RoboCup Rescue 2016 Team Description Paper RoManSa (South Korea)

Yi Taek Kim, Han Soul Kim, Su Yeon Lee, Hyeon Seok Lee, Dong Hoon Baek, Hyun Gon Kim, Tae Min Hwang and Ju Hoon Back (advisor)

Info

Team Name:	RoManSa
Team Institution:	Kwang Woon University
Team Leader:	Yi Taek Kim
Team URL:	None

Abstract. To begin with, robot that we developed has rubber track on both side. Sub-rubber tracks are designed to help robot to easily go ramp terrain and hazardous environments. In addition, installation of rubber sponge on each rubber track increases friction which can help to run well. Therefore, ROSA can pass the harsh environment and stairs through enough motor's torque and friction between robot and ground. Besides, ROSA's two middle rubber tracks are able to easily separate into four parts, and these modules have advantages which can adjust size of robot in different disaster situations. To be more specific, by the free movement of six robot arms of ROSA can help robot to find victims. ROSA is able to find way using SLAM (Simultaneous Localization and Mapping) and navigation function. Finally, ROSA's all firmware operates based on the ROS. We will present how we accomplished this tasks.

I. INTRODUCTION

Disaster has happened all the time in the world. Almost all of disaster claimed many people's lives and depressed many people. Nuclear accident occurred at March, 2011, in Fukushima. The sinking of the Sewol ferry occurred at April, 2014 in my country, South Korea. We could not do anything for disaster's victims.

So, the team which only consists of undergraduate students at the *Kwang Woon University in South Korea* started project for humanity as participating competition which held in South Korea. The competition was called Mini DRC. We had to pass harsh missions which were made up with obstacle, hurdle, ladder, opening door, closing valve, lifesaving and going down the stairs. Unfortunately, they passed only two missions (obstacle, hurdle). Our robot failed at third mission, climbing the ladder. But we solved network problem which was only solved by two teams of ten teams. In addition, we transmitted video using compressed image to guarantee real time operation. As a result, we had a second place in this competition. Please see the Figure 1. We cannot help but stop making rescue robot because many people are dying all over the world due to various disasters right now. Therefore, we had decided to participate ROBOCUP-RESCUE for humanity. In this paper, we will present our robot 'ROSA'. The RoboCup Rescue competition requires a lot of abilities which can overcome unknown environments. So, we used *'Turtlebot'* open platform which was made by Yujin Robot company in South Korea. First, we connected our robot to '*Turtlebot'* electronically so that ROSA and '*Turtlebot'* could operate together. In addition, we made our robot's main driving part with four modules which make it possible to separate each other easily. Second, we designed the free movement of six robot arm which can help operator to find victim. 6DOF manipulator was made up of eight 'Dynamixel' which generates continuous torque (5.3Nm). Third, we used motor's revolutions, depth camera ('Kinect') and LIDAR in order to implement SLAM (Simultaneous Localization and Mapping) for our robot. Lastly, our team used CCD camera module upon 6DOF-arm to recognize QR code and victim. Therefore, most of all, we are going to participate in yellow, orange and red arena. Additionally, we will try to accomplish blue and black arena.

Fig. 1. Photo of Mini DRC (10/31/2015)

Fig. 2. Photo of ROSA

II. SYSTEM DESCRIPTION

A. *Hardware*

- Locomotion

Basic ROSA's movement operates by rubber track based vehicle. ROSA's main body consists of two parts. One part is middle rubber tracks and the other part is sub-rubber tracks. Please see the Figure 2.

First, we will explain middle rubber track. Most of tracked vehicle has disadvantage which can't change robot size because robot's size is decided by track's length and width. We thought robot's size has to change easily because there are many circumstances according to the size of disaster area and situations. So, ROSA's main driving part was designed to change robot's size. To be more specific, our robot's body is made up with four modules which include motor and Dynamixel individually. Motor provides torque to rubber track, and Dynamixel is designed to adjust sub-rubber track's angle. One module is combined with motor, Dynamixel, shaft and aluminum gears which provide power to middle track and sub-track. Please see the Figure 3.

