# Failure Recovery with Shared Autonomy

Bharath Sankaran, Benjamin Pitzer and Sarah Osentoski

***Abstract*— Building robots capable of long term autonomy has been a long standing goal of robotics research. Such systems must be capable of performing certain tasks with a high degree of robustness and repeatability. In the context of personal robotics, these tasks could range anywhere from retrieving items from a refrigerator, loading a dishwasher, to setting up a dinner table. Given the complexity of these tasks there are a multitude of failure scenarios that the robot can encounter, irrespective of whether the environment is static or dynamic. For a robot to be successful in such situations, it would need to know how to recover from failures or when to ask a human for help.**

**In this paper, we present a novel shared autonomy behavioral executive to addresses these issues. We demonstrate how this executive combines generalized logic based recovery and human intervention to achieve continuous failure free operation. We tested the systems over 250 trials of two different use case experiments. We observed that our current algorithm drastically reduced human intervention from 26% to 4% on the first experiment and 46% to 9% on the second experiment. This system provides a new dimension to robot autonomy, where robots can exhibit long term failure free operation with minimal human supervision. We also discuss how the system can be generalized.**

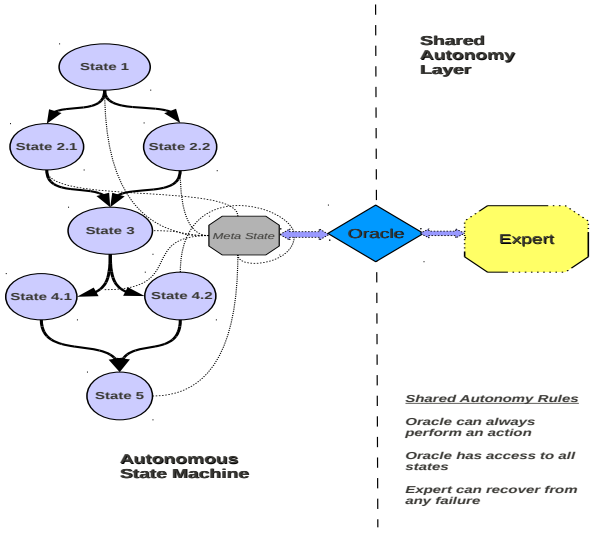



Fig. 1. Shared Autonomy Topological Representation

## I. Introduction

For robots to be successful in any domain outside industrial applications, a fundamental requirement that needs to be addressed is long term autonomy. Ideally one would like to restrict human supervision to tasks such as initiation of a desired activity or intervention during failure. Such an approach in personal robotics would require systems to be more robust and reliable than demonstrated by current state-of-the-art. Since personal robots enagage in complex interactions with their environment, achieving this robustness becomes a challenging task. As Meeussen et al [1] have noted, in the event of a failure the robot should be capable of asking for help, but the question remains as to how and when this should be done. In this paper, we examine some of these questions.

Robot autonomy is an extremely difficult task even in highly constrained environments. In the personal robotics domain, robotic assistants are expected to perform roles similar to that of a maid or a busboy: clearing a table, getting drinks from a refrigerator etc. In the current state-of-the-art high level tasks like these are accomplished through executives which execute high level robot actions to accomplish these tasks. Such a task level executive architecture has been demonstrated by Bohren et al in the *SMACH* system [2]. Such systems are useful as they are specified via high level tasks which are analagous to human instruction. This form of a system enables rapid prototyping for application development in comparison to imperative scripting approaches or model based task-planners. Task executives primarily act as a procedure definition system for task planners, which make them extremely flexible and reusable while being analogous to human style instruction. Though general enough to incorporate any robot action to accomplish a task, such systems are highly prone to failure. Since these executives operate on high level actions, any system built to recover from the failure of these high level actions needs do so without the knowledge of the underlying low level mechanisms. Also when scripting these systems to recover from failure, it is not possible to account for every failure case during the design phase. In these scenarios identifying the source of failure and troubleshooting the specific subsystem is not plausible due to the unknown complexity of the underlying system.

