UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Technology

Hui Shi

Expanding the Open-source ROS Software Package opencv_apps with Dedicated Blob Detection Functionality

Bachelor's Thesis (12 ECTS)

Curriculum Science and Technology

Supervisor(s):
Associate professor of robotics engineering, PhD Karl Kruusamäe
Specialist of robotics, MSc Sandra Schumann

Tartu 2022

Expanding the Open-source ROS Software Package opencv_apps with Dedicated Blob Detection Functionality

Abstract:

As ROS (Robot Operating System) has become the most widely used open source software development kit for robotics applications, integrating computer vision tools such as OpenCV into ROS is highly demanded. One of the more established ROS wrapper packages for OpenCV - opencv_apps - however is yet to include dedicated blob detection functionality. This thesis follows the open-source software development (OSSD) process to expand the opencv_apps package by wrapping OpenCV SimpleBlobDetector within an easy-to-use ROS nodelet that also enables fine-tuning detector parameters in run-time via dynamic_re-configure. Experimental validation shows that the blob detection functionality performs well on object recognition and object tracking, and it is seamlessly integrated in the opencv_apps package.

Keywords:

ROS, robot vision, opencv_apps, blob detection, open-source software development

CERCS: T125 - Automation, robotics, control engineering, T121 - Signal processing, T120 - Systems engineering, computer technology

Avatud lähtekoodiga ROS-i tarkvarakimbu opencv_apps laiendamine laigutuvasti funktsioonaalsusega

Lühikokkuvõte:

ROS (Robot Operating System) on robootika rakenduste puhul kõige laialdasemalt kasutatud avatud lähtekoodiga tarkvararaamistik, ning seetõttu on OpenCV masinnägemise tarkvara integreerimine ROSiga väga nõutud. opencv_apps on enimkasutatud ROSi kimp, mis võimaldab rakendada OpenCV'd ROSis, kuid see ei kata kõiki OpenCV objektide tuvastamise funktsionaalsuseid, laigutuvasti (blob detection) nende seas. Käesoleva lõputöö eesmärgiks on opencv_apps ROS kimbu laiendamine OpenCV laigutuvastamise funktsionaalsusega. Töö tulemusena valmis vabavaraline ROSi sõlm, mis võimaldab käitusaegselt häälestada tuvastaja parameetreid, kasutades dynamic_reconfigure süsteemi. Töö empiirilisest analüüsist lähtub, et OpenCV laigutuvasti on terviklikult integreeritud opencv_apps ROSi kimpu ja see hõlpsasti kasutatav ROSi keskkonnas.

Võtmesõnad:

ROS, masinnägemine, opencv_apps, laigutuvasti, tarkvaraarendus

CERCS: T125 - Automatiseerimine, robootika, juhtimistehnika, T121 - Signaalitöötlus, T120 - Süsteemitehnoloogia, arvutitehnoloogia

Table of Contents

Abbreviations 6

1 Introduction 7

1.1 Background 7

1.2 Motivation 7

1.3 Objectives and Contributions 7

2 Literature Review 9

2.1 ROS (Robot Operating System) 9

2.1.1 ROS Nodelet 10

2.1.2 ROS Parameter Server and dynamic_reconfigure 11

2.1.3 ROS visualization tool – RViz 12

2.1.4 roslaunch 13

2.2 Robot Vision 13

2.2.1 Computer Vision tool – OpenCV 13

2.2.2 Blob Detection 14

2.2.3 Comparison of available ROS wrappers for blob detection using OpenCV 14

2.2.4 opencv_apps 15

2.3 Open-source Software Development 18

2.3.1 Software Development 18

2.3.2 Open-source Software Development 19

3 Requirements 21

3.1 Objective 21

3.2 Functional Requirements 21

3.3 Non-functional Requirements 21

4 Design 22

4.1 Implementation of blob_detection_nodelet 22

4.2 Implementation of dynamic_reconfigure 24
4.3 Implementation of debug modes 25
4.4 Deployment 25
5 Results 26
5.1 Experiment 1 - Benchmarking blob detection nodelet 26
5.2 Experiment 2 - Object recognition and object tracking 27
5.2.1 Setup and task objective 27
5.2.2 Test results 27
5.3 Discussion 29
6 Summary 30
References 31
Appendix 33
Non-exclusive licence to reproduce the thesis and make the thesis public 34

Abbreviations

BSD - Berkeley Software Distribution

CI - Continuous Integration

GPL - General Public License

GUI - Graphical User Interface

OpenCV - Open Source Computer Vision Library

OSS - Open-source Software

OSSD - Open-source Software Development

ROS - Robot Operating System

RViz - ROS Visualization

XML - Extensible Markup Language

YAML - Yet Another Markup Language

1 Introduction

1.1 Background

ROS (Robot Operating System) is an open source robot software development framework providing a standard software platform to the robotics community. Because of its characteristics such as supporting cross-language and cross-system development, module-based, free and open-source, it is widely adopted by the robotics industry and academia [1]. Robotics software developers around the world are contributing to ROS by developing various ROS packages that can be used as modules in any ROS based robot systems. Other developers can download the existing packages and use their functionalities in their robot system directly, and only develop the new features needed. Among these packages, it is particularly needed to develop vision packages integrated with computer vision tools like OpenCV (Open Source Computer Vision Library), as computer vision tools are widely used in the field of robot vision [2], [3].

1.2 Motivation

Various ROS vision packages integrated with OpenCV's functionalities have been developed [4], [5], [6]. Among these packages, to the best of the author's knowledge, only opencv_apps covers the functionalities used in robot vision the most, such as edge detection, people recognition and motion analysis. In addition, opencv_apps is well maintained and well documented, resulting in the package being easy to install and to use. However, opencv_apps does not yet have the blob detection functionality which is commonly used in robot vision for object recognition and object tracking [7], [8], [9].

1.3 Objectives and Contributions

The objective of this thesis is to expand the opencv_apps package with dedicated blob detection functionality. The development followed the open-source software development process and complies with the opencv_apps package structure. ROS nodelet is implemented in C++ for running multiple functionalities in the same process with zero copy transport between algorithms and avoiding network traffic. ROS dynamic_reconfigure is deployed for updating parameters at runtime without restarting the program. The reconfigured parameters can be saved in a YAML (Yet Another Markup Language) file for later reuse in the deploy mode. Extensive experiments were carried out to demonstrate that the blob detection nodelet

performs well on object recognition and object tracking, and it is seamlessly integrated in the opencv_apps package.

2 Literature Review

This chapter explains the concepts related to the background and the technologies used to achieve the objective of this thesis. First, ROS and its features applied in this project are described. Second, robot vision and the related technologies involved in this project are discussed. Third, the open-source software development and its process are explained.

2.1 ROS (Robot Operating System)

ROS is an open source robot software development platform for developing robot applications [1]. ROS provides a development environment allowing a global level of collaboration of robotic software development [10]. It supports cross-language and cross-system development and it is module-based, free and open-source. There are two features of ROS that should be highlighted in terms of contributing to making robot development easy and efficient. The first is the reusability of the program. For instance, a robot software developer can download the existing ROS packages and use their functionalities in their robot system directly, and only develop the new features needed [10]. Developers can also contribute to the community by sharing the packages that they developed. The second highlight is that ROS is communication-based [10]. To illustrate, in order to achieve modularization, a node (a program that contains minimal functions) exchanges data with other nodes through messages published on specific topics at runtime. The communication between the nodes are managed by a ROS Master node [10]. A simplified schematic of how the nodes work with each other in ROS is demonstrated in Figure 1.

