Chapter

Manipulating Complex Robot Behavior for Autonomous and Continuous Operations

Chengliang Liu, Liang Gong and Wei Zhang

Abstract

Service robot control faces challenges of dynamic environment and complex behavior, which mainly include eye-hand coordination and continuous operations. However, current programming scheme lacks the ability of managing such tasks. In this chapter, we propose a methodology of software development paradigm for the continuous operation of the dual-arm picking robot. First, a dual-arm robot is built for picking with the purpose of selectively harvesting in plant factory. Second, a hierarchical control software is framed by means of "Sense Plan Act" (SPA) paradigm. Third, based on the previous design, programming concept, and the ROS system, the sub-node programming of visual module, motion module, eye-hand coordination module, and task planning module are implemented with a state machine-based architecture. The experimental results show that if total number of targets within the visual field is not more than three, the average picking time is less than 35 s. The fluency of concurrent task management shows the feasibility of manipulating complex robot behavior for autonomous and continuous operations with the finite state machine model and task level architecture.

Keywords: dual-arm robot, complex behavior, continuous operation, robot operating system (ROS), finite state machine

1. Introduction

With the development of technologies such as industrial robots and computer image processing, a series of research and experiments on intelligent picking robots have been carried out in Japan and other related countries, such as tomato, apple, and grape picking robots [1]. Research on picking robots has focused on two parts: the first is a hardware device that can achieve rapid picking, that is, how to design a stable, efficient, and adaptable mechanical and visual sensing system, and the second is to design intelligent software for continuous operations, that is, to accurately identify, distinguish, and locate targets and to solve the problem of task planning and task scheduling during continuous picking operations.

In the research field of picking robots, Kondo et al. [2] designed a tomato picking robot using a 7-degree-of-freedom manipulator. It used fingers and pneumatic nozzles in conjunction with a color camera to complete the picking operation. The experiment achieved a picking success rate of about 70% [3]. Tanigaki et al. [4]

developed a cherry picking robot using a four-degree-of-freedom manipulator and a specially designed end effector with suction and shear functions. The visual part uses a light emitter, a photodetector, and a scanning device. The fruit was picked in 14 s, and the success rate was about 84%. The CROPS plan completed in 2014 was jointly completed by many European countries and units and aims to develop a modular picking robot system for different mission scenarios. The greenhouse bell pepper picking robot platform completed in the experiment in the Netherlands [5] used a 9-degree-of-freedom manipulator, two color CCD cameras and a depth-measuring camera, and its end effector was also equipped with a small camera to complete the picking with higher accuracy. Taqi et al. [6] have developed small household cherry tomato picking robots that can achieve very accurate picking tasks in specific environments.

The continuous operation of the picking robot can make intelligent decisions on multi-task under multi-objective scenarios and plan the operation according to the picking needs. The research goal is that the picking robot system can intelligently select and pick fruits that meet the picking conditions, thereby greatly improving the degree of automation of the picking process and improving the quality of the harvested fruits. Because the picking robot is still in the laboratory research stage, the research on continuous operation is also very limited, and it is basically in its infancy. Japan's Nagata et al. [7] used the shape to judge and classify strawberry quality. The accuracy in the experiment is acceptable, but the speed is much slower than artificial. Zhao et al. [8] investigated the visual recognition of apple maturity, using multi-spectral laser beams to complete fruit identification and positioning and ripeness judgment. Guo [9] and others judged strawberry maturity based on HIS color space algorithm. In the field test, the accuracy of the goal of picking fruits with maturity of 80% or more was more than 90%. Wang [10] and Ling et al. [11] carried out research on selective harvest information acquisition and path planning of tomato picking robots. Multi-sensor information fusion method was used for tomato quality detection and classification and selective picking decision. The appearance maturity of the fruit is detected by the H-means in the computer image, the fruits are classified in real time according to the agricultural industry standards, and the selective harvesting decision is made through the progressive identification of feature information and the fusion decision. At harvest time, path planning is performed on multiple targets through an optimization algorithm and then boxed by level.

This chapter mainly focuses on developing an intelligent software system for the continuous operation of the dual-arm picking robot in a plant factory. Typically, the semi-structure environment of the greenhouse poses challenges for autonomous operations of the robot, and the complex tasks mainly include identification and positioning under variable light conditions, selective picking in multi-cluster growth environment, and complex multi-task programming. For the difficulty of developing the software system, a hierarchical modular software system framework is designed. Moreover, a scheduling method of functional modules is designed based on the idea of finite state machine for the complex multi-task planning problem of tomato continuous picking process. The task scheduling design based on the Finite State Machine (FSM) reduces the difficulty of development work and improves the efficiency of development.

This chapter is organized as follows: Section 2 briefly describes the hardware structure of a dual-arm robot, and Section 3 presents the software framework with the highlights of deploying SMACH- and ViSP-based nodes in a ROS development environment. The experimental results are also included at the end of Section 3.

2. Hardware design of a dual-arm robot

The semi-structured operating environment in the plant factory is relatively complicated, such as the occlusion of fruit branches and leaves, the challenging grasping shape of the fruit, the changing light, and the variety of tasks, which requires to consider the multi-tasking ability of the robot when designing the hardware of the picking robot. At the same time, the complexity of the picking environment requires that the execution of the robot be robust to the environment in which the target is located, only in this way can it pick fruits in different states. Based on the above two design goals, we designed a dual-arm picking robot to simulate human picking operations.

2.1 Design of dual-arm robot body

The mechanical structure of the robot body mimics human arms, and the left and right arms each have three joints: a vertical lift joint, a boom rotation joint, and a forearm rotation joint. The vertical lifting joint is driven by a servo motor to drive the roller screw. The actual effective stroke is about 300 mm. The big and small arm joints are driven by a servo motor connected to a harmonic reducer. There is a waist joint between the body and the base, which can provide 360° rotary motion. The single-arm movement of the robot is similar to that of the SCARA robot. The vertical positioning is achieved by the lifting joint, and the rotation of the large and small arms realizes the positioning in the plane. There are three degrees of freedom in motion. Each of the left and right forearms is designed with a mounting flange surface, and end effectors can be installed as required. The dual-arm robot base is installed on a mobile cart and is transported along the track to different picking points for picking operations.

Due to the limitation of the freedom of the robot body, for complex tomato picking environments, it is not enough to rely only on the freedom of the arms, so we can use the design of the end effector to increase the freedom of our robot and enable the robot to flexibly complete the picking operation. We designed a shearing end effector on the cutting hand of the dual-arm robot and designed a suction-type end sleeve on the auxiliary hand to fix the target tomato and assist the hand in picking (**Figure 1**).

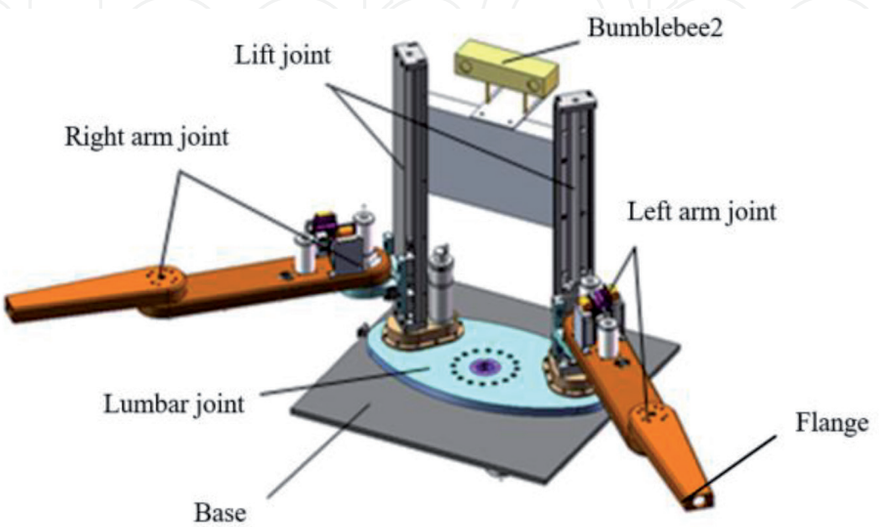

Figure 1.
The structure of the robot body.

2.2 Dual-arm robot coordinate system

As shown in **Figure 2**, the world coordinate system (x_w, y_w, z_w) of the dual-arm robot is built on the waist and coincides with the waist joint coordinate system. The left and right arm coordinate systems (x_l, y_l, z_l) and (x_r, y_r, z_r) use the right-hand principle, and the span direction at the zero position of the x axis is different from the x axis of the world coordinate system by +45° and −45°. The z-axis direction is vertically upward and is at the zero position at the lowest point. The x–y plane is parallel to the world coordinate system x–y plane. The binocular camera coordinate system is established at the intersection of the right-eye visual axis and the camera lens. The coordinate system adopted by the bumblebee2 camera is the left-hand principle. In order to be compatible with the entire system, the y-axis direction is reversed.

We use the tf function package [12] provided by ROS to maintain the transformation relationship between coordinate systems. After the robot model is built according to urdf, the system will automatically broadcast the transformation relationships between all the coordinate systems. The picking robot coordinate system is shown in **Figure 3**.

