SOSVR Team Description Paper
Robocup 2017 Rescue Virtual Robot League

Mahdi Taherahmadi, Sajjad Azami, MohammadHossein GohariNejad, Mostafa Ahmadi, and Saeed Shiry Ghidary

Cognitive Robotics Lab, Amirkabir University of Technology (Tehran Polytechnic), No. 424, Hafez Ave., Tehran, Iran. P. O. Box
{sajjadaazami,14taher}@gmail.com

Abstract. This paper describes the approach used by Team SOSVR of CEIT department of Amirkabir University of Technology for participation in the 2017 RoboCup Rescue Simulation, Virtual Robot competition. We address our methods focused on the autonomous exploration of disaster sites in order to aid victims in a simulated area. Our system provides software solutions for the addressed problem based on ROS framework and uses Gazebo as a simulation environment. In 2017, the team mainly focuses on full autonomy of system which will be possible by improving vision system, robust navigation, efficient exploration and SOSVR utility packages. As a contribution to the RoboCup community, major parts of the used software will be released and documented as open source software for ROS.

Keywords: RoboCup, USAR, Virtual Robot, Gazebo Simulation, ROS, Navigation, Multi-Agent, Victim Detection

1 Introduction

Using autonomous robots in tough situations such as earthquakes, tornados, and urban disasters to aid victims is obviously safer, faster, and more efficient. RoboCup virtual robot league has challenged this problem in a simulated environment. Team SOS of CEIT department of AUT has been participating in 2D simulation league since 2002 with outstanding results. Development of Gazebo, ROS, and new Virtual Robot league in RoboCup lead us to found a new branch of SOS team in order to work on more realistic and crucial challenges. Team SOSVR established in late 2015 within Cognitive Robotics lab of the department driven by the goal of creating a team of heterogeneous robots to be used in USAR situations. The team participated in RoboCup 2016 for the first time focusing on developing a well-documented base code to be used by ROS-Gazebo community.

For RoboCup 2017, we have solved base code challenge and the main focus is on autonomy and vision based multi-agent exploration. Also, heterogeneous robots are going to be used for the first time by the team. Major additions and changes compared to previous year participation are:

1. Development of various Convolutional Neural Networks to create robust victim detection system.
2. Using PID control for navigation purpose.
3. Improving the autonomous state machine.
4. Using heterogeneous team of robots, including UAV and different UGVs.
5. Development of SOSVR Exploration, a novel exploration system for autonomous efficient exploration of UGVs.
6. Improving SOSVR Controller, Human-Robot interface developed in 2016 by Qt framework in order to provide a control panel for human agent to control robot team.
7. Base code release, which is well documented and is available under team's GitHub repository(https://github.com/SOSVR/).

2 System Architecture

In this section, we are going to describe new developments of the previous system. To reach the purpose of robust navigation and exploration, we need a reliable perception system. We have enhanced some parameters of SLAM packages we used before to fulfill our needs. We are planning to work on this part in future, for this year, we have focused on multi-robot exploration, task planning, and victim detection strategies.

2.1 Navigation

PID control is by far the most common way of using feedback in natural and man-made systems. [2]

We have developed a custom PID control loop feedback mechanism in Gazebo controller plugin which continuously calculates an error value e(t) as the difference between a desired set point and a measured process variable and applies a correction based on proportional, integral, and derivative terms which give their name to the controller type.

$$u(t) = k_p e(t) + k_i \int_0^t e(\tau)d\tau + k_d \frac{de}{dt}$$

The controller attempts to minimize the error over time by adjustment of a control variable u(t), in our case u(t) is the speed of robot which we are going to calculate the error arising from a sudden change in rotational speed of wheels that causes some shakes in the robot. the unstable robot may distort explored map so victim detection process will not be effective.

At the formula Kp, Ki, and Kd, all non-negative, denote the coefficients for the proportional, integral, and derivative terms, respectively (sometimes denoted P, I, and D). [4]

For more explanation here you can see effects of varying PID parameters (Kp, Ki, Kd) on the step response of a system. [3]

For better understanding, this is a block diagram of a PID controller in a feedback loop. r(t) is the desired process value or "set point", and y(t) is the measured process value.

