INFORMATIK 2011 - Informatik schafft Communities
41. Jahrestagung der Gesellschaft für Informatik , 4.-7.10.2011, Berlin
www.informatik2011.de

A Software System for Autonomous Rescue Robots

Zaheer Aziz, Dirk Fischer, Tobias Kotthäuser, and Bärbel Mertsching

GET LAB, University of Paderborn
33098 Paderborn, Germany
{last name}@get.upb.de
http://getwww.uni-paderborn.de

Abstract:
Rescue robots are prime examples of cyber-physical systems which are characterized by a tight combination of the system's computational and physical elements. In this paper we analyze the requirements of such a system and propose a generic formulation that can be used to design software frameworks for this field. We present the rescue robot system *GETbot* organized and configured according to the design proposed. Finally, testing in order to validate the efficiency and robustness of individual modules as well as their interconnection is addressed.

1 Introduction

Natural disasters result in casualties of various types. One of the important immediate tasks in such situations is to get trapped survivors out of the zone of imminent danger. Having a huge potential of being extremely helpful rescue robotics is gaining increasing research attention. These systems could be used for different purposes such as constructing a map of the disaster site indicating accessible areas, searching for living survivors, delivering life saving material to victims, and conveying their location and routing human paramedics. On finding a victim these systems may also transmit detailed sensor readings reflecting the victim's vital functions, such as body temperature, carbon dioxide emission, and appearance to the rescue base station. These system must also be able to explore an area as large as possible in order to increase the probability of saving lives. Apart from these mission specific tasks rescue systems must also care about their self sustainment for example avoiding collisions, keeping away from terrain beyond their hardware capabilities, and refrain from dropping from high cliffs.

Rescue robot systems can be found in two major categories, namely, teleoperated and autonomous. The teleoperated systems provide facilities to an operator to control a robot and observe its sensor readings from remote locations. E. g. human controlled victim search using video and thermal cameras are described in [KSS05] and [KUS$^+$06]. Another system presented in [KL09] can navigate autonomously in rubble and perform human assisted search for body heat. Autonomous rescue systems have to perform all involved tasks without any contact to a human operator. A variety of sensing and maneuvering

erschienen im Tagungsband der INFORMATIK 2011
Lecture Notes in Informatics, Band P192
ISBN 978-3-88579-286-4
weitere Artikel online:
http://informatik2011.de/519.html

technology can be found on these systems. An example of an autonomous rescue system using video and thermal imaging can be seen in [KK07]. Application of hyper spectral imaging for detecting presence of victims is discussed in [TPPB08]. A system able to classify 3D terrain using a rotatable LRF can be seen in [FK09]. Despite all these efforts, the current rescue robots remain prototypes and still have to prove their value as the recent earthquake in Japan showed. Here only teleoperated systems came into use at a very late stage of the catastrophe.

Following the above task description rescue robots are prime examples of cyber-physical systems which are characterized by a tight combination of the system's computational and physical elements. Sensors, embedded processors and network components monitor and control permanently the entire robot which is subject to intrinsic and extrinsic physical processes. Environmental changes which cannot be predicted influence computations and vice versa. These systems *are considerable challenges, particularly because the physical components of such systems introduce safety and reliability requirements qualitatively different from those in general- purpose computing. Moreover, physical components are qualitatively different from object-oriented software components. Standard abstractions based on method calls and threads do not work* [Lee08].

Rescue systems consist of a complex combination of heterogeneous hardware and software components working concurrently on different time scales. Establishing timeliness by coordinating and synchronizing the components of the system turns out to be complicated since the number of components and their interdependency become fairly large. Therefore programming models and methods are required which support spatiotemporal actions and reasoning in highly dynamic environments. If no adequate software architecture with different abstraction levels exist, the maintenance and trouble shooting become difficult or impossible. Demanding flexibility and scalability for such systems result also in major problems to deal with. A small change in one component may force rewriting of many other modules if interdependency is not carefully handled. Hence, there is a need of standardization of software development framework for increasing the pace of development in this field. An important issue is the testing of the whole system in an environment providing the challenges expected in a catastrophe scenario.

This paper addresses the above mentioned problems. An analysis of the requirements of a robotic system specialized to rescue applications and a generic formulation are provided that can serve as a basis for the design software frameworks in this field. Furthermore, a design for arranging working modules of a rescue system according to the proposed framework is described. Some recently developed tools are used to implement this modular system architecture. A solution for a complete system level software testing is also discussed that can be helpful in rapid performance analysis of robotic software.