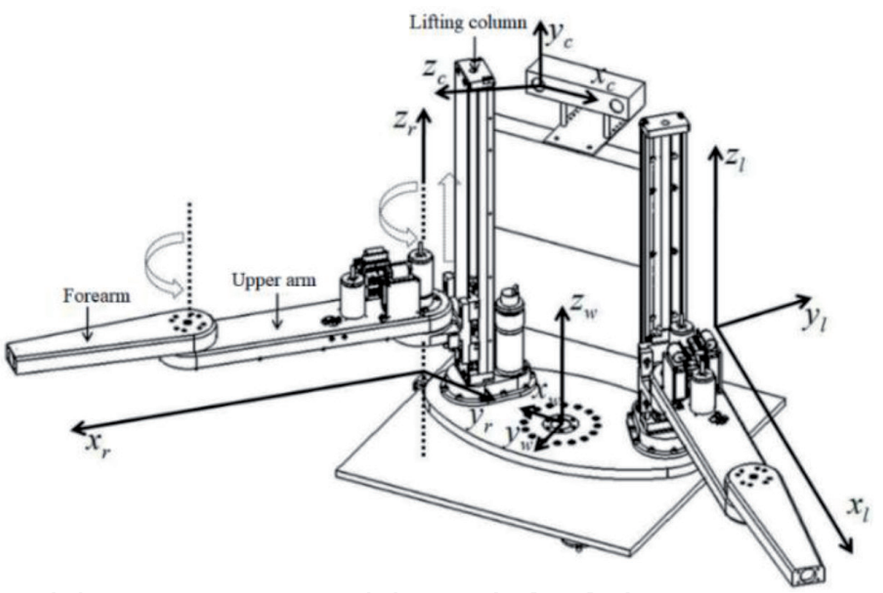

Figure 2.
Robot coordinate system.

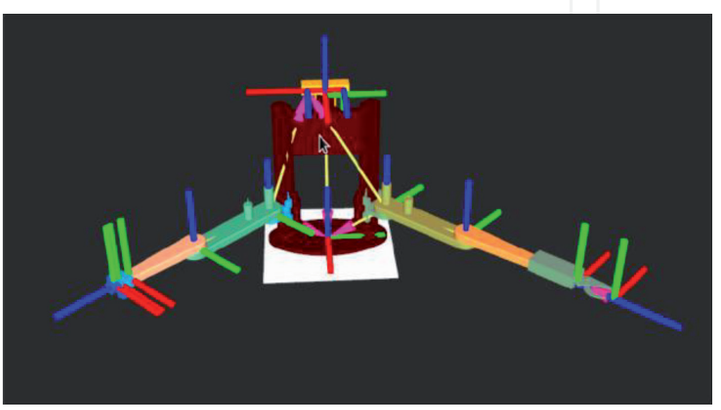

Figure 3.
The picking robot tf coordinate system under Rivz.

Based on the above coordinate system, using the *rosrun tf view_frames* command, we can view the tf tree of the dual-arm picking robot, as shown in **Figure 4**.

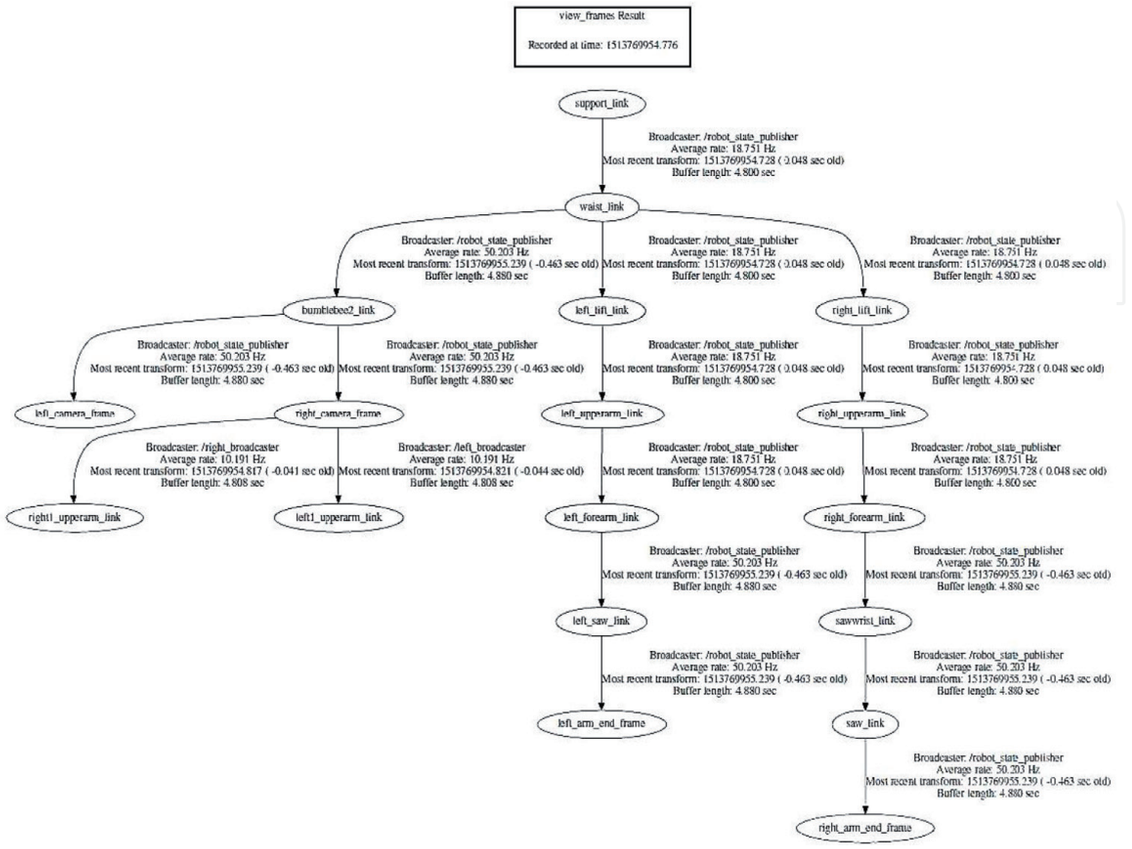

Figure 4.
The tf tree of robot.

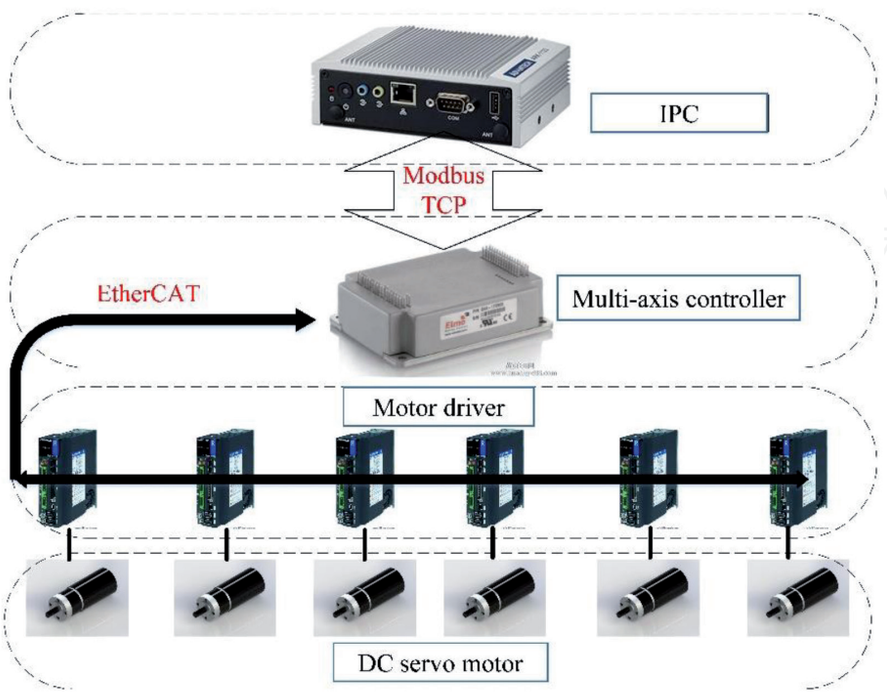

Figure 5.
Picking robot communication architecture.

2.3 Communication architecture of the dual-arm picking robot

The communication architecture of the dual-arm picking robot is shown in **Figure 5**. The motor driver is a Gold series motor driver produced by Elmo and is equipped with the same series of multi-axis controllers. The manufacturer has provided a complete motor driver to multi-axis controller communication protocol and communication protocol implementation and does not require customers to conduct secondary development. The multi-axis controller uses Modbus TCP communication as the communication between the lower computer and the industrial computer. It is connected to it through an Ethernet cable.

3. Software frame design

In the process of programming, we generally manually divide the tasks of the robot. Once the tasks are effectively divided, we can stack the tasks with the smallest functional components, so the difficulty of task division is how to use human prior knowledge to divide the minimum granularity of robot skills, which is a learning process. At the same time, how to easily and effectively combine the divided components is also an important part of completing the task simulation.

The system architecture describes the functional structure of the subdivision and the topological relationship between them and a series of specifications that need to be set for subsequent development. The basic requirements of software engineering include modularity, code reuse, and function sharing. Using a common framework is helpful for decomposing development tasks and code migration. Robot software also follows the general rules of software engineering. Architecture is how you break up the robot's functions and organize your code. A clear architecture that matches the project directly determines your development efficiency and even the success or failure of the final function. There are two main approaches to robot system architecture: SPA architecture and behavior-based architecture.

3.1 SPA

The software system architecture is "Sense Plan Act"(SPA), as shown in **Figure 6**. The robot maps the external environment space through sensors and uses a certain modeling method to structure and model the perception information and then analyzes the model to plan the robot's actions. Finally, the action instructions are executed in the environment to achieve a complete interactive process.

The typical software architecture in a SPA robot system is a three-layer architecture: the perception layer, the planning layer, and the motion control layer, as shown in **Figure 7**. The perception layer receives and processes the sensor data, the planning layer plans the motion trajectory, and the motion control layer ensures the accurate execution of the movement. The SPA robot software system architecture

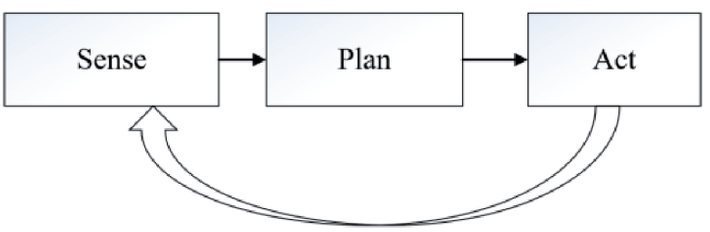

Figure 6.
SPA work pattern.

