Human Supervision of Multi-Robot Systems

Donald Bourque(RBE), Thomas DeSilva(CS), Nicholas Otero(RBE)

April 30, 2015

Report submitted to:

Professor Taskin Padir(RBE/ECE)

Professor Sonia Chernova(RBE/CS)

This report represents the work of WPI undergraduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on its website without editorial or peer review. For more information about the projects program at WPI, please see http://www.wpi.edu/academics/ugradstudies/project-learning.html

Abstract

This project involves the creation and testing of a framework for human supervision of multi-robot systems (HSMRS). The framework serves as a guide for others to implement their own systems. The framework contains abstractions of relevant entities such as Agents and Tasks, workflows which depict the interactions between the entities, and design guides for integrating the system. A survey of multi-robot system research was conducted and areas of strength, such as distributed coordination and utilities, were chosen as inspirations for the final design. The framework is tested using a scenario involving ground coverage tasks, similar to what would be required of systems deployed for search and rescue and exploration. An analysis of the test reveals how many of the framework's specified requirements have been met as well as why some of the requirements have not been met.

Acknowledgments

We would like to acknowledge Professors Padir and Chernova for advising our MQP as well as Joseph St. Germain and Clearpath Robotics for providing us with equipment and advice.

Contents

TABLE OF AUTHORSHIP

	Author
1 Introduction	
1.1 Motivation	Team
1.2 Background	
1.2.1 Heterogeneous Multi-Robot Systems	Donald
1.2.2 Human-Robot Interaction	Team
1.2.3 Areas of Application	Nicholas
2 Design	
2.1 Requirements	Team
2.2 Inspirations From Research	
2.2.1 Iocchi et al.	Donald
2.2.2 Shah et al.	Donald
2.2.3 Scerri et al.	Donald
2.2.4 Trade Study Results and Conclusion	Donald
2.3 Framework Structure	
2.3.1 Task, Progress, and Prerequisite	Thomas
2.3.2 Agent, Robot, and Human	Nicholas
2.3.3 TaskList and Role	Thomas
2.3.4 UtilityHelper, AttributeWeights, and AgentState	Thomas
2.4 Framework Workflow	
2.4.1 Task Creation	Thomas
2.4.2 Bidding	Thomas
2.4.3 Task Completion	Thomas
3 Methods	
3.1 Graphical User Interface	
3.1.1 Design Philosophy	Nicholas
3.1.2 Framework Implementations	Nicholas
3.1.3 Situational Awareness and Interaction	Nicholas
3.2 Robot Selection and Modification	
3.2.1 Turtlebots	Nicholas
3.2.2 Hermes	Thomas
3.2.3 Husky A100	Donald
3.3 Navigation Stack Development	
3.3.1 Localization	Nicholas
3.3.2 Planning And Execution	Nicholas

		Author
B.4	Human	
B.5	Prerequisite	
B.6	Progress	
B.7	Robot	
B.8	Role	
B.9	Task	
B.10	TaskList	
B.11	UtilityHelper	

List of Figures

List of Tables

1 Introduction

1.1 Motivation

In scenarios such as search and rescue, disaster response, and sample return missions, teams of humans are currently needed to complete tasks to reach an objective. However, these scenarios and tasks are often dangerous for humans, or could be completed more quickly, safely, or reliably by an autonomous system [1]. Unfortunately, the level of autonomy available in current systems is insufficient to operate a robot effectively and reliably in these types of scenarios without the aid of a human [2].

To address this shortcoming, a human can be introduced into the system. Humans offer quick, highly-abstract situation assessment and decision-making capabilities to semi-autonomous robots, improving reliability and adaptability. This is advantageous in unstructured or highly dynamic environments where performing tasks correctly and efficiently may be computationally intractable. Furthermore, because the scenarios noted above demand a range of sensing, manipulation, and mobile abilities that are typically not available on a single robotic platform, the incorporation of multiple different robotic systems is advantageous. Systems which include these different capabilities will henceforth be known as heterogeneous systems.

There is a need for a control system which can effectively utilize and coordinate a human supervisor and a team of heterogeneous, semi-autonomous mobile robots [2]. The purpose of this project is to fulfill this need by creating a framework for human - robot team coordination and to demonstrate it in a situation representative of the aforementioned scenarios.

This paper defines cooperation as multiple agents working to accomplish a common, abstract goal and defines a robot as exhibiting semi-autonomous behavior as being capable of autonomous operation while performing a set of non-trivial tasks, but allowing for or requiring the input of a human operator in order to be fully functional. With these definitions in mind, the following goals for this project were identified:

- Develop a framework to enable teams of heterogeneous robots to be used for various applications while taking advantage of human supervisors to aid in tasks which the robots cannot or should not perform on their own,
- Demonstrate this framework in representative test missions.

These goals were fulfilled by completing the following objectives:

- Implement algorithms for task allocation of members of a team of heterogeneous semi-autonomous robots based on their unique sensors, actuators, and other characteristics,
- Design an effective user interface (UI) for a human supervisor,
- Create algorithms for inferring human intent from UI and relaying this information to the robot team,
- Devise methods of testing, evaluating, and validating the system.

1.2 Background

The problem of creating a human supervised multi-robot system (HSMRS) can be divided into two major disciplines: multi-robot cooperation and human-robot interaction. These two disciplines are both active areas of study in Robotics. In this section, information relevant to HSMRSs gathered through studies into both these areas is presented. Additionally, a selected sample of applications of HSMRSs are discussed in order to understand some of the expectations and challenges of such systems.

1.2.1 Heterogeneous Multi-Robot Systems

Multi-robot systems are becoming an increasingly important topic in robotics research because of the large potential applications of such systems [3, 4]. The potential benefits of utilizing multi-robot systems become apparent in certain problem spaces when comparing the success of human teams with that of single humans. A coordinated chain of neighbors passing along buckets of water requires less effort on behalf of each person individually and the team as a whole performs more efficiently [3]. With a strategy, the problem can be divided into tasks that need to be completed and roles to be played to carry out these tasks. These roles can be assigned to people based on their preferences and capabilities in order to take advantage of the individual skills of each person; thus, humans cooperating in teams both can complete tasks more effectively than and are capable of performing tasks impossible for a single person. Similarly, a team of robots can exhibit these qualities as well.

Just as a single person may possess the skills to perform every task needed to achieve a defined goal, a single robot may be equipped to complete every kind of task it could be involved in without help. Unlike humans, robots do not tire or become distracted, but depending on the complexity and time constraints of the tasks involved, it may be completely infeasible to develop a robot that is capable of performing every possible task. Typically in a human team, people with different specializations are brought together to form a group with a broad combined skill set to solve a problem. Multi-robot systems can be formed the same way. Instead of attempting to create a single, perfectly reliable, multi-purpose robot, robots with different capabilities may be combined into a team which can accomplish the required tasks. Even if these specialized robots are individually less robust and reliable,the overall robustness and reliability of the system will actually improve [3] and the potential areas of application for the system will expand with the use of multiple robots.

Devising a system to control multiple robots introduces the problem of coordination, but robotic systems in general already face many challenges that become magnified with the incorporation of more robots. Before any coordination methods can be implemented, each robot must be individually capable of navigating its environment and avoiding potential obstacles. To do this, they each must be able to sense their environment, plan their paths through the environment, and communicate information about the environment and their individual states back to the system; thus, each robot must be capable of acting semi-autonomously.

1.2.1.1 Communication

Once a team of semi or fully autonomous robots has been assembled, the pertinent information needs to be identified and certain communication policies must be established. Specifically, what sensor and state data will be shared (e.g. robots' individual capabilities for performing the different tasks, progress updates on task completion, or instructions for beginning and terminating of team plans) must be determined while designing the multi-robot system. This communicated information can be processed in either a centralized or decentralized manner, and is used in activities such as world modeling or for coordinating team plans and task execution.

The Azzurra Robot Team's implementation of a heterogeneous multi-robot system for the RoboCup competition provides an example of these communication policies.[5] RoboCup is a robot soccer competition in which teams of humanoid robots, shown in Figure 1, autonomously compete against each other. The team is considered heterogeneous since each robot has varying hardware and different control schemes. The team uses a decentralized system where each robot is fully autonomous and interacts with the other teammates, but relies solely on local control [5]. This differs from a centralized system in which robots require higher-level instructions to be explicitly given to them.

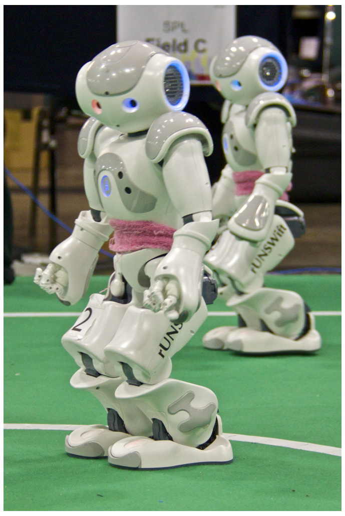


Figure 1: Robots competing in RoboCup [6].