2 Generic Formulation of a Rescue System

As introduced in the previous section, a rescue assistance system has to perform different concurrent complex tasks under a synchronized strategy. In order to cope with these demands a system model with different abstraction levels is required. To fulfill these requirements a generic formulation of a rescue system is needed. There exist proposals for architectures of general purpose robots for example [SSK08] and [O'H11]. Work can also be found in the direction of rescue robot architecture, for example [CFOP08]. However, the main problem there is the tight coupling of algorithms with sensors which can hinder parallel processing of algorithms that need input from the same sensors. We propose here an architecture for rescue systems that rectifies the mentioned problems.

The set of software components controlling the system can be categorized into at least three layers according to the level of their tasks: The layer nearest to hardware should contain elementary processes. The secondary layer should consist of building-blocks that convert raw sensor data into information usable by high level control algorithms at a third layer. A master control algorithm may also be considered in the third layer that orchestrates the individual components to let the whole system perform the task of rescue operation. We discuss the important components involved in each layer in the following subsections.

2.1 Elementary Level Components

We consider an encapsulation of hardware drivers and the application program interface (API) to hardware control functions, for each sensor and actuator, as part of the elementary layer. In a generic context, software components to sense (acquire data from individual sensors), making sensor saccades (rotating the sensor head), performing robot motility, and doing self tracking belong to this level.

In terms of sensing there should be a software component for each sensor responsible to make data of that sensor available for the rest of the system. Some sensors could be mounted on a rotatable head for which the related software should allow focusing towards a given point in space. Such sensor movements are necessary due to the limited field-of-view of sensors which is usually too small to perceive the entire required section of the environment at a given location with one gaze.

Self motility of a system is necessary to increase volume of the search space by reaching different locations. Self tracking can be a helpful tool to estimate the system's position and orientation after going through spatial transformations. Under many situations in rescue scenarios global positioning information may not be available or its resolution may not be sufficient for localization purposes. Hence, odometry can serve as a useful solution for self tracking.

2.2 Secondary Level Components

Under this category we put those system components that directly use the elementary layer and could run in parallel without causing a conflict with other components. Commonly, these components can be used as building-blocks or feedback mechanism for implementing high level behaviors. In a generalized point of view, this layer includes modules for the interpretation of data in the current sensor view, environment modeling, and integration of self into the environment.

There may exist a separate software component for each sensor for the purpose of current view interpretation. The complexity of these tasks may range from determination of absolute locations of points in range data for the detection of objects in 3D point clouds. Apart from processing data from a single sensor, fusion of multiple sensors may also be used for these purposes. The environment modeling modules can construct models of the environment in one or more of the various possible formats ranging from occupancy grids to 3D object structures. In context of self-environment integration the system detects its own location and orientation in the modeled environment.

2.3 Highs Level Components

The third layer consists of modules which contain algorithms, each of them implementing a high level behavior. They may compete with each other to access or control one or more of the elementary level resources. It may not always be possible to let these processes run in parallel, rather a coordination scheme has to be designed to achieve the objectives of the application domain. In the area of rescue robots the modules of target existence check, target conformation, getting close-up sensor readings of the target, obstacle avoidance, navigation between two spatial locations, and exploration strategy are counted as high level system components.

It could be argued that modules like obstacle avoidance, point-to-point navigation, and target existence check should be regarded as secondary level components. Generally, movements for obstacle avoidance and navigation to reach a planned target location conflict with each other. Hence, according to the proposed definitions they belong to the third level layer. In our design of the rescue system the target existence check is done after stopping the robot and performing a head scan around the current location. Therefore, it belongs to the third layer due to competing for motility resources.

Target confirmation requires focusing of a sensor head towards the candidate location and target close-up requires navigating the robot near to the target position and focusing the sensor head to get detailed readings. An exploration strategy has to maximize the area scanned by the system while minimizing repeated scanning of already visited areas.