In this paper, we present an architecture which performs logic based recovery actions on such task executives. Since these task executives are defined as state machines, the recovery architecture operates on the states of the state machine to enable efficient failure recovery. The recovery architecture also incorporates an expert interface to enable human intervention in the case of unrecoverable failures. The system also adaptively learns an online failure rate for each

B. Sankaran is with the GRASP Laboratory, University of Pennsylvania, Philadelphia, PA 19104, USA., bharath@seas.upenn.edu

B. Pitzer and S. Osentoski are with Robert Bosch LLC at the Research and Technology Center North America, Palo Alto, CA 94304, USA., {Benjamin.Pitzer,Sarah.Osentoski}@us.bosch.com

Preprint submitted to 2012 IEEE International Conference on Robotics and Automation. Received September 16, 2011.

action in the state machine to determine the level of human intervention required in the event of a failure. Though failure detection is a crucial aspect for failure recovery, it is an entirely separate research topic in itself. For the sake of this work we assume that the system has the inherent ability to detect failure. The novel contributions of this work are :
1) A generalized logic based recovery algorithm where the user does not need to identify every possible failure case.
2) A novel method for combining expert interfaces to assist in failure recovery through shared autonomy.

In the following sections, we describe the related work to this paper, followed by which we present a System Description and Overview of task executives and shared autonomy. Then we elaborate on the Behavioral Executive followed by the Expert Interface. In the final two sections we present our Experimental Results and Conclusions.

## II. Related Work

Long term autonomy has been addressed by many researchers in their system specific capacities. Robots which aim for long term autonomy have been demonstrated in numerous cases, for example where robots have been used tour guides. To address failure recovery in these robotic systems, varying levels of human intervention are directed towards altering the environment of the robot to ensure successful operation.

RHINO [3], the first robotic tour guide, subsequently MINERVA [4], Mobots [5], Robox [6] and Jinny [7] navigate around museum environments with varying degrees of success. Since these robots operated in environments which were highly dynamic, their success depended on efficient localization. Hence occasional modifications had to be made to their map of the environment to ensure their localization was accurate. This form of human intervention involves directly altering the environment to ensure that the robots can function in a robust fashion. Though this technique is effective for localization tasks, in most robot autonomy tasks modifying the environment can have consequences as some modes of the operation of the robot may depend on its perception of the environment.

In the space exploration industry where failure free operation is absolutely critical to mission success [8], systems tend to demonstrate a high level of robustness by either combining autonomy and human input or relying entirely on human assisted operation. Though the problem of generalized task level failure recovery has not been explicitly addressed, task specific recovery has been addressed by previous work. There have been two areas of research in this field. One where task specific recovery behaviors are scripted into the system, like the ability to handle navigation failures in the Urban Car Challenge [9]. The other where some form of human input has been injected into a system to achieve a high level of robustness in human environments. Researchers at Bosch [10] and Willow Garage [11] have also argued for human intervention in robotic systems for long term autonomy, but the level of human intervention required still seems to be an open research question. We try to address some of these issues in this paper.

## III. System Description and Overview

For executing high level task oriented commands, the current state-of-the-art mostly utilize task-level executives. Although the theory for these executives have existed for decades [12], there has been little research on generalized failure recovery in these systems. The most effective of these systems are used to build and execute hierarchical concurrent state machines [13] [14] to enable seamless integration of the various tasks required to achieve the robot's goal or mission. In general, the robot's goal can be specified in terms of a set of hierarchal tasks which are captured by these systems. For instance, in a drink fetching application, the set of tasks would involve opening a container, looking for a drink, retrieving the drink from the container and closing the container. Such simple tasks, in general encapsulate a variety of components like planning, perception, reasoning, navigation, etc. In such cases we use a shared autonomy intervention at various stages of the task execution to recover from failure. We define shared autonomy as a partial or full replacement of an action that the robot has to execute to accomplish a specific task. In this paper we use this definition of shared autonomy to define levels of human intervention for failure recovery while keeping the original system fully autonomous.