Fig. 3. Photo of robot's main driving part (one module)

Second, we will present four sub-rubber tracks. Four sub-rubber tracks help robot to get through uneven terrain (orange and red arena). Sub - rubber tracks were designed to work efficiently when climbing stairs or getting through from obstacles. Four sub - rubber tracks are attached at outside of each middle rubber track. If robot is needs to get through the obstacles, sub - rubber tracks helps to adjust the angle of robot to get through the obstacles. Furthermore, if robot needs to climb the stairs, the robot adjusts to proper size for climbing the stairs. We use Dynamixel (XM-430) which was made by ROBOTIS, South Korea, which isn't released yet. Dynamixel can adjust four sub-rubber track's angle very sensitively. In addition, we use mechanical gear ratio (4:1) which is made up with two pulleys between sub-rubber track's frame and Dynamixel. There is long hole in front of sub -rubber track's frame. This hole can adjust tension of sub- rubber track. Please see the Figure 4.

Fig. 4. Photo of robot's sub-rubber track

- Body

ROSA's body frame can be changed by various disaster situation because ROSA is composed of four operation modules. In other words, ROSA's body frame is not fixed and the size and design can always be changed by the distance of the four modules. We will present photo to understand easily. Please see the Figure 5.

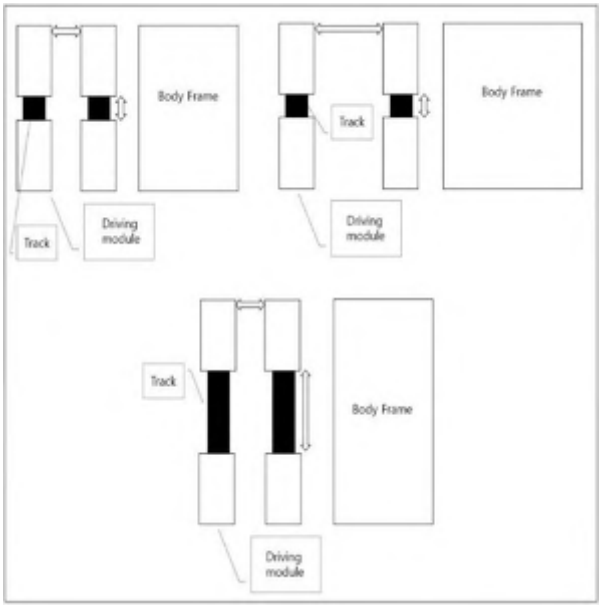

Fig. 5. Photo of ROSA's body frame construction

- Robot Arm

Manipulating objects, finding and checking the victim's condition are the most important tasks in rescue missions. To understand victim's condition very well, Robot must have the degree of freedom. More we have, more we can deal with. In addition, most of the victims are in the box which has a small hole. It is difficult for robot to approach the box deliberately. For these reasons, we designed robot arm which has 6 DOF (Degree of freedom). We referred to the good manipulator like PUMA560. It gave us some tips about how to design the robot arm easily and completely

- Robot arm Hardware

Our robot arm consists of eight motors, 3D printer cylinder, aluminum links and gripper. We used the Dynamixel motors

because it can be connected to each motor easily by using cable. It also has enough power torque for its size to lift the rigid structure. Our robot arm is able to reach 100cm height. So it can observe and touch victims very well. Please see the Figure 6.

The link1 has to tolerate the largest weight and power, so we used two motors to there and one motor used to each remaining joints. The main links can rotate with 55rpm (12V) without considering destruction. And end effector of robot arm is attached to 3DOF which provides the manipulator with a free motion.

Our robot arm's design is very simple because it is designed for considering economy and minimum power to use. We also used 3D printer which can make a complicated frame strong and easily.

We will change the design of robot arm a little bit. To get more power and transmit, we will use a timing belt or another motor in second link and attach other frame for 2D camera. By using the 2D camera, we can examine the rescue circumstance in detail. We used the Raspberrypi2 before but we'll change it to ASUS (VivoPC) and it will become a main board of our team. The OpenCM 9.04C can be connected to Asus board by USB port. The reason why we used Raspberrypi2 before is that we would like to control each part of system comfortably, and Raspberrypi2 is good for examining the robot arm system. Lastly, we'll attach many kind of sensor in robot arm near the end effector so that it'll be able to observe the victim completely.

Fig. 6. Photo of 6-DOF robot arm

- Power Battery

We will use battery that can generate 36V of voltage and 4.4A of current to motor's power input. In addition, we will use extra battery which can provide energy to the main computer and microcontroller. Therefore, two batteries will be used to ROSA because total system can be shut down by motor overload.