2.4 Inter-Component Coordination

Encapsulation of sensor input and motor control within a software component can cause coordination problems when integrating these components in an entire system. Letting sensor data, intermediate results, and control commands flow on a shared data stream is a better option. This means in the practical implementation that modules handling sensors only provide data to the stream and secondary (or higher) level components may pick it for processing. This can usually occur in parallel for different sensors without causing

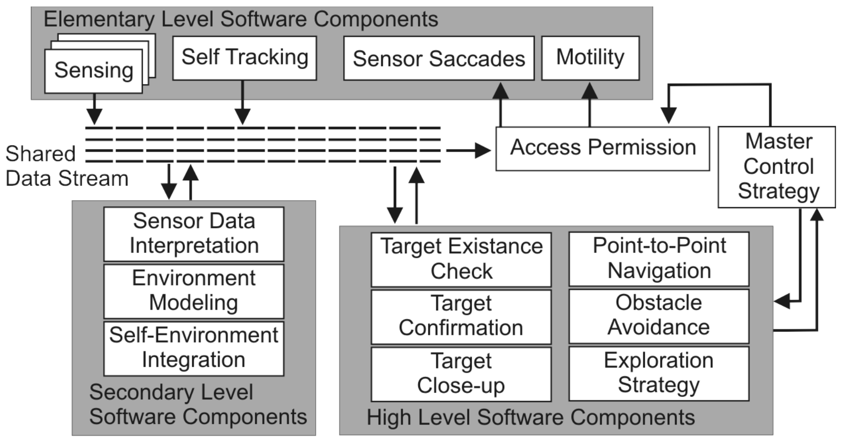

Figure 1: A generic formulation of a software system for autonomous rescue robot.

conflicts among them. For elementary components related to actuators it has to be assured that only one maneuver action can be done by an actuator at a time. Conflict resolution or pre-planned coordination has to be incorporated for different system components trying to send control commands to the actuators. These commands might conflict with each other: e. g. a navigational behavior may require a right turn of the robot whereas the obstacle avoidance may ask for a left turn.

A generic solution for coordination is to use a finite state machine (FSM) that sets the system into a particular state according to a master control strategy designed to achieve the system level objectives. Only a specific software module will be allowed to access an actuator under a given system state.

In the light of the above mentioned formulation, a generic abstraction of a rescue system may be sketched as given in fig. 1. On the elementary level, the sensors usually need only write-access to the shared data stream while actuators have read-access. The secondary level should have read and write access however, its components should not address the actuators directly. The high level modules use processed information from secondary level and may also merge it with the raw sensor data to reach a decision for further actions. The master control coordinates between the high level modules and allow only one command to a particular actuator at a time.

3 Rescue Robot System GETbot

In this section, we present the robotic rescue system, GETbot which has been organized and configured according to the design proposed in section 2. The complete system is depicted in fig. 2(a). The design of the GETbot is based on a Pioneer 3-AT which is a four wheel skid steering drive system. In terms of sensors, it is equipped with encoders to provide basic odometry information, a laser range finder, a monocular camera and a

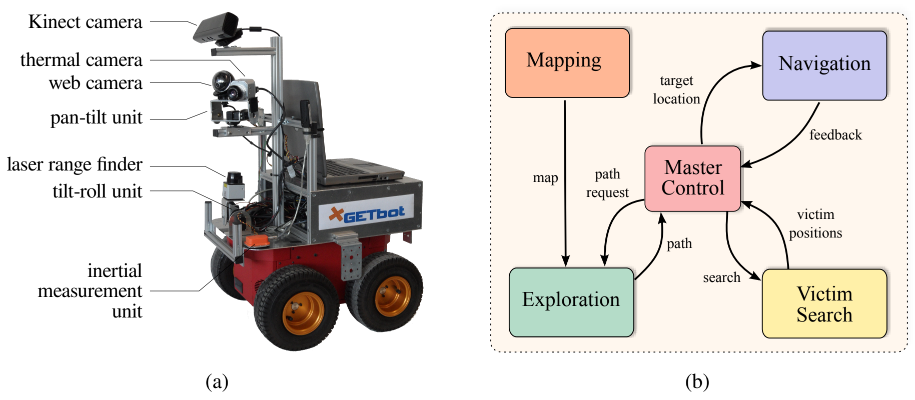

Figure 2: Rescue robot system GETbot: (a) Platform with its sensors and actuators. (b) Interaction of the main components within our rescue system (simplified).

3D camera to visually monitor the area, a thermal camera to measure the temperature of objects present in the environment, and an inertial measurement unit. The actuators include two sensor heads, each capable to move with two degrees of freedom.

Fig. 2(b) illustrates the scheme of inter-component interactions in our rescue system. A map of the environment is continuously updated using range and odometry data. The exploration module uses this map to determine a path to the next optimal exploration target. The master control module requests a path from the exploration module. All intermediate points on this path are forwarded to the navigation module one by one to safely drive to the next location to be explored. Feedback of the navigation module is used to examine conditions, e.g. is the target point reached or not. Each time the system reaches a new exploration location, the master control triggers the victim search module. The number and positions of possible victims are used to decide whether to continue exploring or approach to the closest victim location.

