Fully Autonomous Operations of a Jacobs Rugbot in the RoboCup Rescue Robot League 2006

Kaustubh Pathak, Andreas Birk, Sören Schwertfeger, Ivan Delchev, and Stefan Markov
School of Engineering and Science
Electrical Engineering and Computer Science (EECS)
Jacobs University Bremen
Campus Ring 1, D-28759 Bremen, Germany
a.birk@jacobs-university.de, http://robotics.jacobs-university.de

Abstract — This paper describes the latest version of an integrated hardware and software framework developed for autonomous operation of rescue robots. The successful operation of an autonomous rugbot - short for "rugged robot - was especially demonstrated during several runs at the RoboCup world championship 2006 in Bremen. The design of the autonomous system is described in detail with an emphasis on extendibility and the specific requirements of a typical unstructured rescue scenario.

Keywords: *Autonomous System; Safety, Security, and Rescue Robotics (SSRR); Behavior-Oriented Control; Navigation; Exploration*

I. Introduction

Existing professional rescue robots are optimized with respect to locomotion and ruggedness. They are proven to be useful, field-able devices [1][2][3][4][5][6], but they have their limitations. Especially, they require teleoperation by a user. Due to the high cognitive demands on the operator in purely teleoperated mode [7], any bit of intelligence added makes the systems more useful. There are additional reasons to strive for intelligent functionalities up to full autonomy on the robots. First of all, there are technological aspects like the limitations of communication systems. Second, there are the logistic aspects of rescue operations. Human rescue workers are a scarce resource at accident sites. A single operator should hence supervise as many robots working in parallel as possible. A more detailed discussion of the scopes of autonomy for rescue robotics can be found in [8].

Fig. 1. The RoboCup rescue competition features a very complex test environment (left), which includes several standardized test elements. The Jacobs team demonstrated at the world championship 2006 a combined usage of a teleoperated with a fully autonomous robot (right).

The framework for autonomous operation of a rescue robot presented here is an extension of an earlier system [9]. The new framework has been successfully run in the Robocup 2006 competition, where the Jacobs team made it as the only participant with intelligent functionalities on board of the robots into the final round. Fig. 2 shows a screenshot of the GUI used by the operator to view robot's progress and interact with it on victim detection. Fig. 2 shows a map autonomously generated by the robot during one of the runs at Robocup rescue league. The identified victims are indicated by their numbered IDs on the map. The RoboCup Rescue League in general offers an interesting option to explore the prospects of intelligent rescue systems, as also indicated by related work, e.g., on navigation and exploration for rescue [10] or autonomous victim detection [11].

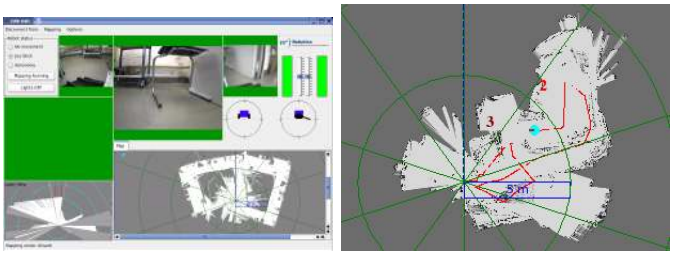

Fig. 2. A screen-shot of the GUI running on an operator station (left). A map completely autonomously created during an actual run at Robocup 2006 (right). The numbers on the map are the victim IDs.

II. Hardware Description

The Rugbot rescue robot platform is a complete in-house development based on the so-called CubeSystem, a collection of hardware and software components for fast robot prototyping [12]. Rugbots are tracked vehicles that are lightweight (about 35 kg) and have a small footprint (approximately 50 cm x 50 cm). They are very agile and fast on open terrain. An active flipper mechanism allows Rugbots to negotiate stairs and rubble piles. Some additional information on the robot as well as its teleoperation software can be found in [13]. A more detailed description of the locomotion mechatronics of the robot is given in [14].

Fig. 3. An autonomously detected victim. The top image shows an overview photo taken by a spectator of the robot after it detected a victim - the arm in the box. The bottom images are the robot's view of a different detected victim. The left image is from the webcam on the robot, and the right image is from the thermocam on the robot.