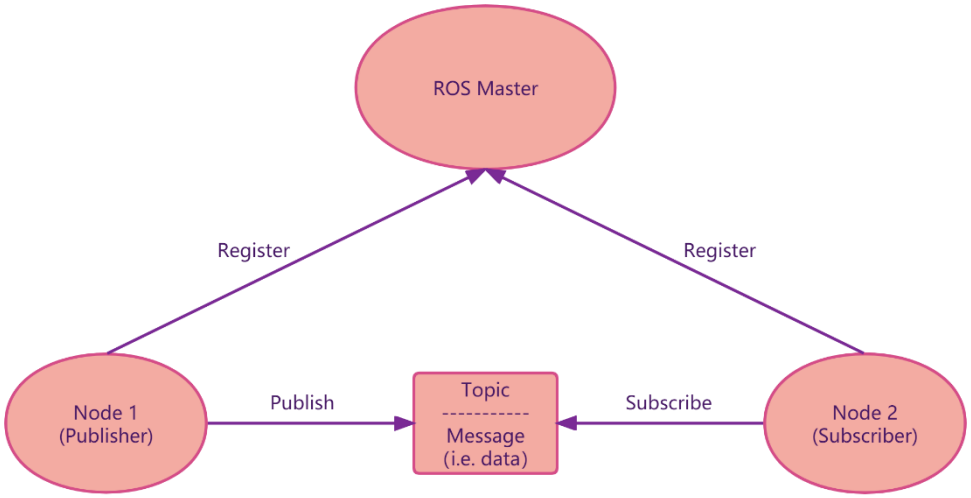


Figure 1. *Schematic of the communication between ROS nodes*

2.1.1 ROS Nodelet

Nodelet is a modified form of ROS node [11]. The nodelet package is designed to run multiple nodelets in the same process, and each nodelet is executed as a thread [12]. Also, instead of transferring the actual data between these nodelets, only the memory address/memory pointers are passed around [13]. This design allows the communication between the nodelets to achieve high efficiency without overloading the network, as there is zero copy transport between them [13]. At the same time, these nodelets are able to communicate with external nodes [11]. Nodelets are especially useful when there is a need for transferring high volumes of data between the nodelets, for example, transferring data from cameras and 3D sensors [11], [12].

To implement ROS nodelet, a plugin architecture of ROS called pluginlib is used [11]. Pluginlib allows the nodelets to be dynamically loaded or unloaded as plugins to the main process [11]. Figure 2 explains how to implement nodelets. First, a base class which inherits a standard ROS nodelet base class and contains the common functionalities used by all the nodelets is implemented in a C++ program [11]. The C++ program also includes pluginlib/class_list_macros.h and nodelet/nodelet.h to access pluginlib APIs and nodelets APIs [11]. Second, each needed nodelet containing a subclass with a separate functionality is implemented, and the class is exported as a plugin to the pluginlib of the package. Finally, through a nodelet manager responsible for dynamically loading and unloading nodelets, the nodelets can be dynamically loaded or unloaded as plugins on a single process [11].

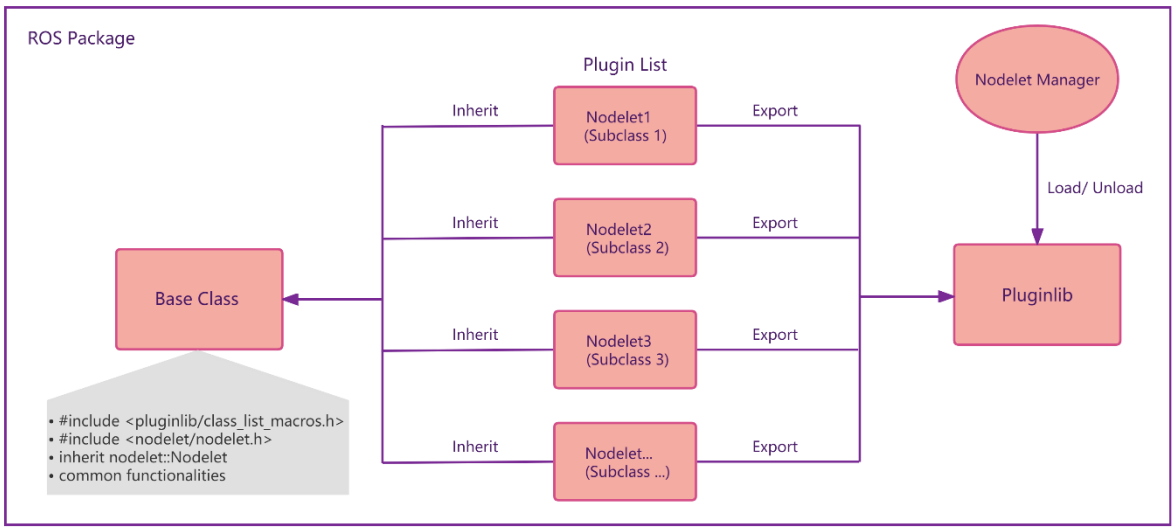


Figure 2. *Schematic of how to implement nodelets*

2.1.2 ROS Parameter Server and dynamic_reconfigure

Parameter Server

In ROS, parameters, for example, variables in a node, can be managed by a parameter server [11]. Parameter server as a shared server is a part of the ROS Master node. Through the parameter server, parameters can be stored in a central location and all nodes can access these values. A scope of a parameter can be set to specify the nodes under which namespace can access (read, write, modify and delete) its value on the parameter server [11]. Parameters can be stored in a file and loaded into the server when a system is launched [14].

dynamic_reconfigure

The dynamic_reconfigure package is an extension package of ROS parameter server [11]. This package allows the updating of parameters on the server at runtime and pushing the updated values to the nodes that need them [14]. It allows the parameters of a node to be dynamically reconfigured without restarting the node. The dynamic_reconfigure package is used when the program needs to frequently change the variable values or tuning the values to find a suitable set of values for performing specific tasks [14].

To integrate the dynamic_reconfigure feature in a node, first is to create a Python executable .cfg file using the dynamic_reconfigure API, this is for defining the parameters desired to be dynamically reconfigured. This file contains a list of parameters and their names, types, ranges, default values, etc. [15]. The next is to make the node dynamically reconfigurable by adding a dynamic reconfigure server in the code. A callback function in the node is called and receives the updated parameter values when the dynamic_reconfigure client tools change the values [16].

Rqt_reconfigure is a dynamic_reconfigure client tool for viewing and editing the parameters that are accessible via dynamic_reconfigure in the parameter server. It is a plugin of the ROS GUI development tool rqt. As the example shown in Figure 3 [17], it provides a GUI to the users to view and modify the parameter values of the nodes that are dynamically reconfigurable. There are various ways to change the values , such as using trackbar, checkbox and drop-down box. These ways can be pre-defined in the .cfg file [17].