The individual software components of the system according to the three levels of task hierarchy are discussed in the following subsections.

3.1 Elementary Level Software Components

For each sensor and actuator installed on the robot, an object-oriented control software is provided that initiates the hardware drivers and allows data reading and writing. For the case of sensors, obviously, only the reading interfaces are needed. For actuators, writing interfaces allow sending control commands whereas reading functions can be used to query status information from the device.

3.2 Secondary Level Software Components

For a rescue system the environment is mostly unknown as the previously known structure might be damaged. Unknown environments require to build a model of the explored environment along with sufficient details about the objects that are present in that area. In terms of sensor data interpretation the components making major contribution to rescue application are analysis of the data from thermal, video, and 3D cameras. A brief description of these components of GETbot is given below.

3.2.1 Environment Modeling and Self-Environment Integration

The environment models may exist in terms of geographical maps and/or 3D representations. The correct mapping procedure fundamentally needs an accurate estimation of the robot position, i. e. performing a self localization. The main input for this process is the range data obtained from a laser range finder. Mapping and robot localization has been an area of research for quite a time, hence, several algorithms are available that perform environment modeling and integration of the robot into this model by determining its current position and orientation. Localization and mapping are usually coined together as simultaneous localization and mapping (SLAM) [Fre06]. The current system makes use of an iterative technique to perform scan matching for the localization purpose [MML06]. This method proved successful for different environments however, scan matching has its own failures which are attributed to noise and limitations of the employed sensors. Lately, we have implemented a generic SLAM framework to make use of different SLAM techniques to map the explored area [GSB07]. Further research is underway to make these processes more robust and efficient.

3.2.2 Thermal and Video Image Processing

The thermal signal is the most reliable symptom from living survivors. Two features are analyzed to determine the existence of a victim in the current thermal view; the first is the human body temperature and the second is the extent of those regions possessing body temperatures. In the context of visual perception there exist special challenges in the rescue scenario. The target can appear in various colors that are hard to define in advance, for example different skin shades and clothing. We have developed a simplified solution by defining a few colors that could not belong to a victim, e. g. dirt, rocks, mud, and ash, etc. After rejecting these defined colors the rest will lead to signals that most probably belong to the target. In order to optimize the processing time, thermal and visual image processing is activated when the system goes into the state of running the high level component for scanning for victims (discussed below).

3.2.3 Terrain Analysis

An important part of the autonomous navigation in unknown and unstructured environments is the assessment of the terrain the robot intends to maneuver on to ensure a save

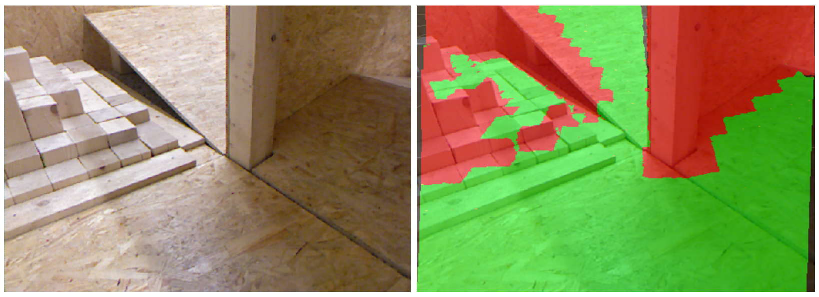

Figure 3: Terrain Analysis: (a) RGB image of the terrain in front of the robot. (b) Classified point cloud. Red patches represent terrain with obstacles which is not accessible to the robot while green patches indicate areas which can be passed.

navigation. This is important for avoiding damage to the robot that can occur by going into terrains beyond the capacity of its actuators. A laser scanner monitors the area on a fixed level and thus, covering only obstacles at exactly this height. Hence, it can not detect obstacles above or below the scanning plane. To solve this problem, sensors generating three-dimensional data of the environment can be used to estimate the structure of terrain in front of the robot. GETbot uses a Kinect camera which allows a fast and reasonably high resolution 3D image data acquisition. Fig. 3 shows the result of the implemented method based on a principal component analysis as described in [LVHH06]. Feedback of this elementary module helps the high level exploration module in determining further explorable paths.

