Software Development in a Multi-Agent Robotic System

Index

1. Abstract

As a part of a voice-controlled multi-agent robotic system, this research built several submodules covering a wide range of applications including object detection, self-localization and mapping and so on. The main goal of this research project is to develop a framework for multi-robot collaboration and by providing them with the ability of completing complex tasks like searching and bringing objects. By illustrating the overall structures and separately describing each submodule, this report gives detailed instructions of a multiagent robotic system that we have extended based on a previously developed modules, and we will explain how the software was developed. Also, the report can serve as guidelines to future researchers on the project for a quick start of the whole project.

2. Introduction

The multi-agent robotic system is becoming a more and more important part of robotic researches. Not limited to the types of the robots, multi-agent robotic system provides developers with expandability so that different robots can be added conveniently for specific tasks. Multiple robots also mean more possibility of robust design and higher efficiency compared to a single one. However, designing an effective collaboration platform for multiple robots is challenging due to many factors like methods of communication and calculation capacity of the whole system and of each separately. With a good encapsulation of communication protocols, ROS (Robot Operating System) gives developers good chances of building their multi-agent robotic systems. Inspired by advanced development of machine perceptions and the need of multiple robot collaboration, we worked on the software architecture of a multiagent robotic system consisting of two wheel-robots, a drone and one Amazon's Alexa. This project is aimed at developing software supporting practical tasks including object detection using vision techniques, routeplanning-based motion and SLAM (Simultaneous localization and mapping), and therefore makes exploration and development for industrial use – more complex tasks like fetching object.

The report is organized in the following way. The first part illustrates the background of the project and describes some related work. The Experiment Setup section describes the overall architecture of the proposed system and gives details about each submodule. The last part concludes the report and states our expectation for further works.

3. Experimental Setup

3.1. The structure of our system

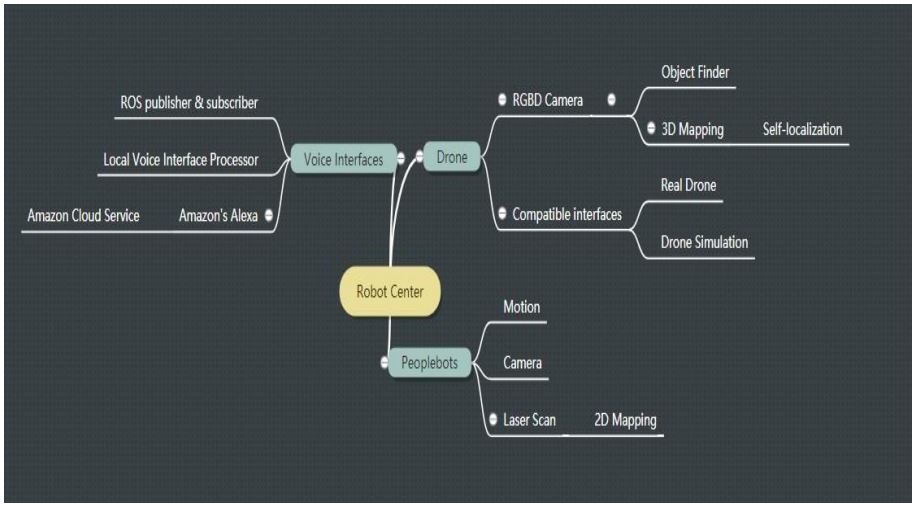


Figure 1 System Architecture

At current, the multi-agent robotic system is equipped with two identical wheel robots, one drone and one Amazon's Alexa. The core part of the whole system is the information sharing. Taking fetching objects for example, according to our design, the drone is used for information gathering like object detection. Then it signals wheel robots to go to a specific place. After choosing a better wheel robot regarding the route costs and job occupation, the chosen one would march to the location provided by the drone. At this level, the user manipulates the gripper to reach the object using the voice control system. Finally, the system sends back the robot (with the object) to the user's location. Every task may involve several-time communication and support of multiple robots. Thus, how to optimize the communication methods and how to make full use of each robot is the essence of the system design.

As list is the basic information of our system. a) Hardware Environment

- Wheel Robots: Peoplebot
- Drone: Erle-copter Erle-Brain 3
- Voice Parser: Amazon Alexa

b) Software Environment

- All of the robots are equipped with Linux Operating System.
- ROS version: Indigo (Peoplebots) and Kinetic (PC)

c) Module Development

The multi-agent robotic system uses a ROS structure. ROS will compile on the folders recursively in the (ROS_DIR)/src There are two ways to use a package:

1. By apt-get
   ```
   sudo apt-get install ros-{kinetic | indigo}-xxx
   ```

2. By compiling the source
   Developers can add a new directory at the src directory and create a CMakeLists.txt to specify the usage of the package.
   Also, supported by the ROS community, developers can download a ROS package and make revisions based on the files.