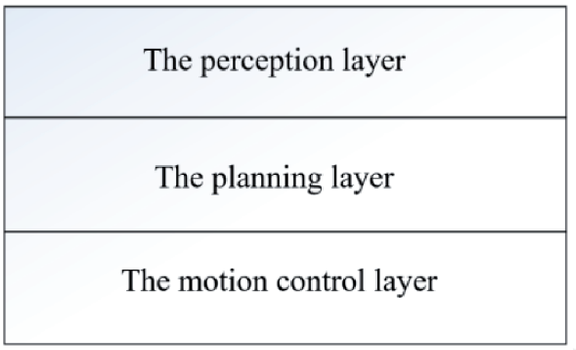

Figure 7.
SPA layered design.

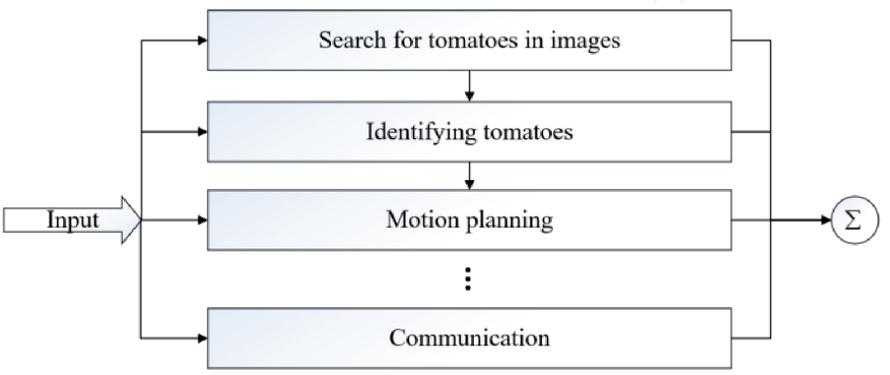

Figure 8.
Action-based software system architecture.

pays more attention to the perception and modeling of the world because this is the basis for the accuracy of subsequent planning and movement.

3.2 Behavior-based architecture

The behavior-based software system architecture is a top-down software design. The small functions of each robot are packaged into individual small modules. All functional modules can be executed in parallel without prioritization. A robot task can be understood as an organic composition of functional modules (**Figure 8**).

To a certain extent, all robot actions are responses to stimuli (inputs). This stress mode avoids the thinking logic in the SPA architecture and facilitates the rapid action response. In order to achieve the task, we can design a control scheme to change the stress level of the action. Therefore, we need a global controller to coordinate the choice of actions in order to achieve our intended purpose. The behavior-based software design framework has good flexibility, but it increases the difficulty of control. When multiple actions can affect the output, problems are easy to occur.

Therefore, combined with the SPA software architecture and behavior-based software architecture, we design a software framework that combines the advantages of both architectures for continuous operation of a dual-arm picking robot. Its characteristics are as follows:

1. Hierarchical modular design: The software architecture absorbs the advantages of the SPA architecture and also adopts a hierarchical design. The layered design is mainly logical, which makes it easier for users to understand the working mode of the robot. At the same time, it also absorbs the advantages of behavior-based architecture, that is, functional modularity. Based on the

analysis and understanding of the robot's internal architecture, we divide the functions of the robot into seven modules, each of which is functionally independent of each other. Combining the above two is a hierarchical modular design, which divides the functional modules into a specific layer according to the attributes of the functions, thus strengthening the logic of the system.

2. Finite state machine control: In order to solve the shortcomings of behavior-based architecture, we have designed a task planning module based on finite state machines, which is used to schedule and control the execution order of each functional module to complete a specific task.

3.3 Hierarchical modular design

Layering is an important concept in software design. The division of layers provides a framework for business decomposition and simplifies many thinking processes. Considering the design characteristics of the software system and the functional features of the robot, the entire software architecture can be divided into four layers: presentation layer, application layer, sense layer, and data layer. The hierarchical modular design architecture is shown in **Figure 9**.

1. Presentation layer: Presentation layer has more business logic requirements. We designed the presentation layer based on the QT architecture. The entire presentation layer includes several main components as shown in **Figure 10**: the RVIZ module displays the model of the robot and other visual information; the image module displays the video image information collected by the current robot; the node module monitors all current node information; and the DashBoard module provides users with a function module for manually operating the robot; shell module provides command line functions; console module displays all log information executed by the system; reconfigure module provides users with a convenient tool for changing model parameters; and diagnostic module provides real-time robot monitoring information.

2. Application layer: Application layer focuses on the task execution of a single robot. It is separated from the implementation of specific functions and uses a combination of function modules to coordinate a task.

3. Sense layer: Sense layer is responsible for the interaction between the software system and the hardware. There are both a visual module responsible for

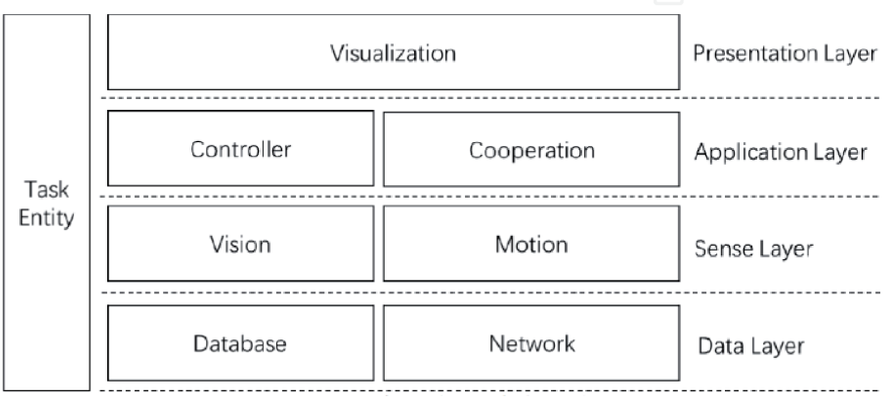

Figure 9.
Layered modular architecture design.

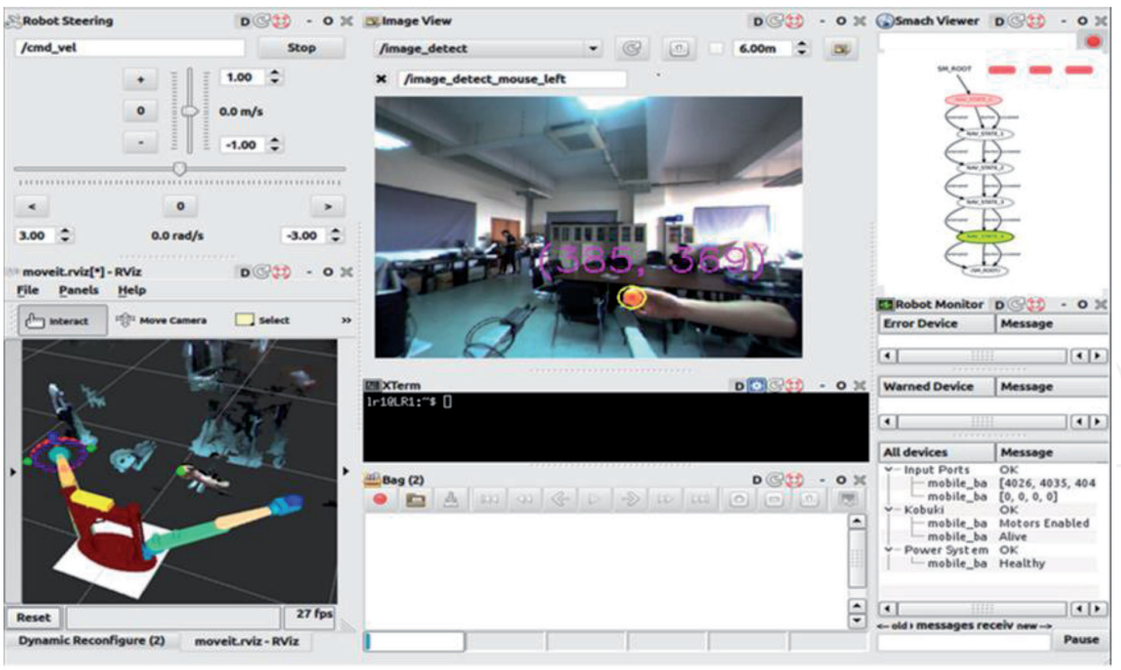

Figure 10.
Presentation layer.

environmental perception and a motion module responsible for the motion control of the upper and lower computers. Sense layer is a description of the robot's capabilities.

4. Data layer: Data layer serves the data generated by the system. Part of the data generated by the system is stored in a local database for real-time decision making of the system. One part is uploaded to the server through the network and is fused with data from other robots and other time dimensions to plan the continuous operation of the robot.

The criteria for the division of functional modules are to reduce coupling, relatively independent functions, and high code repeatability. According to the robot's task module, the software system can be divided into motion (*motion_pkg*), control (*control_pkg*), vision (*vision_pkg*), visualization (*visualization_pkg*), collaboration (*coorperation_pkg*), database (*database_pkg*), and network (*network_pkg*). Each function module is represented as a function package at the file system layer. There can be multiple nodes in a package, and different nodes can be written in different programming languages.

3.4 Cooperative control of dual-arm picking based on FSM

3.4.1 SMACH

Finite state machine (FSM) is a mathematical model of computational science. The objects it represents can be in a limited number of states at any one time. When an external input occurs, the system responds to the external input, and the FSM can conditionally transition from one state to another. This process is called an action. In computer science, finite state machines are widely used for modeling application behavior, hardware circuit system design, software engineering, compilers, network protocols, and computation and language research. FSM can be defined by the present state, condition, action, and substate. The specific interpretation is as follows:

1. Present state: The current state.

2. Condition: The premise of triggering an action can also be considered as an event. When a condition is filled, an action will be triggered.

3. Action: The operation performed when the conditions are met and can be regarded as a unit of calculation or transaction processing. After the action is completed, it can be transferred to a new state, it can still be in the original state, or it can be terminated.