3.3 High Level Software Components

The high level software components of GETbot are divided into two blocks. The first one, labeled navigation, bundles the modules *exploration strategy*, *point-to-point navigation*, and *obstacle avoidance* while the second one which is called victim search groups the modules *target existence check*, *target confirmation*, and *close-up to target*. The functionality of these two groups of components is as follows:

3.3.1 Exploration

This set of software components deals with the issues of exploring an unknown environment and ensuring a safe trip of the robot through the environment. One of the main tasks of a rescue system is to explore maximum possible area without colliding with any object present in the environment. The exploration strategy implemented on GETbot is based upon frontier-based exploration [FO05]. This strategy collects the unexplored locations as a set of target points on the, so far, constructed map. Hence, the strategy is tightly coupled with the SLAM procedures discussed above. The map of the explored area is utilized to

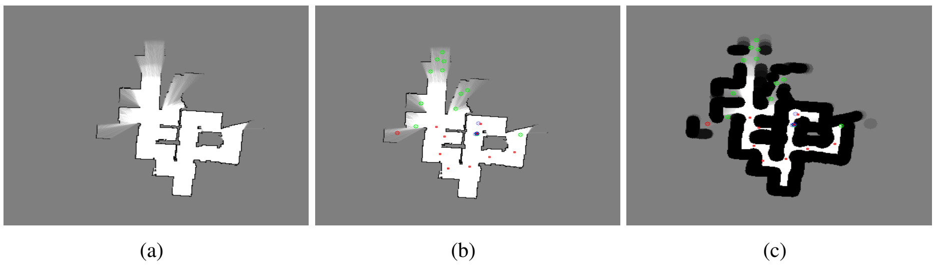

Figure 4: Results of the robot navigation module: (a) Occupancy grid map of the so far explored environment. (b) Map displaying target points: Targets still to be investigated are marked by green circles, already covered target points are marked by red rectangles, and unreachable points by red circles. (c) A safe map generated to be used by the path planner.

carry out the path planning between the current robot location and one of the target points. Obstacle detection and avoidance is running continuously while the robots is moving. Obstacle avoidance has been achieved after implementing variations of a smooth nearness diagram [DB08] algorithm. Fig. 4 presents results of the exploration bundle obtained during one of the experiments.

3.3.2 Victim Search

In the current status of the system, the robot stops after covering a certain amount of distance in the exploration state and switches into victim search mode to scanning. This strategy for mode switching has at least three advantages over continuous victim scanning during exploration. Firstly, the camera-head movements from right to left or vice versa during motion of the whole system can result in non-scanned patches in the explored environment. Secondly, motion blur due to robot movements combined with head rotations can be avoided. Thirdly, more computational resources would be required to run exploration and scanning algorithms in parallel.

The search process includes rotating the camera head carrying the thermal and video camera in a 180 degree span at two levels of tilt angles. The step between the pan and tilt angles are adjusted in such a way that there is a minimal overlap in the consecutive frames of thermal input. A panorama image is created using filtered input provided by the thermal image processing module discussed in the secondary level components. The warm region in the panorama image satisfying the most conditions to be a human body is focused by the sensor head to get further data in order to confirm existence of a human victim. The confirmation process includes analyzing the victim hypotheses delivered by the video camera. The intermediate results and victim localization is shown in fig. 5. After confirmation, the robot moves closer towards the victim location using a local navigation scheme and delivers detailed sensor readings after reaching the target location. Details of this victim search scheme were published in [AM10].

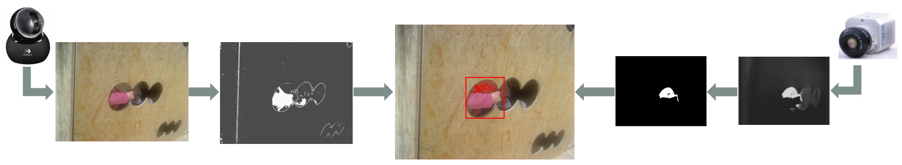

Figure 5: Results of victim confirmation after focusing sensor head on the selected candidate: The hypotheses in thermal camera input are used to identify candidate locations where victims could be present and the sensor head is focused on the most probable candidate. Color information in the video data is used to confirm existence of the target.

3.4 Inter-Component Coordination and Master Control

In order to realize communication and control as shown in fig. 1 a convenient software framework is needed. Since our research focus lies mainly on the intermediate and high level system components, we decided to choose one of the existing robotic frameworks instead of developing a novel one. We evaluated state-of-the-art robotic middleware applications such as Player [GVH03], OpenRDK [CCIN08], Orocos [SDLC$^+$08] and Robot Operating System (ROS) [QGC$^+$09]. Important requirements for an appropriate framework to use were: open source software, hardware abstraction, drivers readily available for a large number of sensors, actuators and robots, network support, facilities to support testing and parametrization of the components, detailed documentation and an active developer community. ROS came up as a suitable choice since it is the most flexible framework which comes with detailed documentation, tutorials and a wide range of ready-to-use software modules.