   While developing, developers only need to work on his own src/(A_MODULE)/ and use
   ```
   catkin_make –pkg (A_MODULE)
   ```
   only to compile the single package. Use rosrun / roslaunch for their ROS nodes.
   Notes: (http://wiki.ros.org/roslaunch) rosrun:
   to run a ROS node
   roslaunch: a XML formatted file to specify parameters and procedures of rosrun

3.2. Application of ROS

ROS (Robot Operating System) acts an essential role in information transportation and package management. In our project, ROS encapsulates the operations of compilation and network. In order to have further development of the system, one needs to know the basic design ideas of ROS.

ROS provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is licensed under an open source, BSD license[1].
ROS can be defined a middle part of a whole multi-agent robotic system. It is responsible for the work communicating upper and lower level as well as among different robots. It can be described as a huge frame containing the following four important components for development[2].

It provides users with:

a) Encapsulated protocols for communication
   ROS node is able to communicate by publish-subscribe methods. By establishing a local network and bringing all the local ROS nodes online, one can process the shared information.
   For example, if we want to notify the center of our system that a target object is found by one of the robot. Our robot need to “say” that it found the object and what it is.
   In the robot program, there should be lines like:

```
ros::init(argc, argv, "object_seeker");
ros::NodeHandle n; std_msgs::String
msg;
ros::Publisher objectSeekerPub = n.advertise<std_msgs::String>("obj_seeker", 1000); //
Where the obj_seeker is the topic name
msg.data = getObjData().str(); //Here simply encode all the object data in a string data.
Practically, custom messages are used for complex structures.
objectSeekerPub.publish(msg);
```

If we at current type the following command: `rostopic list`

we will find a topic named “obj_seeker”

Rostopic echo is useful to find out what publisher is sending out and therefore a good debugging tool. `rostopic echo (SOME_PREFIX)/obj_seeker`

Correspondingly, the subscriber at the center of the system should look like this:

```
void objSeekerSubCallback(const std_msgs::String::ConstPtr& msg){ … } void
subProcesser(){
ros::init(argc, argv, "object_seeker_sub");
ros::NodeHandle n;
ros::Subscriber sub = n.subscribe("obj_seeker ", 1000, objSeekerSubCallback);
ros::spin();  //enter a loop handling callbacks
}
```

b) Development tools

ROS has many useful tools for developers. However, they are easy to use and we will only give a brief introduction to some of them instead of details of the tools.

- RVIZ: displays visualization of all kinds of information [3].
- rqt_plot: dynamically draws plots(and lines) according to published topics [4].
- rqt_graph: shows dependencies among different parts of the system [5].

c) Thriving community supporting and updating programming libraries ROS community is of high quality. Many companies and users put much of their emphasis on the ROS development. Therefore, besides the basic ROS packages provided by ROS, there are lots of ones supported by companies and individuals. Almost everything is open source, so codes are shared and updated frequently to improve development efficiency.

4. Voice Interface

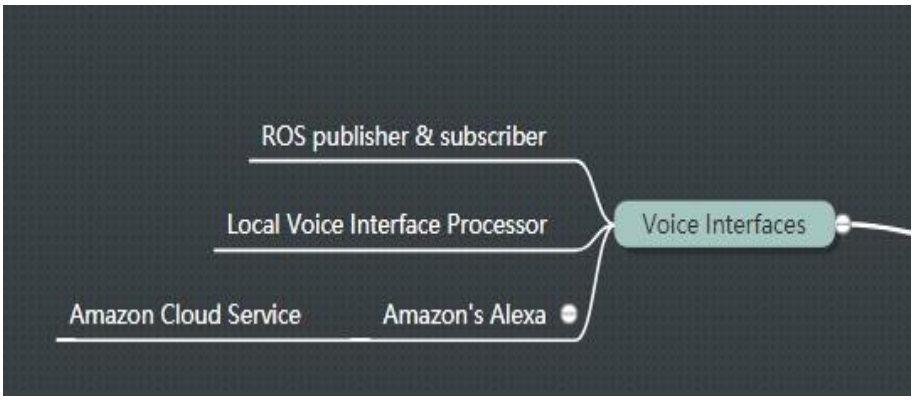


Figure 2 Voice Interface Architecture

Our system is a voice-controlled. Robot actions are triggered by voice instructions. Voice instructions reaching the receiver of Alexa will be parsed by Amazon Cloud Service and the results will be passed to a local parser, which according to the type of the commands, and then be published on ROS. Subscribers listening to corresponding topics will use callback functions to handle the distributed events.

4.1. Introduction to Amazon's Alexa

Alexa is the Amazon's cloud-based voice service available on tens of millions of devices from Amazon and third-party device manufacturers. With Alexa, you can build natural voice experiences that offer customers a more intuitive way to interact with the technology they use every day. The collection of tools, APIs, reference solutions, and documentation make it easy for anyone to build with Alexa[6].

4.2. Usage and coding method

Python part: See the bobby_iot_voice_interface.py C++
part:
The python program above use a ROS publisher broadcast the parsing results of the voice instructions. Voice_interface.cpp will be able to distribute the corresponding messages to different topic subscribers in the system. Then each subscriber handles the message published.

5. Drone

5.1. Real Drone

We use an Erle-copter as the drone part of our system. The drone mainly works for environment gathering, 3D mapping and object detection at current stage. As following, we will introduce methods of drone manipulating, current challenges and some tips for future developers.