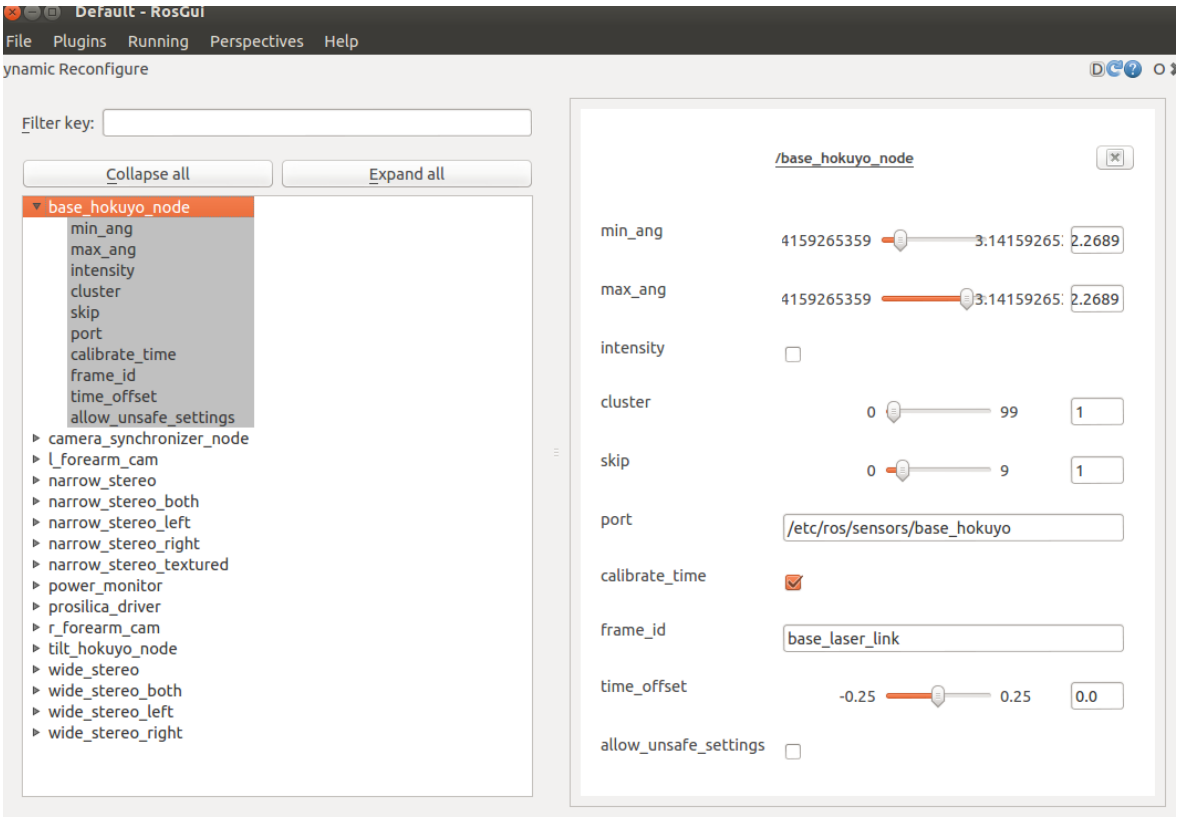


Figure 3. *rqt_reconfigure GUI* [17]

2.1.3 ROS visualization tool – RViz

RViz (ROS Visualization) is a 3D visualization tool. The main purpose of RViz is to visualize ROS messages, allowing verifying the data visually [10]. Various types of data can be visualized in RViz by subscribing to the corresponding topics, such as the distance data from a Laser Distance Sensor, the Point Cloud Data of a 3D distance sensor, and image data from a camera [10]. An example of the visualization of image messages in Rviz is shown in Figure 4 [10].

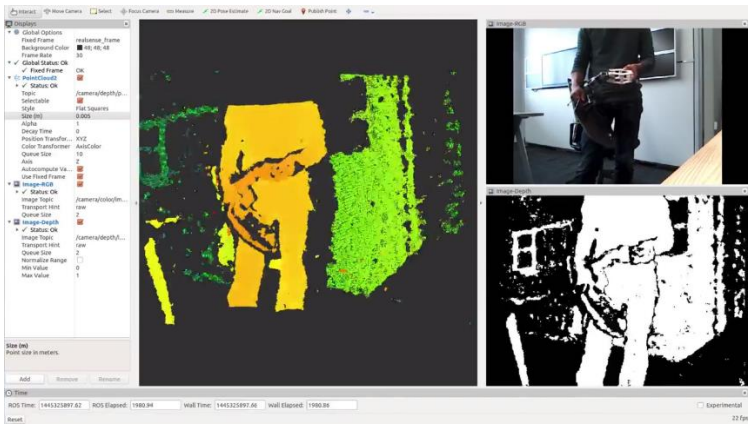


Figure 4. *Distance, infrared, color image value obtained from Intel RealSense* [10]

2.1.4 roslaunch

ROS applications often involve several interconnected nodes with many parameters. The roslaunch package provides the tools to launch multiple nodes and a Master node at once and other initialization requirements [18], for instance, setting parameters on the parameter server. The nodes to run and the parameters to set are specified in a .launch configuration file using XML (Extensible Markup Language). Then, by a single command line command roslaunch followed by the package name and the .launch file name, the nodes can be launched with the specified initialization configuration [18].

2.2 Robot Vision

Vision is the most powerful sense for a robot as it is for humans [19], [20]. Without direct physical contact, vision provides a robot tremendous amounts of information about its surroundings and enables the robot to interact with the environment [19]. A robot vision system often refers to a system that enables robots to perceive the external world visually and perform a wide range of tasks such as navigation, object tracking and manipulation, surveillance and higher-level decision-making [20]. In order for the robot to process visual data and make physical actions accordingly, a robot vision system often involves a combination of vision sensors (e.g. cameras) and computer algorithms [21]. Computer vision tools are widely used in robot vision systems for processing image data. For ROS based robot systems, vision modules integrated with computer vision libraries such as OpenCV are highly demanded by the worldwide robot developers, as these modules can be used in their existing ROS based robot systems directly. One of the OpenCV's functionalities that is commonly used in the field of robot vision is blob detection for object recognition and object tracking [7], [8], [9]. One of the more established ROS wrapper packages for OpenCV - opencv_apps - however is yet to include dedicated blob detection functionality. In this section, OpenCV, blob detection, and opencv_apps will be introduced.

2.2.1 Computer Vision tool – OpenCV

OpenCV (Open Source Computer Vision Library) is an open source software library for computer vision, machine learning and image processing [22]. OpenCV has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Mac OS and Android. It is a giant library containing more than 2500 optimized algorithms. These algorithms have a large range of applications, such as face recognition, object identification, human actions classification, tracking movement objects [22]. With all its powerful functionalities, OpenCV is widely used in the field of robot version [2], [3].

2.2.2 Blob Detection

One of the OpenCV's functionalities that is commonly used in the field of robot vision is blob detection for object recognition and object tracking [7], [8], [9]. A blob is a region in an image that shares some common properties, such as color and brightness. For example, in an image containing some oranges on a white table, the regions of the oranges are blobs. The purpose of blob detection is to identify these regions [23]. However, opencv_apps does not currently have a dedicated nodelet for blob detection. This project aims to integrate the OpenCV functionality SimpleBlobDetector to the opencv_apps package. SimpleBlobDetector [24] is an algorithm for extracting blobs from an image and it returns the coordinates of the center points of the blobs as locations and the radiuses as sizes [24]. The blobs can be filtered by color, area, circularity, inertia ratio, convexity [24].

2.2.3 Comparison of available ROS wrappers for blob detection using OpenCV

The currently available ROS wrappers for blob detection using OpenCV do not have an easy to use blob detection functionality (Table 1). As shown in Table 1, the package cmvision allows the blob detection only based on blob colors, the user can not specify the shapes of the blobs. Also, cmvision is not easy to build and not easy to use, though it is well documented. The package ros_color_detection is not maintained and incompatible with recent distributions of ROS. The well established package opencv_apps has 26 functionalities that are used in the field of robot vision. It is well maintained, well documented, easy to use and widely adopted by the ROS community. However, it does not have a dedicated blob detection functionality. Therefore, the author proposes a blob detection solution to the opencv_apps package.