Software modules in ROS are called *nodes* and each node runs as a separate process. The communication between nodes is possible using two concepts; at one hand a node can subscribe certain topics while other nodes are publishing on these topics. This publish/subscribe messaging pattern conforms to the proposed design of a shared data stream. On the other hand nodes can offer services which all other nodes can call. An important role in an ROS application plays a node called *master*. The master provides name and parameter services, in this way nodes can find topics and services at run-time.

The software components of our system described above, that communicate with each other as sketched in fig. 2(b), have been implemented as nodes for ROS. Communication between these nodes has been established by deploying subscribers and publishers based upon the concerned data types.

In our design of a rescue system, the conceptual base of a hierarchical finite state machine (HFSM) has matched well with the requirements of the master control component. A library package is available in the ROS framework called *SMACH* (State MACHine) that facilitates to deploy the concept of HFSM (http://www.ros.org/wiki/smach). We have used the same to implement the master control component that runs the system under different pre-defined states and coordinates the activities of the individual components by changing these states.

4 Testing Environments

Environments to test a complete behavior of a rescue system is hard to construct and expensive to maintain. In order to evaluate progress of research in this direction, yearly robot competitions (RoboCup Rescue League) are held in several countries. A simulated disaster environment is provided with challenging rescue related tasks to robots. The test environment of these competitions is of course available for only a few days in an year. Maintaining such a facility is not feasible for every research group because of the huge requirements in space. After introducing the test environment of RoboCup arenas in the next subsection we present a solution for the test environment problem.

4.1 RoboCup Rescue Arenas

The RoboCup Rescue competition [JWM03] provides a possibility to test the functionality of a robot designed to carry out rescue tasks in different scenarios. There are three different testing arenas, namely, yellow, orange and red. While the orange and red arenas are meant for testing the mobility of remote controlled robots, the yellow arena is designed with the focus to tackle the development of fully autonomous robots. For the autonomous arena the robot has to be able to explore difficult terrain and must be equipped with sufficient abilities to search and identify the location of victims. The victims are typically simulated by dolls whose body temperature is emulated with heating pads.

4.2 Virtual Prototyping for Robot Rescue Systems

The development of software components for autonomous rescue applications relies on a plenitude of repetitive testing methods. In order to validate the efficiency and robustness of individual software components as well as their interconnection, varying testing conditions, such as hardware configurations or the arrangement of the testing environment have to be incorporated. Naturally, such test procedures are eminently time consuming. Expediting the testing by introducing collected real-world data might be one solution; due to the system complexity and the necessity to interact with the environment does not make it generally suitable in the realm of most autonomous rescue systems.

The prototyping in this work is performed with the help of the simulation framework SIMORE [KM10] which satisfies the requirements of rescue systems and environments. SIMORE allows the simulation of sensors, actuators and entire mobile robot platforms in virtual 3D environments as illustrated (see fig. 6). SIMORE is built upon open-source libraries, such as the *OpenSceneGraph* toolkit for scene rendering and the *Open Dynamics Engine* to accomplish a solid physical behavior during robot-environment interaction.

In general, simulation frameworks such as SIMORE provide various advantages; virtual environments have – compared to real world scenarios – less limitations to the number of used sensors, actuators and robots as well as their arrangement in the environment. Virtual environments support the reproducibility of scenes with manually adjustable complexity with respect to the number and the appearance of objects; distinct algorithms can be benchmarked with identical input scenarios. Of particular importance is the opportunity to test validity and performance of complex algorithms without any risk of damage to the system or the environment subject to ill-conceived processing or control. Finally, simulation in virtual environments offers high debugging capabilities and enables the developers to focus on principal algorithmic issues; disturbing effects, such as unintended robot motions or sensor inaccuracies can be neglected; virtual sensors provide ground truth data for all sensor measurements and virtual actuators can be controlled with high precision and individual rates.

The activity of the software components stated in section 3 can now be transparently prototyped with SIMORE due to its interfaces to the ROS middleware. Analogous to the hardware system, the data flow of sensors and actuators in SIMORE is adapted to ROS communication techniques. Therefore, single software components cannot distinguish, whether they communicate with real hardware or the simulation. Hence, prototyping the entire software system can be accomplished without the actual hardware of the robot.