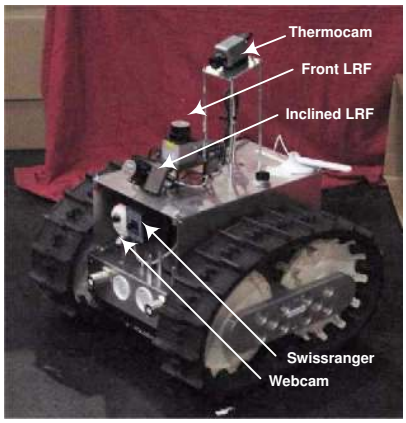

Fig. 4. The autonomous rescue robot *Rugbot* with some important onboard sensors pointed out.

The robot's onboard sensors can be categorized as follows:

1) Odometry Information
   a) The *CubeSystem* has the provision of returning Odometry data using Serial communication.
   b) XSense MTi Gyros for providing the robot heading direction or yaw, pitch, and roll. This yaw is used to correct and recalibrate the drift in the odometry returned by the *CubeSystem*.
2) Cameras
   a) A Panasonic KX-HCM280 pan-tilt webcam with optical zoom.
   b) A Philips USB Cam for front view. Optionally, a back camera, and side view cameras of the two tracks can be installed.
3) 2D Range Detection
   a) Two Hokuyo URG-04LX Laser Range Finders (LRF): One for frontal obstacles (*LRFF*) and another inclined (*LRFD*) for detecting immediate movement impediments like ditches, and blocks. These range finders have a field of view (FOV) of 240^o comprising of about 680 beams.
4) 3D Range Detection
   a) a stereo-camera model STH-DCSG-STOC-C from Videredesign for 3D frontal obstacle range detection.
   b) a Swiss-ranger SR-3000 from CSEM for 3D frontal obstacle range detection.
5) Other sensors
   a) a FLIR thermocamera for temperature information in a range of -40^oC to 120^oC with 0.08^oC resolution.
   b) a CO_2 detector

III. Software Framework

The software framework is coded in C++, and consists of two main modules: a server program running on the robot onboard computer, and a graphical user interface running on an operator station. The communication between these two modules is handled using the Neutral Messaging Language (NML) memory buffers of the NIST RCS library [15]. This is shown in Fig. 5.

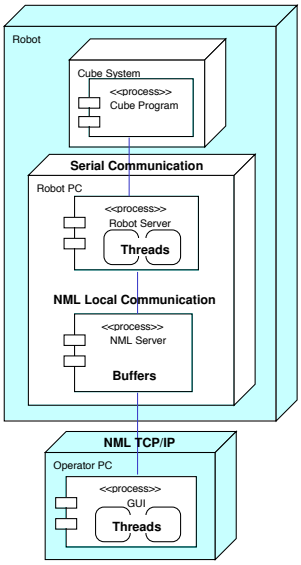

Fig. 5. UML Deployment diagram showing the overall system.

A. NML Server

This server is spawned by the robot server on initialization. Thereafter, it takes care of buffering messages between the robot-server and its clients. Messages arriving in a buffer overwrite previous messages. Currently, the framework has ten buffers. The most important being: the actuation commands buffer for sending operator joystick speed and flipper commands to the robot server, and other buffers for receiving the samples of various sensors, and results of mapping algorithms.

B. The Multi-threaded Robot Server

The server program is multi-threaded and runs on the SUSE Linux O/S. There are separate threads to handle the following tasks.