- Hardware and Software

  - Driving part

We used *'Turtlebot'* main board to connect *Turtlebot* with ROSA because *Turtlebot*'s open platform is convenient to apply to ROSA. *Tutlebot*'s open platform includes various open source, especially SLAM and mapping. We applied this open source to ROSA with a slight change. We electrically replaced *Turtlebot*'s motor driver with our motor driver which can control ROSA's motor. In addition, we changed digital input which enters in *Turtlebot* to ROSA's motor driver digital input. So, we could control efficiently about robot's locomotion. In other words, *Turtlebot* and ROSA are connected completely.

In the future, we will use different main board, ASUS VivoPC which is installed Ubuntu 14.04 LTS, and we will control Arduino Uno R3 microcontroller through main board's order. Then, microcontroller controls ROSA's motor driver digital input, PWM (Pulse Width Modulation) input, and motor driver can control ROSA's main operation motors (four Maxon motors). We will use remote control for moving ROSA. Then, we will develop autonomous driving function to ROSA gradually. Therefore, ROSA has two functions which are made up with autonomous control and remote teleoperation control. Please see the Figure 7 (Red part).

  - Robot Arm

Robot arm should be connected to other main systems such as main board, camera, and main wheel. So we used the ROS (robot operating system) program which is able to combine each projects by node and master. To do this, at first, we installed the Ubuntu 14.04 in Raspberrypi2 (B) because its compatibility is better than any other OS. We used the Qtcreator to program our own source. We combined Raspberrypi2 and OpenCM 9.04 to control the Dynamixel. Please see the Figure 7 (Blue part).

In the ROS, there are so many packages and information about the Dynamixel, so we were able to use them usefully.

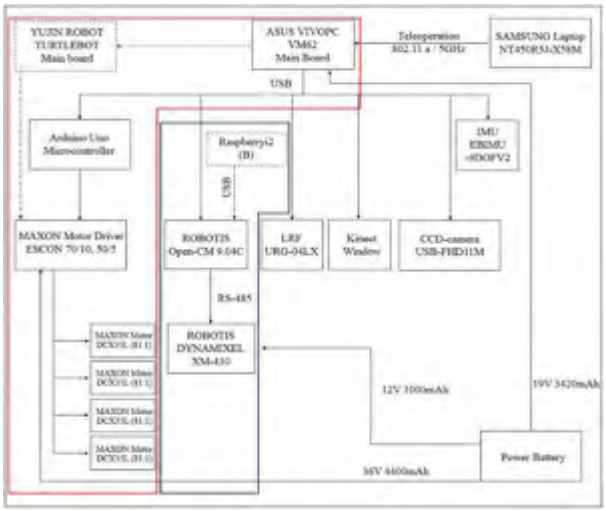

Fig. 7. Photo of all system design

B. *Software*

- SLAM

SLAM (Simultaneous Localization and Mapping) means that robot is to estimate robot's location using an attached sensor. At the same time, it creates a map of an unknown environment. Therefore, SLAM is absolutely necessary in Robocup-Rescue League (yellow and black arena). Typically, sensor that is used on robot to locate estimation is motor's encoder, IMU sensor and LIDAR.

Our team measures the encoder value of the driving attached to the *'Turtlebot'* main board by the revolution of the wheels. The reason why we used the *Turtlebot* and *Kobuki_node* package is to bring up the base launch file of the *Turtlebot*. So, position of the robot is calculated as

approximation through dead reckoning. Dead reckoning can obtain a moving object's location and direction through Encoder value and speed-o-meter without the external sensor. But errors of the calculation are pretty occurred. So, it corrects the position based on information of the environment obtained by the distance sensor or camera. Also, we used *openni_launch* package to operate the Kinect. To increase speed on Kinect mapping at the *rviz*, we proceed the TF process which converts 3D screen printed out by the Kinect into 2D map. We used a *depthimage_to_laserscan* package to convert the distance information data of the Kinect into the image. There are various location estimation method. For example, there are *Kalman filter, Markov localization, Particle filter*, etc. *Kalman filter* has disadvantage of that it applies only to the linear system. But particle filter is applied to a nonlinear system. In the real world, most of the robots and the sensors are non-linear system. So, we'll use the latter algorithm. Finally, we used the *rviz* package to confirm the SLAM result. SLAM through Kinect and Encoder values has disadvantages of slow processing speed because there is a lot of data to process. In addition, the robot tracks the position by itself, but accuracy is low.