Table 1. *Comparison of available ROS wrappers for blob detection using OpenCV*

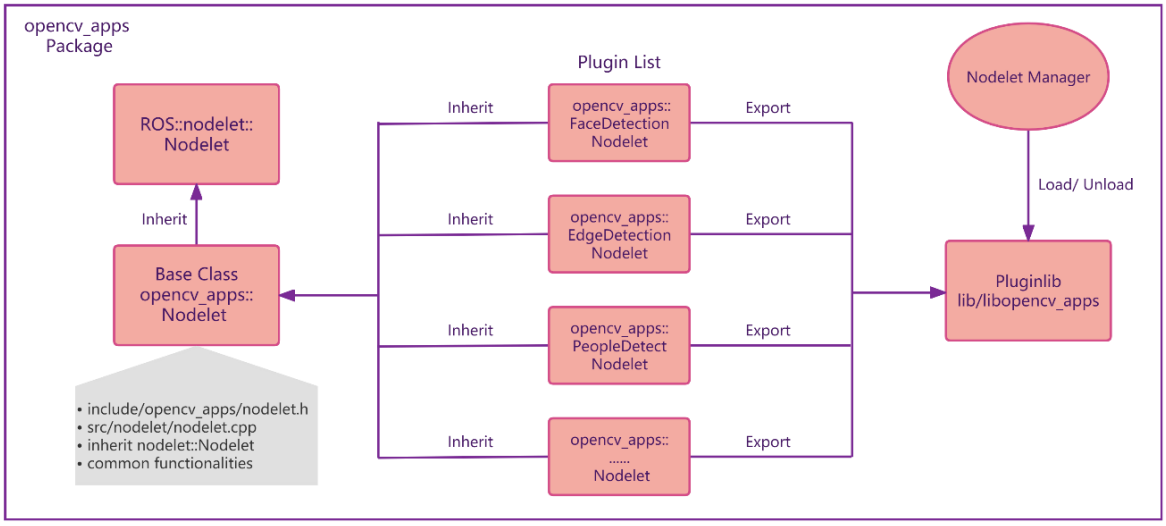
Package Name	Advantages	Issues
cmvision [6]	• well maintained • well documented	• only one functionality - fast color blob detection • blob detection is only based on blob colors, can not based on shapes • not easy to build • not easy to use
ros_color_detection [5]	N/A (was not tested as the package is incompatible with recent distributions of ROS)	• only one functionality - color detection • not maintained • incompatible with recent distributions of ROS
opencv_apps [6]	• 26 functionalities • well maintained • well documented • easy to use	minor issues in some nodes: https://github.com/ros-perception/opencv_apps/issues

2.2.4 opencv_apps

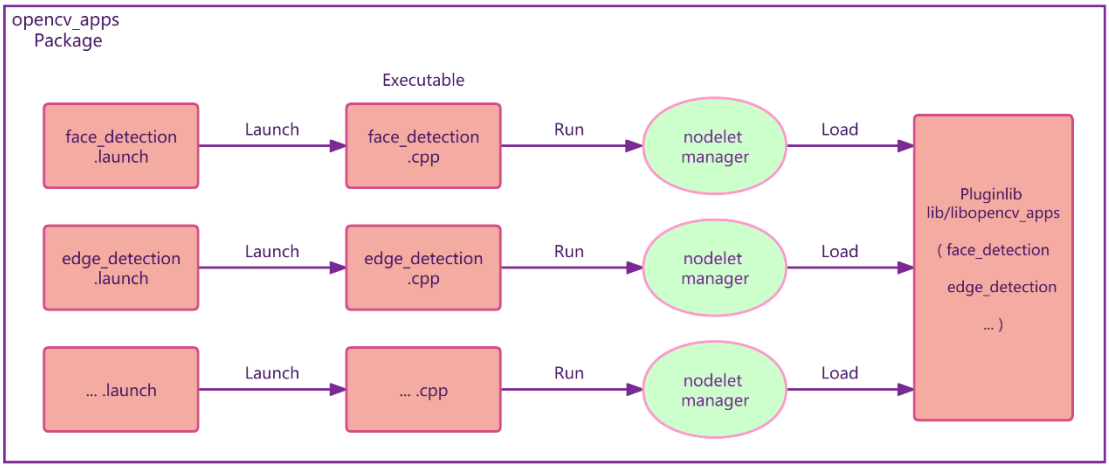
ROS opencv_apps is a package that integrates various OpenCV's functionalities into ROS. It provides various nodelets that run OpenCV's functionalities and publish the results as ROS messages [6]. The opencv_apps package covers a wide range of the functionalities used in robot vision, such as edge detection, structural analysis, people recognition and motion analysis [6]. ROS developers can use opencv_apps directly in their existing robot system and use its functionalities without writing the corresponding OpenCV code. The nodelets take image messages as input by subscribing to the specified topic and publish the result as ROS messages. For instance, the face_detection nodelet first processes the input image to find the faces and label the detected faces on the original image by circles. Next, this nodelet publishes the result image with labeled faces as image messages as well as publish array messages containing the locations of the detected faces in the image coordinates [6]. Then other nodes in the robot system can use the information by subscribing the topics with these messages. For example, a node for navigating the robot can use the location of a face to navigate the robot to that person.

Package architecture

The opencv_apps package architecturally replies on ROS nodelet. Figure 5 shows the simplified architecture of opencv_apps. In Figure 5a, the base class opencv_apps::Nodelet which inherits the ROS::nodelet::Nodelet class and contains the common methods used by all the nodelets is declared in *include/opencv_apps/nodelet.h* and defined in *src/nodelet/nodelet.cpp*. Each nodelet containing a subclass with a separate functionality (e.g. face detection, edge detection) is implemented in the src/nodelet folder, and the class is exported as a plugin to the pluginlib *lib/libopencv_apps* of the package for the dynamic loading. In Figure 5b, each functionality's executable is generated by copying the template file *src/node/standalone_nodelet_exec.cpp.in* and replacing the @NODELET_NAME@ in its content with corresponding nodelet name; this is implemented in the *CMakeLists.txt* as shown in Figure 6 (the macro *opencv_apps_add_nodelet* is then called for each functionality in the *CMakeLists.txt*). An executable starts a nodelet manager which then loads the corresponding plugin from the pluginlib. For example, by running the *face_detection.launch* file, it launches the *face_detection* executable which starts a nodelet manager as well as loads the *face_detection* plugin. However, starting a new nodelet manager for each functionality does not follow the design purpose of ROS nodelet which is using one nodelet manager to load multiple nodelets into one process to achieve high efficiency [13].



(a)



(b)

Figure 5. *Architecture of opencv_apps*

```
## Macro to add nodelets
macro(opencv_apps_add_nodelet node_name nodelet_cppfile)
  set(NODE_NAME ${node_name})
  set(NODELET_NAME opencv_apps/${node_name})
  configure_file(src/node/standalone_nodelet_exec.cpp.in ${node_name}.cpp @ONLY)
  add_executable(${node_name}_exe ${node_name}.cpp)
  SET_TARGET_PROPERTIES(${node_name}_exe PROPERTIES OUTPUT_NAME ${node_name})
  target_link_libraries(${node_name}_exe ${catkin_LIBRARIES} ${OpenCV_LIBRARIES})
  list(APPEND _opencv_apps_nodelet_cppfiles ${nodelet_cppfile})
  list(APPEND _opencv_apps_nodelet_targets ${node_name}_exe)
endmacro()
```

Figure 6. *The macro opencv_apps_add_nodelet in CMakeLists.txt*

The nodelets (subclasses) of opencv_apps share the same working flow. Figure 7 shows a simplified working flow of a nodelet. First the *onInit()* method calls the *Nodelet::onInit()* method of the base class to initialize nodehandles nh_ and pnh_. It also calls the *onInitPostProcess()* method for the post processing of initialization of the nodelet. In the *onInit()* method, a dynamic_reconfigure server with the callback function *reconfigureCallback()* is created. The *subscribe()* method creates a subscriber which subscribes to an image topic and the callback function *imageCallback()* is called when there is a new image message received. The *imageCallback()* then calls the *doWork()* method which processes the image and publishes the results as ROS messages. However, though these methods are used in all the subclasses, they are not included in the base class.