When deciding whether to implement a centralized or a decentralized coordination framework certain assumptions must be made about the system's operating environment as each has its own benefits and drawbacks. Centralized systems typically allow for optimal solutions for planning and task allocations [3] as they combine and coordinate information in an efficient way [7], but require greater communication overhead as well as demand consistent communication and that the robots be able to reliably reconstruct global information from their sensors [5]. Decentralized systems offer increased flexibility and redundancy [3] leading to a greater level of robustness. Each robot gathers its own belief about the state of the environment and the state of the other robots and uses this to determine which actions it should take. This characteristic allows the robot to still function when it has completely lost communication with the rest of the team. A decentralized system lessens the communication overhead needed as well as allows for inconsistent communication, but at the potential cost of sub-optimal solutions for team coordination. The RoboCup competition frequently has communication failures, and it proves to be difficult to accurately reconstruct a global view of the field [5]. For these reasons and because other teams will be competing for bandwidth, most teams, including the Azzurra Robot Team, choose to use a decentralized means of coordination.

1.2.1.2 Task Allocation

The objectives for the system must be broken down into a number of tasks that need to be completed to meet the goals. It is common to then associate these tasks with different roles to be played by the robots. Before assigning robots to roles, the robots' utility must be calculated for each role. Utility functions provide a quantitative measure of the estimated effectiveness of each robot in a given role [5] using the known information about the state of the robot and the environment. To develop utility functions, the variables that are relevant for the execution of the task associated to a role must first be identified. The utility function could be a linear combination of these variables with coefficients assigned to each one (as in Eq. 1). These coefficients will be different for each robot and may need to be experimentally determined. With the utility functions of each robot calculated for a given role, an optimal choice of robot can then be determined for a role.

$$U = \Sigma cx \tag{1}$$

In the task domain of a RoboCup soccer match, the Azzurra Robot Team defined roles for the robots to play such as forward, defense, or support, and has developed an algorithm for establishing a formation for the team based on the environment conditions. For the example of a robot considering playing as a forward position, the robot's utility function will consider its distance and direction to the ball, the distance and direction to the opponent's goal, and the presence of obstacles in the trajectory from the robot to the ball [5]. The coefficients for each of the variables have been experimentally derived for each robot in order to achieve desirable behaviors with the system.

Tasks can be of different priorities and thus so can the roles that take responsibility for completing them. The role assignment process must be such that the most crucial roles are being covered. It is also possible that not all robots are able to perform every role, creating a constraint satisfaction problem. Other possibilities include more than one robot playing the same role at the same time, there being a required number of robots assigned to a role, and robots being capable of fulfilling more than one role at a time. These specifications will need to considered when designing a multi-robot system. The actual assignment of roles to individual robots occurs differently depending on the whether the system is centralized or distributed. Centralized system coordination systems may utilize combinatorial optimization algorithms such as the Auction algorithm or the Hungarian Method [8] to seek optimal solutions, but require greater communication overhead in return.

In decentralized systems, each robot calculates the optimal role assignments for itself as well as every other robot based on its current belief of the state of all robots on the team. This can be done by calculating the utility functions of all robots for all roles and using the greedy algorithm with a priority queue of roles [5] to choose the best robots to perform each role. As long as each robot has the same data, which should be the case during consistent communication, calculated role assignments will the same. Intermittent communication will potentially cause temporary suboptimal allocations and a total loss in communication will cause the robots to all assume the highest priority role. Since each robot is performing all these calculations for itself without explicitly relaying commands to the other robots, the communication overhead needed is less than in centralized system.

Multi-robot systems will also typically utilize task planning procedures such as Simple Temporal Networks and Temporal Constraint Satisfaction Problem algorithms. These procedures perform task assignment and synchronization such that concurrent executions of tasks are valid [7]. Ensuring task execution and recovering from failures are common challenges of multi-robot systems [3] and multi-robot systems can combat these obstacles through dynamic team task planning. By providing status updates on task execution progress the system has the opportunity to become more robust. Human supervisors could further provide a means for validating success and failures of task executions. They do this by offering high level capabilities native to human critical thinking.

1.2.2 Human-Robot Interaction

The performance of robot teams can be greatly enhanced with the addition of human supervision [2, 9]. The effectiveness of human supervision is determined by the nature of interactions between the human and the robotic system. There are many modes and degrees of Human-Robot Interaction (HRI). Humans may directly control robots, give robots high level commands to execute semi-autonomously, or guide their behavior and actions with queues. Operators may use graphical interfaces, voice commands, gesture control, eye-tracking, etc. The level of control a supervisor has to coordinate a robotic team, the interface with which they they gain information about the team and the tasks, and the manner in which they issue commands are all important HRI decisions which greatly influenced the effectiveness of the HSMRS framework [9].

1.2.2.1 Situational Awareness

As a metric of interface effectiveness, Situational Awareness (SA), as defined by Endsley, is "The perception of elements in the environment within a volume of time and space [Level 1 SA], the comprehension of their meaning [Level 2 SA] and the projection of their status in the near future [Level 3 SA]" where the "environmental elements" pertain to the goals and decision tasks of the operator [10]. It is important to note that these may be different from the goals and decisions of the team as a whole. For example, an operator of a single bomb-disposal robot investigating an Improvised Explosive Device (IED) in a war zone needs to make very different decisions - and thus needs a different SA - from the team leader coordinating the operation as a whole. The former needs a clear view of the trigger mechanism and the robot's manipulators while the latter needs to know the positions and activities of personnel in the surrounding area. Figure 2 shows a graphical user interface which might be important to a team member coordinating the tasks of the team. Either member could be distracted by the other's decision making elements, even though they are part of the same team. The goal of SA systems design is to provide enough information to enable all three levels. However, too much information causes information overload, requiring excessive cognitive demand, which degrades human performance. It is also the goal of SA systems design to balance and manage this information and its presentation so it is easily interpretable to the user. This is why SA is constrained to a specific "volume" of time and space which is sufficient to encapsulate the situation about which the user is making decisions. Although good SA is not a guarantee of good decision making, it is usually requisite for consistently good decision making.

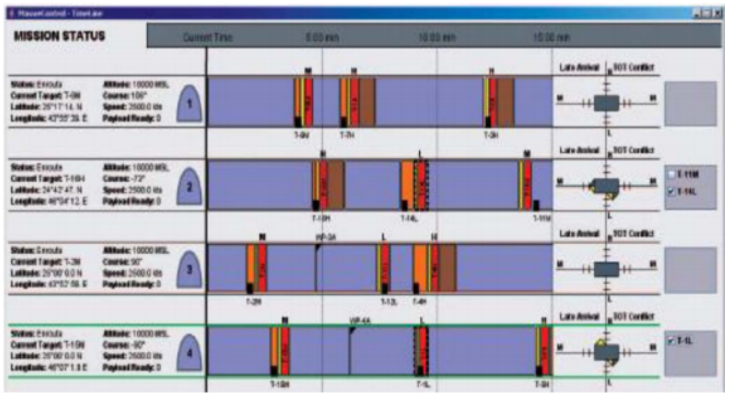


Figure 2: A graphical user interface showing a timeline for the tasks of a team of robots [9].

The first two levels of SA are accomplished, respectively, by the choice of information that an interface conveys and its manner of presentation of that information. For the two types of robot GUIs introduced above, the decisions the operators must make are on different levels, and thus the environmental elements they must perceive must be appropriate to those levels. The teleoperator is concerned with the specific state of their robot and its immediate surroundings, while the supervisor is concerned with the general state of multiple team members and their activities. In either case, the basic function of the interface is to help the user become aware of and attend to important information to facilitate level 1 SA (perception.) For level 2 SA, the users must be able to assimilate the information. This involves combining, interpreting, storing, and retaining elements [11]. The design of an interface that presents information comprehensibly is difficult but critical. Level 3 SA is largely dependent on the operator, but can be enhanced by system behavior discussed below.

1.2.2.2 Robot Behavior

Robot behavior may be considered another type of interface in that the actions of a robot team member will be interpreted by the supervisor to determine the current and future status of that robot [12]. This state information will then be incorporated into the supervisor's decision making. Therefore, robot behavior must be carefully considered so it is not only effective at performing tasks, but also easily understandable to human teammates that are coordinating those tasks. (Note that optimal behavior and understandable behavior are not always reconcilable.) Robot behavior is also one of the most important design considerations for level 3 SA: projection. There are two types of projection we may consider: projection of the state of the task domain and projection of the dynamics of the system being controlled or operated in that domain. Usually, only experts are able predict what will happen in a situation and act in anticipation of future events. This domain-specific mastery comes with years of experience and practice and unfortunately cannot be imparted through a coordination framework or interface. However, it is possible to devise a system which facilitates a user's projection of its future state. In this case, if a supervisor can understand the behavior of robotic team members, they can more easily project the future status and actions of those team members. The implicit information conveyed by robot behavior is supplementary to explicit system predictions provided by the interface [12].