Therefore, we also used LIDAR Hokuyo URG-04LX and IMU sensors to implement SLAM. Using two sensors have advantages. First, Lidar gets image to a 2D screen. So, amount of data processing decreases. If the robot position is estimated by using only the encoder value, the considerable error occurs because slope of robot center is not verified. To overcome this weakness, we used IMU sensor which includes acceleration and gyro sensor. So, it can estimate a more exact location. Finally, we tested our SLAM function using two sensors. Please see the Figure 8.

Wheel Encoder – ROSA is equipped with driving module attached to the *'Turtlebot'* main board. Therefore, we get the encoder value through the amount of rotation of the wheels. Odometry data is used for speed control. Also, the slam is based on the encoder value.

RGB-D Camera - We use the Microsoft Kinect sensor. RGB-D camera is mainly used for victim detection and seeing the picture. This camera is mounted on the top of robot.

IMU - The accelerations and angular rate are measured through a 9DOF inertial sensor EBIMU-9DOFV2. The sensor is useful to implementing a slam.

Laser Scanner - ROSA is equipped with the Hokuyo URG-04LX LIDAR. We draw the 2D map using this sensor. URG-04LX is attached at the front of robot. Process speed of data is fast because this laser scanner deals with 2D Axes.

Fig. 8. Photo of SLAM test

- Navigation

In real disaster situation, we will require the robots to find their own way because we don't know what it looks like in real disaster circumstance. Accordingly, we need four essential functions for making navigation. Which are knowing the location of robot, making map, optimized path, and avoiding obstacles.

Our robot, ROSA has measurement functions to know its location. Measurement functions will be calculated by dead reckoning that approximates position. And then we need a sensor to make 2D map. First, we used a Kincet sensor to receive information about x, y, z axis value and then we changed those value into x, y of it. However, this method has the sort of disadvantages. Some of them have low accuracy and speed. So, we decided to use Kinect sensor as looking for victim and added LRF sensor (HOKUYO's URG-04LX). It is one of the greatest way to measure the distance of objects. By using them, we could make detailed 2D map.

To complete autonomous robot, we have to make path searching and planning robot's destination. It assumes the localization where the robot is at now. In this situation, there are many kind of methods for assuming localization of robot. Among the many methods, we decided to use a variant of MCL (Monte Carlo Localization) called AMCL (Adaptive Monte Carlo Localization). The path planning produces a movement path plan (trajectory) from the current location to target point on the map. We will make robot's movement divided into two path planning. One is a global path planning of the entire map and the other is a local path planning of some areas around a robot.

During sensing, estimation of location, and migration route planning, there will be a lot of obstacles. To avoid obstacle, we are planning to use the package of migration route plan such as *Ros's Move_base, Nav_core*, etc. which are based on Dynamic Window. Finally, we tested navigation using only LIDAR sensor. Please see the Figure 9.

Fig. 9. Photo of navigation test
(Yellow: Destination / Red line: Path Planning)

- QR code

In real disaster circumstance, there are not hints about victims or surrounding environment. However, in RoboCupRescue, there are a lot of QR codes. These QR codes include lots of information about victim's location and environment. So, if robot recognizes them, we have enormous amount of information. Due to enormous amount of information, it is necessary to recognize QR code fast and accurately.

First of all, QR codes have regular patterns. They have three big patterns and one small square pattern. So, if these patterns are used with very good algorithm, regardless of any orientation of QR codes, robot can recognize QR codes with right orientation. Thus, we made vision recognition algorithm as following.

First robot browses the surrounding. In this behavior, if robot finds a QR code, robot will recognize the QR code with normal orientation. Then it can recognize 4 patterns of the QR code. After this process (gray conversion, binary code conversion, decoding), robot can recognize the QR code as digital code including 1s and 0s. With these series of data processing, robot can give the information to controller. With that information, our robot can find victims and knowing about surrounding environments.

We will execute this QR code's algorithm on Linux OS(Ubuntu) and Source code is consisted of C language. The source code includes *'opencv'*(opencv.org) and *'zxing'* library. *'opencv'* library is consisted of C language, but *'zxing'* library is consisted of Java language. So our team converts the *'zxing'* library to C language. Lastly, we tested our algorithm which can recognize QRcode. Please see the Figure 10.