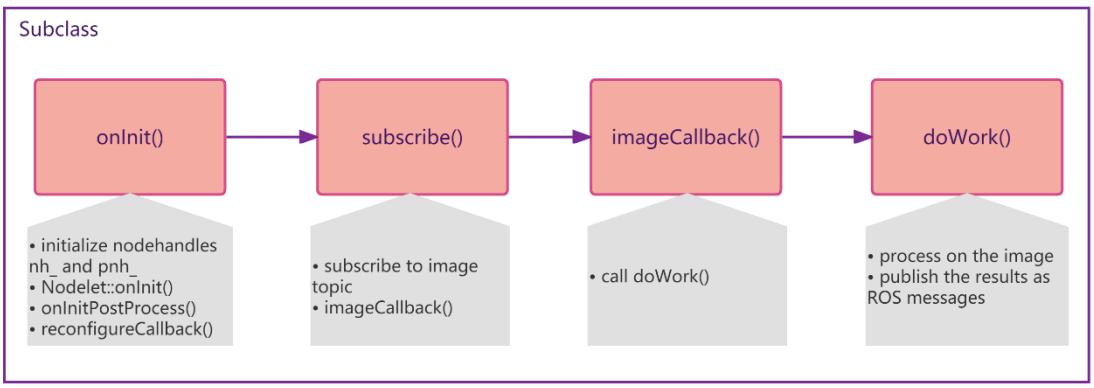


Figure 7. *Working flow of a nodelet in opencv_apps*

2.3 Open-source Software Development

Software is a collection of instructions organized in a particular order that instruct a computer to perform specific tasks [25]. Software is hardware independent, making the computers programmable [25]. In this section, the concepts and technologies related to software development and open source software development will be explained.

2.3.1 Software Development

Software development is the process of building a software product according to users' requirements [25]. It is a systematic process that includes requirements analysis and specification, design and development, testing, deployment, maintenance and support [25]. Software developers use programming languages (e.g. C++, Java) to implement software products [25].

In software development, continuous integration (CI) is the practice of automating the integration of code changes from multiple developers into a single project [26]. CI allows developers to merge their code changes frequently (usually each person integrates at least once a day) into a central repository where each integration is verified by an automated build [26]. The automated build (including compilation, release, and automated testing) allows errors to be detected before the integration [26], [27]. CI reduces problems that occur during the integration and increase the efficiency of the development of a cohesive software [27].

The foundational dependency of the CI process is a source code version control system, such as Git [26]. A version control system records changes of a file or set of files over time

in order specific versions can be recalled later [28]. Git is an open source distributed version control system that can handle version management of small to very large scale projects efficiently [29]. Git allows the developers to revert files or the entire project back to a previous state, compare changes over time, see which developer in the team last modified something that might be the cause of a problem, etc. [28]. A project using Git is hosted on a version control hosting platform such as Bitbucket, GitHub, and Gitlab, and these version control hosting tools have support and features built in for CI [26]. To contribute to the same project and work parallelly, multiple developers first clone (copy) the Git repository (codebase) of the project from the Git hosting platform to their local machine (computer) [30]. Then each developer creates a branch (version) from the main branch for developing different features and makes changes and commits the changes on that branch [30]. To merge a branch to the main branch, a developer can open a pull request (propose changes to the main branch) to notify other developers that the new set of changes are ready for integration [26], [30]. Next, a reviewer of the team performs a code review of the new code and functionality and approves or denies the pull request and makes edit suggestions [26]. Pull requests and code review are essential practices to effective CI [26].

2.3.2 Open-source Software Development

Open-source software development utilizes the software development process and its related technologies described in the previous sub section. At the same time, open-source software development has some unique characteristics. Open-source software (OSS) is a term defined to describe software whose source code is available to the public, and the copyright holder of such software retains some rights under the software license and allows users to use, learn, modify, and distribute the software free of charge to the public [31], [32]. Unlike closed source (or proprietary) software, whose source code is only available to the person, team, or organization who created it, the source code of OSS is available to the public [33].

OSS encourages the principles of open exchange, collaborative participation, rapid prototyping, transparency and community-oriented development [33]. To illustrate, OSS allows contributors from numerous places in the world to collaborate and contribute to the software. They can obtain, scrutinize, make additions or improvements to the software. The goal is that the joint efforts of many people will produce software that is increasingly useful and reliable to the end users [32]. However, some OSS products have disadvantages when compared with proprietary software, such as lower security, lack of documentation, less user-friendly, lower customized support, nonexistence of extensive tech support [32], [34], [35].

Open source licenses regulate the way people use, change and distribute OSS. The most popular license used for OSS is the GPL (General Public License) license and it is used by approximately 70% of open source products [32]. GPL gives the right to individuals to use the software for any purpose, alter, share and freely distribute the changes one makes to the software [32]. GPL forces the modified versions of the software to meet the same criteria listed in the license of the original software [32]. That means the changes of the software should be also free of charge and open source [32]. At the same time, there are less restrictive OSS licenses allowing derivative works to be released under different terms,

such as the BSD (Berkeley Software Distribution) license. BSD license allows the changes of a software to be closed source and used for commercial purposes [36]. The opencv_apps package uses the BSD license.

3 Requirements

3.1 Objective

The objective of this thesis is to expand the open-source ROS software package opencv_apps with dedicated blob detection functionality, which integrates OpenCV's SimpleBlobDetector functionality.

3.2 Functional Requirements

• Implementation of ROS nodelet in C++ for running multiple functionalities in the same process with zero copy transport between algorithms and avoiding network traffic.

• Implementation of ROS dynamic_reconfigure for updating parameters at runtime without restarting the program.

• The reconfigured parameters can be saved in a YAML configuration file for later reuse in the deploy mode.

• Visualization of image messages published by the nodelet in RViz.

• Being able to switch between debug modes and initialization configurations using roslaunch.

3.3 Non-functional Requirements

• The development follows the open-source software development process and complies with the opencv_apps package structure.

• Ubuntu Linux 18.04 or higher for running ROS Melodic or later distributions.

• opencv_apps (the version updated on 11th, Feb., 2022)

• OpenCV 3.0.0 or higher

• Git and GitHub

• C++ language for the implementation of ROS nodelet.

• Python language for the implementation of dynamic_reconfigure.

4 Design

The project followed the open-source software development process including six steps:

1) Need identification
The needs of opencv_apps package and blob detection functionality are explained in the literature review (under section 2.2).
2) Planning & Designing
The analysis of the architecture design of the opencv_apps package is done in the literature review (under section 2.2.4).
The requirements of developing and integrating the blob_detection_nodelet into the opencv_apps package is explained in the Requirements chapter (Chapter 3).
3) Implementation
The implementation of the blob_detection_nodelet will be introduced in this chapter.
4) Testing
Testing of the solution will be introduced in the next chapter (Chapter 5).
5) Deployment
Deployment will be explained in this chapter.
6) Maintenance
Maintenance will be discussed in the discussion.

4.1 Implementation of blob_detection_nodelet

The blob_detection_nodelet follows the architecture of a nodelet (subclass) of opencv_apps as described in section 2.2.4. This section will focus on introducing the implementation of the *doWork()* method which takes image messages as input, does blob detection and publishes the detection results as ROS messages. Figure 8 shows a simplified flow chart of the *doWork()* method. First, it converts the image message into OpenCV image. Second, it checks if the *debug_view* is turned on (*debug_view* can be turned on/off in the *blob_detection.launch* file). When the *debug_view* is turned on, the program shows the debug images in OpenCV windows and attaches OpenCV trackbars for tuning the blob parameters dynamically. If the value of a parameter is changed by a trackbar, the new value is then updated to the parameter server. Third, it does thresholding and morphological operations if it is turned on, making the image ready for the blob detection. Next, it creates a new blob detector if the blob parameters change and performs the blob detection. Finally, it converts the detection results (radius and coordinates of the center point of the blob) to ROS messages and publishes them. The message type is a custom type *BlobArrayStamped* defined in *msg/BlobArrayStamped.msg* and the structure of the messages is shown in Figure 9. *BlobArrayStamped* message type consists of two message types. The type *Blob[]* is an array which includes the two message types described in *Blob.msg*. Radius of a blob is in the type of *float64* and the message type of the center point of a blob is *Point2D* which is

defined in *Point2D.msg*. The debug images are also converted to ROS image messages and published if there is a subscriber.