1) One thread *each* for all the onboard sensors and actuators, viz. Gyro, Cube serial communication, LRF (front and inclined), CO_2 sensor, stereo-camera, swiss-ranger.
2) The image capturing of the themo-cam and various webcams is done by a palantir server [16].
3) One main thread for sensor data collection and NML communication with the client. The latter includes sending actuator and speed commands to the Cube thread.
4) One thread *each* for all the mapping threads. Currently we have a basic occupancy grid based mapping, a SLAM algorithm based on scan-matching [17], and a 3D occupancy grid based algorithm.
5) Autonomy thread for autonomous operation of the mobile robot. This thread analyzes sensor data and generates actuator and speed commands which are sent to the Cube thread via the main thread. The autonomy thread can be started and stopped from the remote operator GUI. This allows the operator to take charge in difficult situations, and drive the robot using a joystick. The autonomy thread in turn spawns auxiliary threads for automatic victim detection. Currently we have implemented the following algorithms.
   a) A thread which detects motion in the webcam image when the robot is stationary.
   b) A thread which scans the thermocam images for warm or heated objects.

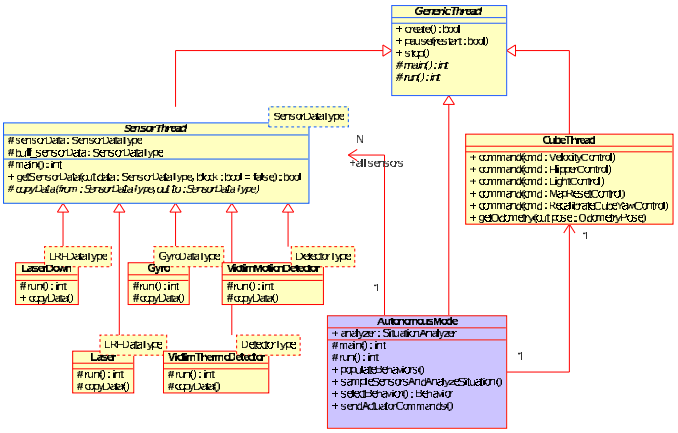

Fig. 6. UML class diagram showing the principal threads involved in the autonomous mode. Mapping and camera threads are not shown for clarity.

As shown in Fig.6, all thread classes derive from a generic thread class. All sensor-thread classes are derived from a templated *SensorThread* class which takes care of mutex locking and data copying. Similarly, all mapping-thread classes are derived from a generic *MappingThread* class framework which takes care of distributing odometry and LRF sensor data to all mappers and transmission of the resulting map to the NML buffer, from where the operator GUI can fetch it.

TABLE I
PARAMETER LIST

V_{Rot}	1250	mm/s
V_{Fwd}	1750	mm/s
ΔT_S	0.5	s
N_{Beams}	682	
h_{LRFD}	400	mm
f^R_{min}	0	
f^R_{max}	3/8	
f^F_{min}	3/8	
f^F_{max}	5/8	
f^L_{min}	5/8	
f^L_{max}	1	
d^F_{min}	280	mm
d^F_{max}	600	mm
$d^F_{\text{IgnoreSide}}$	600	mm
d^F_{SD}	310	mm
d^F_{SHD}	600	mm
d^F_{SideMax}	700	mm
γ	$0.693/(d^F_{\text{SHD}} - d^F_{\text{SD}})$	

C. The Autonomy Thread Main Loop

Remark 3.1 (Basic Idea): A *behavior* is a complex sequence of motions executed by the robot in response to a *situation* detected through its sensors. Behaviors can be aggregated together to form a new behavior. We distinguish between a behavior and a *primitive motion* like in-place rotation or pure translation. There is always an active (current) behavior which *handles* the situation, i.e., computes motion and actuator commands for the robot.

A new behavior is selected at each time instant based on the robot's perception. If a behavior has not finished its complete sequence of motions, it can ask the autonomy thread to consider running it in the next sample instant. This is, however, not guaranteed. This mechanism allows all behaviors to be interrupted in mid-run if a situation with a higher severity occurs which can best be handled by another behavior.