5.1.1. Current methods of drone manipulating

One can manipulate the drone by following one of these 3 methods:

APM Planner:

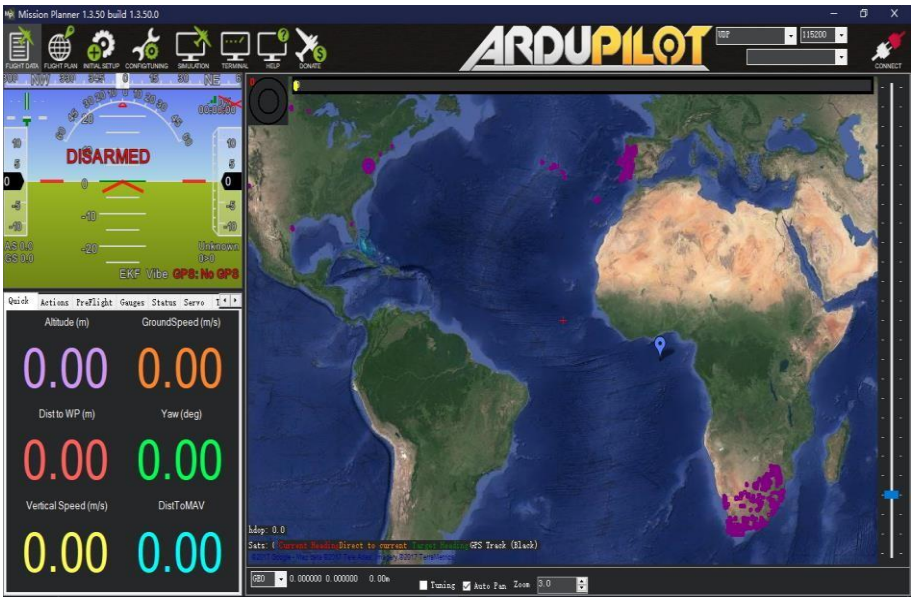


Figure 3 APM Planner

1. Turn on the Erle-Brain 3
2. Waiti for Erle-copter “frambusa” Wi-Fi brings itself online
3. Connect PC to the Wi-Fi
4. Click “Connect” button at the right-top.
   UDP 115200 Local
   Port: 6000 5. Access
   the drone.
   Detailed instructions for calibration, parameter configurations can be found in [7].

RC Controller:

In APM Planner, RC Controller should be calibrated. A regular calibration is expected. Pulling left throttle down right, the drone will be armed and ready to fly. Slowly push the left throttle can lift the drone up. After landing, pull the left throttle down left to disarm it. Switches enables different modes such as ALTHOLD, ROVER and LANDING. Those can also be set in the RC Controller calibration.

Direct Access to Drone Operating System

One can also access to the drone using SSH or a keyboard and HDMI. Remember to use Erle-copter's Wi-Fi to make the SSH.

5.1.2. Current Challenges

We are now confronted with a yaw problem and therefore not able to proceed this part of development.

Problem Description: The drone keeps rotating around yaw axis itself while taking off, and thus do not have enough push force onto the sky.

We have tried several methods to handle the problems:

1. Recalibration of the RC controller, the compass, gyros and ESCs.
   However, those seemed of no effects.
   We noticed:
   a) There are BAD AHRS and ERROR COMPASS VARIANCE problems of the drone but no solutions to them have been figured out.
   b) There is a compass drafting even if the drone lies still on the table. Perhaps there is a magnetic field interference, but we do not think it should be that strong.
2. We changed two ESCs for motors seemingly rotating slower before taking off. However, the drone still cannot find its balance point and rotate around raw axis itself.
3. We disassembled the drone and reassembled it to exclude assembly problems.

5.1.3. Tips for future drone developers

After several failures, we managed to conclude some analysis based on our experiments and understanding:

Battery: Erle-Brain 3 uses Li-Po battery for flight. The battery is very fragile, and any over-charge or over-discharge may destroy the chemical balance in the battery. It is dangerous to charge it without monitoring. Generally, a complete charge takes around 8 hours. However, we would recommend charging it immediately after use of 10-15 minutes. That charge will only take 1-2 hours.

Raspberry Pie: The "Brain" is actually a raspberry pie. The computer is power-consuming so please use an independent power to supply its need when running programs. Only use the battery as power when using it in a flight.

Electrical Speed Controller (ESC): This electronic device gives instructions to the motors. We replaced two of them with brand new ones but made no effects. Also, we tried to use different electrical current through the ESC to motivate the motors, but nothing worked.

RC Controller: The drone is manipulated by the RC Controller only when the controller module is plugged into the drone. Make sure that the controller module will not be swept by the propellers otherwise the drone will get out of control.

5.2. Simulation Environment

It is unsafe to use the drone if we are trying undebugged codes. The drone itself can be over-controlled if the code does not run in a proper way. That is why we choose a simulation environment and decide to make any test in the simulation environment before downloading the code into the drone. Also, tests in the simulation environment can make development more effectively.