4. Substate: The state after the present state transition. When different actions occur and different conditions are generated, a state may transition to a different substate. Once the transition is completed, it becomes the present state.

As shown in **Figure 11**, a task can be represented by a state transition diagram.

SMACH [13–15], which refers to "State Machine," is a powerful and scalable Python-based library for hierarchical state machines. The SMACH library does not depend on ROS and can be used in any Python project. The *executive_smach stack*, however, provides very nice integration with ROS, including smooth *actionlib* integration and a powerful SMACH viewer to visualize and introspect state machines. The SMACH core library is lightweight and mainly provides two interfaces: State and Container.

State: The state represents the state being executed. Each state has some potential outputs. The State class outputs the result by implementing the blocking function *execute()*.

Container: The container is a collection of one or more states that enforces some strategy. The simplest container is a State machine. A SMACH state machine can be viewed as a state flow graph, where each node is an execution state (the robot is performing a certain action), and the edges connecting the nodes represent transitions between states. The State machine itself can also be regarded as a state and has its own output, so they can be combined in layers to complete a complex task.

Figure 12 shows an example state machine [16].

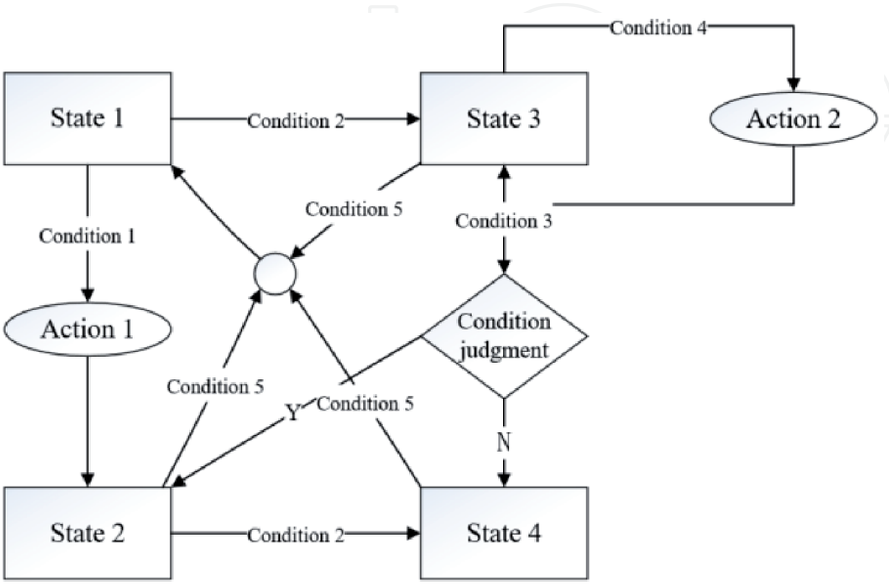

Figure 11.
Finite state machine.

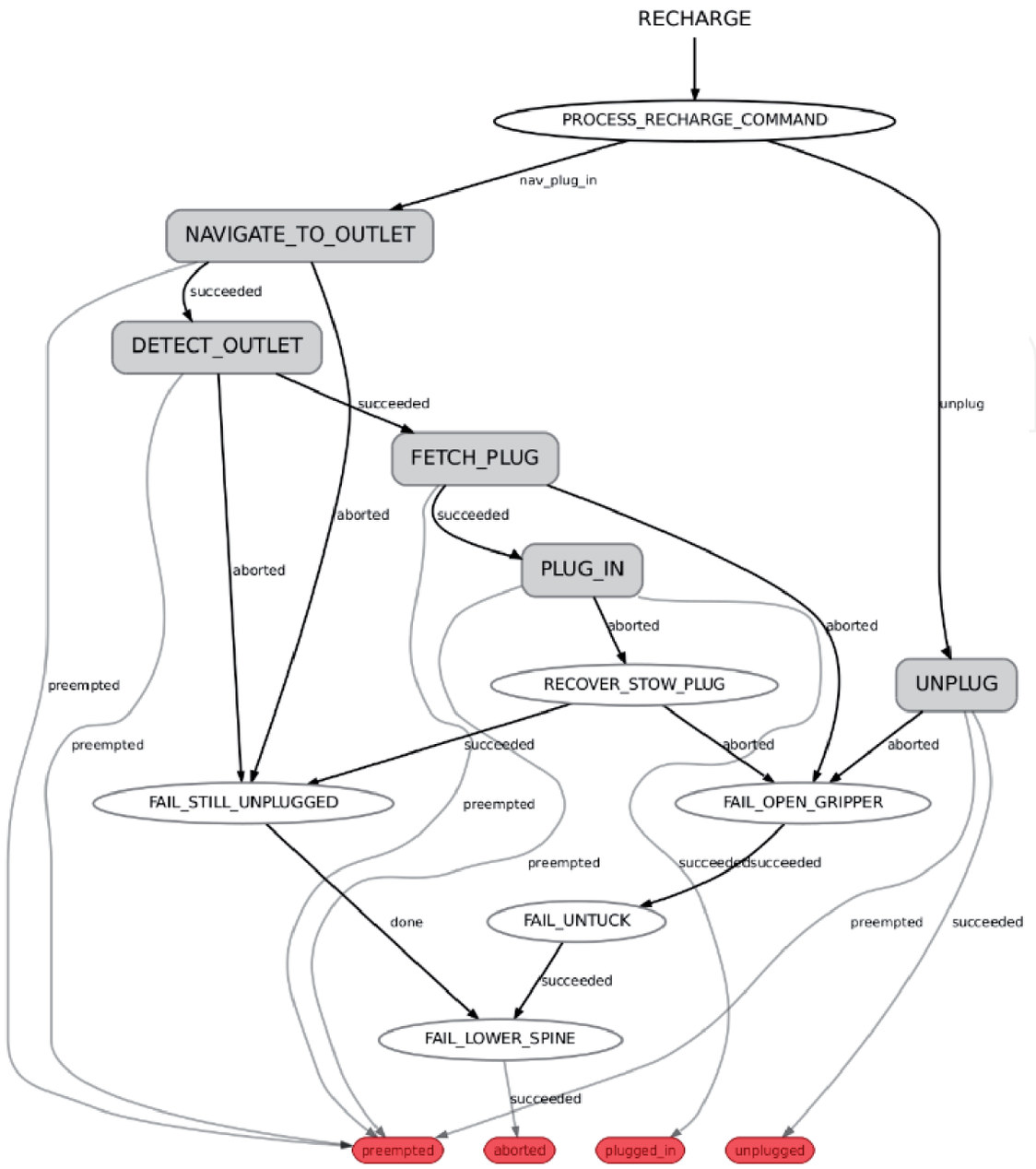

Figure 12.
SMACH state machine example.

SMACH uses action files to define communication protocols between different states. The structure of the action file is simple and clear, as shown in **Figure 13**. Three data definition areas are separated by three underscores. The first area defines the message format of the request, and the middle area defines the returned result (result message format, the bottom area defines the intermediate information feedback) message format. Each area can contain multiple data type, and the system will automatically compile the action file into three message files during the compilation process, so the message format for communication between states is actually the message format provided by ROS.

SMACH provides a general state type to support invocation while providing a special state class *SimpleActionState* as a proxy for *actionlib*. During the construction of the *SimpleActionState* object, the corresponding *actionlib* client is started by default. The user can define a goal in the constructor and create a callback function to process the data returned by the *actionlib* server.

```
# Define the goal
trajectory_msgs/JointTrajectoryPoint target_joint
bool arm
---
# Define the result
int32 succeed
---
# Define a feedback message
int32 percent_complete
```

Figure 13.
File format of action.

3.4.2 Hierarchical concurrent state machine design

For the picking robot to perform a continuous picking task, we can use the state transition process to describe it. The detailed description of the state transition is as follows: first enter the startup state, start the picking robot platform, wait for the initialization of each component, and perform a startup self-test. If any component fails to initialize and is in a fault state, the startup fails, and then the system is placed in the error state, stopping working and the task ends.

If the startup is successful, the platform moves into the state, the mobile platform moves to the first picking point, and data collection state and the tomato scanning state are started at the same time. The data collection state collects tomato information in the current status and uploads it to the database of the server; the tomato scan state checks whether there are tomatoes suitable for picking in the viewing area. If not, restart the platform moving state and move the robot to the next picking point; if so, first analyze all the tomato position information and pass the spatial attitude information of the first target tomato to the kinematics solution state according to the predetermined rules, and then program performs kinematics calculation and motion planning. If the tomato is unreachable, the information of the next target tomato is passed to the kinematics solution state, and so on until the last tomato is reached. If it is determined that the tomato is unreachable, the platform is moved to the next picking point. If it is judged that the tomato is reachable, the calculated right arm motion information is transmitted to the robot motion state, and this state sends the trajectory information of the right arm motion to the lower computer and simultaneously detects the joint motion position information during the execution of the lower computer. After the right arm moves to the target position, it enters the suction state. The system starts the suction device to fix the tomatoes and move them to a suitable position, which is convenient for the left arm to cut hands. The kinematics solution state is started again, and the left arm motion information is solved and transmitted to the robot motion state to control the left arm to move near the target. The start of the visual servoing state is close to the target tomato precisely, and the cutting state is started after the arrival, the pneumatic shear transposition of the left arm is started, and the tomato is cut. After completing a tomato pick, pick the next goal planned. Repeat until the last picking point.

According to the task execution process and state transition process described above, the designed state transition diagram is shown in **Figure 14**.

In SMACH, we use *SimpleActionState* to directly simulate the server side of *actionlib* and define a state machine with 10 states to control the robot to complete a comprehensive picking job.