Fig. 8 shows an overview of one step of the autonomy thread (shown as an object of `AutonomousMode` class) main loop using a UML collaboration diagram. The static relationships between participating classes is shown in Fig. 9. At each sample time instant within this loop, all the sensors threads are sampled and their data analyzed to fill in a *current situation* object. The design of the `Situation` class is depicted in Fig. 9. A Situation object $\mathcal{S}$ has boolean flags $\mathcal{S}.f_i, i = 1 \ldots N_{\text{Situations}}$, for flagging various runtime conditions like nearby obstacles, dangerous pitch or roll, whether the robot is stuck or near a fall, whether any of the automatic victim detection algorithms found a victim, etc. Each such flag $\mathcal{S}.f_i$ has its own *statically assigned* priority $\mathcal{S}.f_i.p$. Each flag is associated with an overall *runtime* severity value $\mathcal{S}.f_i.s \in [0, 1]$. Furthermore, this severity value is also provided *directionally*, e.g. if an obstacle is flagged, one can check all the different directions in which an obstacle has been detected, and what the relative severities $\mathcal{S}.f_i.s[\text{Direction}] \in [0, 1]$ are. How situations are analyzed and how severity values are arrived at will be

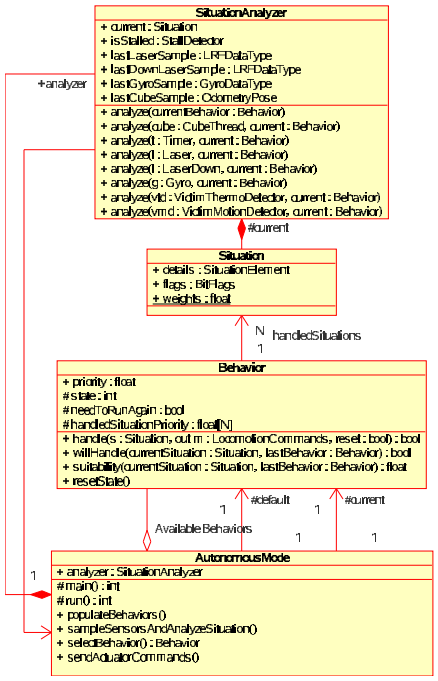

Fig. 7. UML class diagram showing the interrelationships between the classes responsible for the behavior selection process.

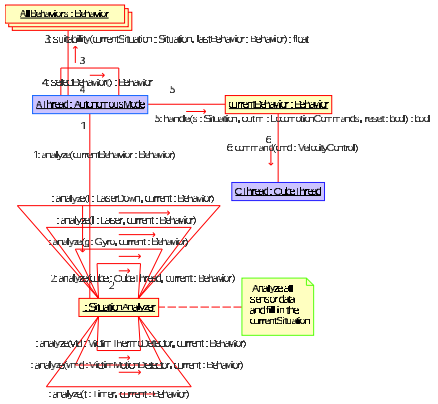

Fig. 8. One step of the main loop illustrated using UML Collaboration diagram.

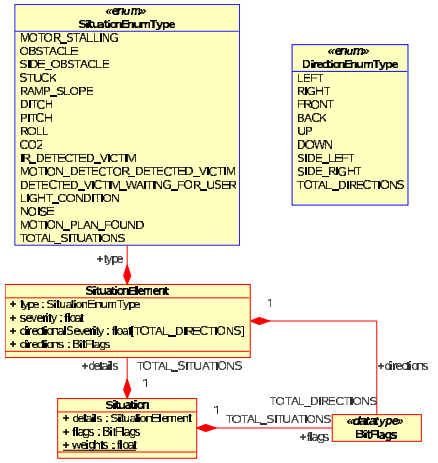

Fig. 9. UML class diagram showing the design of the situation class.

discussed for a few representative sensors in the following Sec. III-D.

D. Analyzing Sensor Data and Assigning Severities

1) Detection of Robot being Stuck: This detector sets the $\mathcal{S}.f_{\text{Stuck}}$ flag. It samples robot odometry (x, y, θ) every ΔT_S seconds and keeps a window of last N_w data sets. It then finds the radial and angular distance between the most recent and the most past data points. If this is within a threshold, the $\mathcal{S}.f_{\text{Stuck}}$ is set. Note that the robot has $\approx N_w \Delta T_S$ seconds to get out of a stuck situation, before this flag is set, and *Back off* behavior takes over.