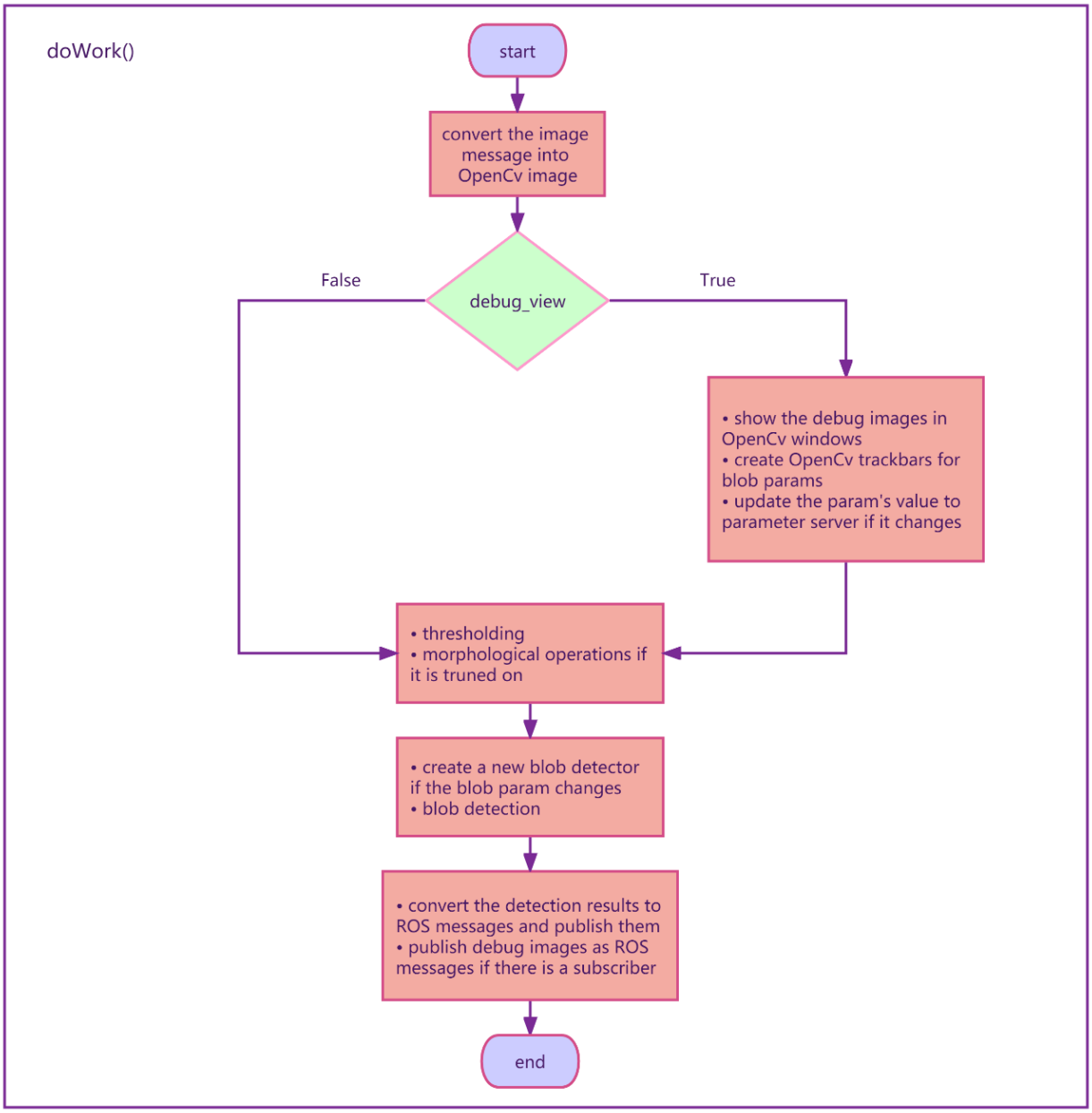


Figure 8. *Flow chart of doWork() method*

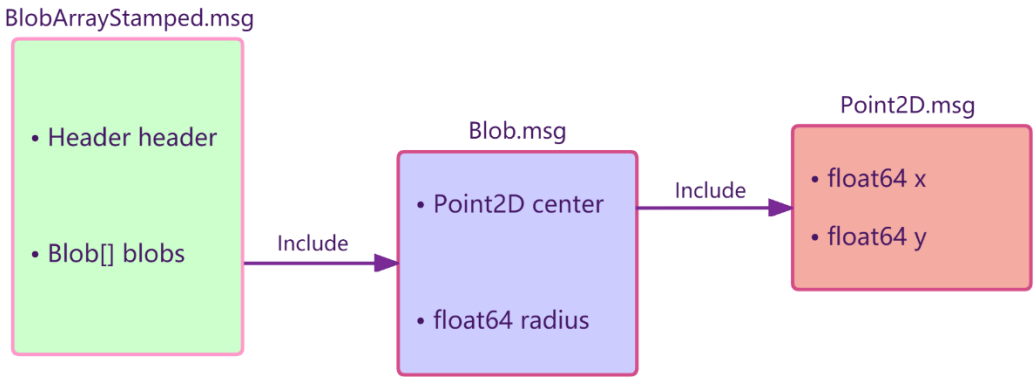


Figure 9. *Structure of the message type BlobArrayStamped*

4.2 Implementation of dynamic_reconfigure

Figure 10 shows the design of dynamic_reconfigure in the blob_detection_nodelet. This design allows the parameters to be dynamically reconfigured by both ROS dynamic_reconfigure and OpenCV trackbars. When a dynamic_reconfigure client tool, for example rqt_reconfigure, changes the value of a parameter and sends a request to the reconfigure server, the callback function *reconfigureCallback()* is called. It receives the updated value and changes the value of the parameter in the nodelet accordingly. It also updates the value of the OpenCV trackbar corresponding to that parameter. When the value of a parameter is changed by an OpenCV trackbar, the callback function *trackbarCallback()* is called which enables the *need_config_update_* feature. This feature then updates the new value to the reconfigure server.

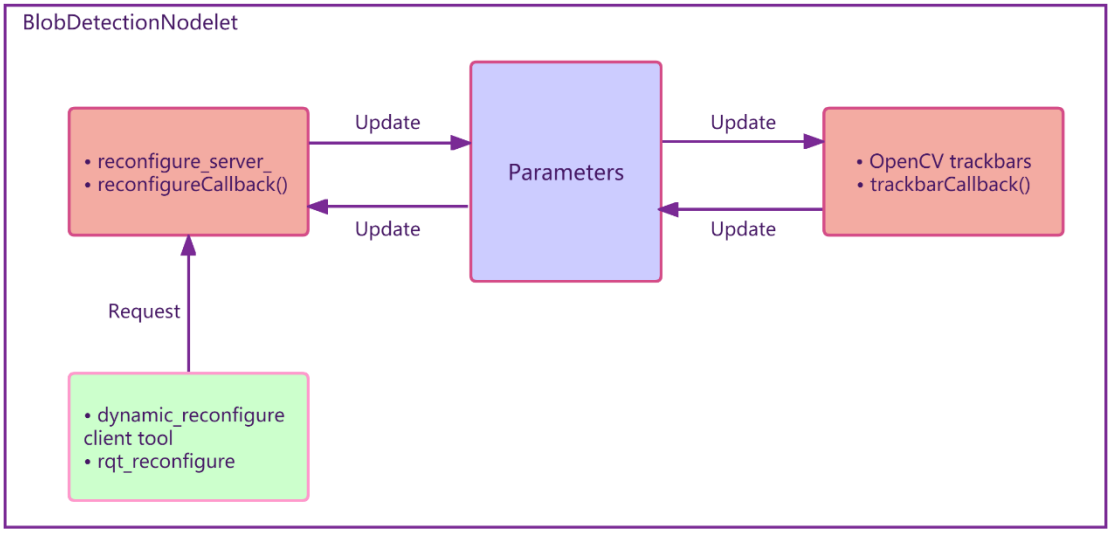


Figure 10. *Design of dynamic_reconfigure in blob_detection_nodelet*

4.3 Implementation of debug modes

In addition to the *debug_view* (OpenCV debug view) feature of the original design of opencv_apps. The author developed two more debug modes - ROS debug mode and deploy mode. This feature is implemented in *launch/blob_detection.launch*. The ROS debug mode launches the blob_detection nodelet, the rviz node with the configuration which subscribes to the debug images published by the nodelet to visualize the debug images, and the rqt_reconfigure node. The deploy mode launches the blob_detection nodelet with configured parameters loaded from a YAML file. One suggestion to use the modes is that under ROS debug mode, configure the parameters and save the configured values to a YAML file by the following command : rosparam dump config/blob_detection_config.yaml blob_detection. Then switch to the deploy mode.