1. DO_START: start state.

2. DO_MOVEBASE: mobile platform mobile status.

3. DO_TOMATO_SCAN: tomato scanning status.

4. DO_SPATIAL_TEMPORAL: data collection status.

5. DO_KINEMATICS: state of kinematics solution.

6. DO_MOVE_ROBOT: left and right arm movement status.

7. DO_MOVE_ENDEFFECTOR: end effector status.

8. DO_ERROR: fault status.

9. DO_STOP: emergency stop status.

10. DO_VISUALSERVOING: visual servoing status.

3.5 Major software node design

Based on the software framework design and operation requirements of the dual-arm picking robot, based on the functional division of each part, a vision module, an eye-hand coordination module, and a task planning module are mainly designed and installed in the industrial computer. Each module further refines

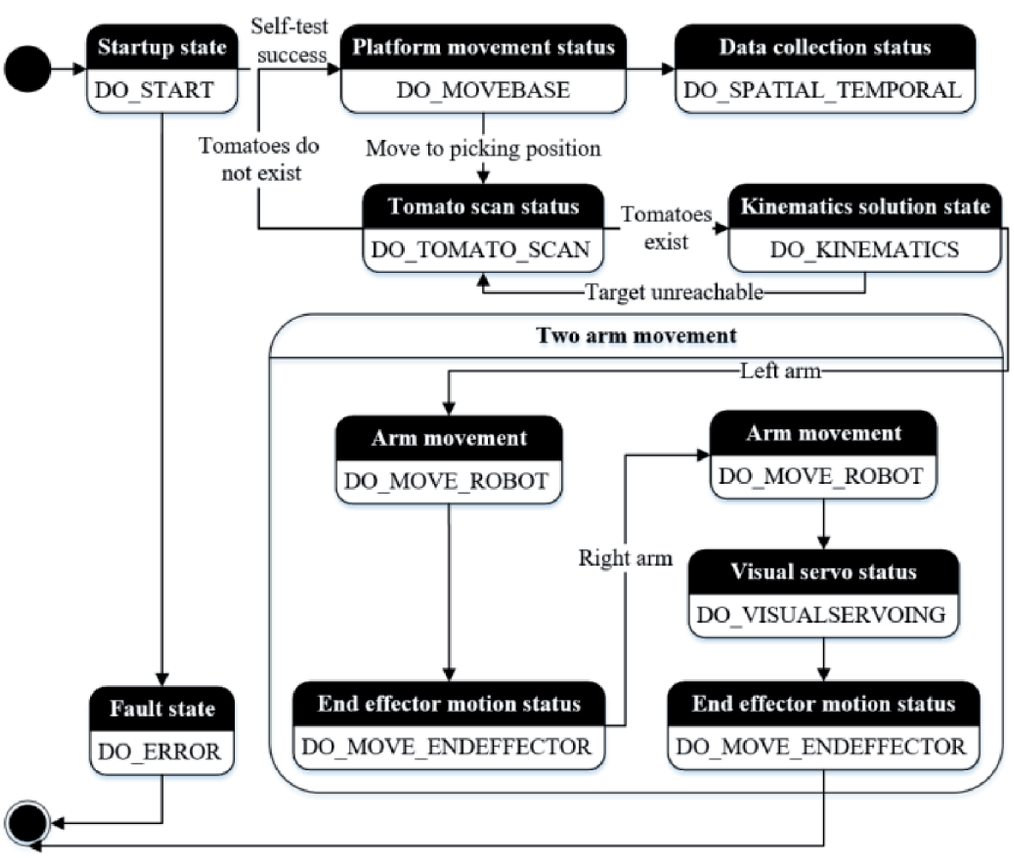

Figure 14.
Continuous picking status flow.

the functions and can be divided into functional nodes. Nodes are the minimum functional modules in ROS, which regard as the ultimate goal of the design.

3.5.1 Vision module node

In the dual-arm acquisition robot eye-hand system, the main component of the "eye" is the camera, including a binocular camera mounted on the robot's head and a monocular camera mounted on the arm. Based on the ROS framework, we designed three nodes to complete the environment perception function: binocular camera image acquisition node (*dual_eye_image_capture*), monocular camera image acquisition node (*single_eye_image_capture*), and image processing node (*image_processing*). The actual recognition effect is shown in **Figure 15**. There are three valid tomatoes in the image. The system recognizes all tomatoes and marks the positions of the tomatoes in the picture that need to be picked first according to the rules.

The binocular camera acquisition node uses the two original images collected by the left and right sensors of the Bumblebee2 camera to finally generate five images: left and right eye corrected color images, left and right eye corrected gray images, and 3D point clouds (**Figure 16**).

The collection process of the binocular camera is shown in **Figure 17**. The camera's original data are read, and the data are packaged into a Bell template image; then, three color information is extracted from the Bell template image and assembled into the original color image. The eye image data are used to obtain corrected left and right eye color images and grayscale images. Next, the left and right eye images are used for stereo matching through the principle of triangulation to generate a 3D point cloud. In the end, all the five images generated were published, and the algorithm used in the image acquisition process was provided by the camera SDK.

Based on the above process, we designed the binocular collection program UML as shown in **Figure 18**.

3.5.1.1 Monocular camera image acquisition node

For monocular vision, we use a Daheng Mercury series industrial camera MER-500-7UC, which uses USB2.0 digital interface and provides free SDK and secondary

Figure 15.
Tomato identification interface.

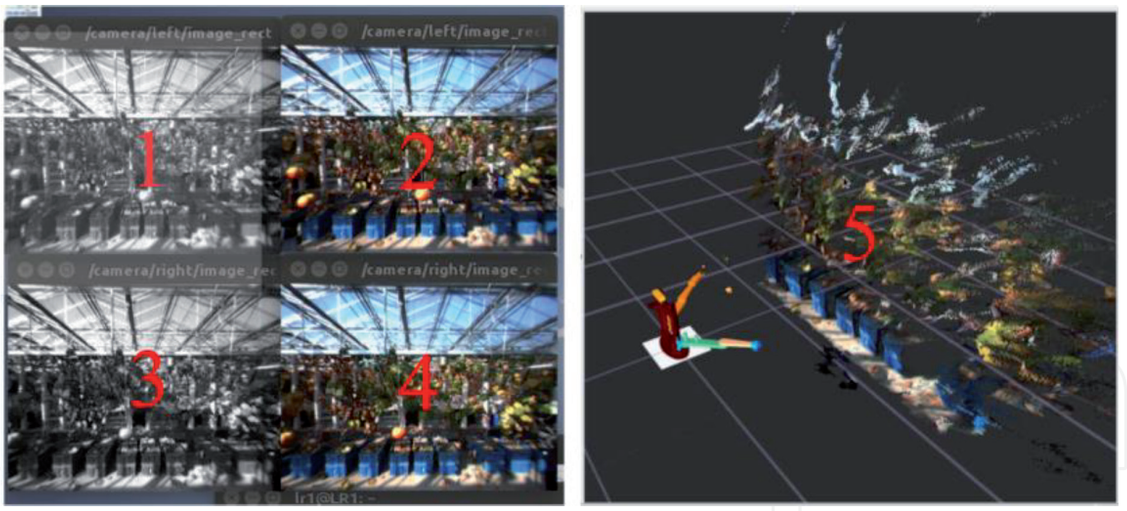

Figure 16.
Five pictures generated by Bumblebee2 camera. 1. Gray image of left eye after correction. 2. Color image of left eye after correction. 3. Gray image of right eye after correction. 4. Color image of right eye after correction. 5. Point cloud image.

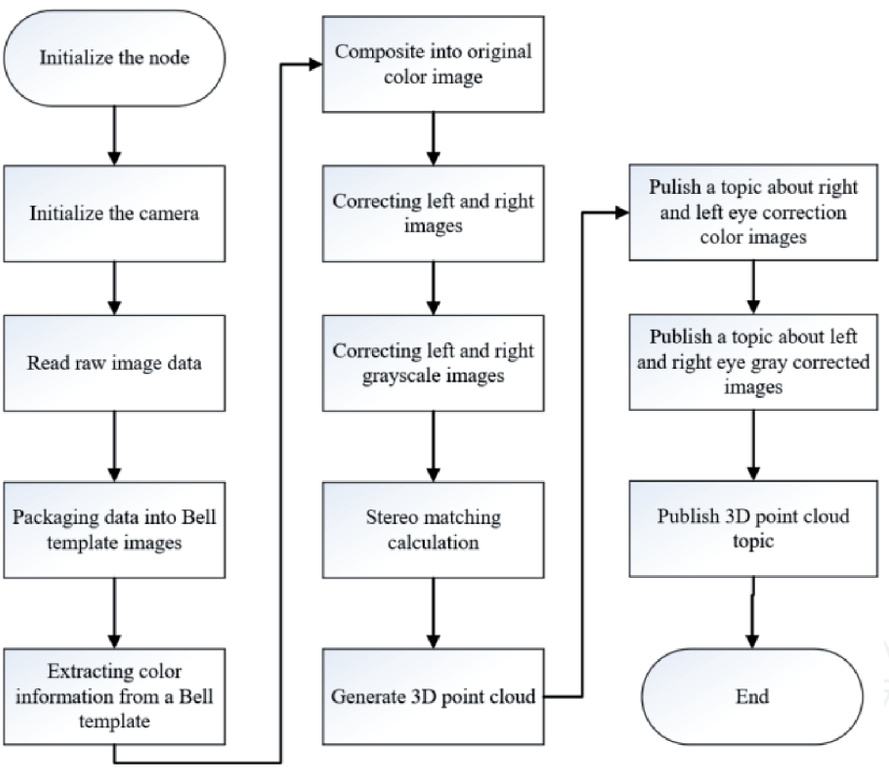

Figure 17.
Binocular image acquisition flow chart.

development example source code under windows platform and Linux platform. We use the *usb_camera* package provided by ROS to collect monocular images, as shown in **Figure 19**.

3.5.1.2 Image processing node

The image processing node receives the collected planar image and point cloud image and provides different image processing function interfaces according to different business requirements. In the current task requirements, image processing nodes are required to complete the accurate two-dimensional recognition

Figure 18.
Binocular acquisition node UML design.

Figure 19.
Image captured by monocular camera.

and accurate three-dimensional positioning of tomatoes. Therefore, the two-dimensional image dataset of the scene and the three-dimensional point cloud data are also required. Next, we introduce us from two directions. Image processing node design: first is the architecture design and functional flow of the image processing node as a functional interface, and the second is the specific implementation of related image processing algorithms.