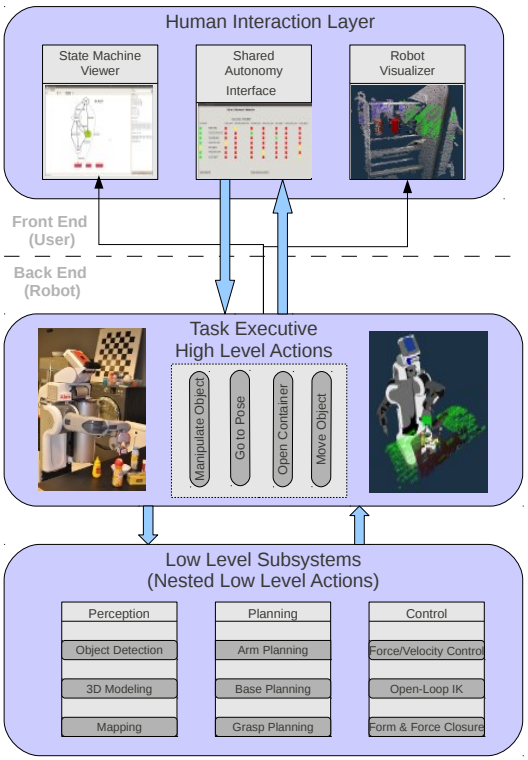



Fig. 2. System Architecture

### A. Shared Autonomy

High level robot tasks can be broadly classified as either some form of information acquisition, information analysis, decision making or plan execution. Since these high level tasks are used in a task executive, our shared autonomy interface directly interacts and replaces these tasks in the event of an expert recovery. When queried, the expert has the ability to intervene and supersede any stage of the robot's task execution pipline. The generalized architecture for our system is depicted in Fig 2 There have been attempts to preempt failure through planning [15], but our system addresses situations which require recovery since all failures cannot be preempted.

## IV. Behavioral Executive

Most state machines that are designed for robot applications are non ergodic in nature, i.e every state cannot be reached from every other state in the state machine. The non-ergodicity of these state machines makes task level recovery impossible, as the state machines cannot revisit the states that it had occupied before the current failure state. This is demonstrated in the example state chart [16] in Fig 3(a). For instance if the action in state FOB fails, the state machine fails. Designing recovery behaviors in this state machine is trivial, given its relative simplicity. The state machine could be redesigned to revisit a select set of states in the event of a failure. Though trivial given the number of states in the state machine, this approach cannot be generalized as it is tough to reason about relations between states in large state machines. This tends to be the case in state machines designed for most robot applications. In general, robot state machines to perform a certain task can be extremely complex. Unless explicitly defined by the designer of the state machine, crafting recovery behaviors to account for all possible failures is non trivial and generally intractable. To address this specific issue we introduce a generalized failure recovery mechanism, through the introduction of a meta state called the *oracle* .

### A. Oracle State

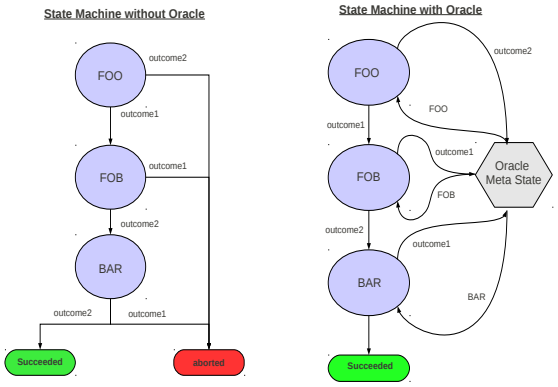
The *oracle* is a meta state which can help the state machine transition from any failed state to any other state in the state machine. If the state machine were formulated as a graph, then the introduction of the *oracle* makes all nodes in the graph accessible as it shares an edge with every node in the graph. This is illustrated in Fig 3. If an action fails, it results in the failure of the state containing the action. In this case, the *oracle* executes a set of logic based recovery behaviors before querying the expert. These logic based recovery behaviors either execute a set of preconditions for the current state to succeed or transition the state machine to a previous state without any preconditions. States without preconditions are Markovian and hence transitioning to them does not violate any consistencies in the state machine. In the event of encoutering a high failure state the oracle queries an expert for help.