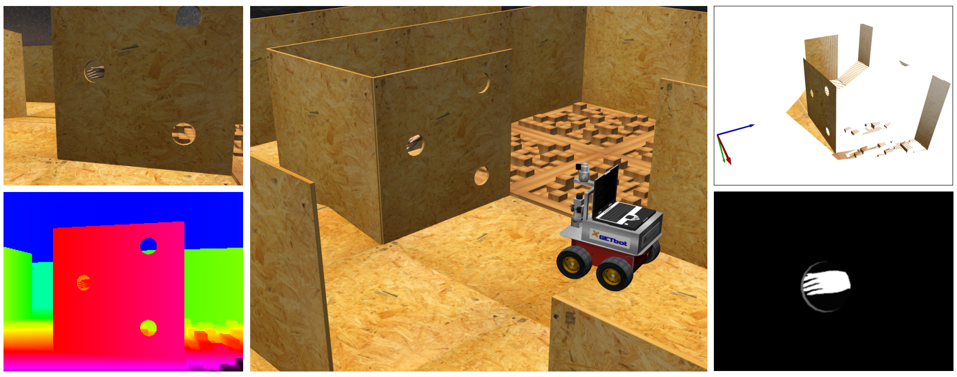

Figure 6: Sample of a simulated rescue scenario and synthesized sensor outputs. Center: User's view of the simulator. It depicts a robot that is supposed to detect victims hidden in the environment. A potential evidence of a victim's presence is given by the hand sticking out of the left hole of wall faced by the robot. Top left: Output of a standard camera. Bottom left: Depth image. Top right: 3D scan with object attributions. Bottom right: Thermal scan.

To satisfy the material demands of a rescue mission SIMORE features the replication and design of independent virtual catastrophe scenarios that can be rapidly combined to individual rescue missions. Hence it allows to test the efficacy of a system in a given situation before sending the actual hardware robots into a mission.

Since virtual simulation is not able to capture the complexity of the real world entirely, it cannot always be verified that an algorithm that has been successfully tested in the simulation bears the same success as under real conditions. However, if a tests fails under simplified simulation conditions, the same tests are likely to fail in the real applications as well.

5 Conclusion

In this work, we discussed the requirements for a robotic software framework involving the reliable coordination of software modules accessing multiple hardware resources. Using the example of a rescue robot system, we presented a generic formulation comprising elementary, secondary level and high level software components.

Other existing architectures for robotic cognitive systems have either encapsulation or tight coupling between system components belonging to different levels of processing hierarchy (see [VMS07] for a review). This hinders scalability on the one hand and the potential of adaptation to distributed and parallel processing on the other. The advantage of the proposed architecture is the lose coupling among the components lying at different levels of processing hierarchy. Hence, it favors parallelism and distributed computing for deployment as a cyber physical system.

We introduced our rescue platform and the software modules significant for the performance of a rescue robot framework, namely mapping, exploration, navigation and searching for victims. Further on, we have shown how to prototype individual modules as well as their interplay within defined testing and simulation environments. Software development for the same hardware and simulated platforms was done under a tightly coupled architecture as well as the one proposed in this work. Challenging circumstances with strict deployment deadlines and multiple developers concentrating on different modules provide a testbed for the effectiveness of the software architecture. RoboCup competitions are an example of such situations where parallel development and enhancement of various modules have to be done in a strictly limited time. Using the tightly coupled infrastructure the integration and coordination of software modules became extremely difficult. The proposed architecture, on the other hand, allowed a smooth development by heterogeneous experts and facilitated quick integration of modules into a unified system. The robot simulation framework introduced in this paper facilitates simultaneous testing of individual modules and their integrated functionality.

The main problem using loosely coupled distributed components is the bottleneck of data communication stream. This can be an issue for robotic systems as some sensors deliver large packets of data continuously. Sometimes, real-time output cannot be expected from the system when different components have different speeds of delivering results. These problems can make the scalability an issue. For application oriented robots this is likely to happen because they require a large number of input, processing and output channels.

References

[AM10] M. Z. Aziz and B. Mertsching. Survivor Search With Autonomous UGVs Using Multimodal Overt Attention. In *IEEE International Workshop on Safety, Security & Rescue Robotics 2010*, July 2010.

[CCIN08] D. Calisi, A. Censi, L. Iocchi, and D. Nardi. OpenRDK: a modular framework for robotic software development. In *IEEE/RSJ International Conference on Intelligent*

Robots and Systems, 2008. IROS 2008, pages 1872–1877, 2008.