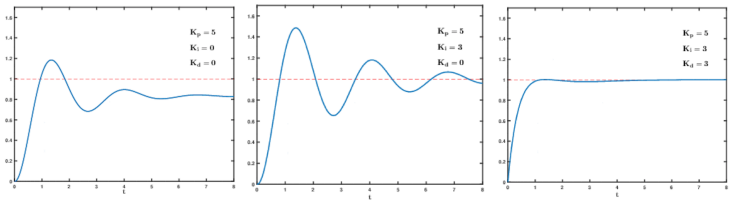

Fig. 1. effects of varying PID parameters on the step response of a system

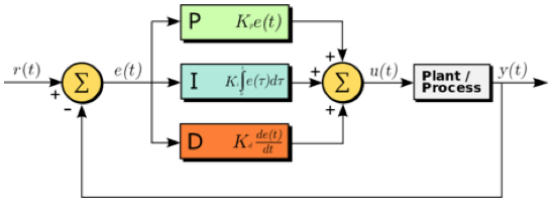

Fig. 2. PID controllers in a feedback loop

2.2 Autonomous Exploration

Controlling the robot by a human operator for long periods of time is hard, time-consuming, inefficient, and in some cases even impossible because either a direct connection or a human operator is not available. In the multi-robot scenario, the situation gets even worse. Since each robot needs at least one operator for reliable functioning a lot of human resources are required. As a result, autonomous exploration is of great importance.

This year we have created our own autonomous exploration system which operates based on different states a robot can have and the overall situation of all other operating robots. Each state specifies the action the robot should undertake to successfully explore its surrounding area and move to the next state. Each State for a given robot depends on the percentage of the overall exploration completed, the percentage of exploration of surrounding area of the robot completed, the previous state of the robot and other factors explained below. These states can form a state machine (as shown in figure 3) which can accurately describe the planning and behavior of our robot, therefore, we have used SMACH to create the and join the state together.

The exploration system has two different modes named joint mode and direct mode. Each one will be explained separately.

Percentage of exploration As described earlier the percentage of exploration which we denote POX from now, is an important factor for the functionality of our system. Figure 4 shows the global and local costmap of our robot used by move base. Assuming we have a resolution of 0.05 and width and height of 36 for our global costmap we realize that the matrix corresponding to the cost of cell

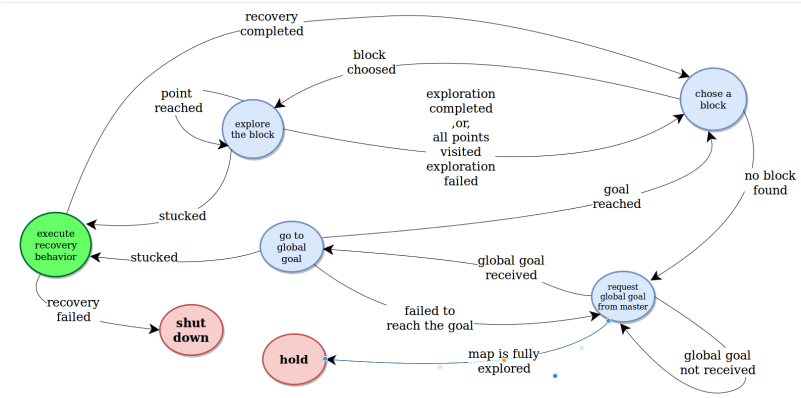

Fig. 3. The state machine of robot during exploration, Red color indicates a final outcome, Blue indicates a state, and Green indicates another state machine.

in the map has 720 rows and 720 columns. Since processing this matrix needs a large amount of computational power we can break this matrix into blocks with a fewer element and smaller size. The size of smaller blocks is mostly arbitrary so we assume we have broken the matrix into 9 blocks with 240 rows and 240 columns each.

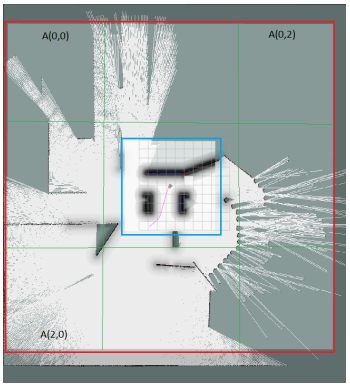

Fig. 4. The red square is the global costmap and the blue square is the local costmap. note that A(0,2) is fully unknown while A(2,0) is almost fully explored.