4.4 Deployment

The developed solution is available in the public repository of the IMS robotics lab of the University of Tartu on GitHub (Appendix1). A pull request based on the developed blob detection functionality has been sent to the opencv_apps organization on GitHub (Figure 11). Further review is needed from the opencv_apps team before the integration.

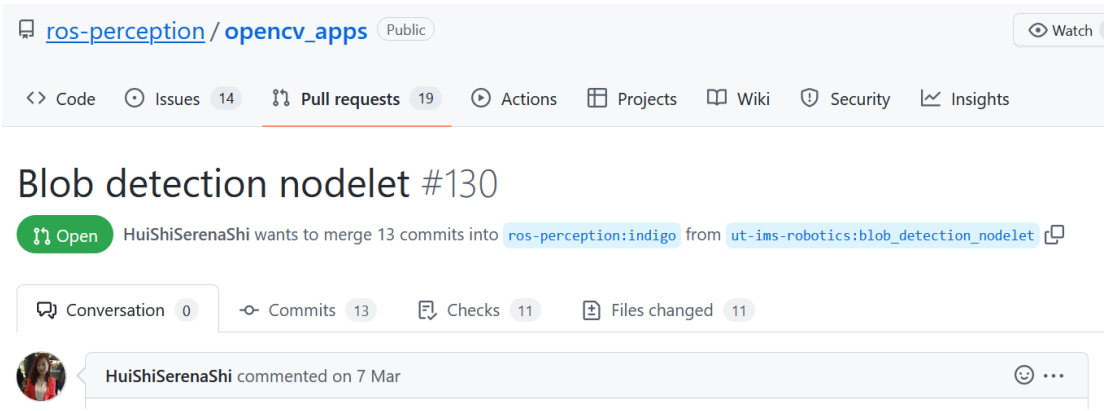


Figure 11. *Screenshot of the pull request on Github to merge the developed solution*

5 Results

5.1 Experiment 1 - Benchmarking blob detection nodelet

The performance of the blob detection functionality was benchmarked on two computational systems, a personal computer and the on-board computer of a mobile robot Robotont (Figure 12). Different blob parameter configurations were tested to see how it affects the blob detection performance (the frequency of the publishing of the messages of the detected blobs, the frequency is obtained using rostopic hz [37]). Table 2 shows the testing results, the first column describes the blob parameter configurations, the second and the third column show the frame rate obtained from the tested computers under different conditions. The frame rate results indicate that, on the tested computers, different configurations of blob detector parameters do not affect the detection performance. The frame rate is on average 30 FPS under all the conditions.

Figure 12. *Mobile robot Robotont*

Table 2. *Benchmarking results of blob detection nodelet*

Blob Parameter Configuration	Frame Rate of PC (FPS) (r7 5800h, 16GB RAM, NVIDIA RTX 3060, camera resolution 1280*720)	Frame Rate of Robot Computer (FPS) (i5-7260U, 4GB RAM, Intel Iris® Plus Graphics 640, camera resolution 1280*720)
no filter	30.015	30.150
filter by area	30.009	30.274
filter by area + filter by shape	30.034	29.756

5.2 Experiment 2 - Object recognition and object tracking

The developed blob detection functionality was tested on the service robot TIAGo for object recognition and object tracking.

5.2.1 Setup and task objective

Computer specification: i5-7260U, 4GB RAM, Intel Iris® Plus Graphics 640, camera resolution 1280x960

Object recognition:

The robot is surrounded by 4 different shapes, triangle, square, hexagon, circle, as shown in Figure 13. The task is to detect specific shapes under specific blob parameter configurations. The robot should spin and stop for 3 seconds at the shapes with the property that meets the configuration.

Object tracking:

In this task, the author is seen as an object. The robot should recognize and follow the author.

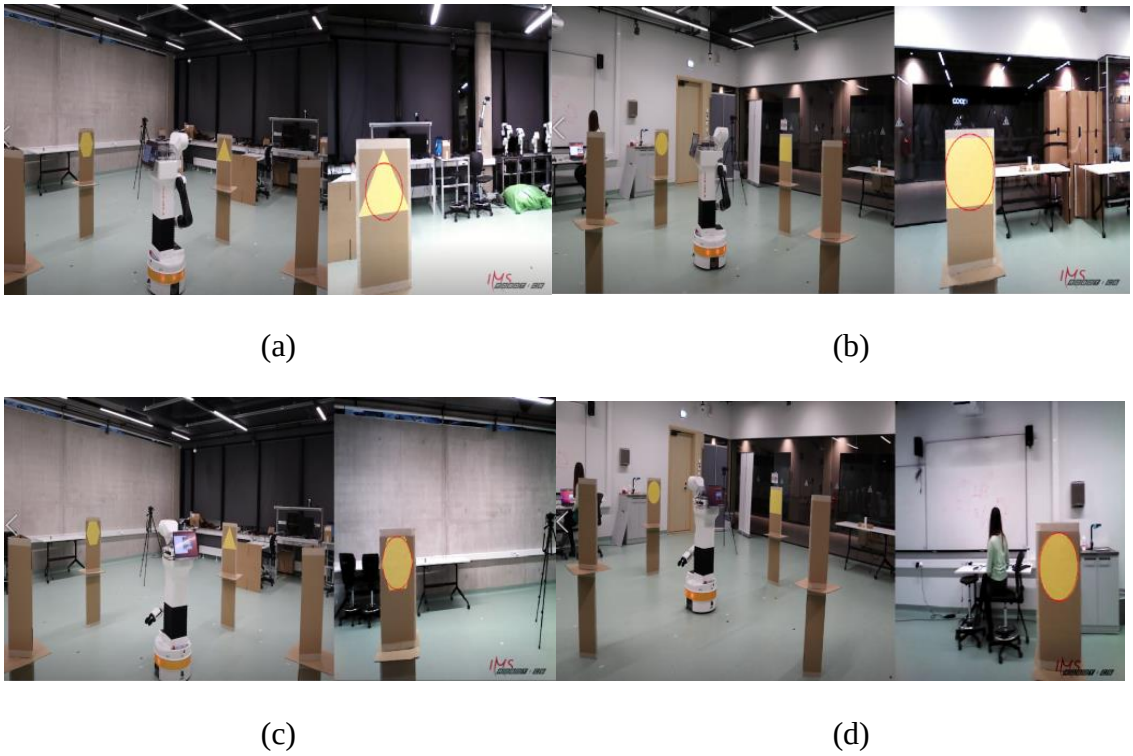
Figure 13. *Setup for object recognition*

5.2.2 Test results

Object recognition:

The robot was first configured to detect only triangle and square among the four shapes, and the test was successful as shown in Figure 14a-b.

The robot then was configured to detect only hexagon and circle among the four shapes, and the test was successful as shown in Figure 14c-d.



(a) (b)

(c) (d)

Figure 14. *Test results of object recognition*

Object tracking:

The robot was configured to recognize the author, and the robot is able to follow the author as shown in Figure 15. When the author is out of the robot's field of view, the robot is able to spin around to look for the author.