5.2.1 Simulation Environment: Introduction and Installation

The environment structure is as follows:

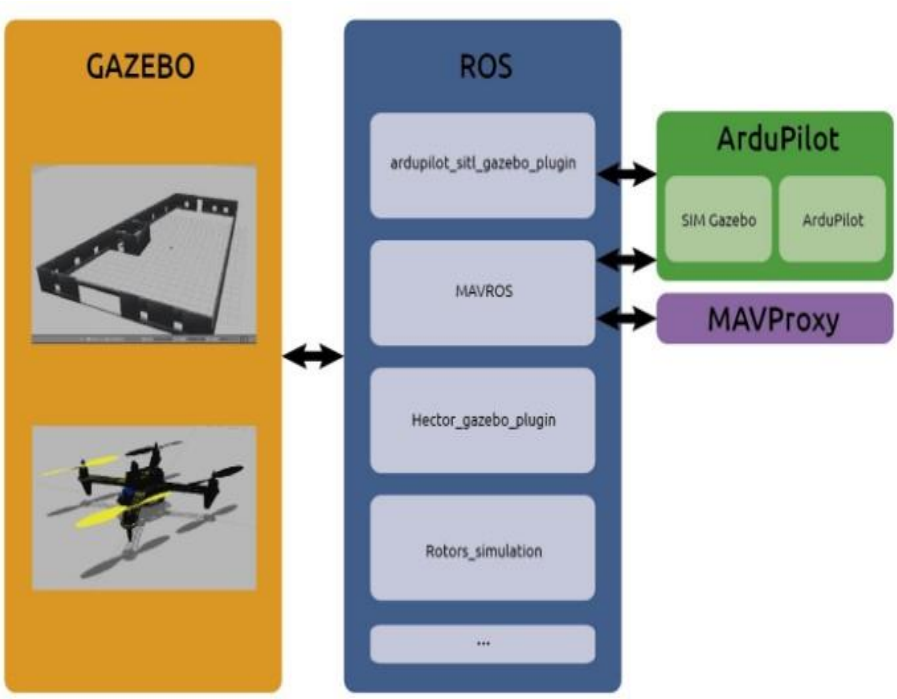


Figure 4 Simulation Architecture

Installation[8]

Gazebo is the simulation environment with ROS interfaces. The general idea is to send instructions using publisher and correspondingly, Gazebo subscribes the topic and informs the drone.

Note: It would be better to use ros-indigo-xxx for the simulation because some of the kinetic packages do not support perfectly. However, ROS kinetic can also work if properly configured.

5.2.2 Command Line Mavros Manipulation[9]

In one terminal, source your ros directory devel/setup.bash

```
cd ~/simulation/ardupilot/ArduCopter
../Tools/autotest/sim_vehicle.sh -j 4 -f Gazebo
```

Source your ros directory devel/setup.bash
In another terminal, source your ros directory devel/setup.bash `roslaunch ardupilot_sitl_gazebo_plugin erlecopter_spawn.launch`

Now, in the first terminal, the drone will follow instructions like:

```
mode GUIDED
arm throttle
takeoff 2 &
mode LOITER
arm throttle
rc 3 1600 rc
3 1500
```

5.2.3 C++ Manipulation

To show the vision of the virtual drone:

```
Rosrun            image_view            image_view            \
image:=/erlecopter/front/image_front_raw
```

Take off in C++

```
ros::ServiceClient arming_cl =
n.serviceClient<mavros::CommandBool>("/mavros/cmd/arming");
mavros::CommandBool srv;     srv.request.value = true;
if(arming_cl.call(srv)){
        ROS_ERROR("ARM send ok %d", srv.response.success);
    }else{
        ROS_ERROR("Failed arming or disarming"); }
```

As we can see, ROS srv /mavros/cmd/arming called. It is the ROS interface of Gazebo Simulation.
Furthermore, take the vision for example. The topic is /erlecopter/front/image_front_raw. We can use launch file remapping from that topic to a target topic, image input of find_object_2d for example, therefore we can use find_object_2d package for the object detection in the virtual environment.

5.2.4 Simulation-Real Interface

Actually, the vision example above is a method of transplantation. The major distinctions between a virtual drone and a real drone are the methods of override.
Here [10] is the override methods for a real drone.

After the completion of a real drone, just remap the original interfaces to a new name will run.

5.2.5 Development Guidelines

It is dangerous to make attempts on a real drone with a undebugged code. That is why we develop a simulation environment. Every time new codes are finished, they are supposed to be tested in a virtual environment to guarantee that no mistake occurred in virtual flights. However, there is differences between software simulation and hardware. Therefore, be careful even it seems a perfect code in the simulation environment.
Please check example files in the simulation part to find out all the override methods and how each service is used. It is maybe a good idea not to use any service and compile your own service for a specific task to improve the program efficiency.

6. Peoplebot

6.1. Motion

Motions of Peoplebot involves more advanced programming techniques. It directly communicates with hardware APIs. For instance, it needed mutex conflict handling while programming in case of access sharing districts.