2) The Front Laser Range Finder (LRFF): The assignment of severities based on LRF data involves experimentation to determine some critical parameters (refer Table I). The basic idea is that if the robot sees no obstacles in its front, it should go ahead ignoring the side obstacles. Otherwise, a side obstacle causes the left and right wheel speeds to be modulated which makes the robot to veer to a side to avoid collision. The left and right severities are assigned as shown in Algo. III.1. The argument *Dir* takes the values of *Left* and *Right*. Thereafter, the front severities are assigned as shown in Algo. III.2.

Algorithm III.1: ASSIGNSIDESEVERITIES(Situation $\mathcal{S}$, Dir)

Find Side beams set $\mathcal{F} = [N_{\text{Beams}} f^{\text{Dir}}_{\text{min}}, N_{\text{Beams}} f^{\text{Dir}}_{\text{max}}]$.
Find shortest beam d in $\mathcal{F}$ ignoring error beams.
if $d \leq d^F_{\text{SideMax}}$
then
- **if** $d \leq d^F_{\text{SD}}$
  - **then** $\mathcal{S}.f_{\text{Obstacle}}.s[\text{Dir}] = 1.0$
  - **else** $\mathcal{S}.f_{\text{Obstacle}}.s[\text{Dir}] = e^{-\gamma(d - d^F_{\text{SD}})}$

else
- **comment:** Obstacle flag not set.

Algorithm III.2: ASSIGNFRONTSEVERITIES(Situation $\mathcal{S}$)

Find front beams set $\mathcal{F} = [N_{\text{Beams}} f^F_{\text{min}}, N_{\text{Beams}} f^F_{\text{max}}]$.
Find shortest beam d in $\mathcal{F}$ ignoring error beams.
if $d \leq d^F_{\text{min}}$
then $\mathcal{S}.f_{\text{Obstacle}}.s[\text{Front}] = 1.0$

else if $d \geq d^F_{\text{IgnoreSide}}$
then
- $\mathcal{S}.f_{\text{Obstacle}}.s[\text{Front}] = 0.1$
- $\mathcal{S}.f_{\text{Obstacle}}.s[\text{Left}] \leftarrow 0.2\mathcal{S}.f_{\text{Obstacle}}.s[\text{Left}]$
- $\mathcal{S}.f_{\text{Obstacle}}.s[\text{Right}] \leftarrow 0.2\mathcal{S}.f_{\text{Obstacle}}.s[\text{Right}]$

else $\mathcal{S}.f_{\text{Obstacle}}.s[\text{Front}] = \frac{d^F_{\text{max}} - d}{d^F_{\text{max}} - d^F_{\text{min}}}$

3) The Inclined Laser Range Finder (LRFD): Initial versions of the autonomous robot only had the front LRF which scans the neighbourhood for obstacles in a horizontal plane above the ground. This is sufficient for mapping, but it can only see obstacles which are at its height (h_{LRFD} in Table I). With this configuration, the robot cannot detect any stairs or dangerous edges from which it might fall down. To enable the autonomous robot to navigate in more difficult terrain, an inclined LRF (LRFD) was installed. The main idea behind this

is to measure the height or depth of obstacles or falls in front of the robot. Subsequently, severities for front, left and right can be (re)calculated. Additionally, the distances of obstacles on the sides can be used for the calculation of the side-left and side-right severities.

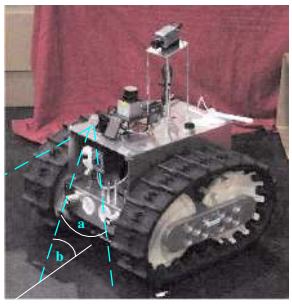

Fig. 10. Schematics of the inclined LRF. All beams lie on an inclined plane. $\measuredangle b = 77^o$.