The entire software system is based on the C/S model architecture, using the *actionlib* function package provided by ROS, with the task planning node as the server, and requesting computing resources from the client of each functional unit. Image processing nodes are no exception. After receiving the image processing instructions and image data, the instructions are parsed to clarify the functional requirements, and then the required image data are extracted, input into the algorithm function for processing, and the results are finally returned to the server.

The specific processing flow is shown in **Figure 20**. After initializing the node and *actionlib* server, start the service, wait for the goal sent by the client, and subscribe to the processing function. After receiving the instruction, analyze the source of the instruction. If the instruction originates from the spatial positioning of tomatoes, the processing steps are: first, use the tomato recognition algorithm based on image feature fusion to identify all tomatoes in the right eye image space of the binocular camera. If there are no tomatoes, return the results; if tomatoes

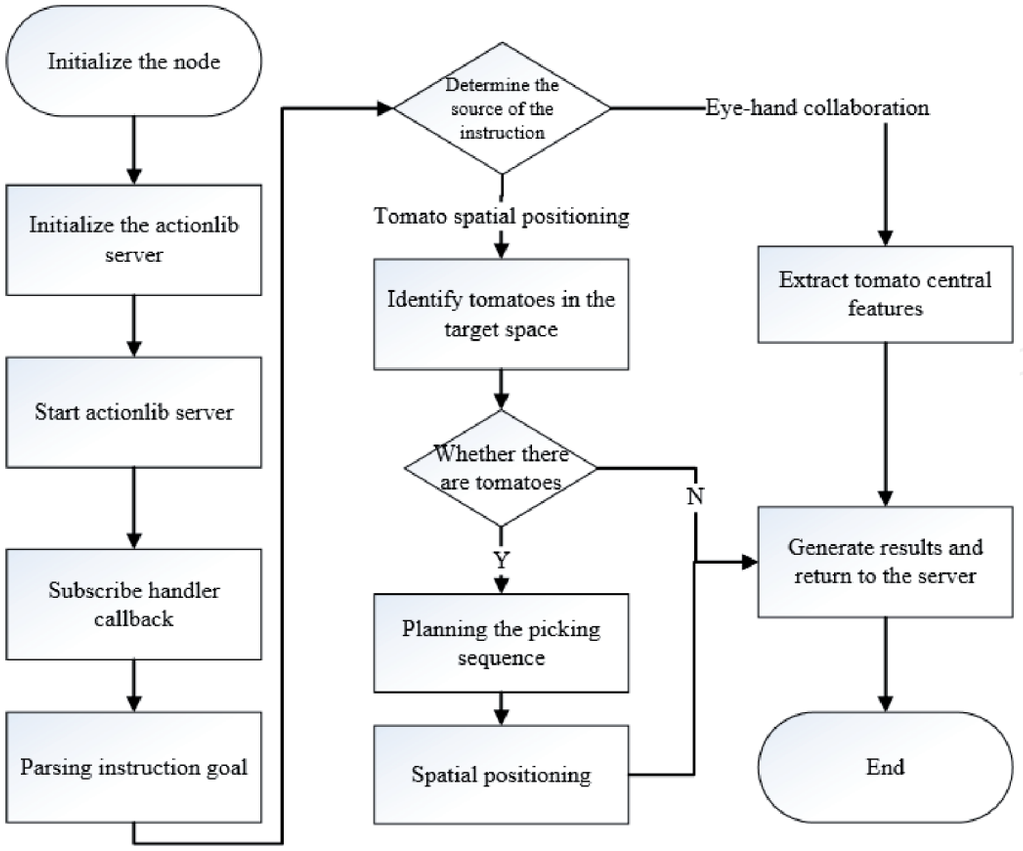

Figure 20.
Image processing node flow chart.

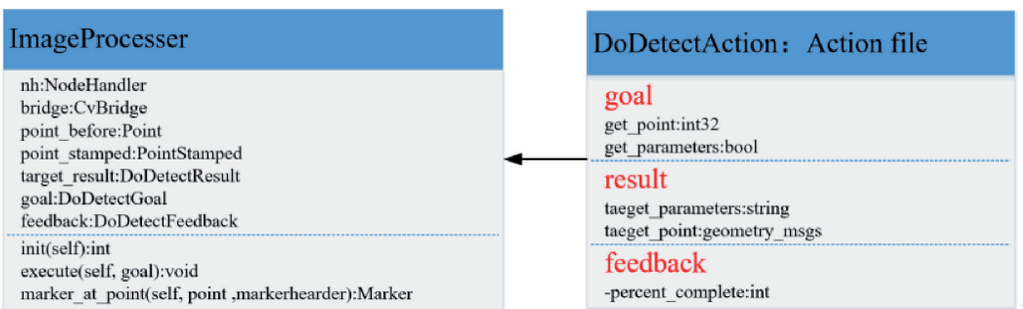

Figure 21.
Image processing node UML design.

are detected, plan the picking order. The rule is from bottom to top, left to right, and calculate the spatial position of the pick point, and finally return the result to the client. If the object recognition result triggers the harvesting task, the image collected by the monocular camera is used to extract the central image feature of the tomato.

According to the above process, the design program UML is shown in **Figure 21**.

3.5.2 Eye-hand coordination module node

3.5.2.1 Eye-hand collaboration process design

Our solution uses an eye-in-hand vision servo solution to achieve eye-hand coordination, as shown in **Figure 22**. The picking robot obtains the image information of the target fruit through a monocular camera installed on the picking hand,

extracts the position information of the tomato features in the two-dimensional image, and makes a difference from the expected position information. The difference is used as the input of the visual servo control algorithm and then calculate the control output in real time, that is, the speed vector of the end effector, and then integrate this speed vector with time to calculate the next point that needs to reach the target position. Cycle back and forth to get a trajectory that gradually approaches the target position. The eye-hand correspondence is converted into the amount of motion of the joint, and the end of the robot arm moves accordingly to approach the target. The implementation process is shown in **Figure 23.**

3.5.2.2 ViSP

ViSP [17] is an open source visual servo framework developed and maintained by the Lagadic team of the French National Institute of Information and Automation. It has the characteristics of hardware independence, scalability, and portability. In addition, ViSP also provides a complete library of basic functions, which can be combined with a variety of visual feature libraries; it also provides a simulation environment and interfaces with various hardware. Based on ViSP, we

Figure 22.
Visual servo program of eye-in-hand.

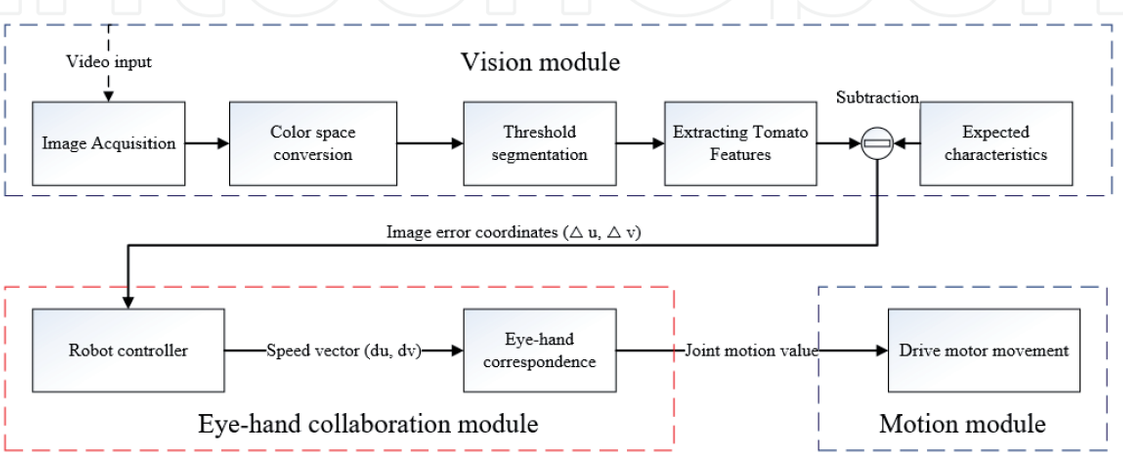

Figure 23.
Eye-hand coordination process.

can complete functions such as visual tracking, fiducial marking, two-dimensional contour tracking, pose estimation, and so on. The goal of ViSP is to provide developers with a tool for rapid development of visual servo functions. The software framework of ViSP is shown in **Figure 24**. The entire framework is divided into three modules: one module provides vision models, vision servo control algorithms, and robot controller interfaces; the second module provides image processing algorithms, tracking algorithms and other machine vision algorithms; and the last module is a visualization module that provides a simulation and visual environment. All these features make ViSP very suitable for use as a core part of our module.

3.5.2.3 Eye-hand collaboration module node design

The complete flowchart of eye-hand coordination is shown in **Figure 25**. After the node is initialized, the system initializes and starts the '*/visual_servo*' *actionlib* service and subscribes to *execute()* to wait for the client to be awakened. After receiving the service request, start the visual servo loop. In the loop, program request the feature position of the tomato image from the vision module and make a difference from the expected position. If the difference exceeds the threshold Δs (Δs=2mm), the program will obtain the camera parameters, initialize the control model, and call the ViSP library function *vpServo()* to calculate the control output speed vector. Then, program integrates the velocity vector with time (t = 1s), motion module controls robot to move to the output position, and requests the tomato image feature position from the vision module again, then makes a difference with the desired position, and loops back and forth until the target image feature. The difference between the position and the desired image feature position is less than the threshold Δs, the visual servo loop is ended, and our execution result is returned.

Figure 26 shows the design of the eye-hand coordination node class. There are mainly two classes. The *VisualServoCycleNode* class is responsible for the loop and interaction with other modules. The *VisualServoControlNode* module is responsible for controlling the operation of the algorithm.

