RoboCupRescue 2015 - Robot League Team

The EXPLORERS (CHINA)

AUTHORS: **WangWei[1], LiaoZhihan[2], WangSheng[3]**

EMAIL: **WEI_NWPU@163.COM**

TELEPHONE: **+8618302976735**

ADDRESS: **Changan campus of Northwestern Polytechnical University,Xi'an,China**

[1]A junior at Northwestern Polytechnical University

[2]A sophomore at Northwestern Polytechnical University

[3] A sophomore at Northwestern Polytechnical University

Abstract:

We are the team named "THE EXPLORERS" from Northwestern Polytechnical University. Our rescue robot can fulfill the task of stepping over the obstacles, mapping, autonomous navigation, and other functions. It has achieved the basic purposes of urban search and rescue. Our Robot bases on Robot Open System, and ideas of our own team were added. At the same time, the mechanical parts and circuits of the robot are all designed independently by our team. We firmly believe that it will have a good performance in the game.

1. Team Members and Their Contributions

- WangWei Controller development
- ChenXiaojun Mechanical design
- Liangli Liuyuan Circuit design
- LiaoZhihan The software design
- WangSheng The software design
- ZuoPanfei Advisor
- LiaoZhihan Operator

2. Operator Station Set-up and Break-Down (10 minutes)

We need at least a table to put our two computers, one of them is used for remote control and mapping, the other is used to display the image that the camera sends back. We just need an operator.

First we need to check the machinery and circuit part of the robot. If it has no problem, just open the power button of the robot, and connect robots and computers through the network.

3. Communications

Rescue Robot League		
THE EXPLORERS (CHINA)		
MODIFY TABLE TO NOTE ALL FREQENCIES THAT APPLY TO YOUR TEAM		
Frequency	**Channel/Band**	**Power (mW)**
5.0 GHz - 802.11a	5GHZ	500
2.4 GHz - 802.11b/g	2.4GHZ	500

4. Control Method and Human-Robot Interface

Control Method

This robot is mainly controlled by minicomputer which is provided with some ROS packages for the robot .Before running this robot ,there should be a WIFI opened by this minicomputer , then another computer (control terminal) with ROS(or rqt) and control platform must connect to this WIFI， log on to minicomputer as a local administrator by the

SSH, after that , set the ROS_HOSTNAME to the host name of the mini computer, then the operator would run on ROS packages build on the minicomputer from the control terminal， later, some of the nodes that we build in ROS would be activated , and each of them has its own specific function (navigation,arm_control,robot_move_control,flippers_control,image_display.....).After checking all of the nodes executive, the operator could use the human-robot interface to public new topics, then there must be some corresponding nodes to subscribe the topic, after the analysis, these nodes would publish a new topic in ROS , if this topic is going to send to the lower machine to control the robot , there would be a node to subscribe these topics which based on CAN Protocol. Through the node message will be sent from PC to the MCU and be resolved. In the end, we control the action of the robot.

For our robot, there would be two controlling methods:

1. Remote teleoperation

The operator could publish some corresponding topics by the human-robot interface to control some movable parts of the robot , then these topics would be subscribed by some nodes , after the analysis , some new topics would be published in ROS .A node based on CAN Protocol would subscribe these topics.

2. Partial autonomy

There would be a disaster scene map on the human-robot interface, the operator could set a goal on this map, then the ROS package--navigation and AMCL would help the robot to reach the goal safely, and at the same time that the robot move to the goal, operator could send message to make robot stop or adjust the camera mounted on the robot to view different disaster scenes.

3. Full autonomy

If the operator selected this mode, the ROS package gammaping would help the robot to move safely by itself, at the same time the robot is moving, the operator need not to send any missions. There would be a real-time laser scanning map displaying on the window.

Human-robot interface

The human-robot interface is mainly based on the rqt which is a Qt-based framework for GUI development for ROS, and we have also added some plugins made by ourselves to this framework, it would run on the control terminal.