For example, a supervisor watching a robot's progress in navigating to a goal point on a map may predict the robot's Estimated Time of Arrival (ETA) by assuming that the robot maintains the same average speed (a system behavior.) Their estimate could be confirmed by the robot's own estimate, displayed concurrently on the map interface. If this is a firefighting robot heading to put out building fire, and the supervisor is an experienced fire chief that can draw on his experience and predict that the building will collapse before the robot arrives, he can decide to cancel that assignment and give a new task to that team member. This level 3 SA is a combination of user skill and a synergy and concordance between robot behavior and interface information [12].

1.2.2.3 Neglect Tolerance

In applications where multiple robots are being controlled by a single human operator, the issue of neglect must be considered. Neglect of a robot occurs when the human operator is busy performing a task or servicing another robot. The effect of neglect on a robot depends on its level of autonomy and the accuracy of its sensors [13].

In a study of neglect tolerance, Michael Goodrich and his team implemented three levels of autonomy [13]. These levels included full tele-operation, waypoint navigation, and globally scripted navigation. For robots that must be tele-operated, neglect results in complete loss of performance. For robots that require short term goals, neglect results in a steady loss of performance as navigation errors accumulate and a full loss of performance when the robot has reached its goal. Robots that follow global scripted navigation goals can operate for longer periods of time while being neglected, but performance continues to decrease as errors accumulate.

The goal of this study was to learn how to maximize the performance of a multi-robot system in the face of neglect. It was found that using the naive approach of having each robot maximize their own performance led to an outcome similar to game theory's Prisoner's Dilemma. This outcome is a demonstration that distributed maximization of performance does not lead to a system-wide maximization. It was found that by estimating the complexity of the environment, robots were able to determine which levels of autonomy were appropriate to assume. Robots dealing with more complex environments assumed the higher performance tasks while those in simpler environments settled for levels of autonomy which sacrificed performance but were more neglect tolerant [13].

The overall neglect tolerance of a system depends on the levels of autonomy available to the individual robots and the accuracy of their sensors. Fully autonomous robots with noisy sensors will accumulate error in the absence of human correction and thus will perform poorly. Alternatively, robots which demand constant attention from the user but utilize precise sensors will also perform poorly. The number of robots deployable in a system depends strongly on the neglect tolerance of the system as a whole. If the human operator cannot provide enough attention to each of the robots to maintain a non-zero performance, then the system is too large to be well managed.

1.2.3 Areas of Application

Human supervised multi-robot systems(HSMRSs) have the ability to perform diverse tasks in parallel while being fault tolerant. Because most real world jobs require this functionality, HSMRSs are widely applicable. This section will examine a sampling of potential applications of HSMRS, including: industrial assistance, construction, space exploration, and search and rescue.

1.2.3.1 Industrial assistance

Automation has become prominent in many key manufacturing industries. However, because today's industrial robots are confined to highly structured environments and repetitive tasks, only those applications that warrant such expensive robot-centric design and processes are benefiting from robotic technology. Manufacturing of low-volume, complex, or customized items usually still necessitates human labor. Most labor in small manufacturing businesses is still manual and even in highly automated industries, such as automotive assembly, only about half of the work is done by robots [14].

Because automation technology is not likely to displace these remaining manual labor positions in the near future, there has recently been much attention towards robotic assistance of necessary human workers to increase production efficiency. For example, multi-robot fetch-and-deliver robotic systems help provide workers with tools and materials just in time to use them [14]. Robotic welding assistants can hold workpieces for workers to work on or inspect. These kinds of supporting activities will allow workers to focus on their work without needing to worry about their immediately accessible resources.

These robotic assistants also have the potential to decrease the occupational hazard of manufacturing. In 2013, 304 fatalities were reported in the manufacturing industry. Of those, 17% were caused by being struck by an object or equipment and 16% were caused in the transportation of materials [15]. These fatalities can be prevented by using teams of robots under the supervision of workers to operate in dangerous locations and perform hazardous tasks.

1.2.3.2 Construction

Construction jobs are great examples of situations in which multiple workers must complete diverse tasks. Its is also an industry that has seen very little advances in automation. This is because robots working in this field must be able to operate in unstructured environments where they handle a variety of objects which vary in size, shape, and weight, and safely interact with other nearby workers. All of this must also be done in environments with low levels of standardization [16].

HSMRSs have the potential to overcome these challenges and accelerate the pace of automation in construction. The problem of operating in an unstructured environment can be overcome through feedback from the human supervisor. The manipulation problem brought about by the need to handle objects of various sizes, shapes, and weights can be simplified by using a heterogeneous robot team with individuals specialized to handle a small subset of the objects. Because of the already explicit need to work alongside humans, HSMRSs already take precautions to protect collaborating workers.

Similar to manufacturing, automation in construction has the potential to save many lives. According to the National Census of Fatal Occupational Injuries in 2013, construction workers suffered more fatal injuries than any other profession [15]. Of those fatal injuries, 37% of them were from falling, slipping, or tripping. Assistive robots such as those described in the manufacturing section could help decrease the number of fatal injuries due to this cause by limiting workers' movement when in dangerous locations. Additionally, robotic workers can perform tasks in the vicinity of dangerous machines which claimed the lives of 10% of the lethally injured construction workers in 2013 [15].

1.2.3.3 Space exploration and colonization

Space exploration and colonization technologies have gained attention in recent years. Private companies such as Space Exploration Technologies Corporation, Planetary Resources, Mars One, and Deep Space Industries are developing technologies to bring humans and robots farther into the solar system and to return resources back to Earth [17, 18, 19, 20]. For their missions to succeed, they will need workers, human or otherwise, to build infrastructure in space.

Humans have proven to be efficient and effective workers in space. In an MIT Space Systems Laboratory experiment, results showed that human productivity on orbital missions was several times greater than the productivity for similar tasks on Earth [21]. Unfortunately, space poses a myriad of dangers to humans. To protect themselves from these dangers, humans must wear expensive and heavily engineered space suits. In addition, humans need to bring supplies and life support with them on missions, further adding to mission costs.

Although robotic workers play an increasingly important role in space missions, productivity rates are still higher for missions with human presence or supervision over those with fully-autonomous robotic systems [22]. Despite being less productive, robots have a number of advantages when operating in space. Because robots do not require food, water, or life support, these resources do not contribute to launch costs. Robots also do not suffer from fatigue, and thus can work for longer periods of time than humans. Additionally, robots are more expendable than humans, allowing them to take on high risk missions. In the event that a robot is damaged, it can also be repaired or repurposed.

HSMRSs can combine the advantages of human and robotic workers in space. Simple labor can be assigned to robot teams to complete autonomously. For robots assigned complicated tasks, humans can share control while remaining in a safe location. If a task is too complicated for any of the robots on the team to perform efficiently, the team can assist the human worker in executing the task. This will decrease the human worker's exposure to hazardous conditions.

1.2.3.4 Search and rescue

Search and rescue missions have been a large area of interest for multi-robot systems in recent years. This is because these missions often require agents to explore dangerous environments for extended periods of time. Rather than risk the lives of human team members, it is preferable to deploy robots. These environments tend to be unstructured and hard to navigate. These conditions make autonomous robots very ineffective.

An alternative is to have humans remotely tele-operate the robots. This is an effective solution if the human team is the same size or larger than the robot team and if communications between the remote human team and the robot team are unimpaired. While the first condition is reasonable, the second cannot be easily assumed. In situations with high radiation environments, such as the Fukushima Daiichi Nuclear Power Plant, communication between robots and remote human teams is limited [23].

Using a HSMRS, the teams can take advantage of the robots' autonomous capabilities until those capabilities become insufficient to perform a task. The human team can then intervene at a high level or take complete control of the robot until the robot is able to resume on its own. While the robot is operating autonomously, human supervisors are also able to focus on the sensor output instead of micromanaging the robot [2].

2 Design

2.1 Requirements

A Systems Engineering methodology was followed in the design of this framework. This methodology led to many important and relevant results, which ultimately led to the formulation of the requirements for the framework. A full report on the Systems Engineering results can be seen in Appendix A.

The formulated requirements were instrumental in driving the design of the framework. Each requirement was linked to one or more stakeholders to validate its performance. This helped to focus the team on the aspects of the system which would be most desirable by the people using the system. The requirements were also prioritized. This helped the team identify where concessions could be made if they needed to be made. Concessions would occur on the lowest priority tasks.

Table 2 shows the tabulated requirements and their associated meta data. The stakeholder IDs are defined in Table 10.