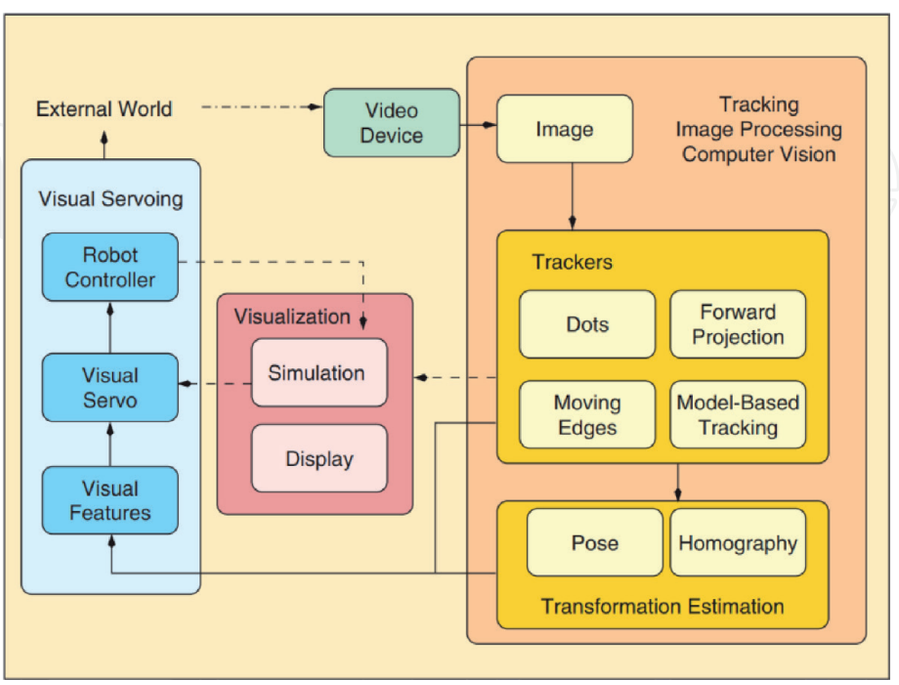

Figure 24.
ViSP software architecture.

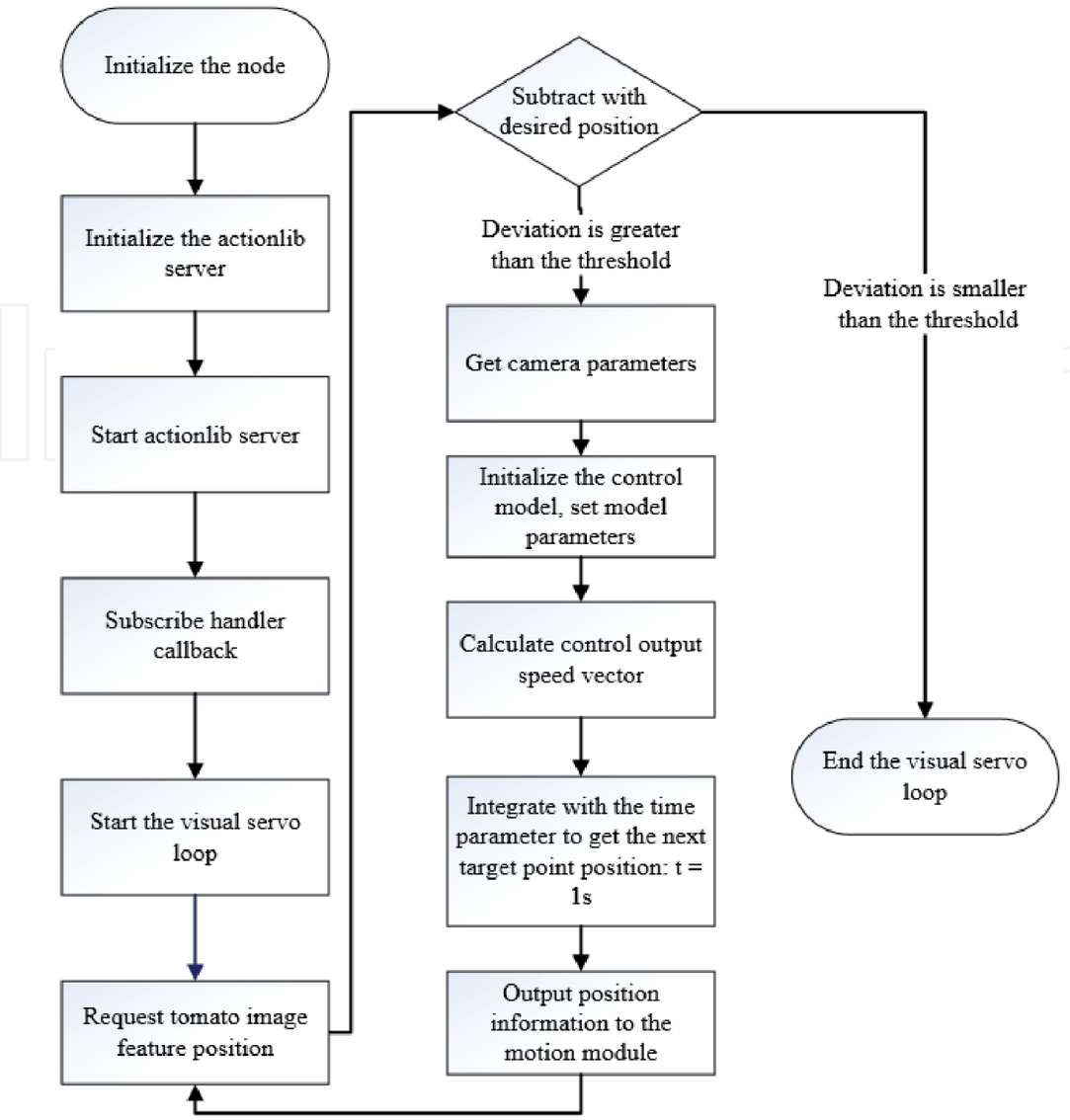

Figure 25.
Eye-hand collaboration node flow chart.

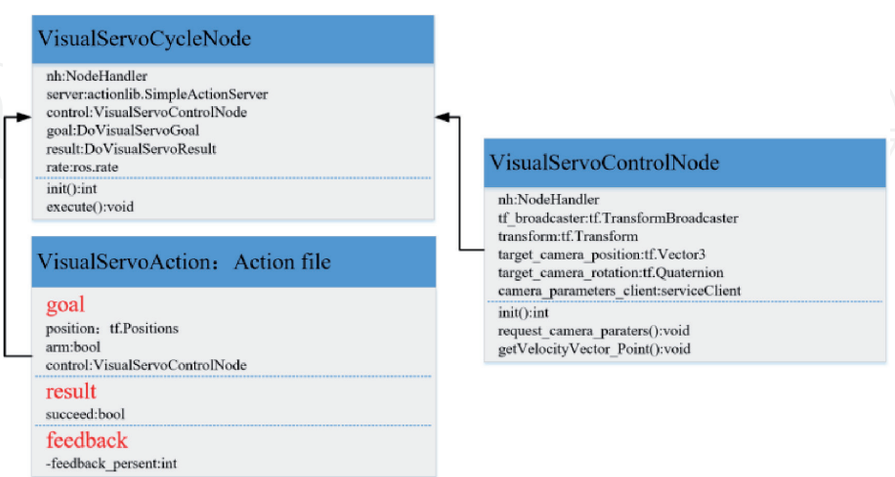

Figure 26.
Eye-hand coordination node class design UML diagram.

3.5.3 Task planning module node

The task planning module mainly completes the design and implementation of a layered concurrent state machine for one pick, as shown in **Figure 27**:

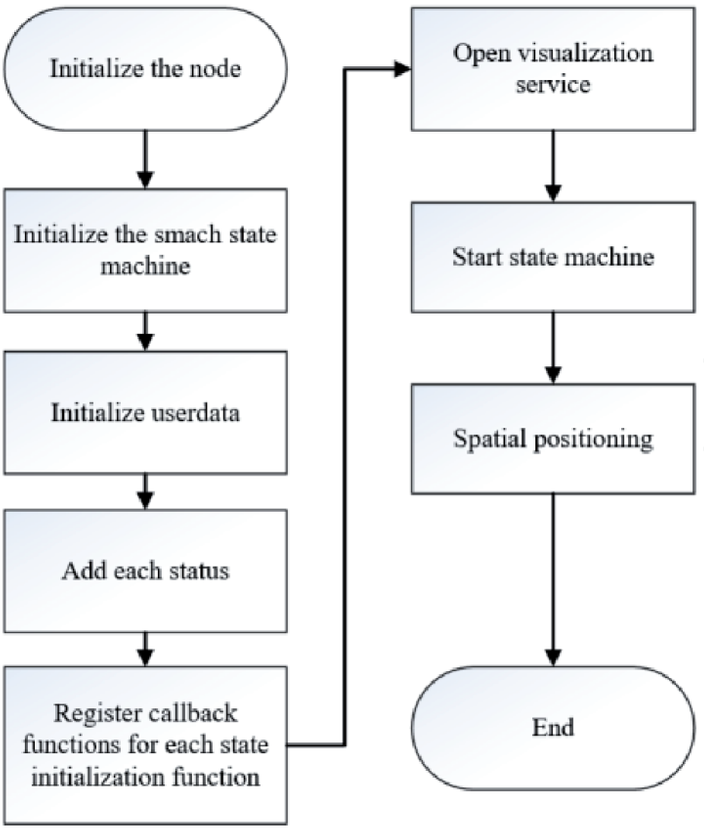

Figure 27.
Task planning node flow chart.