Fig. 10. Photo of QR code detection and decoding

- Victim detection

According to the last year contest material, victims had bio-rhythm signals (sound, thermal ...) of their own. Also, many sensors could detect these signals. However, only one camera could detect victims. So, we decided to use one camera to find victim. This doesn't mean that our team will use the only one camera for victims. Although we uses many sensors, we'll make ROSA to detect victims with only one camera. In short, various sensors used on ROSA are subsidiary role of detecting victims. Camera is major role of detecting victims. If the sight is clear, camera is very good sensor for detecting the victims. We will use the camera model 'USBFHD01M'. This model is very simple USB-camera. We will use the vision of this camera to find victims with vision processing.

USBFHD01M Camera – We use the USB-camera to detect QRcode. We'll load a surrounding image through this camera and decode the code in real-time.

- Vision Process

Vision from the camera has information about surrounding circumstance. Human can recognize the circumstance and find the victims. Robot also can see the circumstance but, it can't recognize what victims are at. So we made the algorithm to find the victims as following:

First, robot takes 6 frames of images per 1 second for its surrounding environment. If there is a victim needs to be rescued in the image, robot will stop taking the images and calculate from the previous image. With our vision algorithm, our robot will distinguish singular points of doll's eyes, nose and mouth. Then robot can recognize that there is a victim. After finding the victim, robot can move its arm to the victim's coordinate calculating according to the inverse-kinematics. Then it will transfer the victim's information to operator. Adding to camera's recognition, our robot will find the victims with other sensors (sound-detecting sensor, CO2 sensor, thermal sensor). That sensors are ancillary equipment to find the victims. Therefore, our robot will detect the victims very effectively.

- Moving Process

After recognizing the victim, robot can investigate the victim in detail. To do so, information of victim's coordinate what vision process gives is calculated by the inverse-kinematics, and DH-arm's (our team robot arm name) end effector will go to the victim's location. Then it will execute to the next action.

- Autonomous

If our robot detects the victims in autonomous mode, it can know victim's location by Cartesian Coordinates(x, y). For solving the inverse-kinematics completely, we must know 3 parameters' values. Although our robot can't know the other parameter's value, we can solve this problem. What our robot can do instead of using complete inverse-kinematics is following :

First, with only x, y value (no z value), robot can locate its end-effector in air, in this step its end-effector is in straight line with victim. Then robot moves its arm to victim, keeping it in straight line. Finally Arm's camera recognizes victim's size. Once it recognizes fixed size of victim, robot knows that its arm is near victim. And robot stops its end-effector.

- Manual

If our robot detects victims in manual mode, it will instruct

to the operator that it has detects victims. Then operator must switch the robot's arm in manual mode. We will use *'AVATAR'* controller. 'AVATAR' will be made up of 6 MX-28 (ROBOTIS in South Korea). These motors are a kind of servo motor. These motors can measure the angles itself. If operator handles the controller, controller's MCU communicates with main PC. And it will control value of each motor's angle values. Then main PC will drive the robot arm's each motor and the shape of the robot's arm. After this process, shape of the robot's arm will become just as same as *'AVATAR'* controller.

In this way, our team will operate the robot to detect victims concretely.

C. Communication

We will use ipTIME A604, Wi-Fi adapter in Robocup Rescue League. It operates on the 2.4GHz and 5GHz. Our team uses 5GHz – 802.11 a frequency to teleoperate between our robot and operator via Wi-Fi adapter. The wireless LAN is used for both autonomous mode and teleoperation mode.

D. Human-Robot Interface

To control our team robot, we have to turn on laptop. We will use for both, laptop's keyboard and joystick if we need. In short, our basic controller is laptop's keyboard and when manipulator (6DOF-arm) is controlled by operator, we might use *'AVATAR'*. Our team's potential user isn't decided. However, we will train the operator in various circumstance. For example, when the robot run in harsh terrain that we make with our hand, operator doesn't have to see the robot directly. Operator must control the robot only depending on laptop screen.

III. APPLICATION

A. Set-up and Break-Down

When ROSA operates in disaster situation, we will use laptop computer to remote control and to see robot's sight. To start the competition, we will power up our team's laptop simply and will do remote access to ROSA's main board (ASUS VivoPC VM62) via wireless LAN.