ID	Category	Requirement Statement	Stakeholder	Verification	Priority
R.01	Core	The system shall incorporate at least two robotic agents.	SH.02	A robot team made of two or more robots is be initialized and deployed.	1
R.02	Core	The system shall be able to represent different robot configurations and capabilities that are comparable to those of existing field robots currently used in the target scenarios.	SH.02	A robot team made of two or more heterogeneous robots is represented.	1

R.03	Core	The system shall be able to utilize the different robot configurations and capabilities specified in R.02	SH.02	Only robots with necessary capability are dispatched to complete a subtask.	2
R.04	Communication	The system shall send relevant information about the agents in the robot team to the supervisor UI.	SH.01	The human supervisor will control the robot team with the information given.	1
R.05	User interface	The supervisor UI shall update its representation of the team and task status at a frequency suitable for effective supervision.	SH.01	User opinion on UI update speed is acceptable in a to-be-determined proportion of cases.	3
R.06	Communication	Components of the system shall be able to send data to other components at a rate suitable for the coordination and supervision of the entire team.	SH.01	Data is transmitted in time for all dependent system components to execute properly.	3
R.07	Communication	The system shall provide robots with tasks to complete.	SH.02	Robots receive tasks.	2
R.08	Core	The system agents shall be able to accomplish tasks relevant to the aforementioned scenarios.	SH.02	Sample tasks representative of the scenarios of interest will be used to verify this requirement.	3
R.09	User Interface	The supervisor shall be able to utilize all capabilities of the system using the user interface.	SH.02	The full capabilities of the system must be enumerated and the user interface must be tested against each one.	3

R.10	User Interface	The user interface shall require only a single device to interact with the user.	SH.02	The user only needs one device in order to run the UI.	4
R.11	User Interface	The device running the user interface shall be mobile.	SH.02	The user can move around their environment with the device running the UI.	4

Table 2: Requirements for the HSMRS project.

The framework needs to enable implementations to have capabilities which would meet these requirements. The team determined the following capabilities through inspection of the requirements:

The framework should enable common activities to:

- Be abstracted and encapsulated.
- Have a unique measure of the capabilities needed to be completed.
- Have an order to be completed in relative to other activities.
- Be bundled with other activities to create more complicated activities.

Additionally, robots should be able to:

- Take responsibility for completing an activity.
- Ask for help if it cannot complete an activity autonomously.
- Communicate regular status updates to other actors in the system.
- Be able to act autonomously or be controlled by a human.

Finally, the framework should give human supervisors the ability to:

- Assign responsibility to a robot to complete an activity.
- Directly control one or more robots.
- Establish situational awareness for whatever task is being focused on.
- Project an intuition about future states of the environment to increase the efficiency of the system.

2.2 Inspirations From Research

While researching into multi-robot systems and human-robot interaction, several sample implementations were reviewed. A trade study was then performed on those with characteristics in line with the system requirements derived from a stakeholders needs analysis described in Table 11 of Appendix A.2. The evaluation criteria for the studies and the associated weighting percentages of each criterion can be found in Table 3. It was determined that the most desirable characteristics of a framework were to contain testable components, have the ability to be generalized to heterogeneous systems, and allow for increased situational awareness of the system. More details on the generation of these weight values can be found in Appendix A.

1. Adaptability to intermittent communication / loss of robots
2. Representation and handling of neglect tolerance
3. Supports Situational Awareness (SA) and ease of input for the user through a UI
4. Ease of implementation/simplicity for development team (schedule risk)
5. Generalization to heterogeneous robot teams
6. Generalization to different task domains
7. Intelligent task allocation
8. Testable - How well performance can be quantified

Criterion #	Weight (%)
1	3
2	7.59
3	17.24
4	13.22
5	19.84
6	7.9
7	6.17
8	25.03

Table 3: Each criterion with their associated weights.

Three specific papers were analyzed in a trade study: "Distributed Coordination in Heterogeneous Multi-Robot Systems"[5], "Fast Distributed Multi-Agent Plan Execution with Dynamic Task Assignment and Scheduling"[24], and "A Prototype Infrastructure for Distributed Robot-Agent-Person Teams"[1]. Each paper was chosen for its novel approach to different challenges inherent to the design of multi-robot systems. The final framework considered much of the evaluated design characteristics presented in these papers.

2.2.1 Iocchi et al.

Iocchi's work in "Distributed Coordination in Heterogeneous Multi-Robot Systems"[5] discusses a framework for the distributed coordination of a multi-robot system based on dynamic role assignment. The Azzurra Robot Team has implemented this approach within its team of heterogeneous robots for the RoboCup robotic soccer competition. This paper was of particular interest for this project due to its task allocation and coordination procedures in a dynamic environment involving role assignment and formation selection. The results for this framework's evaluation can be seen below in Table 4.

Criterion #	Weight (%)	Grade
1	3	3
2	7.59	0
3	17.24	1
4	13.22	3
5	19.84	5
6	7.9	1
7	6.17	4
8	25.03	4
Weighted Total	100	29.78

Table 4: Evaluation results of Iocchi et al.'s framework.

This paper discusses the use of role assignments to provide a high level means of assigning behaviors to the different Robots. Each robot, though very similar, contains different hardware and control software resulting in a heterogeneous team. Utility functions, which map a Robot's fitness for a specific role, are calculated for each robot in a distributed manner allowing for the self-assignment of roles. Data gathered about the world model is then used to determine the desired team formation to use. A greedy-based algorithm is used to determine the role allocations. Roles also contain priority levels which allows the most important role positions to be filled first by the task allocation algorithm. Prioritized roles also allows for the robots to assume the most important roles during intermittent or lost communication between the robots.

Many of the characteristics of Iocchi's framework design were determined to be closely in line with our team's framework requirements. Through its distributed coordination and role priorities the system was tolerant to the intermittent loss of communication posed by the RoboCup field. Also by calculating utility functions, this framework is generalizable to different heterogeneous teams. The task allocation was intelligent through its use of utility functions and role priorities, and the greedy-based algorithm used is simpler to implement than other complex methods that could provide more optimized allocations. The paper also provided many insights into testing the performance of the system. This framework did not incorporate an actively involved human supervisor, and thus did not handle the problem of neglect tolerance or providing substantial situational awareness to the supervisor. Its formation/role design, though very useful for the RoboCup soccer competition, did was determined to not be generalizable to other task domains. Its use of roles seemed very appealing, but it did not allow for the expansion of roles to make the addition of future roles easier. The use of roles in our framework needed to be implemented such that it is expandable and adaptable for use in various applications.

2.2.2 Shah et al.

Multi-agent systems commonly use a planning procedure that performs task assignment and synchronization. Shah et al.'s work in "Fast Distributed Multi-Agent Plan Execution with Dynamic Task Assignment and Scheduling"[24] was the second considered framework in this trade study which describes plan execution with dynamic task assignment as well as scheduling. In it, flexible temporal plans are created for scheduling tasks. The results for this framework's evaluation can be seen below in Table 5.

Criterion #	Weight (%)	Grade
1	3	5
2	7.59	0
3	17.24	0
4	13.22	3
5	19.84	5
6	7.9	5
7	6.17	5
8	25.03	3
Weighted Total	100	29.93

Table 5: Evaluation results of Shah et al.'s framework.

This paper was of interest due to its plan execution and scheduling methods for multi-robot coordination. It discusses the use of Simple Temporal Networks (STN) and Disjunctive Temporal Constraint Networks also known as Temporal Constraint Satisfaction Problems (TCSP). The paper introduces an executive agent named Chaski which successfully performed multi-manipulator coordination of two Barrett Arms. Chaski enables the execution of temporally flexible plans by allowing agents to dynamically update their plan in response to task assignment and schedule changes of the other agents [24]. It guarantees that chosen schedules and action executions remain temporally consistent and valid within the multi-agent plan.

The methods in this paper are useful in dynamic environments that require frequent plan repair [24]. Those described were determined to be greatly generalizable to heterogeneous teams as well as different task domains. The task allocation methods were very intelligent, and the broadcasting of event completions would be very useful in task scheduling for multi-robot systems. This paper did not involve a human supervisor to address the situational awareness or neglect tolerance requirements of our framework. It was also determined that this highly centralized system, which generates event graphs consistent with the given constraints at the start of the system, may be very difficult to implement in a dynamic coordination framework where event lists would need to be dynamically updated for each of the robots.

2.2.3 Scerri et al.

The third framework considered due to its novel approach was Scerri et al.'s work in "A Prototype Infrastructure for Distributed Robot-Agent-Person Teams"[1]. Commonly, proxy-based architectures are used for coordination in teams of heterogeneous agents. This paper argues that these architectures have taken too much coordination responsibility away from the agents and proposed to return it through the notion of adjustable autonomy in which coordination roles can be explicitly given to a robot, agent or person. The results for this framework's evaluation can be seen below in Table 6.

Criterion #	Weight (%)	Grade
1	3	5
2	7.59	3
3	17.24	2
4	13.22	3
5	19.84	4
6	7.9	4
7	6.17	2
8	25.03	5
Weighted Total	100	36.036

Table 6: Evaluation results of Scerri et al.'s framework.

In this framework, a single human supervisor overlooks a virtual fire brigade simulation. The supervisor's job is to minimize the amount of damaged cause by fires in the simulation. The role allocation procedure work by the proxy of each agent determining the fitness of the other agents in the simulation. The agent's proxy will then determine whether or not it should accept a new role or whether or not it should pass it along to another agent. If it chooses to pass it along, it will pass the role assignment choice off to the next best agent to fulfill the role that has not been already been asked. The level of autonomy of the system can be adjusted by varying how many of the agents will be asked before resorting to the human supervisor to find a proper assignment for the queued role. The load on the supervisor was measured by the effectiveness of the system and the level of autonomy set by the supervisors during testing.