**Algorithm 1**: The Oracle's logic based shared autonomy recovery algorithm

**Input**: Set of States $\mathscr{S}$, Current Failed State $\mathcal{F}(s_t)$ and Preconditions of States $\mathcal{P}(\mathscr{S})$ where $s_t \in \mathscr{S}$
**Output**: Recovery State $\mathcal{R}c_t$
**if** $\mathcal{F}(s_t) = \mathcal{F}(s_{t-1})$ **then**
  Attempt = TRUE
$\mathcal{F}(s_{t-1}) = \mathcal{F}(s_t)$;
$i = 0$;
**while** $i < t$ **do**
  **if** $\mathcal{P}(s_i)$ *is NULL* **then**
    Most Recent State, $\mathcal{R}s_{t-1} = s_i$;
  **end**
  $i = i + 1$;
**end**
**if** $\mathcal{F}(s_t) =$ *LOW FAILURE* **then**
  **if** $\mathcal{P}(s_t)$ *is NULL* **then**
    $\mathcal{R}c_t$ = $\mathcal{R}s_{t-1}$
  **else**
    success = execute($\mathcal{P}(s_t)$);
    **if** *success = TRUE* **then**
      $\mathcal{R}c_t = s_t$
    **else**
      $\mathcal{R}c_t$ = $\mathcal{R}s_{t-1}$
    **end**
  **end**
**else if** $\mathcal{F}(s_t) =$ *MODERATE FAILURE* **then**
  **if** $\mathcal{P}(s_t)$ *is NULL* **then**
    **if** *Attempt is FALSE* **then**
      $\mathcal{R}c_t$ = $\mathcal{R}s_{t-1}$
    **else**
      $\mathcal{R}c_t$ = Expert()
    **end**
  **else**
    success = execute($\mathcal{P}(s_t)$);
    **if** *success = TRUE* **then**
      **if** *Attempt = FALSE* **then**
        $\mathcal{R}c_t = s_t$
      **else**
        $\mathcal{R}c_t$ = Expert()
      **end**

    **else**
      **if** *Attempt = FALSE* **then**
        $\mathcal{R}c_t$ = $\mathcal{R}s_{t-1}$
      **else**
        $\mathcal{R}c_t$ = Expert()
      **end**
    **end**
  **end**
**else**
  $\mathcal{R}c_t$ = Expert()
**end**
return $\mathcal{R}c_t$;

The oracle adaptively learns a limited memory failure rate




(a) The State machine without an oracle displays non ergodic behavior

(b) The State machine with the oracle demonstrates how the addition of the oracle makes the state machine ergodic

Fig. 3. Ergodicity with the Oracle

for each state. Since this memory is a tunable parameter the frequency of expert intervention can be set by the designer of the state machine. This enables the oracle to classify the states as LOW_FAILURE, MODERATE_FAILURE or HIGH_FAILURE states. The oracle's shared autonomy recovery algorithm is illustrated in Algorithm 1.

### B. Preconditions

For most actions in states to succeed, there needs to be a set of preconditions that need to satisfied which are crucial to the success of that task. Defining the preconditions, for a state can be determined by the designer of the state machine based on the actions being carried out in the state. For example, if you consider the action of grasping an object, the preconditions of this action would be :
a) An object has been detected
b) A collision map for the environment is available
c) The gripper is empty.
Since these preconditions are task specific and determining them is strictly based on the design of the system, the designer can specify these preconditions during the state machine construction in our current system. During the normal execution of tasks the preconditions would not be invoked as they would have been satisfied by previous actions executed by the state machine. These preconditions are invoked only when a particular action fails. The precondition handling is controlled by the *oracle*

## V. Expert Interface

When the *oracle* queries the expert on the event of an action failure, the expert through an interface can transition the state machine to any state in the state machine for recovery. The expert also has the ability to supersede failure states through user defined shared autonomy interfaces and subsequently transition the state machine into a new state. These shared autonomy interfaces can be used to replace specific actions in the states of the state machine. This solution is particularly useful when the robot encounters situations which are unresolvable autonomously. For instance if a perception action fails to segment an object of interest, the expert can intervene with a user assisted segmentation interface. This will allow the expert to supersede the state performing the perception action and continue with the execution of the state machine. The assumption in this approach is that ultimately the expert can resolve all failures. The two shared autonomy interfaces that we provide and use in our frame work are a perceptual shared autonomy interface and a manipulation shared autonomy interface.

### A. Shared Autonomy

In the perceptual shared autonomy interface, we replace the robot's object detection module with a human assisted interface. The human assisted perception interface lets the human select a region of interest and the lets the system estimate the object based on the grabcut algorithm [17]. Similarly, the manipulation shared autonomy interface consists of a 6 degree of freedom teleoperation module which enables the expert to supersede the planning and control phase during manipulation failures.