B. Mission Strategy

We will make 2D Map by using SLAM and then move ROSA by path planning through calculation. Victims will be recognized by using sensors and image processing. When we detect victim in location that robot is difficult to reach, ROSA will stretch sub-rubber tracks and use friction of track's rubber to approach victim. So it can reach to the difficult location easily. (Yellow arena)

ROSA will stretch sub rubber tracks so it will be able to go up easily without affecting by steps. And it will be able to go up steps, high ramps and obstacles by using friction of rubber. When ROSA gets stuck by the obstacles, sub rubber tracks will adjust angle of ROSA. So, it will be able to escape obstacles by using sub rubber track. (Orange and red arena)

The Blue arena requires accurate control of robot's manipulator. Therefore, we will use ROSA's 6DOF-arm, DH-arm to pick and to place various objects.

Finally, we will aim to accomplish four arena in Robocup Rescue competition.

C. Experiments

We tested our robot's various features. First, we checked the connection between *Turtlebot* and ROSA through remote controlling ROSA. The result was successful. Second, we proceeded with experiment that we tried to set proper motor's speed through current control in motor driver. Third, we attempted to check whether SLAM does work or not through lots of experiments. We confirmed that map was drawn in *rivz*, and we will develop SLAM function gradually in the future. Finally, we keep trying to improve robot's arm control algorithm.

D. Application in the Field

We didn't try with field experiments yet. However, our robot is enough so that it can be applied to real scenarios because ROSA is pretty strong hardware (body thickness 5mm) and has enough motor torque. We will test ROSA's driving ability in field which will be made like a real environment. For example, ROSA will run on messy concrete bricks, climbing stairs and passing through the obstacles.

IV. CONCLUSION

We will improve ROSA's various function. First, we will modify hardware stronger than now as changing part which will be made into 3D printer with aluminum material. Of course, we will use those parts if we need. Second, we will develop software through improving our algorithm. Especially, for the SLAM, although we implemented SLAM in our robot using open package, SLAM has a lot of modification because it requires much time and effort. Finally, to control robot easily, we are going to construct GUI which makes it able to see ROSA's status in operator station.

TABLE I
MANIPULATION SYSTEM

Attribute	Value
Name	ROSA
Locomotion	Rubber tracked
System Weight	15kg
Weight including transportation case	20kg
Transportation size	0.7× 1×0.7 m
Typical operation size	0.6×0.53×0.35 m
Unpack and assembly time	180min
Startup time (off to full operation)	20min
Power consumption (idle/ typical/ max)	65 / 300 / 500 W
Battery endurance (idle/ normal/ heavy load)	120 / 60 / 30 min
Maximum speed (flat/ outdoor)	0.5 / - m/s
Payload (typical, maximum)	2/ 4 kg
Arm : maximum operation height	96cm
Arm : payload at full extend	500g
Arm : degree of freedom	6 (6 revolution type)
Arm : number of used actuator	8 " Dynamixel" motors
Support : set of bat. Chargers total weight	10kg
Support : set of bat. Chargers power	2500W (100-240V AC)
Support : Charge time batteries (80%/100%)	100 / 150 min
Support : Additional set of batteries weight	2kg
Cost (Total)	USD 9109

APPENDIX A
TEAM MEMBERS AND THEIR CONTRIBUTIONS

- Yi Taek Kim — Team Leader/ Mechanical Design
- Han Soul Kim — Robot Arm Design/ Control
- Su Yeon Lee — Mechanical Design/ Motor Control
- Hyeon Seok Lee — Software Design/ Navigation algorithm
- Dong Hoon Baek — Robot Arm Design/ Control
- Hyun Gon Kim — Software Design/ Image processing
- Tae Min Hwang — Software Design/ SLAM algorithm
- Ju Hoon Back — Advisor
- Geon Woo Park (Supporter) — Video producer
- Jae Hyun Yoon (Supporter) — Translate support