6.1.1 Corresponding code analysis

```
cmdvel_sub = n.subscribe( "cmd_vel", 1, (boost::function
<void(const geometry_msgs::TwistConstPtr&)>)
   boost::bind(&RosAriaNode::cmdvel_cb, this, _1 ));
direct_motion_sub                                          =
n.subscribe("direct_motion_command",1,(boost::function
<void(const                          std_msgs::StringConstPtr&)>)
boost::bind(&RosAriaNode::direct_motion_cb, this, _1));
```

This is a part of the code from motion manipulation. It shows how to use mutex lock and how to bind callback functions, which serves as handler to corresponding calls. boost::function and boost::bind are used here for mutex conflict handling. The speed/rotation settings are sharing areas so that an unsynchronised operation may cause fatal errors. (Like “mutex error / multi-thread error”) And their corresponding callbacks are as follows, if we only extract the very core part of the operations(rosaria_modified.cpp):

```
/*Here are the messages from voice interface processor*/
dirCommBool=true;       unsigned int wordCount =
countWords(msg->data);         vector<string> command =
splitStringToWords(msg>data);  ...
/*Handle the voice instructions*/
     int  distance  =  stoi(command[2]);  //parse  instructions
veltime = ros::Time::now();      robot->lock(); //Mutex Lock
        robot->move((double)distance); // Rosaria API
    robot->unlock(); //Mutex Lock Release As for
speed control callback: robot->lock();  //Mutex Lock
robot->setVel(msg->linear.x*1e3);
if(robot->hasLatVel())
robot->setLatVel(msg->linear.y*1e3);
 robot->setRotVel(msg->angular.z*180/M_PI);
robot->unlock(); //Mutex Lock Release
```

The process is almost identical and the only differences are the Rosaria API calls.

6.1.2 C++ Manipulation of Peoplebot Motions

This part is in the commandparse.cpp in the center computer. e.g.

voice instruction: “goto” {alice | bobby} goto room {1 | 2}
{robot_name}/move_base
move_base_msgs::MoveBaseGoal goal

goal.target_pose.header.frame_id = "map";
goal.target_pose.header.stamp = ros::Time::now();

goal.target_pose.pose.position.x = xxx;
goal.target_pose.pose.position.y = xxx;
goal.target_pose.pose.orientation.w = xxx;

voice instruction: “move” / “turn”
{alice | bobby} {(move {forward | backward} {$Distance}) | (turn {left | right} {$Degree})}
/{robot_name}/direct_motion_command
std_msgs::String direct_cmd;
direct_cmd.data = {robot_name} + " " + {move_motion}+ " " + [distance];

6.1.3 Route-planner

The route planner is actually an analyzer giving instructions. It synthesizes virtual map and several algorithms including shortest path, collision avoidance, etc., to provide a theoretical best way for the peoplebots.

```
DWAPlannerROS[11]
The package provide local costmap and global costmap.
~<name>/global_plan (nav_msgs/Path)
The portion of the global plan that the local planner is currently attempting to follow. Used primarily for visualization purposes.
~<name>/local_plan (nav_msgs/Path) The local plan or trajectory that scored the highest on the last cycle. Used primarily for visualization purposes.
```

There are various parameters here for route planners. For example, different weights for different planners are combined together for a final route.

6.1.4 Troubleshoot

Route Planner:

If there is a problem like robots cannot move through the doors, check the route pruner. The pruner is able to cut off the original route to get a shorter path. However, when there is a threshold of collision-avoiding, the pruner's plan could be unreachable. That is why the robots cannot move through the door. By lower the collision-avoiding threshold or simply temporarily turn off the pruner may solve the problem.

6.2. Camera

This is a camera for the drone. The camera gathers colors and depth information for the drone, enabling it to build 3D map, detect target objects and follow specific route. At current stage, the camera works independently. After the completion of real drone, it will be assembled by USB.

6.2.1. Basic information of the camera

VC-C50i

- Pan/Tilt/Zoom camera with 9 preset positions
- 26x Optical Zoom with powered telephoto lens
- Superior performance in low-light conditions
- Built-in Infrared illuminator
- Easy set up and advanced connectivity
- Cascade control facility for up to 9 cameras [12]

This camera uses a motivator called ArVCC4, which provide hardwaresoftware interface for rosaria.

6.2.2. Mechanism of the camera manipulation

The camera is controlled by a library provided by the manufacturers. It uses a standard interface called ArVCC4. We encapsulated a sample to a header file so that one only need to get familiar with KeyPTU.h in rosaria in peoplebots to revise this part of manipulation.
Use the constructor by connecting robot directly to rosaria keyPTU.h

```
myPTU(robot),
myDriveCB(this, &KeyPTU::drive)
(rosaria_modified.cpp)
KeyPTU ptu(node->robot);
```

The process is quite similar to the previous mutex lock process:

```
robot->lock();      //Mutex Lock
myPTU.SOME_OPERATION();   //See   KeyPTU.h   in   the   peoplebot
robot->unlock(); //Mutex Lock Release
```

It is written as a part of the rosaria_modified.cpp in the form of a callback function and registered in the constructor.