So this GUI has the following functions:

1. Controlling module (after clicking specified buttons, operators could send missions to make robot move by this module)

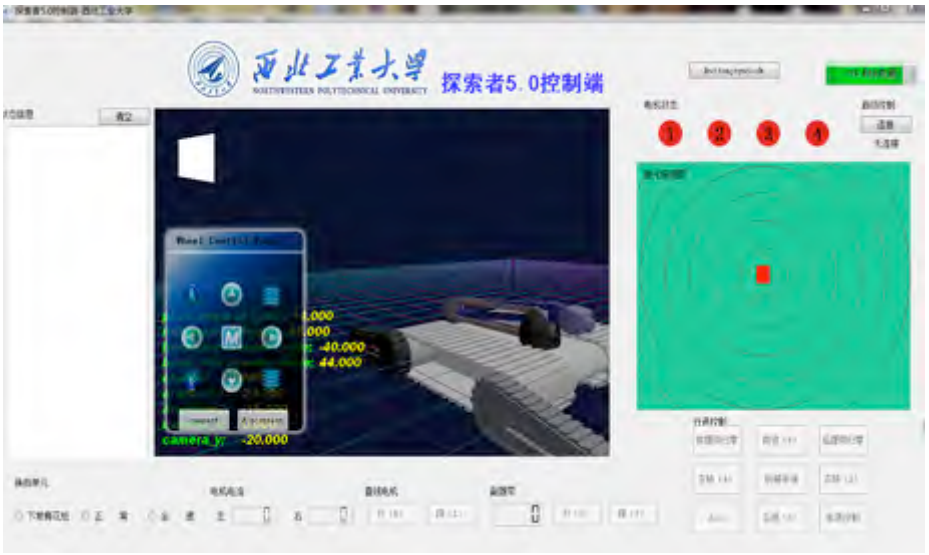
2. Real-time laser scanning map display (while the robot is moving, the map would be changing synchronously)

3. Velocity settings (will be awaken when running navigation package)

4. Camera display (which would get the real-time disaster scene from the camera mounted on the robot)

5. Pose and status of robot (show status and pose of the robot graphically)

6. A stage module (use the ROS package stage_ros , this will show the robot in a occupancy grid map of the disaster scene)



5. Map generation/printing

As our robot is based on ROS, the robot would map automatically with help of ROS package ---- gammaping, this package mainly use the FastSLAM algorithm to generate map, which requires laser scanning and odometry of robot, the FastSLAM algorithm starts with a random distribution of particles. While the robot is moving, the particles and odometry would be changed in real-time. After that, prediction of measurements of the particle would be compared with the actual

measurements. Then the algorithm would assign each particle a weight depending on how well its estimate of the state agrees with the measurements. Finally, it would randomly draw particles from previous distribution based on weights creating a new distribution.

And during the same process, the operators could also change the camera's perspective on the robot and get the view of the disaster scene from the robot, if the operators found out the victims, he/she could click buttons in the Qt control applications, the position would be noted as red pixels in the final map.

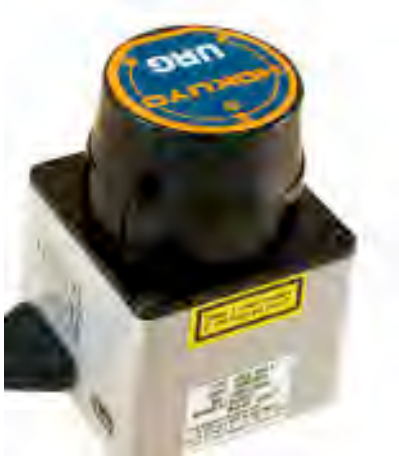
Besides, when we use rviz to fellow the whole process in our task, our operator cold also record the necessary local map and the mapping progress, which could let operator to identify mark victims and arena features of different area when the robot is still working in the stage to map.

After the cycle several times in this algorithm, we could build an occupancy grid map , in which the white pixels represent free cells, the black pixels represent occupied cells , gray pixels are in unknown state and the red pixels represent there may be a victim in that area .

6. Sensors for Navigation and Localization

Our robot is based on ROS, so the robot would use ROS package – navigation and amcl, which requires laser scanning to navigate and odometry to locate the position of robot. Put the 2D laser scanning on the base link of the robot and put rotary encoder near the wheel joint of robot.

Before running the robot, we would set some parameters required by this ROS package, such as the start angle of scan (min_angle), the end angle of scan (max_angle), the time between measurements (time_increment) ,the angular distance between measurements (angle_increment) and the frequency(Hz). When the robot is working, a node in ROS would get the return data from the laser scan, which includes an array stores ranges. After the analysis of these ranges, this node will also publish a new topic to change the pose or velocity of the robot.