APPENDIX B
CAD DRAWINGS

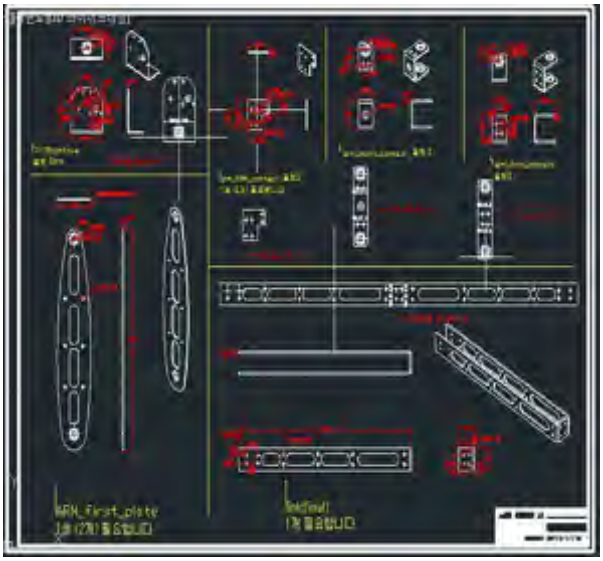

Fig. 11. Photo of 6DOF arm CAD drawing

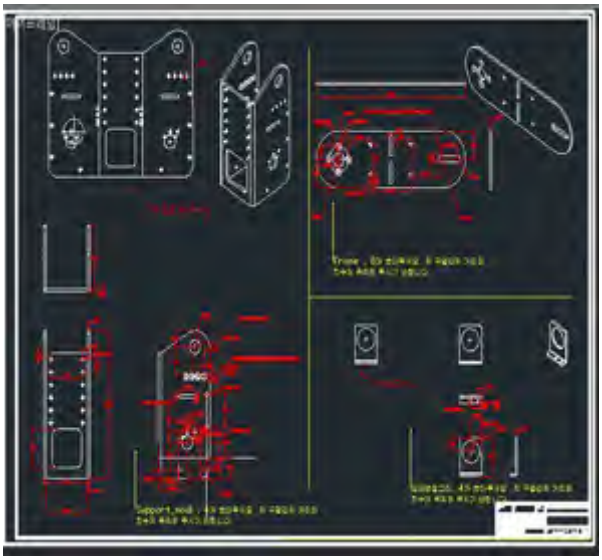

Fig. 12. Photo of driving module CAD drawing

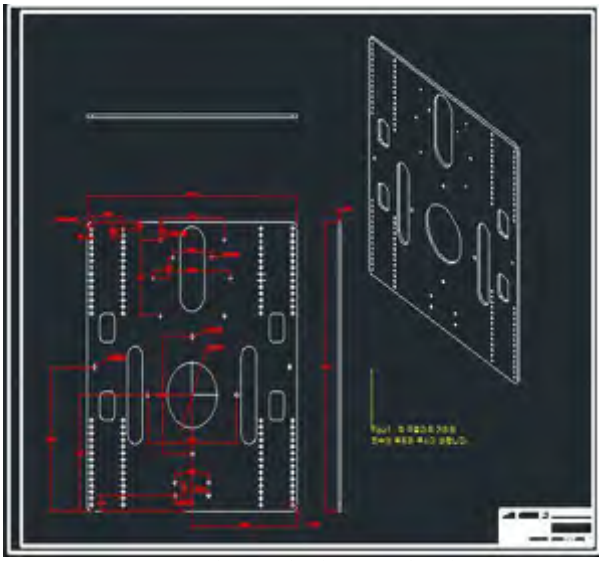

Fig. 13. Photo of body CAD drawing

APPENDIX C
LISTS

TABLE II
OPERATOR STATION

Attribute	Value
Name	Rescue operator
System Weight	1.99kg
Weight including transportation case	2.5kg
Transportation size	0.45 × 0.35 × 0.5 m
Typical operation size	0.38 × 0.25 × 0.23 m
Unpack and assembly time	1 min
Startup time (off to full operation)	10 min
Power consumption (idle/ typical/ max)	15W
Battery endurance (idle/ normal/ heavy load)	9 / 6 / 4 h
CPU	Intel i5-4210U
Cost	USD 903.86