6.2.3. Data transportation

Our current information transportation is limited by Local Area Network. Therefore, methods of compressing images (with loss or without loss) are expected to reduce the occupation of bandwidth. After the arrival of the bag transported, the center computer will decode the compression and get the original (a compressed) image.

6.3. Laser Scan & 2D mapping

Laser scan means using the laser in the front of the peoplebot to detect whether there is an obstacle in a fixed distance. By scanning, the robot can tell whether it should move forward or not. Also, if we have a previously made laser scan map. We can use it for our route planner so that our robot can know roughly where is reachable.
As for the 2D mapping, the laser scan process will provide information of obstacle positions. By assembling the information, a 2D virtual map will be built.

6.3.1. API & Introduction

```
<lasername>_pointcloud (sensor_msgs/PointCloud)
Only   published   if   publish_aria_lasers   parameter   is   true.
Provides laser data as a point cloud. <lasername> will be ARIA's
```

```
identifier for the laser. May be repeated with different laser
names if multiple lasers are configured in ARIA robot parameter
file(s).
<lasername>_laserscan (sensor_msgs/LaserScan) Only published if
publish_aria_lasers parameter is true. Provides laser data as a
laserscan type. <lasername> will be ARIA's identifier for the
laser. May be repeated with different laser names if multiple
lasers are configured in ARIA robot parameter file(s).
```

6.3.2. Laser Scan & 2D mapping Development

Further development should be based on the sensor msgs/LaserScan and sensor_msgs/PointCloud.
Take LaserScan for example to build a virtual fence for the peoplebots:

```
# Single scan from a planar laser range-finder
#
# If you have another ranging device with different behavior
(e.g. a sonar
# array), please find or create a different message, since
applications
# will make fairly laser-specific assumptions about this data
Header header               # timestamp in the header is the
acquisition time of
                          # the first ray in the scan.
                          #
                          #  in  frame  frame_id,  angles  are
measured around
                          #     the     positive     Z     axis
(counterclockwise, if Z is up)
                          # with zero angle being forward along
the x axis
```

```
float32 angle_min        # start angle of the scan [rad]
float32 angle_max        # end angle of the scan [rad] float32
angle_increment  # angular distance between measurements [rad]

float32 time_increment   # time between measurements [seconds]
- if your scanner
                         #  is  moving,  this  will  be  used  in
interpolating position
                         # of 3d points
float32 scan_time        # time between scans [seconds]
 float32 range_min        # minimum range value
[m] float32 range_max        # maximum range value
[m]

float32[] ranges            # range data [m] (Note: values <
range_min or > range_max should be discarded)
float32[] intensities      # intensity data [device-specific
units].  If your
                         # device does not provide intensities,
please leave
                         # the array empty.
```

We can use the ranges and intensities to specify what the laser can see so that add a virtual obstacle to it to block the way of the robot. In this way, we will be able to set a invisible fence for our people robots.

7 ORBBEC Astra Camera

7.1. Introduction to the camera and basic knowledge & calibration

Astra and Astra S were developed to be highly compatible with existing OpenNI applications, making both 3D cameras ideal for pre-existing apps that were built with OpenNI. Astra S has a 0.4- to 2-meter range, while Astra has a 0.6- to 8-meter range. (See full technical specs for both below.)
Astra 3D cameras are excellent for a wide range of scenarios, including gesture control, robotic, 3D scanning, and point cloud development.
Astra and Astra S are already in market. Innovative businesses worldwidehave adopted Astra and Astra S for use with their existing OpenNI solutions [13].

Installation of the Camera [14]
We use a package introduced by the official guide, named rostra_camera.

```
cd  /etc/udev/rules.d/ sudo
gedit 56-orbbec.rules
/* Add rules from
https://github.com/tfoote/ros_astra_camera/blob/master/orbbecu
sb.rules*/
sudo service udev reload
```

```
sudo service udev restart
```

Unplug the camera and plug it again.

Use the following command to run the roscore and the camera node:

```
roslaunch astra_camera astra_camera_node
```

7.2. 3D mapping

RGBD camera supports 3D mapping. The basic methods of implementing the 3D mapping is odometry calculation. In other words, to calculate the current position of the robots, the program needs last frame(s) information. Where it used to be accounts for the specific current position. However, when a frame is rapidly change to another, there is some possibility that the odometry calculation cannot match with the expected position and the whole program may be trapped in a mess until the last frame information is supplemented by other odometry calculations.

7.2.1. Our current toolkits

rtabmap_ros:

This package is a ROS wrapper of RTAB-Map (Real-Time AppearanceBased Mapping), a RGB-D SLAM approach based on a global loop closure detector with real-time constraints. This package can be used to generate a 3D point clouds of the environment and/or to create a 2D occupancy grid map for navigation.

7.2.2. API

```
roslaunch rtabmap_ros demo_robot_mapping.launch
```

This launch file provide a method of building a complete 3D map and then map_server can be used to save the created map. At current, a 3D map of the lab has been built and therefore we can set our drone simulation using the 3D map to have a more vivid simulation.