[CFOP08] A. Carbone, A. Finzi, A. Orlandini, and F. Pirri. Model-based control architecture for attentive robots in rescue scenarios. *Autonomous Robots*, 24(1):87–120, 2008.

[DB08] Joseph W. Durham and Francesco Bullo. Smooth Nearness-Diagram Navigation. In *IROS*, pages 690–695, 2008.

[FK09] T. Fujita and Y. Kondo. 3D terrain measurement system with movable laser range finder. In *SSRR 2009*, pages 1–6. IEEE, 2009.

[FO05] Luigi Freda and Giuseppe Oriolo. Frontier-Based Probabilistic Strategies for Sensor-Based Exploration. In *ICRA*, pages 3881–3887, 2005.

[Fre06] Udo Frese. A Discussion of Simultaneous Localization and Mapping. *Autonomous Robots*, 20(1):25–42, 2006.

[GSB07] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23:2007, 2007.

[GVH03] B. Gerkey, R.T. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th international conference on advanced robotics*, pages 317–323. Citeseer, 2003.

[JWM03] Adam Jacoff, Brian Weiss, and Elena Messina. Evolution of a Performance Metric for Urban Search and Rescue Robots. In *Proceedings of the 2003 Performance Metrics for Intelligent Systems (PerMIS) Workshop*, page 18, 2003.

[KK07] A. Kleiner and R. Kuemmerle. Genetic MRF model optimization for real-time victim detection in Search and Rescue. In *International Conference on Intelligent Robots and Systems*, 2007.

[KL09] Albert Ko and Henry Y. K. Lau. Robot Assisted Emergency Search and Rescue System With a Wireless Sensor Network. pages 69–78, 2009.

[KM10] T. Kotthäuser and B. Mertsching. Validating vision and robotic algorithms for dynamic real world environments. *Simulation, Modeling, and Programming for Autonomous Robots*, pages 97–108, 2010.

[KSS05] M. W. Kadous, R. K. Sheh, and C. Sammut. CASTER: A Robot for Urban Search and Rescue. In *Australasian Conference on Robotics & Automation*, 2005.

[KUS+06] K. Kon, Y. Urano, N. Shiroma, N. Sato, Y. Fujino, H. Fukushima, and F. Matsuno. Development of Robot Teleoperation System in Bad Viewing Condition. In *IEEE International Conference on Robotics and Biomimetics*, pages 427–432, 2006.

[Lee08] Edward A. Lee. Cyber Physical Systems: Design Challenges. Technical Report UCB/EECS-2008-8, EECS Department, University of California, Berkeley, Jan 2008.

[LVHH06] Jean-Francois Lalonde, Nicolas Vandapel, Daniel Huber, and Martial Hebert. Natural terrain classification using three-dimensional ladar data for ground robot mobility. *Journal of Field Robotics*, 23(1):839 – 861, November 2006.

[MML06] Javier Minguez, Luis Montesano, and Florent Lamiraux. Metric-based Iterative Closest Point Scan Matching for Sensor Displacement Estimation. *IEEE Transactions on Robotics*, 22(5):1047–1054, 2006.

[O'H11] K.J. O'Hara. Towards Robot Systems Architecture. In *2011 AAAI Spring Symposium Series*, 2011.

[QGC+09] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. ROS: an open-source Robot Operating System. In *Open-Source Software workshop of (ICRA)*, 2009.

[SDLC+08] R. Smits, T. De Laet, K. Claes, P. Soetens, J. De Schutter, and H. Bruyninckx. Orocos: A software framework for complex sensor-driven robot tasks. *IEEE Robotics and Automation Magazine*, 2008.

[SSK08] F.M.P. StephenBalakirsky, C.J. Scrapper, and T.R. Kramer. A Mobile Robot Control Framework: From Simulation to Reality. In *Simulation, modeling, and programming for autonomous robots: first international conference, SIMPAR 2008, Venice, Italy, November 3-6, 2008: proceedings*, page 111. Springer-Verlag New York Inc, 2008.

[TPPB08] M. Trierscheid, J. Pellenz, D. Paulus, and D. Balthasar. Hyperspectral Imaing for Victim Detection with Rescue Robots. In *International Workshop on Safety, Security and Rescue Robotics*, pages 7–12, 2008.

[VMS07] D. Vernon, G. Metta, and G. Sandini. A survey of artificial cognitive systems: Implications for the autonomous development of mental capabilities in computational agents. *Evolutionary Computation, IEEE Transactions on*, 11(2):151–180, 2007.