2.2.4 Trade Study Results and Conclusion

Overall, Scerri et al.'s work [1] described a framework that most met the needs of the framework design requirements and was thus chosen as a basis for our framework. The final results of the trade study can be seen below in Table 7. A few modifications, however, were needed to be made to this framework in order to arrive at the final framework: the use of proxies for the robots were removed, role assignment was restricted to a utility-based greedy algorithm, and functionality for progress reports was added on tasks.

Criterion #	Weight (%)	Iocchi et al.	Shah et al.	Scerri et al.
1	3	3	5	5
2	7.59	0	0	3
3	17.24	1	0	2
4	13.22	3	3	3
5	19.84	5	5	4
6	7.9	1	5	4
7	6.17	4	5	2
8	25.03	4	3	5
Weighted Total	100	29.78	29.93	36.036

Table 7: Results of the framework trade study.

2.3 Framework Structure

The framework is made up of eleven abstractions of different entities that exist within a Human-Supervised Multi-Robot System. These abstractions describe the tasks that must be carried out, the physical agents of the system, and their respective states. This section will describe the purpose of each of these abstractions in detail. Source code for the C++ abstract framework classes may be found in Appendix B.

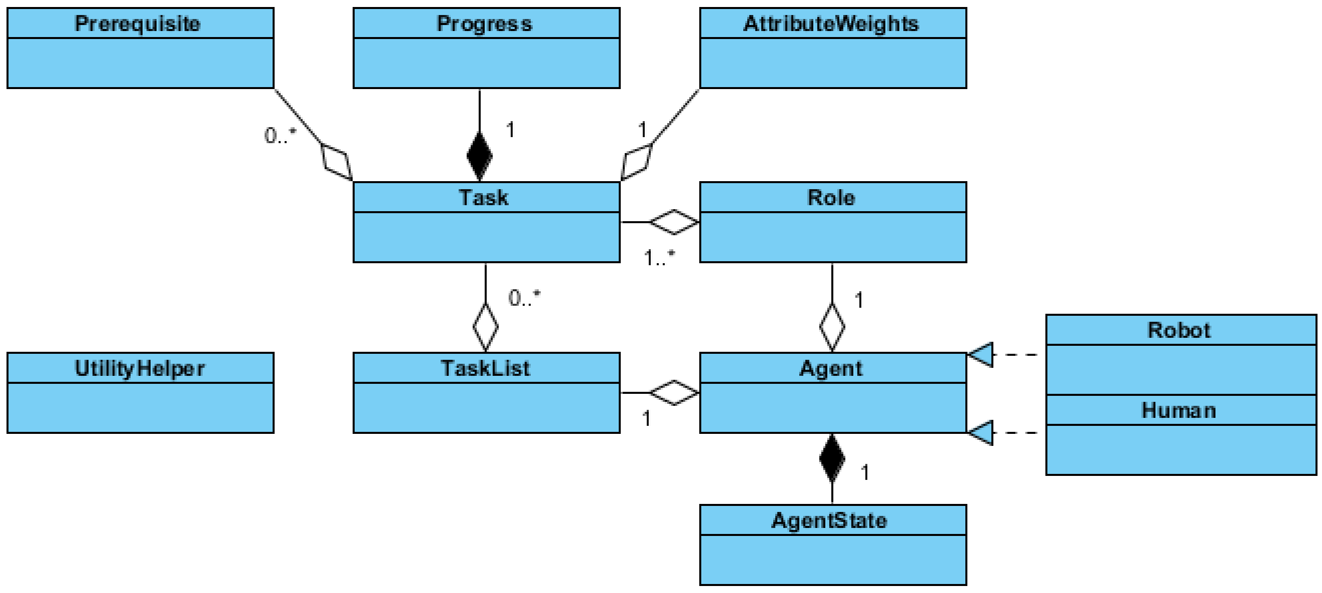


Figure 3: A UML diagram showing the structure of the framework.

2.3.1 Task, Progress, and Prerequisite

The framework stores abstracts common activities into an entity known as a Task. Information about tasks is represented in three different parts: the Task, the Progress, and the Prerequisite. Tasks have a type (e.g. find an object), parameters to further define the desired work (e.g. what object to look for, where to look), and a list of Agents who own the Task, as well as Progress and Prerequisites. Progress keeps track of how much of a task has been completed, and Prerequisites represent conditions which must be fulfilled before a task can be started. Tasks can be queried for any of this information, as well as have their ownership adjusted. Tasks can gain owners through the automatic bidding process, or by direct assignment from the supervisor. The supervisor may also revoke an Agent's ownership of a Task.

2.3.2 Agent, Robot, and Human

Three of the framework abstractions involved physical actors within the system. In this framework, humans and robots can work side by side to accomplish tasks. Therefore the shared characteristics of these two entities could be abstracted into an Agent. Agents have the ability to receive tasks, to claim tasks, and to execute tasks. They also have the ability to communicate with the system's human supervisor. Communications could include general feedback and sending standard updates.

A concrete type of an Agent is a Robot. Robots are able to do everything an Agent can do with the addition of their own abilities. Robots must bid on tasks before claiming them. This bidding serves as the Robot's judgment to ensure that only the Robot(s) best suited for a task can claim it. To support bidding, Robots also have the capability to produce information about their internal states, such as their attributes, their state, and their utilities for a specific task. (The bidding process is explained in Section 2.4.) A critical ability held by a Robot actor is to ask for help. Asking for help is important because it improves the neglect tolerance of the system and helps stop Robots from wasting time and resources. Finally, Robots are able to request that a task be added to the list of tasks that need to be completed by the system.

The other type of Agent is a Human. While humans are able to complete tasks like Robots, their role in the framework is biased towards supervision. Humans are capable of assigning tasks to the team as well as approving tasks requested by Robots. They are also able to respond to help requests by robots and to take full command of the robot through tele-operation. Another capability of Humans is to assign roles. This is an important ability because it allows the Human to exercise their judgment in predicting the future state of the environment and to prepare the team to address tasks that don't yet exist. Roles and their significance are discussed in further detail in Section 2.3.3. As a necessity for supervision, Humans also collect and view all of the information generated by the system, from Robot states to perceptions of the environment.

2.3.3 TaskList and Role

The TaskList and Roles are organizational tools that the framework uses to associate Agents with Tasks. The TaskList is a list of all the Tasks that the supervisor has instructed the system to complete. Tasks in the TaskList are sorted by their priority (which is assigned by the supervisor), and the TaskList can be queried for either the next available Task of highest priority, or for a specific Task with a given ID number. Each Agent keeps its own copy of the TaskList in order to reduce load on the supervisor's computer, and Agents are responsible for keeping their TaskLists up to date.

Roles afford the supervisor course-grained control over task allocation by allowing them to restrict a Robot to performing certain tasks (as opposed to the fine-grained control they have through individual direct task assignment). A Role contains a set of Task types, and when a Robot is assigned a Role by the supervisor, that Robot will cease bidding on any Task not specified by its new Role. Supervisors can revoke Roles at any time to allow Robots to bid normally again. The main use of Roles is to allow the supervisor to prevent higher-capability robots from crowding out lower-capability ones such that all of the work is being performed by a small subset of the available workers.

For example, imagine a scenario in which there are two types of tasks: picking fruit and digging holes. A supervisor has a large fleet of robots, half of which are exceptional at both tasks–the ”good” robots–and the other half of which are merely passable at digging holes and incapable of picking fruit–the ”bad” robots. Under normal circumstances, good robots would always beat bad robots when bidding for tasks, and thus the only time a bad robot would get to work would be if all of the good robots were already busy and a hole needed digging. If all of the good robots are busy digging holes when the supervisor requests that fruit be picked, then no fruit will be picked until all of the holes have been dug.

Without Roles, the only way that a supervisor can remedy this situation is to manually reassign good robots from hole-digging to fruit-picking whenever a bad robot is available to take its place and finish digging. However, a supervisor who forsees this problem can assign Roles to the good robots to prevent them from taking hole-digging tasks, which in turn will allow bad robots to take those tasks, increasing the overall productivity of the system. The supervisor can dynamically adjust the number of good robots allowed to dig holes through Role assignment to accommodate the amount of fruit-picking required at any given time.

