored code instructions correspond to the functional blocks executed sequentially in the vision function block program, and the data structure of the code instructions consists of instruction execution number, Function block identifier, and Function block. It consists of position information and Function block parameter information. At compile time, a temporary variable is created for each connecting line object, because the data transmission represented by the connection relationship between instructions at execution time is also realized through variables.

The running environment executes the functions of the vision program by parsing the code file and sequentially calling the algorithm functions in the algorithm runtime library. The input parameters of the code instruction include energy flow information and data flow information. When the code instruction is executed, the energy flow information controls the execution of the instruction function. The data flow provides the operation data of the instruction execution. The execution result is fed back to the graphical interface through the position information stored in the intermediate code instruction.

4. Functional Verification of Vision Function block Programming Software

This article uses C++ language to develop the vision Function block programming software running on the Windows platform. The client interface is shown in Figure 4. It mainly includes four parts: navigation area, project management area, program editing area, and output area. Among them, the navigation area mainly has menus and toolbars to provide software-related functions. The project management area mainly displays the function blocks supported by the software in the form of a tree diagram. The program editing area is an area for function block program design, which is displayed as a uniformly distributed grid array. The output area is used to display relevant prompt information during program editing, compiling, and running.

![Figure 4. Programming Example](image)

The network in Figure 4 is a function block program developed by the vision programming software to identify the model code of the controller tag. The execution flow is shown in the figure. The number displayed in the upper left corner of the function block is the instruction execution sequence number. Among them, function block 1 "Pic_In" is an image input instruction to import image resources into the programming platform. Function block 2 "CV_Gray" is a grayscale conversion instruction to convert an image into a single-channel gray image. Function blocks 3, 5, 7, 10, 11 "Pic_Out" is an image output instruction that saves the processed image data to a local computer. Function block 4 "CV_Gaussian" is a Gaussian smoothing instruction that reduces the effect of random noise on the input image through
3*3 Gaussian filtering. Function block 6 "Adaptive_Threth" is an adaptive binarization instruction that converts a gray image into a binary black and white image. Function block 8 "CV_Not" is an inverse color instruction that performs inverse color processing on the input binary image. Function block 9 "Label_Recognition" is a sign recognition instruction to performs positioning of the label and character recognition on the pre-processed image, and finally obtains the model of the controller as PEC8000. Figure 5 shows the intermediate effect diagram of the original image processed by the overlay function block.

![Image](image_url)

**Figure 5. Intermediate effect diagram during image processing**

### 5. Summary

In this paper, the vision programming method based on graphic programmable technology is studied. The IEC61131-3 function block specification is used to classify and modularly package the vision algorithms and functions in the OpenCV vision library and establish the programming specification and instruction set for the vision function block program. The instruction model and program model of the vision function block program are designed, and the vision function block interface and the function block program are described by using XML language according to the model. By researching the function block program editing and compilation algorithms, programming and operating environment of the vision function block program are developed by using C plus plus language, and the functions of editing, compiling, and running the vision function block program are realized. This method of writing vision programs in a graphical manner is more in line with human thinking habits, simplifies the difficulty of developing vision programs, and allows developers to focus more on the design of vision requirements, which is of great significance to the development of machine vision technology.

### Acknowledgement

This research was financially supported by the National Key Research and Development Project (2018YFB1700204).

### References

[1] Luke Butters. Using Machine Vision to Command a 6-axis Robot Arm to Act on a Randomly Placed Zinc Die Cast Product[C]. Proceedings of The 2nd International Conference on Control and Computer Vision.2019:14-18.

[2] Zhu Z G, Yao C. Application of attitude tracking algorithm for face recognition based on OpenCV in the intelligent door lock[J]. Computer Communications, 2020.

[3] Chen Y F, Wu L N, Yue S L, Lin Q. Pose Measurement Base on Machine Vision for the Aircraft Model in Wire-driven Parallel Suspension System[J]. Energy Procedia, 2011, 13.

[4] Chen D X, Guo L, Nie J X. Design of Visual Automatic Programming Platform for Medical Images[J]. control engineering, 2010, 17(02): 256-259.

[5] Jean F, Gebali A, Beugeling T, et al. An Educational Visual Prototyping Environment for Real-Time Imaging[C]. 2012 Frontiers in Education Conference Proceedings. IEEE Comp Soc, 2012:1-6.
[6] Zhao D Z, Li W, Yang J Z. An XML-based process definition language for medical image understanding[C]. International Conference on Computer Application and System Modeling. IEEE, 2010:V1-679-V1-683.

[7] Konstantinides K, Rasure J R. The Khoros software development environment for image and signal processing[J]. IEEE transactions on image processing: a publication of the IEEE Signal Processing Society, 1994, 3(3): 243-252.

[8] Liu Z G, Li D, Wang S y, Zhang C h. Structure Design and Implementation of a Configuration System for Image Processing Application Development[J]. Computer Measurement & Control, 2016, 24(02): 307-310.

[9] Ren Y F, Wu X Y, Wang Y G. DESIGN OF VISUALISED INTERACTIVE COMPUTER VISION SYSTEM[J]. Computer Applications and Software, 2012, 29 (09): 171-174.

[10] Zhu G T, Lu Y. COM-based Design of the Image Manipulation System[J]. Computer Simulation, 2005 (04): 283-286.

[11] Peng Z L, Zhu C H, Peng J. Control of LabVIEW-based Visual Intelligent Robot Grasping Ping-Pong [J]. Electric Engineering, 2018 (18): 3-6.

[12] Hu L S. Design and implementation of configuration-based image processing development platform [D]. Central South University, 2012.

[13] Qin H, Xia Q G, Fu Y. PLC Programming Environment Based on IEC61131-3 Standard[J]. Computer Engineering, 2009, 35(23): 246-248 + 251.

[14] Tekli G, Chbeir R, Fayolle J. A visual programming language for XML manipulation[J]. Journal of Visual Languages and Computing, 2013, 24(2):110-135.