As shown in the figure 4 for the matrix A the block in the top right corner of the matrix is A(0,2) and the block in bottom left corner is A(2,0). For the block A(n,m) the Px(A(n,m)) is defined as below:

$$Px(A(n,m)) = \sum_{i=n\times 240}^{(n+1)\times 240} \sum_{j=m\times 240}^{(m+1)\times 240} G(A(n,m)_{(\mathrm{i,j})})$$

Where Gx is a function that returns -1 if and only if x = -1 and returns 1 otherwise. For a fully explored block Px is 57600. While for a completely unknown block Px is -57600 and for block and Px = 0 means that about half of the block has been explored. Therefore, we get the following formula for calculating POX of the block:

$$POX_{\mathrm{A(n,m)}} = \frac{Px(A(n,m)) + 57600}{115200} \times 100$$

now that we have the POX of a block we can decide whether we need to explore the block or not. Choosing the minimum POX for a block to be considered for exploration or not is somehow arbitrary but based upon the test it seems 80 percent is a good ratio.

Joint mode In the joint mode, we have a merged map for all robots and we also have the position of all the robots relative to each other. In this situation, each robot will divide the map of its surrounding area to several blocks with equal sizes. then, it will compute how much a block is explored by determining how many cells (pixels)within a block have known values. Based on the current state of the robot it may pick a block to explore. In this case, the robot will try to visit several points in the block which have been chosen by the algorithm. Different outcomes are possible and in the aforementioned state machine all these outcomes are handled and they will result in different states.

Assuming the robot does not pick a block either because it cant find an unexplored block or its not possible to get to a chosen block or even failing to explore a block after doing all the behavior and possible actions robot will signal the master node which in response by checking the global merged map and finding a goal for the robot far out of its surrounding area and with a small number of explored cells.

Direct mode This mode is very similar to the joint mode especially when the robot tries to explore its surrounding area. Since in this mode we have neither a merged map nor the location of robots relative to each other, instead of contacting a master node and waiting for a response from the master node, the robot will now divide its global map into super blocks and tries to find a goal in an unexplored area.

However, if any time during the operation our map sharing node manages to create a global merged map for all robots then direct mode will be disabled and exploration will continue in joint mode.

2.3 Victim Detection

Our implementation is based on Caffe[7]. Caffe is a deep learning framework and training parameters are mainly the same as in original papers. Details are in Table 1. SqueezeNet v1.1 was chosen from different versions of SqueezeNet because of less computation and it is fine tuned for our purpose. We used a Nvidia Geforce GTX 980 with 16 gigabytes of RAM to train our models.[4] To train

Table 1. Training parameters details. Batch size were limited due to lack of memory. Learning Rate is multiplied by gamma every stepsize.

	SqueezeNet
Iteration	2000
Batch Size	64
Base Learning Rate	0.001
Learning Rate Policy	step
Step Size	500
Gamma	0.1
Momentum	0.9

our models we gathered images from three victims(see Figure 5 for samples). 500 images of each victim were selected. We also gathered 1500 random non-victim images from gazebo environment. This 1500 victim and 1500 non-victim images were used as our dataset. We make these images publicly available and encourage others to add images of their own to help gather a large dataset of different kinds of victims for future research.

Fig. 5. Sample of used data set.

Different experiments were conducted to evaluate the performance of the model. First, victim and non-victim images were shuffled separately and 66% of each were chosen for training, the rest were chosen for the test set. Due to randomness in training procedure, the model was trained 5 times and average accuracy on the test set is 99.7%.

Time needed for every forward pass of architectures is reported in Table 2. Our measurements were on two different GPUs(GTX 980 with 2048 Cuda cores and GT 620m with 96 Cuda cores), a general purpose laptop CPU and a minicomputer CPU. As expected, SqueezeNet is fast and runs in a reasonable time on robot's system.

We are planning to use and test other NN architectures like Tiny YOLO until RoboCup event in July.

Table 2. Average test time of every architecture on different platforms.

	GTX980(ms)	GT620m(ms)	Core i5 2.50GHz(ms)	Celeron Dual Core 1.10GHz(ms)
SqueezeNet	2.13	16.63	48.55	173.33

2.4 Human-Robot Interface

SOSVR Panel is our Control Panel visualizer which is a plugin for rqt that allows human operator to see robot state, map, camera feed, and switch between robot controllers. This year, we have updated the plugin, bug fixes have been done and it is publicly available on team's repository. Any contribution on this package is appreciated. See figure 6 for a screen shot.