7. Sensors for Victim Identification

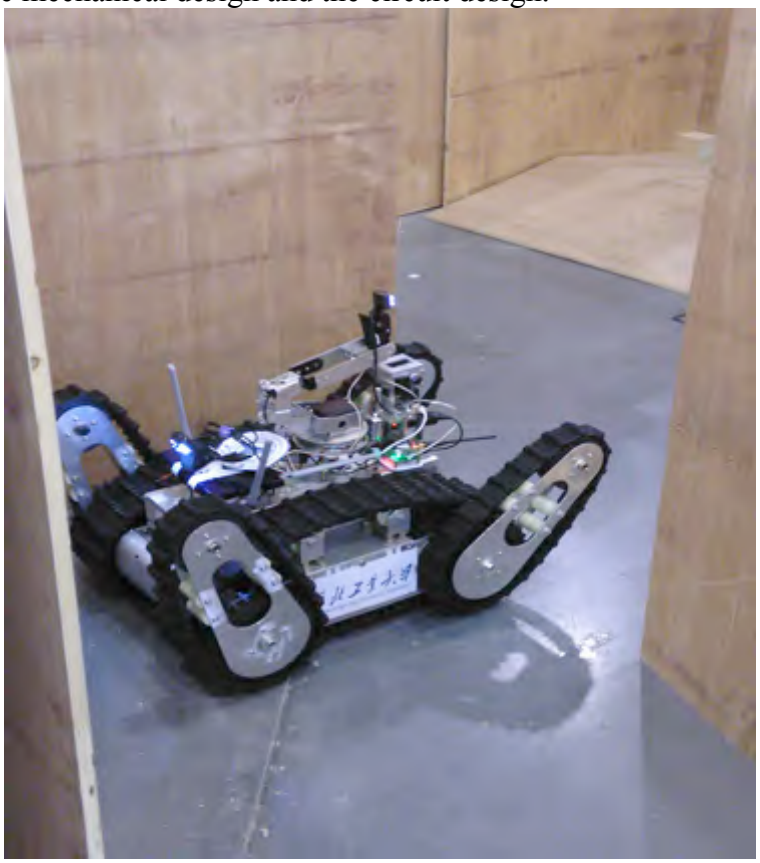
We will use a camera fixed on the robot's arm to get the necessary images from the stage to find victims. The image took by the camera will be operated in the mini pc by four nodes and sent to the controller terminal wirelessly. Our operator will in turn judge victims in images. The victims' localization will be recorded in an array and be marked on the global map.

We will use four nodes provided by image_common which is a stack provided by ROS. The node named camera_info_manager provides a C++ interface for saving, restoring and setting camera calibration information by a service named sensor_msgs/SetCameraInfo. The node called camera_calibration_parsers contains routines for reading and writing camera calibration parameters.

The node named polled_camera defines a ROS interface for requesting images from a polling camera driver which include camera_calibration_parsers and camera_info_manager. Polled_camera will also publish two topics named sensor_msgs/Image and sensor_msgs/CameraInfo, call service named polled_camera/GetPolledImage. We will also use a node named image_transport to change the format of the images such as JPEG to PNG if necessary.

8. Robot Locomotion

After completing the mechanical drawings in the SolidWorks, we divide the mechanical drawings into some components, and add corresponding joints between these components which could generate a ".urdf" file in computer. Afterwards, we use the ROS package – joint_state_publisher (reading the ".urdf" file and finding all of the non-fixed joints and publishes a JointState message with all those joints defined) and robot_state_publisher(computing and broadcasting the 3D pose of each link in the robot) to display this robot in Rviz. If everything is ok, we would add several plugins to this robot, for instance, laser_sensor plugin and skid_steer_drive_controller plugin to make our robot more completed. We use Gazebo to test the algorithm in these packages, and simulate robot in the complex virtual disaster environment. This could give us a better suggestion to make improvements in the mechanical design and the circuit design.



9. Team Training for Operation (Human Factors)

For someone who wants to use our system for controlling robot, he should make sure that he could meet the following requirements:

1. Briefly learning about our robot. He will know the function of our robot and the parameters' effects to the robot, especially to the robot's velocity.

2. Being familiar with the linux shell commend line operation and could start the program by the commend line in terminal.

3. Understanding the laser scanning map, so he could control the robot to move more safely and get useful and important information returned by the camera and laser scan.

Of course, if the user wants to debug this robot by itself, it is recommended that the user is skilled at ROS and simple electrical knowledge.

10. Possibility for Practical Application to Real Disaster Site

After finishing the mechanical design, we build this robot according to ROS's style, and use some necessary ROS packages to make the robot's software more robust. By testing the robot using the method of simulating complex virtual disaster environment in Gazebo, better advises could be given to make improvements in the mechanical design thanks to the robust physics engine of Gazebo. With all the work above, we could make sure that the mechanical is fully adjustable in the real disaster environment.

11. System Cost

minicomputer:	**￥3000**
Hokuyo laser scanning:	**￥7000**
Mechanical parts:	**￥50000**
Circuit part:	**￥5000**
Camera:	**￥1000**
Others:	**￥4000**

12. Lessons Learned

References