7.3. Self-localization

As the report mentioned above, odometry calculations play a very important role in building a 3D map. Also, they are essential in the sel localization. In fact, the self-localization is based on the 3D map. The technique is called SLAM (Simultaneous localization and mapping). It is based on the target that robots will find its current position without any prior knowledge of the map. Also, the robots will be able to build a complete map during the observations.

In this project, the self-localization program has an approximate estimation of the kinetic model. It recorded every odometry calculation to form very complex odometry transform matrices. Therefore, the way (after odometry

transformation) of reaching every reached point from the very beginning point is recorded. After the whole map is built, the robot will be able to tell where it is.

7.3.1. Our current toolkits

Name: rtabmap

```
roslaunch rtabmap_ros rtabmap.launch localization:=true
```

The localization mode will then be able to scan the whole space and build a map just following the process mentioned above.

7.3.2. Next-step goal for precise localization

Besides 3D mapping, we can also use 2D mapping constructed by Laser Scan and gyro information to have relatively higher precise localization effects.

We would recommend to use robot_localization: robot_localization is a collection of state estimation nodes, each of which is an implementation of a nonlinear state estimator for robots moving in 3D space. It contains two state estimation nodes, ekf_localization_node and ukf_localization_node. In addition, robot_localization provides navsat_transform_node, which aids in the integration of GPS data[15].

7.4. Computer Vision

We also applied computer vision techniques to the development of our robotic system. We provided our drone with perception. Due to the limited of hardware, we were not able to use Neural Network methods for object detection. It is possible if needed. One may transfer a captured picture or video flow to the center computer with GPU to process the picture/video flow there. Then transfer back the results. Undoubtedly, this method can consume large amount of bandwidths and therefore, perhaps not a better choice.

7.4.1. Find object algorithms

We used a package called find_object_2d. The package implemented several algorithms including SIFT, SURF, FAST, etc.

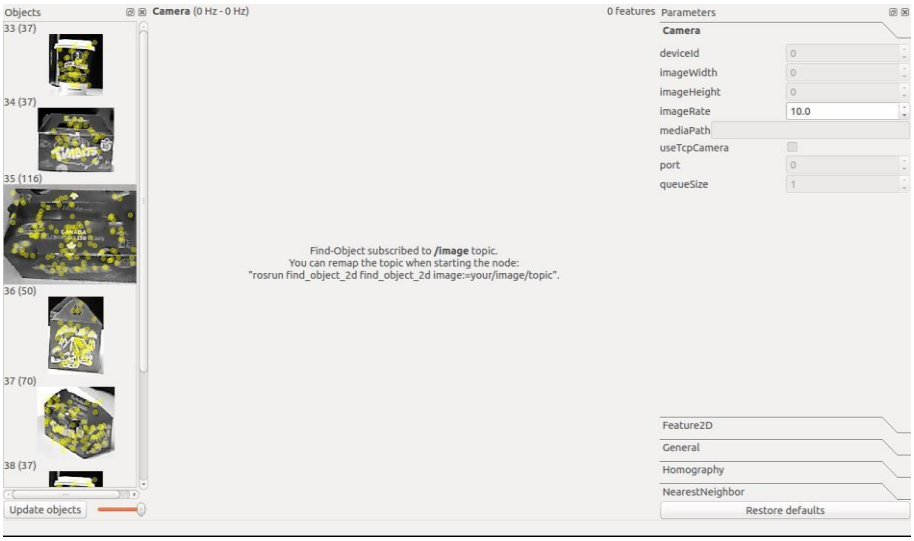
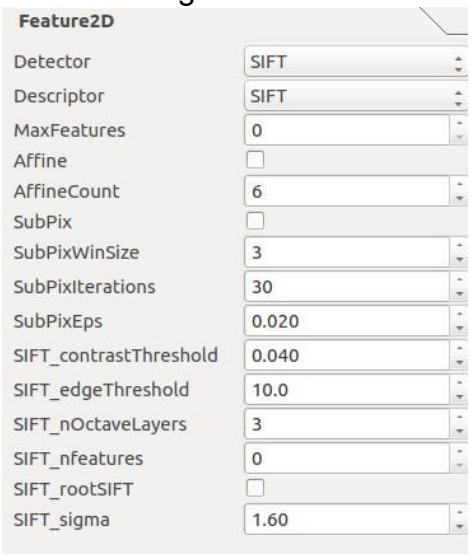


Figure 5 Find Object User Interface

Now we take SIFT for example.

SIFT (Scale-invariant feature transform) is an algorithms make detections according to a given image (given character points). There are invariants in very close frames. By finding the invariants, SIFT managed to “guess” a transformation on the detected object happened between the last frame and the current one.

Then we have the parameter settings as follows:



Furthermore, the program calculated the odometry of the detected objects and the camera itself. So, it can work as an aid to our self-localization. Also, we can use the odometry to precisely fetch the object detected.

7.4.2. Next-step goals

Our next-step goal for vision is to detect objects in a longer distance. Our current capability is around 1 - 2 meters. However, that is not effective enough if we equip our drone with the camera. The drone need to plan a

long distance unless we improve the detection range of the camera and the algorithms
Also, we are working on the number recognition based on the camera. We will enable our drone capturing images and recognize, for example room numbers.