The device faces forward and is inclined at the relatively steep angle of about 77^o. All its N_{Beams} beams are evenly distributed over the opening angle of the laser scanner of 240^o. Zero degrees hits the ground at the front of the robot and corresponds to beam number 341. About 90 beams to the left and 90 beams to the right still hit the ground in front of the robot. The beams which leave the sensor at 90^o are parallel to the ground, while the first and last beams go backwards in the air. Only the forward-going beams which are between -50^o and 50^o are used for obstacle avoidance. The sensor returns the distance (d_b) from the sensor to the obstacle. Eq. (1) is used to calculate the height h of the obstacle *relative to the ground*. This value depends on the angle at which the beam leaves the sensor ($\measuredangle a$) as well as on the inclination of the sensor itself ($\measuredangle b$) (refer Fig. 10).

$$h = d_b \cos(a) \cos(b) - h_{\text{LRFD}} \tag{1}$$

With the robot standing on flat ground, the values for h for all beams are then calibrated to 0. The error observed in h ranges between 5 mm in the centre and 30 mm on the sides. The (vertical or horizontal) distance between the robot and those obstacles is not calculated. The detected obstacles are all very close (the beam hits the ground a distance of about 20 cm), so that this information would not be of interest. The use of gyro roll and pitch data for the calculation of the height of the obstacles has its pros and cons. If this data is not used, the robot will see flat ground in front of it even when standing on an inclined plane. On a triangular ramp a big fall would be detected once the top of the ramp is reached. If the pitch of the robot is compensated by simply adding this value to the inclination of the sensor ($\measuredangle b$) during the calculation this fall is much smaller. Since in reality such a fall on a ramp is not dangerous for a robot, the pitch compensation is preferred. The downside of this compensation is that while going up or down a ramp the robot will see a (small) obstacle or fall in front.

Remark 3.2 (Severity Computation): The robot is about 33 cm high and is quite rugged. It is no problem for it to fall down up to $h_u^f = 18$ cm. Climbing up obstacles, especially in autonomous mode, is more difficult. If an obstacle is higher than the radius of the track wheels ($\approx$ 13 cm), overcoming it becomes really challenging even when being tele-operated. Therefore, the maximum height which the robot should attempt is set to about $h_u^o = 13$ cm. An obstacle height below $h_l^o = 3$ cm and a fall depth less than $h_l^f = 4$ cm is ignored. Between those minimum values and the maximum values the severity rises from zero to one. These severities are put in the situation object $\mathcal{S}$ as in Sec. III-D.2. The directional severity values already filled in by analyzation of LRFF data are overridden by those of LRFD data only if they are more critical.

4) The Victim Motion Detection: We have coded a basic movement detector which activated periodically after stopping the robot. Two successive frames are taken from the USB camera and compared. If a certain amount of pixles change above a certain threshold, a movement is detected. Several movement detections in a row lead to the assumption that a moving victim is found.

This simple movement detector works well and is easy to implement. But it has some drawbacks. The first is that a calibration phase is needed which can take up to 15 seconds. A second problem is that the detection quality might suffer if the light conditions change. The most important drawback is that the robot has to be stationary for somewhere between two and eight seconds, every time the scene is checked for movement.

E. Selecting a Behavior

Now, we look at the actual logic for selection of a current behavior from the set of available ones. Each behavior $\mathcal{B}$ has an overall priority $\mathcal{B}.p \in [0, 1]$, and a list of flags $\mathcal{B}.f_i$ corresponding to the situation flags $\mathcal{S}.f_i$ that it can handle. Each such behavior to situation-flag association also has a corresponding priority $\mathcal{B}.f_i.p$. The suitability of a behavior $\mathcal{B}$ for *handling* a situation $\mathcal{S}$ is computed as in Algo. III.3. Let the behavior activated in the last sample time instant be B_L. The selection of a suitable behavior then happens according to the flow presented in Fig. 8.

Algorithm III.3: BEHAVIOR::SUITABILITY($\mathcal{S}, B_L$)

$s \leftarrow 0$
for $i \leftarrow 1$ **to** $N_{\text{Situations}}$
  do { **if** $\mathcal{S}.f_i$ **and** $\mathcal{B}.f_i$
    then $\{s \leftarrow s + (\mathcal{B}.p)\ (\mathcal{B}.f_i.p)\ (\mathcal{S}.f_i.p)$
if this $= B_L$ **and** MotionSequenceNotFinished
  then $s \leftarrow s +$ LastRepeatPriority
return (s)

As shown in Fig. 7, the autonomy thread object is associated with the following behavior objects:

1) *A default behavior*: This behavior is activated when no severe situation is flagged.
2) *A current behavior*: This is the last behavior activated.