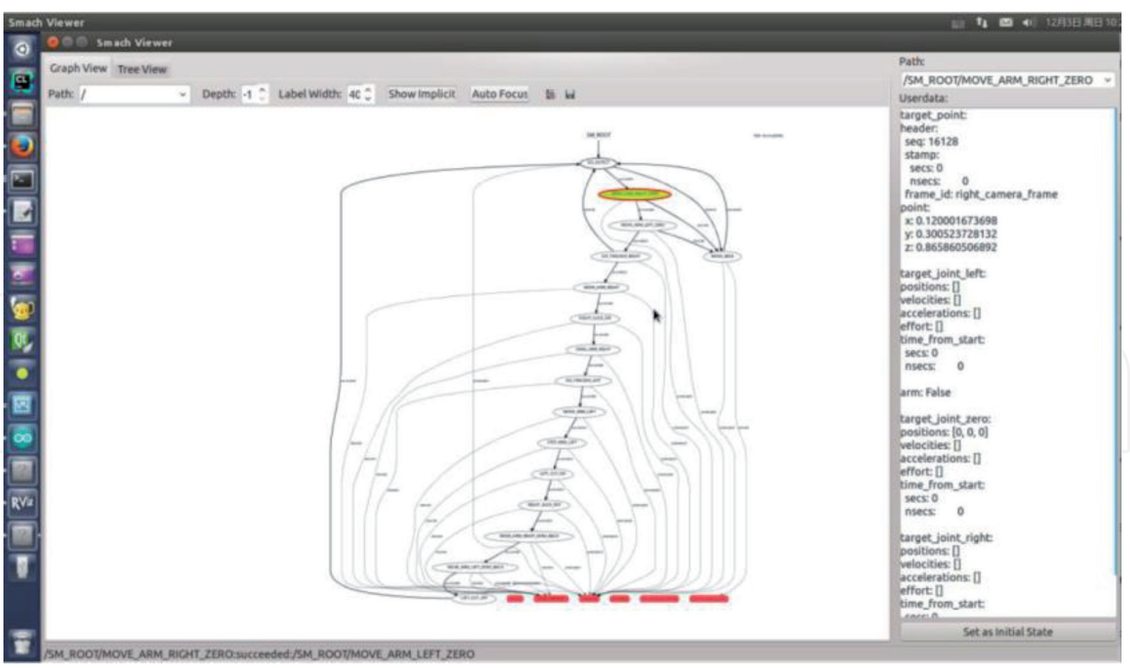

Figure 28.
FSM in SMACH_viewer.

First we initialize the node, state machine, and user intermediate data, and then add the transformation relationship between the states of each state machine according to the state transition of the task design. Use the transition keyword to control the transition from the current state to the secondary state. At the same time, since each state is *SimpleActionState*, each state implements an *actionlib* client by default. You need to add an initialization function and a callback function *callback()* for each

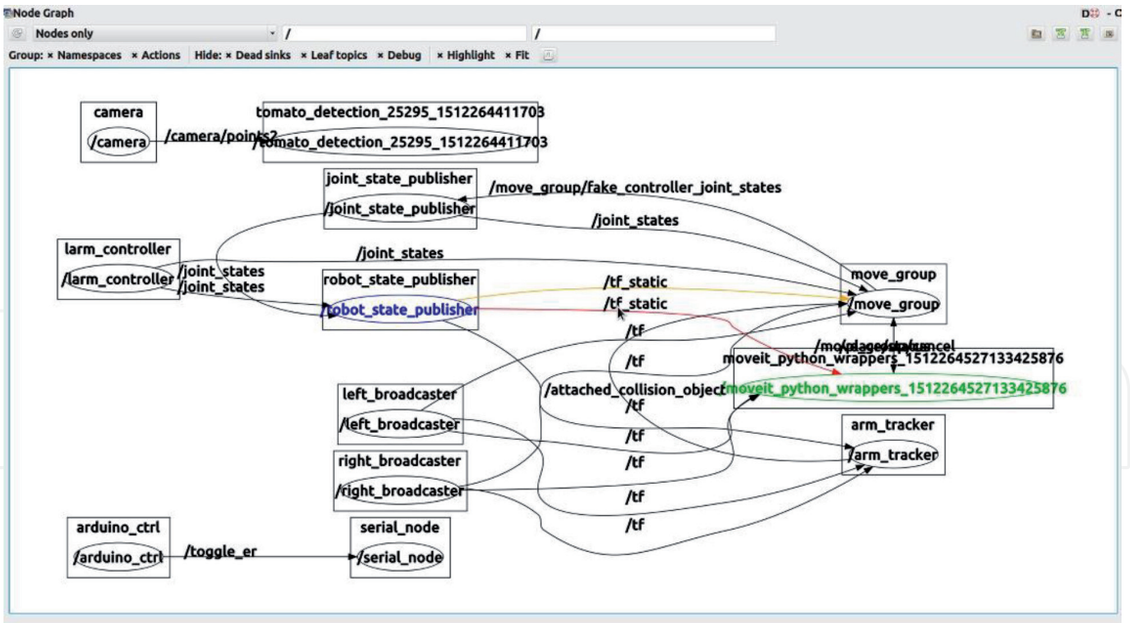

Figure 29.
System function node diagram.

state. Start a state machine visualization service *IntrospectionServe*r in the node, so that we can view the state transition diagram in *SMACH_viewer* and can monitor the state transition in real time. The data details of each state are shown in **Figure 28**.

3.5.4 System node diagram

Figure 29 shows that the running node diagram after all ROS nodes in the system is turned on. The node diagram is generated using the *rqt_graph* command. Each rectangular box represents a topic. The oval box represents a node, and the arrowed lines represent the subscription relationship between each other. Visualization of the node diagram makes the system architecture intuitive.

Since most of the eye-hand coordination and motion control are concurrent, the fluency of multitasks is verified under two plant factories and three greenhouses with different fruit status and illumination variations. The experimental results show that if total number of targets within the visual field is not more than three, the average picking time is less than 35 s.

4. Conclusion

The contribution of this research mainly orients around the software engineering for manipulating the complex robot behavior. Although service robot leverages ROS for rapid development, classical tasks such as eye-hand coordination and continuous operation in an open scenario have not been systematically addressed. In this chapter, we advocate that if the complex robot behavior can be structured, then they can be modeled as Finite State Machines (FSM), and a "Sense Plan Act" (SPA) process can be implemented with a formal software architecture. Meanwhile, we demonstrate that ViSP and SMACH in ROS are beneficial frameworks for developing a dual-arm robot for autonomously harvesting the fruits in plant factory, which embodies the complexity of multi-task planning and scheduling in natural scenes. The experimental results show that the software engineering paradigm effectively improves the system reliability and scalability of the dual-arm harvesting robot.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (No. 51775333) and the Scientific Research Program of Shanghai Science and Technology Commission (No. 18391901000).

Author details

Chengliang Liu, Liang Gong* and Wei Zhang
School of Mechanical Engineering, Shanghai Jiao Tong University, Shanghai, China

*Address all correspondence to: gongliang_mi@sjtu.edu.cn

References

[1] Song J, Zhang T, Xu L, et al. Research actuality and prospect of picking robot for fruits and vegetables. Transactions of the Chinese Society for Agricultural Machinery. 2006;**37**:158-162

[2] Kondo N et al. Fruit harvesting robots in Japan. Advances in Space Research. 1996;**18**:181-184. DOI: 10.1016/0273-1177(95)00806-P

[3] Zhao Y, Wu C, Hu X, et al. Research progress and problems of agricultural robot. Transactions of the Chinese Society of Agricultural Engineering. 2003;**19**:20-24

[4] Tanigaki K et al. Cherry-harvesting robot. Computers and Electronics in Agriculture. 2008;**63**:65-72. DOI: 10.1016/j.compag.2008.01.018

[5] Hemming J et al. A robot for harvesting sweet-pepper in greenhouses. In: Proceedings International Conference of Agricultural Engineering. Zurich; 06-10 July 2014

[6] Taqi F et al. A cherry-tomato harvesting robot. In: 18th International Conference on Advanced Robotics (ICAR); 10-12 July 2017. Hong Kong. New York: IEEE; 2017. pp. 463-468

[7] Nagata M et al. Studies on automatic sorting system for strawberry (part 3) development of sorting system using image processing. Journal of the Japanese Society of Agricultural Machinery. 1997;**59**:43-48

[8] Zhaoxiang L, GANG L. Apple maturity discrimination and positioning system in an apple harvesting robot. New Zealand Journal of Agricultural Research. 2007;**50**:1103-1113. DOI: 10.1080/00288230709510392

[9] Guo F et al. Fruit detachment and classification method for strawberry harvesting robot. International Journal of Advanced Robotic Systems. 2008;**5**(1):41-48. DOI: 10.5772/5662

[10] Wang X. Study on information acquisition and path planning of greenhouse tomato harvesting robot for selective harvesting operations [thesis]. Zhenjiang: Jiangsu University; 2012

[11] Ling X, Zhao Y, Gong L, Liu C, Wang T. Dual-arm cooperation and implementing for robotic harvesting tomato using binocular vision. Robotics and Autonomous Systems. 2019;**114**(4):134-143

[12] Foote T. tf: The transform library. In: IEEE Conference on Technologies for Practical Robot Applications (TePRA); 22-23 April 2013. Woburn. New York: IEEE; 2013. pp. 1-6

[13] Bohren J, Cousins S. The SMACH high-level executive [ROS news]. IEEE Robotics and Automation Magazine. 2010;**17**(4):18-20. DOI: 10.1109/MRA.2010.938836

[14] Mcgann C et al. Model-Based, Hierarchical Control of a Mobile Manipulation Platform. Thessaloniki, Greece: ICAPS Workshop Planning and Plan Execution for Real-World Systems; 2009

[15] Meeussen W et al. Autonomous door opening and plugging in with a personal robot. In: IEEE International Conference on Robotics and Automation; 3-7 May 2010. Anchorage. New York: IEEE; 2010. pp. 729-736

[16] Joseph H. Getting Started with Smach [Internet]. 2018. Available from: https://wiki.ros.org/smach/Tutorials/Getting%20Started [Accessed: 21 March 2020]

[17] Marchand E et al. ViSP for visual servoing: A generic software platform with a wide class of robot control skills. IEEE Robotics and Automation Magazine. 2005;**12**(4):40-52. DOI: 10.1109/MRA.2005.1577023