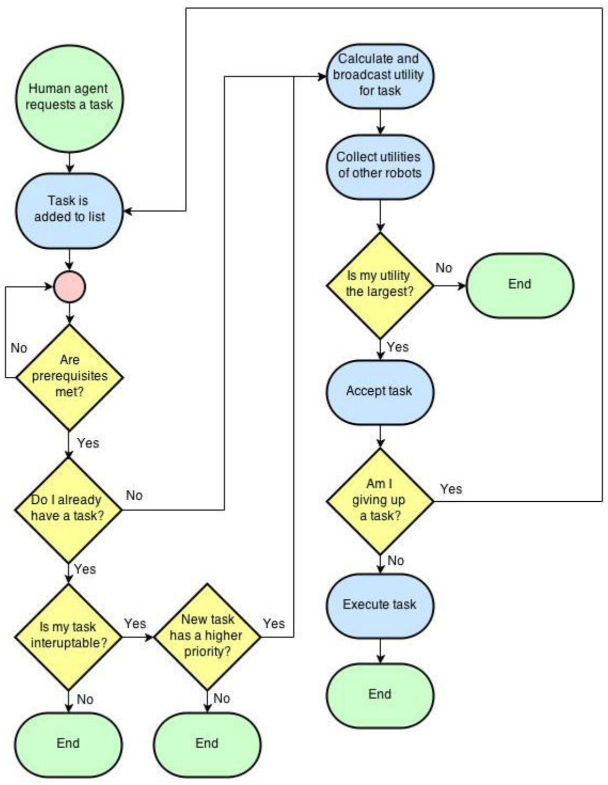
2.3.4 UtilityHelper, AttributeWeights, and AgentState

These three abstractions assist Robots with bidding. The UtilityHelper is capable of calculating a Robot's utility for a given Task using information from the Robot's AgentState and the Task's AttributeWeights. The AgentState keeps track of various attributes of an Agent, such as positon or maximum drivebase velocity. Any information about an Agent which is relevant to completing Tasks is stored in the AgentState, and each Agent is responsible for keeping their AgentState up to date. AttributeWeights are a static description of how important certain attributes are to completing a Task. For example, if a Task is to move to a location, important attributes to check would be an Agent's current position relative to the given location, and the Agent's maximum speed. In general, the UtilityHelper will combine the sums of the products of each attribute a in an Agent's AgentState with its respective weight w to determine the utility U of a robot for a task:

$$U = \Sigma a w \tag{2}$$

Special cases may be defined for individual Task types, such as incorporating a realistic time-to-travel calculation for Tasks requiring robots to drive to a different location instead of the default naive linear combination of attributes.

2.4 Framework Workflow



Human agent requests a task
Task is added to list
Are prerequisites met?
No
Yes
Do I already have a task?
No
Yes
Is my task interuptable?
Yes
No
End
New task has a higher priority?
Yes
No
End
Calculate and broadcast utility for task
Collect utilities of other robots
Is my utility the largest?
No
End
Yes
Accept task
Am I giving up a task?
Yes
No
Execute task
End

Figure 4: A visual representation of the framework workflow from an Agent's perspective.

2.4.1 Task Creation

The workflow starts with the supervisor adding a Task to the TaskList. Once the supervisor has specified the necessary parameters, the Task is sent to all connected Agents so they can add it to their respective TaskLists. Upon receiving a Task, each Agent evaluates the Task's Prerequisites to determine if the Task is ready for execution. If so, an auction for the Task is started.

2.4.2 Bidding

To start an auction for a Task, an Agent publishes a bid for the Task, which consists of the Agent's name, its utility for the Task at hand, and the Task itself. Other Agents receiving this bid must then decide whether or not to post a competing bid. An Agent may bid on a Task if one of the following conditions is met:

- The Agent is idle.

- The Agent is the owner of a preemptible Task of lower priority than the Task being auctioned. (A preemptible Task is one which may be interrupted without input from the supervisor, and whether or not a Task is preemptible depends on its type.)

If the Agent may bid, then it queries its UtilityHelper to calculate its utility and proceeds to publish its own bid. Regardless of whether or not the Agent was allowed to bid, it will listen for more bids on the same Task for the next few seconds and keep track of who is currently winning the auction and with what utility score. After the bidding period is over, the Agent (or Agents, in the case of Tasks that allow multiple owners) with the highest bid publishes a claim to the Task, and the other Agents record the winner in their TaskLists. If an Agent erroneously publishes a claim to a Task, the other Agents will publish corrections in the form of the real winner's bid. When ownership of the Task has been settled, the owning Agents begin execution.

2.4.3 Task Completion

When a Task is complete, the owning Agent publishes a message informing other Agents and the supervisor. The owning Agent will then check its TaskList for a new Task to auction, and other Agents will mark the Task as complete. Any Prerequisite reliant on the newly-completed Task will have its state set to being fulfilled.

3 Methods

In order to verify that the designed framework met the identified requirements, a test scenario was created. The framework was implemented for a team of mobile robots to operate in this scenario. A number of elements needed to be created, modified, and integrated in order to make this possible. This section will detail those elements, which include: a graphical user interface, robotic platforms, a simple but generalized navigation stack, and a set of tasks for the robots to complete. Finally, it will discuss how these elements were combined for the test scenario.

3.1 Graphical User Interface

The graphical user interface(GUI) is an essential part of any non-autonomous system. Through it, users can acquire situational awareness about the system's environment and internal state as well as send commands to the system. Because the system assembled for this project was composed of many robots which could all be operating in very diverse local environments, the role of the GUI was made even more critical. This section will discuss the philosophy behind its design, its integration with the proposed framework, and its methods of interacting with the user.

3.1.1 Design Philosophy

The purpose of the GUI was to allow the user to gain information about each robot's environment and internal state and be able to control each robot either completely or semi-autonomously. As noted in Section 1.2.2.1, all of this needed to be done while minimizing the cognitive load of the user. Additionally, the user's focus could not be so narrow that other robots in the system suffered from neglect tolerance, as discussed in Section 1.2.2.3. To accomplish this, a list of elements necessary for minimal situational awareness was created.

The user must be able to:

1. See each robot which is connected to the system.
2. Determine the status of each robot in the system.
3. See each task given to the system.
4. Determine the progress of each task.
5. Perceive each robot's local environment independently.
6. Locate each robot in the global environment.

Because the system is assumed to be heterogeneous in the types of robots utilized, these minimum criteria may not be sufficient for every configuration. To avoid changes to the established elements in the case of a new requirement, the GUI needs to be modular. Each requirement could be met through the use of a module. The GUI is split into four sections, known as hubs, which contained modules of similar types. Any module which is used to bring situational awareness to the states of tasks would be grouped into the task hub, located on the left side of the screen. For modules pertaining to the robots, they were grouped into the robot hub located on the right side of the screen. Modules which required a large amount of screen space or which did not fit into the two other categories were placed in the situational awareness hub. Finally, a hub was available for any modules which provided the user with feedback about the system and also did not require a relatively large amount of screen space. Figure 5 depicts this abstract layout.

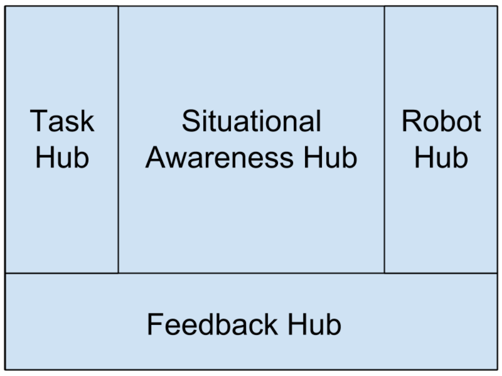


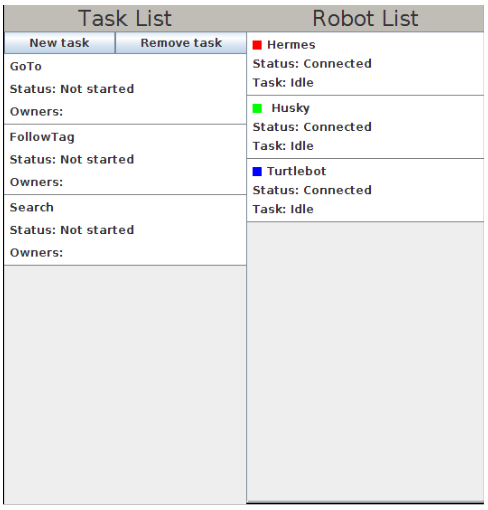
Figure 5: An abstract view of the GUI Layout

Having the requirements and a general layout in mind, a few decisions were able to be made. Firstly, Java's Swing library is well formulated to handle the types of modular designs desired. Modules could be designed as JPanels and swapped in and out of the hubs with two lines of simple code. Also, bindings for ROS had previously been developed for Java (ROSJava), allowing it to integrate easily with the rest of the system. Due to these reasons, Java was chosen as the programming language of the GUI.

The second decision was motivated by the adoption of ROS. Information was received and sent to and from the GUI through messages and topics. Each topic required a separate class to handle messages. Actions on the GUI and on the GUI's underlying data then needed to be performed from each class. Due to this widely distributed design, code would need to be duplicated, maintenance would be tedious, and there would be many opportunities for error. To counter this, a single class was created to handle similar types of operations, and the distributed ROS classes could leverage those. These centralized classes and the modular GUI elements fit well into the Model View Controller(MVC) object oriented design pattern. The modules could be represented as views, the centralized classes as controllers, and the underlying data as models. For this reason, the MVC design pattern was adopted for this GUI.