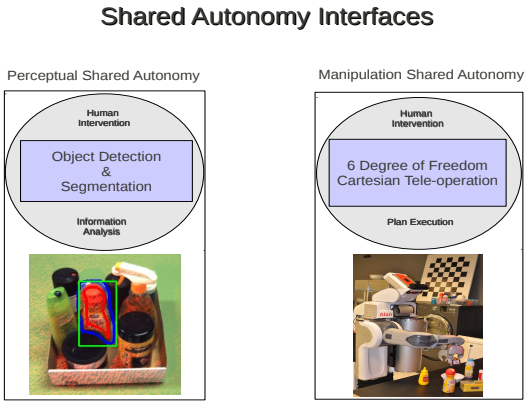



Fig. 4. The following diagram shows two shared autonomy interfaces. 1) The **Perceptual Shared Autonomy** interface is used to replace object detection with human assisted object detection and segmentation. 2) The **Manipulation Shared Autonomy** interface is used to replace planning and control with a 6 degree of freedom teleoperation device.

## VI. Experiments

To demonstrate the efficiency of our approach, we have built our system on top of the existing *SMACH* [2] architecture present in ROS [18]. *SMACH* is a task executive where tasks can be specified in the form of a state machine. The state machine is defined in terms of states, where these states consist of actions that need to be executed. Transitions are enabled by mapping the outcome of these states to the other states in the state machine.

### A. SMACH Failures

In *SMACH*, tasks relating to the accomplishment of a goal are specified as states in a state machine. Based on the outcome of the task execution, the state machine transitions from one state to another. In the underlying definition of a *SMACH* state machine, every outcome of the state in the state machine is mapped to a transition. In the event of an unsuccessful outcome, the state machine either transitions to some user specified state or it aborts. The aborted state machine would technically result in a failed attempt at the task. Our experiments were performed on state machines designed for the PR2 [19] robot built by Willow Garage. We tested the system on two separate use cases. The first use case involves a simple pick and place operation and in the second use case, the robot is commanded to retrieve a drink from the refrigerator.

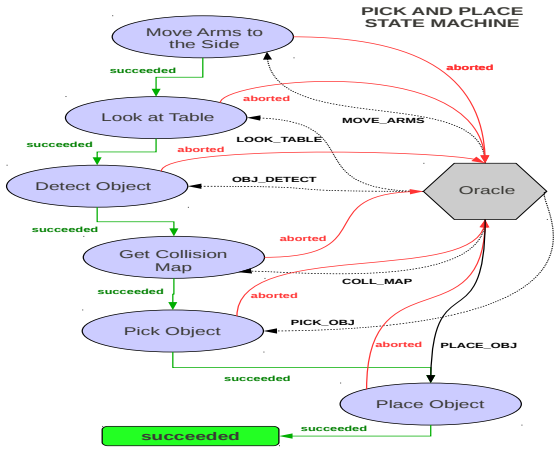



Fig. 5. State machine for the pick and place operation

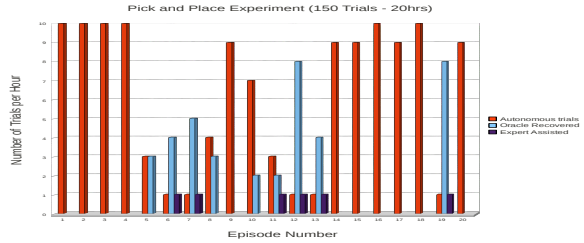



Fig. 6. Time series graph showing frequency of oracle recoveries and expert assists in 150 trials of autonomous operation over 20 hours.

*1) Use Case 1: Pick and Place Operations:* In our first use case, we evaluate the robot's long term performance on a simple pick and place operation. Here the robot is tasked with the detection of an object on a tabletop and moving it to a new location. The order of actions performed by the robot include, moving the arms out of the line of sight → pointing of the head of the robot → detecting the tabletop → estimating the collision map → picking the object → placing the object. As earlier noted these are high level task specifications outlined in the task executive and they do not detail low level behavior. The low level behavior embedded into these tasks are not visible to the task executive. The state machine of this system with the *oracle* is illustrated in Fig 5. We performed this test for 150 cycles over a period of 20 hours.