Additionally, the autonomy thread has the following repertoire of available behaviors. In each case, the way a behavior

handles a situation, i.e., computes motion commands is also described in brief. The motion commands ususally consist of the left and right wheel speed (v_L, v_R) in mm/s, though, they could also activate auxilliary actuators like the flipper. Refer to Table I for parameter definitions.

1) *Obstacle Avoidance*: This is the default behavior which handles the flag $\mathcal{S}.f_{\text{Obstacle}}$. It is a stateless purely reactionary behavior and as such does not require to be run again to complete its sequence. Therefore, variable `MotionSequenceNotFinished` mentioned in Algo. III.3 is always false.

$$\begin{aligned} v_L &\leftarrow V_{\text{Fwd}}\left(1 - \mathcal{S}.f_{\text{Obstacle}}.s[\text{Forward}]\right), \\ v_L &\leftarrow v_L - V_{\text{Rot}}\mathcal{S}.f_{\text{Obstacle}}.s[\text{Right}]. \end{aligned} \quad (2)$$

$$\begin{aligned} v_R &\leftarrow V_{\text{Fwd}}\left(1 - \mathcal{S}.f_{\text{Obstacle}}.s[\text{Forward}]\right), \\ v_R &\leftarrow v_R - V_{\text{Rot}}\mathcal{S}.f_{\text{Obstacle}}.s[\text{Left}]. \end{aligned} \quad (3)$$

2) *Largest Opening*: This behavior does not handle any situational flags directly, but is aggregated within other behaviors, e.g. Back off. It makes use of the LRFF data to find the largest opening amongst obstacles for the robot to escape, and rotates the robot to that direction. As noted in Remark 3.1, this behavior can be interrupted before reaching its end.
3) *Back off*: It handles situational flags $\mathcal{S}.f_{\text{Stuck}}$, $\mathcal{S}.f_{\text{Ditch}}$, $\mathcal{S}.f_{\text{Roll/Pitch}}$, $\mathcal{S}.f_{\text{MotorsStalling}}$. Essentially, it backs up a certain distance and then calls *Largest Opening*. As noted in Remark 3.1, this behavior can be interrupted before reaching its end.
4) *Victim Found*: It handles the following situational flags:

$$\begin{aligned} &\mathcal{S} \cdot f_{\text{IRDetectedVictim}}, \\ &\mathcal{S} \cdot f_{\text{MotionDetectorDetectedVictim}}, \\ &\mathcal{S} \cdot f_{\text{DetectedVictimWaitingForUser}}. \end{aligned}$$

This behavior has the highest priority, and cannot be interrupted by another behavior. It basically stops the robot and waits till a user confirmation is received or a timer times out.
5) *Motion Planner*: This behavior is run periodically based on a situational flag set by a timer. It samples the motion planner thread which tries to find paths to unexplored regions based on the generated occupancy grid map. The behavior runs only when no critical situations are flagged. It rotates the robot in the direction of the planned path.

IV. Conclusions

This paper presented an integrated hardware and software framework for autonomy of a rescue robot. This framework was field tested in Robocup 2006.

Acknowledgments

The authors gratefully acknowledge the financial support of *Deutsche Forschungsgemeinschaft* (DFG).

Please note the name-change of our institution. The Swiss Jacobs Foundation invests 200 Million Euro in **International University Bremen (IUB)** over a five-year period starting from 2007. To date this is the largest donation ever given in Europe by a private foundation to a science institution. In appreciation of the benefactors and to further promote the university's unique profile in higher education and research, the boards of IUB have decided to change the university's name to **Jacobs University Bremen**. Hence the two different names and abbreviations for the same institution may be found in this article, especially in the references to previously published material.