Notes: No matter the drone or the peoplebot, they do not have enough calculation ability to process the object detection, especially number recognition. A transportation using Local Area Network is expected and thus the center computer will be able to process it and give answers to a corresponding robot.

8 Results and Discussion

8.1. Results

The ROS-based multi-agent robotic system is of very good expandability and now is equipped with the features mentioned above: voice_interface, drone_simulation, RGBD camera modules and peoplebot modules. The software developed proved to be of strong robustness and due to the good supports of ROS, we managed to make optimizations for a better efficiency in our system. During the process of development, there are plenty of practical problems such as

1. incompatible version of ROS packages to the ROS system.
2. system library version does not support ROS packages
3. lack of the corresponding dependencies for some ROS packages

Usually, we chose to make revisions in the error source file to support the current versions. Also, adapters for some packages to the system libraries were sometimes needed. Such problems honed my ability of putting forward solutions and bring the solutions to reality.

8.2. Discussion

Most of the development in the report do not involve complex collaborating calculations. Although the whole system has been built with high capability of communication, a more comprehensive program is expected to synthesize each part to prove the efficiency of the multi-agent robotic system.
Regarding the cross-platform communication problems, we managed to setup our design on Ubuntu16.04 x64(Desktop), Debian x64(PC) and Ubuntu ARM(Drone). The main idea of handling cross-platform is to provide interfaces like ROS. If machines support ROS, it will become convenient for us since all the API on different platforms tend to be identical. However, if ROS is not supported on some specific hardware instruction set (for example, different instruction set leads for different implementation of C programming which ROS is based on), we would recommend using C

programming. Using C-based UDP/TCP transportation as an alternative communication method is a good choice. Then for long-term use, an encapsulation of UDP/TCP to publisher/subscriber is expected to be implemented.

Another problem with different operating systems is the difficulty or even possibility of transplantation. One architecture method we would recommend if LAN bandwidth allowing, is to set a central processing unit. Like, in our experiments, the PC, it serves as a central brain for communication. Comparing to wheel robots and drone, PC has much better calculation ability, much larger storage and is adaptable to different powerful GPUs. Therefore, no matter what platforms are involved, the only problem is to publish and subscribe corresponding message topics. Such functions are supported by almost all the operating system enabling network connection.

9 Conclusion

It is clear that a broad range of applications including computer vision, multiagent robotic system design, hardware-software combination, etc. is assembled to our system. This is a strong evidence for the capability and expandability of the whole system. Also, we tried to develop the software standing at the angle of the whole project. Every module has its own clear interfaces so that even a new developer will be able to use the interfaces for other development. This will prove to be a solid base for further development of the whole project. The system is now equipped with the ability of voice control, information transportation and submodules including object detection, self-localization, mapping, etc. Also, the existed modules can serve as good examples for future developers of the system to implement more complex functions.

10 Acknowledgements

I would like to provide my gratitude to my professor Dr. Fakhri Karray and my co-supervisor Dr. Chahid Ouali. They gave me support and encouragement to help me complete the whole project. Also, I would like to thank Mahmoud Abdul Galil and Mahmoud Nasr, who helped me to start the project and provided me with great help in this period of time.

11 References

[1]. “Wiki.”http://wiki.ros.org/

[2]. Gerkey, Brian. “What Is ROS Exactly? Middleware, Framework, Operating System? Edit.”What Is ROS Exactly? Middleware, Framework, Operating System? - ROS Answers: Open Source Q&A Forum, answers.ros.org/question/12230/what-is-ros-exactly-middleware-frameworkoperating-system/.

[3]. “Wiki.”http://wiki.ros.org/rviz
[4]. “Wiki.”http://wiki.ros.org/rqt_plot [5].
“Wiki.”http://wiki.ros.org/rqt_graph
[6]. Amazon Alexa, developer.amazon.com/alexa.
[7]. “Autopilot Setup (Landing Page).” *Autopilot Setup (Landing Page) — APM Planner 2 Documentation*, ardupilot.org/planner2/docs/autopilot-setup.html.
[8]. “Simulation Configuration.” *Ros-Erle Doc*, docs.erlerobotic.com/simulation/configuring_your_environment.
[9]. “Launching Erle-Copter Simulation.” *Atom*, docs.erlerobotics.com/simulation/vehicles/erle_copter/tutorial_1.
[10]. “Wiki.”http://wiki.ros.org/move_base
[11]. “Canon.” *Canon Consumer*, 14 Dec. 2004, www.canon.co.uk/for_work/business-products/network-cameras/vc-c50i/. [12].
“Overriding Radio Controller.” *Ros-Erle Doc*, docs.erlerobotic.com/erle_robots/erle_copter/examples/overriding_radio_controller.
[13]. “Orbbec Astra, Astra S & Astra Pro.” *Orbbec*, orbbec3d.com/product-astra/.
[14]. “Wiki.”http://wiki.ros.org/astra_camera
[15]. “robot_localization Wiki.” *robot_localization Wiki — robot_localization 2.4.0 Documentation*, docs.ros.org/lunar/api/robot_localization/html/index.html.