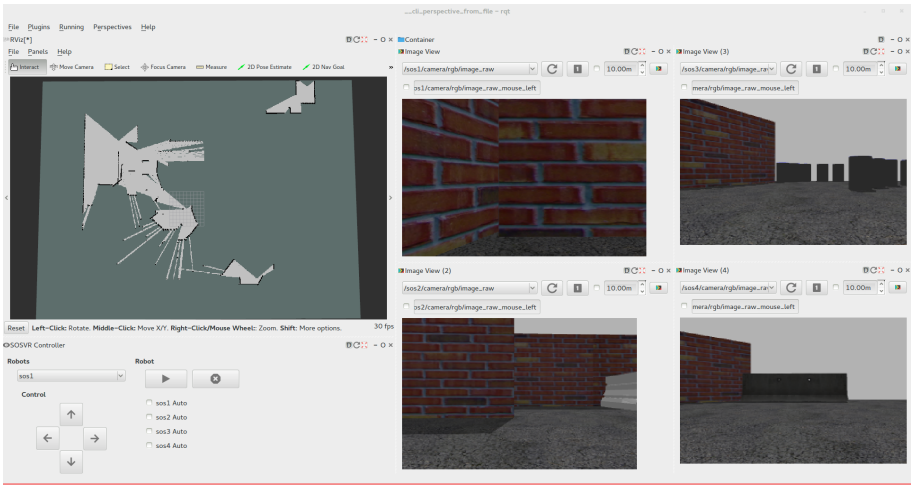

Fig. 6. Screen shot of SOSVR controller. Shared map is on top left. Control buttons are at bottom left and camera feeds are shown in right half of panel.

2.5 Base Code Package

According to our new contributions to the base code released last year, this package is richer and well documented now. It contains standard robot definition(P3AT), SOSVR controller, Teleop, SLAM, SOSVR Exploration, Victim Detection, SOSVR Panel and related Launch files. As such, the new version of base code containing above-discussed works will be released after RoboCup 2017 competition.

3 Innovations

As noted in section 1, this year, we have developed a CNN based robust victim detection system and an exploration system namely SOSVR exploration. Also, we have used PID controller for navigation purpose, improved autonomous state machine and SOSVR controller. Also, we have used heterogeneous robots and their cooperation.

4 Conclusion and Future Works

After all, these new contributions are under development and they need test and improvement. Besides we have in mind to move toward new challenges like 3D mapping, object recognition and manipulation. As well we have to consider realistic wireless communication between mobile robots despite using unconstrained communication, which in near future will be added to the competition.

References

1. Gerndt, R., Seifert, D., Baltes, J.H., Sadeghnejad, S. and Behnke, S., 2015. Humanoid robots in soccer: Robots versus humans in RoboCup 2050. IEEE Robotics & Automation Magazine, 22(3), pp.147-154.
2. Aström, Karl Johan & Murray, Richard M (2010). Feedback systems: an introduction for scientists and engineers. Princeton university press
3. Zhong, J. (2006). PID controller tuning: a short tutorial. class lesson), Purdue University
4. Araki, M. (2009). PID control. Control Systems, Robotics and Automation: System Analysis and Control: Classical Approaches II, Unbehauen, H.(Ed.). EOLSS Publishers Co. Ltd., Oxford, UK., ISBN-13: 9781848265912, 58-79.
5. Iandola FN, Han S, Moskewicz MW, Ashraf K, Dally WJ, Keutzer K. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and¡ 0.5 MB model size. arXiv preprint arXiv:1602.07360. 2016 Feb 24.
6. Taherahmadi, M., Azami, S. & Shiry, S. (2016). SOSvr Team Description Paper: RoboCup 2016, Rescue Virtual Robot League.
7. Jia Y, Shelhamer E, Donahue J, Karayev S, Long J, Girshick R, Guadarrama S, Darrell T. Caffe: Convolutional architecture for fast feature embedding. InProceedings of the 22nd ACM international conference on Multimedia 2014 Nov 3 (pp. 675-678). ACM.