TABLE III
HARDWARE COMPONENTS LIST

Part	Brand & Model	Unit Price	Num
Drive motors*	Maxon DCX35L GB KL	CHF 667.01	4
Drive gears*	Planetary Geared GPX42 81:1		4
Drive encoder*	ENX16 EASY 1024IMP		4
Dynamixel	XM-430 (12V 5.3Nm)	USD 224.83	8
Motor drivers	ESCON 50/5	CHF 229.10	3
	ESCON 70/10*	CHF 336.80	1
Camera	USBFHDO1M	USD 34.01	1
	Kinect Window	USD 123.70	1
IMU	EBIMU-9DOFV2	USD 120.38	1
LIDAR	URG-04LX	USD 1140.00	1
Thermo_graphic	Lepton 500-0659-00 (camera)	USD 178.00	1
CO2 sensor	SEN0159	USD 65.40	1
Microphone	POM-2245L-C10-R	USD 3.30	1
Microcontroller	Arduino Mega Uno (R3)	USD 68.91	1
	OpenCM 9.04-C	USD 14.94	1
	Raspberry Pi 2 Model B 1GB	USD 35.70	1
Computing Unit	ASUS VivoPC VM62	USD 481.53	1
Battery	36V 4400mAh	USD 166.04	1
	11.1V 1000mAh	USD 22.83	1
WiFi Adapter	ipTIME A604	USD 28.14	1
Operator laptop	SAMSUNG NT450R5J-X58M	USD 913.24	1
etc	Kobuki	USD 414.28	1

- We won second place in mini DRC competition which held in South Korea, and we received the 20 Dynamixel motors as a prize. So we were able to save our economic material.
- We marked devices which are supported by Kwang Woon University with an asterisk (*)

TABLE IV
SOFTWARE LIST

Name	Version	License	Usage
Ubuntu	14.04 LTS	open	
ROS	indigo	BSD	
OpenNI	1.5.4.0	BSD	Kinect depth
OpenCV	3.0 alpha	BSD	Victim detection
Hector_SLAM		BSD	2D Mapping
Arduino IDE	1.0.5		Upload board

REFERENCES

[1]. Stefan Kohlbrecher , Johannes Meyer, Thorsten Graber, Karen Kurowski , Oskar von Stryk. Introduction, operator station set-up and break-down, communication and hardware modularity in Hector Darmstadt's TDP. RoboCupRescue 2015, Hefei, China. http://www.robocup2015.org/show/article/90.html

[2]. Stefan Kohlbrecher, Johannes Meyer, Thorsten Graber, Karen Kurowski and Oskar von Stryk. Communication in RRT-Team's TDP. RoboCupRescue 2015, Hefei, China. http://www.robocup2015.org/show/article/90.html

[3]. Farshid Najafi, Mehdi Dadvar, Alireza Hosseini, Soheil Habibian, Hossein Haeri, Mohammad Arvan, Mohammad Hossein Salehzadeh and Alireza Haji Mohammad Hosseini. Software/ hardware architecture in PANDORA team's TDP. RoboCupRescue 2015, Hefei, China. http://www.robocup2015.org/show/article/90.html

[4]. Introduction to Robotics: Mechanics and Control (John J. Craig|Pearson). Inverse kinematic theory, page101~134.

[5]. An improved binarization algorithm of QR code image (Yinghui Zhang ; Chengdu Neusoft Univ., Chengdu, China ; Tianlei Gao ; DeGuang Li ; Huaqi Lin)

[6]. Decoding Algorithm of Two-Dimensional QR Code (Kwang Wook Park, Sang Yong Han, Bo Hyun Jang and Jong Yun Lee -Dept. of Compute Education,† Chungbuk National University Dept. of Digital Informatics and Convergence, Chungbuk National University)

[7]. Joan Sola - Simulataneous localization and mapping with the extended Kalman filter, `A very quick guide... with Matlab code!'. SLAM, EKF-SLAM, Geometry. Pages 2-17. October 5, 2014.

[8]. Heng Zhang, Yanli Liu, Jindong Tan, Naixue Xiong - RGB-D SLAM Combining Visual Odometry and Extended Information Filter. Pages 18742-18766. August, 2015.

[9]. A oroca cafe on website. http://cafe.naver.com/openrt. Kinect-slam instruction reference 2012.

[10]. A oroca cafe on website. http://cafe.naver.com/openrt. URG-04LX-slam instruction reference 2012.

[11]. Sebastian THRUN, Wolfram BUGRARD, Dieter FOX, "Using localization / pose estimation method in PROBABILISTIC ROBOTICS", Cambridge, Mass: MIT PRESS, 2005

[12]. Sebastian THRUN, Wolfram BUGRARD, Dieter FOX, "Using avoid obstacle algorithm in the dynamic window approach to collision avoidance", https://en.wikipedia.org/wiki/Dynamic_window_approach

[13]. Daiki Maekawa, "driving method with navigation and node composition", http://daikimaekawa.github.io/