Figure 15. *Test result of object tracking*

5.3 Discussion

The developed blob detection nodelet follows the opencv_apps package architecture and meets the requirements. Based on the two tested computers, the performance of the blob detection functionality is stable under different blob detector configurations. The blob detection functionality was tested on the mobile robot robotont for object recognition, and it was tested on the service robot TIAGo for object recognition and object tracking. All the tests were successful, and the functionality can be used on any ROS based robot system. To deploy the developed feature, the author has sent a pull request to the opencv_apps repository on GitHub. However, in order for the developed feature to be used by a wider range of ROS developers, the deployment from the opencv_apps team is needed to review the solution and integrate it to the opencv_apps package. The solution is currently available in the public repository of the IMS robotics lab of the University of Tartu on GitHub (Appendix1). The code and demo video of the experiments also serves as examples to the public on GitHub under the IMS robotics organization (Appendix2). The author will be available for future maintenance of the software. However, the maintenance is managed centrally by the opencv_apps organization on GitHub. The author will be available to maintain the software if there is a requirement from the opencv_apps team.

6 Summary

This project expanded the open-source ROS software package opencv_apps with dedicated blob detection functionality, by integrating OpenCV's SimpleBlobDetector functionality. The development followed the open-source software development process and complies with the opencv_apps package structure. ROS nodelet is implemented in C++ for running multiple functionalities in the same process with zero copy transport between algorithms and avoiding network traffic. ROS dynamic_reconfigure is deployed for updating parameters at runtime without restarting the program. The reconfigured parameters can be saved in a YAML file for later reuse in the deploy mode. Extensive experiments were carried out to demonstrate that the developed blob detection functionality performs well on object recognition and object tracking, and it can be used in any ROS based robot system. The developed solution and its demos are available to the public in the repository of the IMS robotics lab on GitHub.

In order for the developed feature to be used by a wider range of ROS developers, the deployment from the opencv_apps team is needed, which is to review the solution and integrate it to the opencv_apps package. The author will be available for the future maintenance of the software if there is a requirement from the opencv_apps team.

References

1. Quigley, M. *et al.* ROS: an open-source Robot Operating System.

2. Brahmbhatt, S. *Practical OpenCV*. (Apress, 2013).

3. B., J., V., S., Purohit, V., Oswald Tauro, D. & J., V. Design and Development of Automated Intelligent Robot Using OpenCV. in *2018 International Conference on Design Innovations for 3Cs Compute Communicate Control (ICDI3C)* 92–96 (2018). doi:10.1109/ICDI3C.2018.00028.

4. cmvision - ROS Wiki. http://wiki.ros.org/cmvision.

5. Hou, P. *ros-color-detection*. (2022). https://github.com/penghou620/ros_color_detection

6. opencv_apps - ROS Wiki. http://wiki.ros.org/opencv_apps.

7. Kiran, D., Rasheed, A. I. & Ramasangu, H. FPGA implementation of blob detection algorithm for object detection in visual navigation. in *2013 International conference on Circuits, Controls and Communications (CCUBE)* 1–5 (2013). doi:10.1109/CCUBE.2013.6718570.

8. Acevedo-Avila, R., Gonzalez-Mendoza, M. & Garcia-Garcia, A. A Linked List-Based Algorithm for Blob Detection on Embedded Vision-Based Sensors. *Sensors* **16**, 782 (2016).

9. Ravipati, D., Karreddi, P. & Patlola, A. Real-time gesture recognition and robot control through blob tracking. in *2014 IEEE Students' Conference on Electrical, Electronics and Computer Science* 1–5 (2014). doi:10.1109/SCEECS.2014.6804526.

10. Pyo, Y., Cho, H., Jung, L. & Lim, D. *ROS Robot Programming*. (2017).

11. Joseph, L. *Mastering ROS for Robotics Programming*. (Packt Publishing Ltd, 2015).

12. Fernández, E., Crespo, L. S., Mahtani, A. & Martinez, A. *Learning ROS for Robotics Programming*. (Packt Publishing Ltd, 2015).

13. nodelet - ROS Wiki. http://wiki.ros.org/nodelet.

14. dynamic_reconfigure - ROS Wiki. http://wiki.ros.org/dynamic_reconfigure.

15. dynamic_reconfigure/Tutorials/HowToWriteYourFirstCfgFile - ROS Wiki. http://wiki.ros.org/dynamic_reconfigure/Tutorials/HowToWriteYourFirstCfgFile.

16. dynamic_reconfigure/Tutorials/SettingUpDynamicReconfigureForANode(cpp) - ROS Wiki. http://wiki.ros.org/dynamic_reconfigure/Tutorials/SettingUpDynamicReconfigureForANode%28cpp%29.

17. rqt_reconfigure - ROS Wiki. http://wiki.ros.org/rqt_reconfigure.

18. roslaunch - ROS Wiki. http://wiki.ros.org/roslaunch.

19. Horn, B. *Robot Vision*. (1986).
20. Ude, A. *Robot Vision*. (2010). doi:10.5772/222.
21. Hall, E. Fundamental principles of robot vision. in (1993). doi:10.1117/12.150210.
22. Home. *OpenCV* https://opencv.org/.
23. Blob Detection Using OpenCV (Python, C++) |. https://learnopencv.com/blob-detection-using-opencv-python-c/ (2015).
24. OpenCV: cv::SimpleBlobDetector Class Reference. https://docs.opencv.org/3.4/d0/d7a/classcv_1_1SimpleBlobDetector.html.
25. What is software development? | IBM. https://www.ibm.com/topics/software-development.
26. Atlassian. What is Continuous Integration. *Atlassian* https://www.atlassian.com/continuous-delivery/continuous-integration.
27. Fowler, M. Continuous Integration.
28. Git - About Version Control. https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control.
29. Git. https://git-scm.com/.
30. Atlassian. Learn Git- Git tutorials, workflows and commands | Atlassian Git Tutorial. *Atlassian* https://www.atlassian.com/git.
31. Laurent, A. M. S. *Understanding Open Source and Free Software Licensing: Guide to Navigating Licensing Issues in Existing & New Software*. (O'Reilly Media, Inc., 2004).
32. Corbly, J. E. The Free Software Alternative: Freeware, Open Source Software, and Libraries. *Information Technology and Libraries* **33**, 65–75 (2014).
33. What is open source? | Opensource.com. https://opensource.com/resources/what-open-source.
34. Verma, M. Advantages & Disadvantages of Open Source Software, Explained! *Quick Code* https://medium.com/quick-code/advantages-disadvantages-of-open-source-software-explained-2fd35acd413 (2020).
35. Pros & Cons of Open Source in Business. *PDF Blog | Investintech PDF Solutions* https://www.investintech.com/resources/blog/archives/7975-pros-cons-open-source-business.html (2018).
36. Atal, V. & Shankar, K. Developers' Incentives and Open-Source Software Licensing: GPL vs BSD. *The B.E. Journal of Economic Analysis & Policy* **15**, 1381–1416 (2015).
37. rostopic - ROS Wiki. http://wiki.ros.org/rostopic#rostopic_hz.

Appendix

Appendix1. GitHub repository of the developed blob detection functionality:
https://github.com/ut-ims-robotics/opencv_apps/tree/blob_detection_nodelet

Appendix2. GitHub repository of the demos of the developed blob detection functionality:
https://github.com/ut-ims-robotics/opencv_apps_demo

Non-exclusive licence to reproduce the thesis and make the thesis public

I, _____Hui Shi_____,

(*author's name*)

grant the University of Tartu a free permit (non-exclusive licence) to

reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, my thesis

_____ Expanding the Open-source ROS Software Package opencv_apps with Dedicated Blob Detection Functionality_____,

(*title of thesis*)

supervised by ___Karl Kruusamäe, Sandra Schumann___.

(*supervisor's name*)

2. I grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in points 1 and 2.

4. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

author's name Hui Shi

27/05/2022