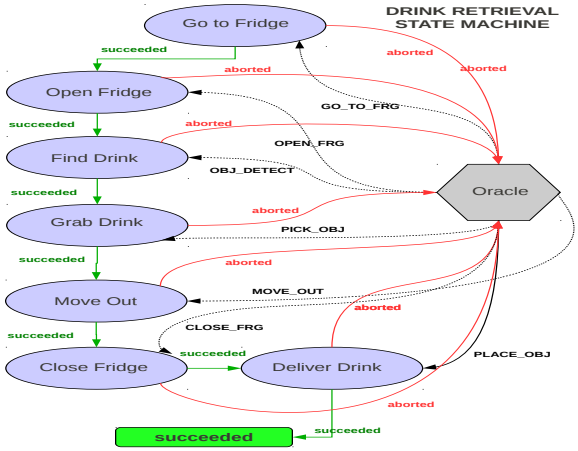



Fig. 7. State machine for the Drink Retrieval Operation

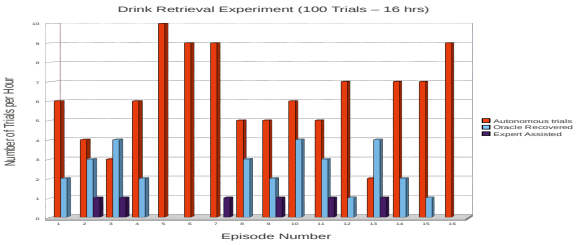



Fig. 8. Time series graph showing frequency of oracle recoveries and expert assists in 100 trials of autonomous operation over 16 hours.

*2) Use Case 2: Drink Retrieval:* In our second use case, we tested the algorithm on a drink retrieval state machine. Here the robot is tasked with retrieving a drink from a refrigerator. The level of complexity involved in the drink retrieval experiment is higher than that of the pick and place operation. Thus the robot state machine includes far greater number of states and hence it is more prone to failure. The order of actions performed by the robot include, raising the torso → opening the refrigerator → moving into the refrigerator → detecting the drink → picking the drink → moving out of the refrigerator → closing the door → moving to home position→ handing the drink. The state machine of this system with the *oracle* is illustrated in Fig 7. We performed this test for 100 cycles over a period of 10 hours. As it can be noted from Table I, the introduction of an oracle drastically reduces human supervision in such systems. In the absence of the oracle, every oracle recovered trial would

TABLE I
SHARED AUTONOMY STATISTICS

| Experiment | Runs | Oracle Recoveries | Expert Interventions |
|---|---|---|---|
| Pick and Place | 150 | 39 | 6 |
| Drink Serving | 100 | 46 | 9 |

have required human intervention. Given these statistics we can note that human intervention for the pick and place experiment was reduced from **26%** to **4%** and in the drink retrieval experiment, human intervention was reduced from **46%** to **9%**. On an average we get a **5 fold%** increase in performance.

## VII. DISCUSSION & FUTURE WORK

In order to attain the goal of long term autonomy, robotic systems need to be made both reliable and robust. However this cannot be exclusively achieved through task-level robustness as it is a component of the framework and not its entirety. There needs to be a considerable emphasis on system-level robustness too. An important design feature for robots systems would include ability to recover from failure. Simply because these systems cannot be designed to preempt every possible failure case.

In this paper, we have outlined a generalized task based recovery architecture which can be readily applied to any robotic system. From our experiences in continuous operations we argue that shared autonomy is an imperative component to achieve reliable long term operation in robotic systems. Sparsely supervised systems with effective failure recovery strategies can help achieve otherwise unattainable goals in robotic applications.

Since the size of such systems have a tendency to grow exponentially with complexity we intend to further investigate possible strategies to perform efficient inference on these systems. In the future we also intend to model these systems to constrain the search space of possible recovery behaviors. All the software used in this paper is readily available for download on `http://www.ros.org/wiki/recovery_shared_autonomy`.

### A. Unrecoverable Failures

Though the *oracle* based recovery can account for task failures, the system however does not account for non software related failures. These would come under the general umbrella of unrecoverable failures, as they cannot be addressed from the software side. This would include kernel panics, hardware failures, electronic failures and battery problems. The only realistic recovery possible in such cases is to store the state of the system externally and debug the system. Then the robot can be restarted from its externally stored state.