The third decision regarded specific modules that would be needed to fulfill the requirements. It was observed that requirements 1 and 2 and requirements 3 and 4 could easily be represented in a single module, and that those two modules would be extremely similar. The similar structure would be a list of rendered panels which show information about each entity. Such a structure had already been created by Shadi Ramadan in a project that the authors were involved in. This code, the SRList, was adopted as the backbone of the modules which would satisfy requirements 1-4. It is shown in Figure 6.

Finally, it was decided that only three other modules were needed to fulfill the rest of the requirements.



GoTo
Status: Not started
Owners:

FollowTag
Status: Not started
Owners:

Search
Status: Not started
Owners:

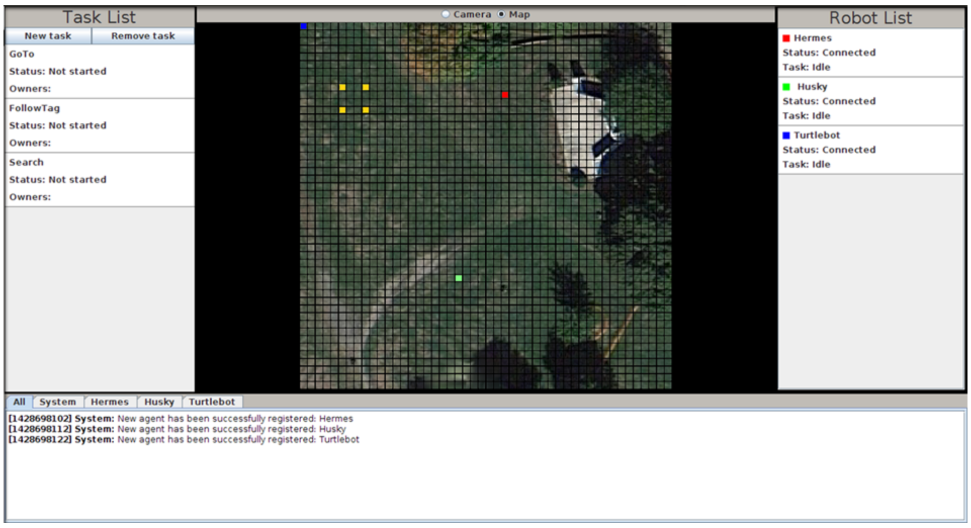
🟥 Hermes
Status: Connected
Task: Idle

🟩 Husky
Status: Connected
Task: Idle

🟦 Turtlebot
Status: Connected
Task: Idle

Figure 6: The SRLists used in the GUI

A video module which displayed a single image stream sent from the robot and an overhead map with robot positions marked would give the user local and global situational awareness. A third module was added into the feedback hub to monitor log statements published by the system which gave the user general information about the state of the system as a whole. The latter two can be seen in Figure 7.



GoTo
Status: Not started
Owners:

FollowTag
Status: Not started
Owners:

Search
Status: Not started
Owners:

🟥 Hermes
Status: Connected
Task: Idle

🟩 Husky
Status: Connected
Task: Idle

🟦 Turtlebot
Status: Connected
Task: Idle

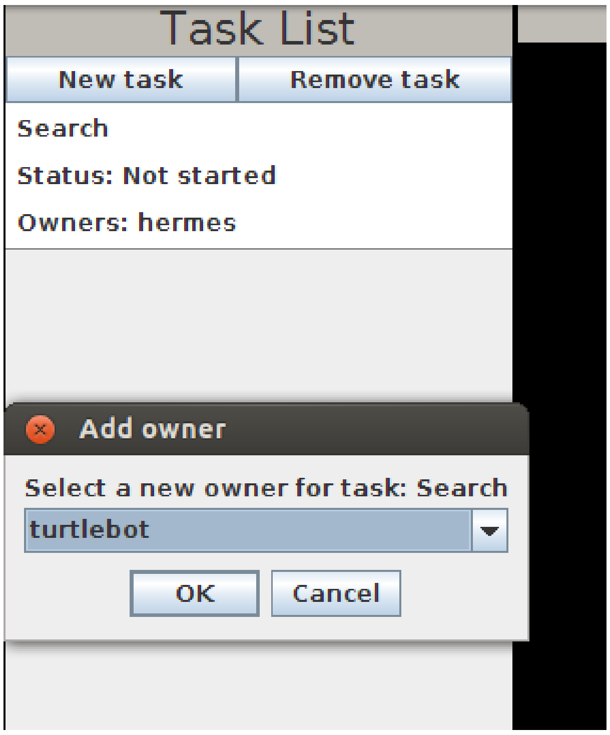
```
[1428698102] System: New agent has been successfully registered: Hermes
[1428698112] System: New agent has been successfully registered: Husky
[1428698122] System: New agent has been successfully registered: Turtlebot
```

Figure 7: The HSMRS GUI

3.1.2 Framework Implementations

In order for the GUI to integrate with the rest of the system, it needed to implement some of the core entities specified by the framework. These included Agents, Tasks, and Roles. The GUI interacted with the agents by invoking their framework APIs through ROS messages. Many of these interactions required the passing of tasks and roles between the GUI and the agents. The entities were modeled as Java classes in the GUI, however they could not be transported through ROS that way. Instead, ROS messages were defined which encapsulated each entity's data. From these messages, language specific representations were created which could be used to send the data through ROS. Typically ROS handles the language specific generation by itself, however to generate Java code additional code needed to be written which leveraged the rosjava_bootstrap package.

After the issue of representing and transferring the entities was solved, the manipulation of the entities needed to be addressed. This was straightforward with the modular design. Manipulations on tasks included creating new tasks, creating new types of tasks, and editing tasks. Creating a new task or a new task type each required their own module which could be swapped into the task hub. Editing a task was done simply using the right click menu and a popup menu, shown in Figure 8, since simple edits did not warrant a new module.



Search

Status: Not started

Owners: hermes

Select a new owner for task: Search

turtlebot

Figure 8: Editing a Task using the GUI

Manipulations on agents and roles were done in a similar fashion to tasks. Agents could be manipulated through a right click menu. Roles could be assigned through the role selection module which was swapped into the robot hub. New roles could be created from a selection of available tasks through another module as well.

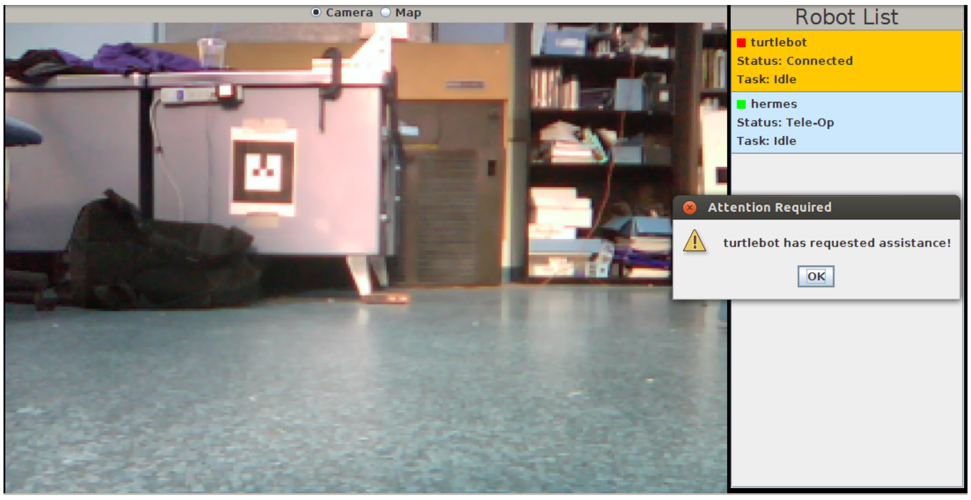
3.1.3 Situational Awareness and Interaction

As mentioned previously, situational awareness was achieved through the use of three modules, an image stream module, an overhead map module, and a textual console module. Just establishing situational awareness is useless, however, without a way to use it to interact with the robot. This section will discuss a selection of situational awareness functionalities and interactions as they are coupled together.

The first noteworthy couple is the image stream and tele-op interaction. Using the image stream module, the user can establish situational awareness of the local environment of a robot. Once established, the user can then tele-op that robot by selecting the option from a right click menu. The robot can be manually guided through difficult to navigate environments or to explore an area first hand using this method.

The second noteworthy couple is the overhead map view and task allocation. Using the overhead map view, the user can see the positions of all of the robots in the system. Additionally, the user can select grid cells of interest and use them as parameters to tasks. This is useful for giving high level navigation commands.