References

[1] T. Kamegawa, T. Yamasaki, and F. Matsuno, "Evaluation of snake-like rescue robot "kohga" for usability of remote control," in *IEEE International Workshop on Safety, Security and Rescue Robotics, SSRR*, 2005, pp. 25–30.
[2] S. Kang, W. Lee, M. Kim, and K. Shin, "Robhaz-rescue: rough-terrain negotiable teleoperated mobile robot for rescue mission," in *IEEE International Workshop on Safety, Security and Rescue Robotics, SSRR*, 2005, pp. 105–110.
[3] A. Davids, "Urban search and rescue robots: from tragedy to technology," *Intelligent Systems, IEEE*, vol. 17, no. 2, pp. 81–83, 2002.
[4] R. G. Snyder, "Robots assist in search and rescue efforts at wtc," *IEEE Robotics and Automation Magazine*, vol. 8, no. 4, pp. 26–28, 2001.
[5] R. R. Murphy, J. Casper, and M. Micire, "Potential tasks and research issues for mobile robots in robocup rescue," in *RoboCup-2000: Robot Soccer World Cup IV*, ser. Lecture notes in Artificial Intelligence (LNAI), P. Stone, T. Balch, and G. Kraetszchmar, Eds. Springer Verlag, 2001, vol. 2019, pp. 339–334.
[6] J. Abouaf, "Trial by fire: teleoperated robot targets chernobyl," *Computer Graphics and Applications, IEEE*, vol. 18, no. 4, pp. 10–14, 1998.
[7] J. Scholtz, J. Young, J. L. Drury, and H. A. Yanco, "Evaluation of human-robot interaction awareness in search and rescue," in *Proceedings of the International Conference on Robotics and Automation, ICRA'2004*. IEEE Press, 2004, pp. 2327– 2332.
[8] A. Birk and S. Carpin, "Rescue robotics - a crucial milestone on the road to autonomous systems," *Advanced Robotics Journal*, vol. 20, no. 5, pp. 595–695, 2006.
[9] A. Birk, S. Markov, I. Delchev, and K. Pathak, "Autonomous rescue operations on the iub rugbot," in *IEEE International Workshop on Safety, Security, and Rescue Robotics (SSRR)*. IEEE Press, 2006.
[10] D. Calisi, A. Farinelli, L. Iocchi, and D. Nardi, "Autonomous navigation and exploration in a rescue environment," in *IEEE International Workshop on Safety, Security and Rescue Robotics, SSRR*, 2005, pp. 54–59.
[11] S. Bahadori, L. Iocchi, D. Nardi, and G. Settembre, "Stereo vision based human body detection from a localized mobile robot," in *IEEE Conference on Advanced Video and Signal Based Surveillance*, 2005, pp. 499–504.
[12] A. Birk, "Fast robot prototyping with the CubeSystem," in *Proceedings of the International Conference on Robotics and Automation*. IEEE Press, 2004.
[13] A. Birk, K. Pathak, S. Schwertfeger, and W. Chonnaparamutt, "The iub rugbot: an intelligent, rugged mobile robot for search and rescue operations," in *IEEE International Workshop on Safety, Security, and Rescue Robotics (SSRR)*. IEEE Press, 2006.
[14] W. Chonnaparamutt and A. Birk, "A new mechatronic component for adjusting the footprint of tracked rescue robots," in *RoboCup 2006: Robot WorldCup X*, ser. Lecture Notes in Artificial Intelligence (LNAI), G. Lakemeyer, E. Sklar, D. Sorrenti, and T. Takahashi, Eds. Springer, 2007.
[15] *Real-Time Control Systems Library*, NIST, Gaithersberg, USA, 2006. [Online]. Available: http://www.isd.mel.nist.gov/projects/rcslib/
[16] *Palantir: a multichannel interactive streaming solution*, Fastpath Research, Milano, Italy, 2006. [Online]. Available: http://www.fastpath.it/products/palantir
[17] G. Grisetti, C. Stachniss, and W. Burgard, "Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling," in *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA*, 2005.

Pathak, K., A. Birk, S. Schwertfeger, I. Delchev, and S. Markov, "Fully Autonomous Operations of a Jacobs Rugbot in the RoboCup Rescue Robot League 2006", International Workshop on Safety, Security, and Rescue Robotics (SSRR): IEEE Press, 2007.

http://dx.doi.org/10.1109/SSRR.2007.4381267