## VIII. ACKNOWLEDGEMENTS

We like to thank Dejan Pangercic of Technische Universität München and Christian Bersch of Bosch RTC for their insightful comments and help in setting up the use case experiments. We also like to Chris Mansley of Rutgers University and Dr. Philip Roan of Bosch RTC for their comments. Lastly we like to thank Dr. Jan Becker of Bosch RTC, for his support and guidance.

## REFERENCES

[1] W. Meeussen, E. Marder-Eppstein, K. Watts, and B. Gerkey, "Long term autonomy in office environments," in *ALONE Workshop,In Proceedings of Robotics: Science and Systems (RSS'11)*, Los Angeles, USA, 2011.

[2] J.Bohren, R. B. Rusu, E. Jones, E. Marder-Eppstein, C. Pantofaru, M. Wise, L. Mösenlechner, W. Meeussen, and S. Holzer, "Towards autonomous robotic butlers: Lessons learned with the pr2," in *International Conference on Robotics and Automation (ICRA'11)*, Shanghai, China, May 2011.

[3] S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Henning, T. Hofmann, M. Krell, and T. Schmidt, "Map learning and high-speed navigation in RHINO," in *AI-based Mobile Robots: Case Studies of Successful Robot Systems*, D. Kortenkamp, R. Bonasso, and R. Murphy, Eds. MIT Press, 1998.

[4] S. Thrun, M. Bennewitz, W. Burgard, A. Cremers, F. Dellaert, D. Fox, D. Hähnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz, "MINERVA: A second generation mobile tour-guide robot," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1999.

[5] I. R. Nourbakhsh, C. Kunz, and T. Willeke, "The mobot museum robot installations: a five year experiment," *Proceedings 2003 IEEERSJ International Conference on Intelligent Robots and Systems IROS 2003*, vol. 4, no. October, pp. 3636–3641, 2003.

[6] R. Siegwart, "Robox at expo.02: A large-scale installation of personal robots," *Robotics and Autonomous Systems*, vol. 42, no. 3-4, pp. 203–222, 2003.

[7] G. Kim, W. Chung, K. rock Kim, and M. Kim, "The autonomous tour-guide robot jinny," in *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS*, 2004, pp. 3450–3455.

[8] M. Maurette, "Mars rover autonomous navigation," *Auton. Robots*, vol. 14, pp. 199–208, March 2003.

[9] C. Baker, D. I. Ferguson, and J. M. Dolan, "Robust mission execution for autonomous urban driving," in *10th International Conference on Intelligent Autonomous Systems (IAS 2008)*, 2008, pp. 155–163.

[10] B. Pitzer, M. Styer, C. Bersch, C. DuHadway, and J. Becker, "Towards perceptual shared autonomy for robotic mobile manipulation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[11] E. Marder-Eppstein, E. Berger, T. Foote, B. P. Gerkey, and K. Konolige, "The office marathon: Robust navigation in an indoor office environment," in *International Conference on Robotics and Automation*, 05/2010 2010.

[12] N. Nilsson, A. I. C. S. International), and S. R. I. M. P. CALIF., *A Hierarchical Robot Planning and Execution System*, ser. Technical note. Defense Technical Information Center, 1973.

[13] B. Lee and E. A. Lee, "Hierarchical concurrent finite state machines in ptolemy," in *Proceedings of the 1998 International Conference on Application of Concurrency to System Design*, ser. CSD '98. Washington, DC, USA: IEEE Computer Society, 1998, pp. 34–.

[14] P. J. Lucas, "An object-oriented language system for implementing concurrent, hierarchical, finite state machines," Champaign, IL, USA, Tech. Rep., 1994.

[15] N. Dantam and M. Stilman, "The motion grammar: Linguistic perception, planning, and control," in *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, June 2011.

[16] D. Harel, "Statecharts: A visual formalism for complex systems," *Sci. Comput. Program.*, vol. 8, pp. 231–274, June 1987.

[17] C. Rother, V. Kolmogorov, and A. Blake, "Grabcut: Interactive foreground extraction using iterated graph cuts," *ACM Transactions on Graphics*, vol. 23, pp. 309–314, 2004.

[18] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[19] W. Garage, "The pr2 robot," http://www.willowgarage.com/pages/pr2/overview.

[20] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *ICRA*, 2011, pp. 1470–1477.