One final noteworthy couple addresses the issue of neglect. As described in Section 1.2.2.3, neglect tolerance measures how well a robot can operate when a user is not paying attention to it. Neglect can occur in this situation if the user is tele-operating a robot and another robot who is acting on a high level task encounters a problem. In case this happens, the robot can ask for help, triggering a reaction from the GUI and ending the neglect of the robot. When a robot asks for help, its entry in the robot list module becomes yellow and a popup alerts the user that it encountered a problem. The user can choose to ignore this warning or proceed to provide aid to the robot. This increases the overall neglect tolerance of the system by decreasing the amount of time that a robot is unproductive once it encounters a problem during a period of neglect. In the example shown in Figure 9, the supervisor is tele-operating Hermes when the Turtlebot runs into an obstacle. The user is alerted to this issue through the mechanism described above.



turtlebot
Status: Connected
Task: Idle

hermes
Status: Tele-Op
Task: Idle

Attention Required

turtlebot has requested assistance!

Figure 9: A Robot asking for help, as seen from GUI

3.2 Robot Selection and Modification

3.2.1 Turtlebots

The Turtlebots, shown in Figure 10, were identified as great platforms to test the framework on. They were created by Willow Garage as a platform for developers to create applications for ROS in the context of mobile robots. The newer versions, which were available to the team, come with excellent odometry, a preconfigured camera,and a simple API. These attributes were invaluable during the early stages of

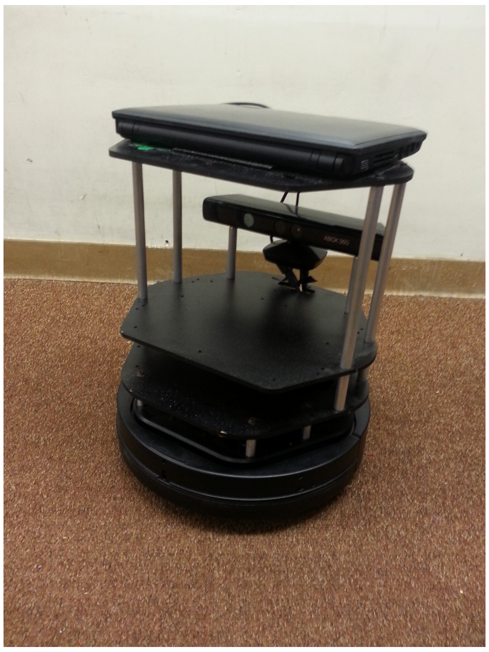


Figure 10: The Turtlebot version 2.

testing.

While excellent platforms out of the box, some modifications were needed in order to make the Turtlebots integrate with the system. The Turtlebots rely on a transform tree for many of their functionalities, and when multiple Turtlebots are deployed in a system, their transform trees clash with one another. Typically, this can be fixed using the ROS parameter tf_prefix. The manner in which the Turtlebots were developed, however, precluded this method. Many of the important frames of reference, such as odom, were created such that the tf_prefix is not resolved on the name. In the case of the odom frame, this means that all robots using such a frame would believe that they started in the exact same location. To avoid this problem, a layer of indirection was introduced into the Turtlebot's transform tree. A new frame was broadcasted which was unique to the Turtlebot, and the odometry messages were intercepted and resent with respect to the unique frame. With this fix, the Turtlebots were able to integrate with the navigation stack and perform tasks within the system.

3.2.2 Hermes

Hermes is a small robot built by students in the Robotics and Intelligent Vehicles Research(RIVeR) Lab. It was created to be a testing platform for localization research. It came equipped with a Mecanum drivebase and a relatively powerful onboard computer, making it ideal for indoor testing. It also had the benefit of being fully assembled and available to the team early in the project, unlike other robots that were considered for use. It is shown in Figure 11.

A failure of the onboard computer's hardware led to its replacement by a less-capable laptop, and a USB webcam was added to allow for the use of visual odometry. In terms of software, Hermes needed a ROS node to control its drivebase and another to publish images from its webcam. The drivebase node uses kinematics derived by previous works [25] to translate ROS Twist messages to speeds for the motor

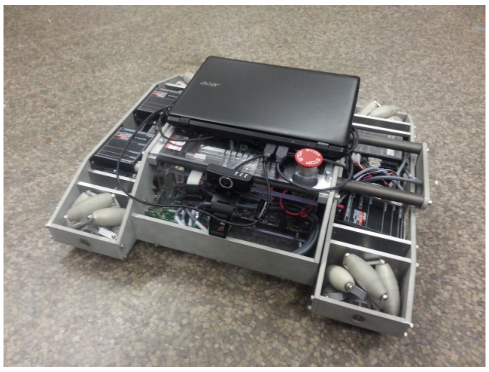


Figure 11: Hermes

controllers, and to translate encoder data to ROS Odometry messages. The webcam node uses OpenCV to access and resize the images before publishing. The camera was calibrated from values generated by the standard ROS camera_calibration node.

3.2.3 Husky A100

The Husky A100 is an unmanned ground vehicle (UGV) from Clearpath Robotics. It is an open source all-terrain mobile robot development platform. With the intention of being used for research, the Husky is highly adaptable with a large top plate for mounting various payloads. This skid-steer drive robot is capable of reaching speeds up to 1 m/s and carrying 75Kg in its latest model. There are drivers and APIs for ROS, C++, and Python for controlling the robot and accessing sensor data such as wheel odometry, motor currents, and battery voltage. The Husky robot used can be seen below in Figure 12.

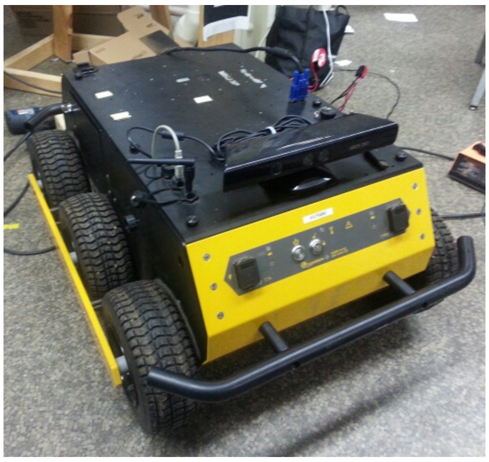


Figure 12: A Clearpath A100 Husky

This robot was chosen due to its all-terrain capabilities and its accessibility to the group. The robot, originally running on Ubuntu 10.04 and ROS Diamondback, required a software update to Ubuntu 12.04 and ROS Hydro. The Husky Hydro packages, unfortunately, were not compatible with the A100 model Husky causing the electronic hardware on the robot to not function properly. As a substitute, an Arduino Uno microcontroller was utilized to interface with the Husky's two Jaguar motor controllers via pulsewidth modulation (PWM). The internal wiring of the robot needed to be adjusted to account for the use of the microcontroller. The ROS rosserial_arduino package was used to interface the motherboard with the Arduino.

A Kinect camera was integrated with the Husky's power distribution board and interfaces with the motherboard via USB. The camera was then calibrated with the standard ROS camera_calibration node and allows for the use of visual odometry. Using a differential drive base as a model for the Husky's skid-steer style base, a ROS node was created to control the Husky's motors in response to published ROS Twist messages. The Arduino was connected to the Husky's quadrature encoders to allow for the access of wheel odometry data and code was written to read the wheel encoder data and turn this information into ROS Odometry messages.

3.3 Navigation Stack Development

The scenario chosen by the team was representative of the types of situations that would be found in the identified areas of application. This means that the tasks required that the robots cover a large amount of ground autonomously. While open source navigation stacks do already exist, they require a large amount of calibration and configuration. Due to the heterogeneous nature of the system, configuring the navigation stack for each of the different robots would be tedious. Instead, it was decided that a simpler and more streamlined navigation stack could be made which would make the integration of additional robots effortless. This section discusses that code base.

3.3.1 Localization

Localization is still an active area of research in robotics, and there are many different tools available to determine the position of a robot within its environment. Unfortunately many of these require expensive sensors or a lot of calibration. Since the goal of this navigation stack was to be simple and streamlined, these tools would not be helpful. Instead, augmented reality(AR) tags [36] were used to establish a robot's position in the environment. Furthermore, the readings from the AR tags were fused with odometry messages to increase the precision of the localization. The choice of AR tags over other methods is discussed in further detail in the Trade Study section of Appendix A.

Localization with the AR tags required only a few steps. First, the ROS package ar_track_alvar needed to be installed on all of the robots. The package required very few configuration steps, and the configuration was common for all of the robots. After doing this, any robot equipped with a camera running the package would be able to determine its 6 degree of freedom pose with respect to the tag. The locations of the tags needed to be predetermined before running the system. The positions of the tags were marked using frames from the ROS tf library. Each tag was given a transform from the map frame to its own tag. This made determining the robot's location extremely easy, as it only required transforming the robot's pose with respect to the tag into the map frame.

Unfortunately using AR tags was not sufficient for reliable localization. Because the entire operational area could not be covered in AR tags, there were occasions when robots could not see tags. In these situations, the robot would not be able to localize. To combat this, odometry data was also collected and fused with the AR tag measurements. The fusion was performed using the ROS robot_pose_ekf package. This package utilizes an extended Kalman filter to estimate a robot's 6dof pose based on measurements from multiple sensors. First, an initial estimate of pose from the AR tags would determine the location of the odometry frame with respect to the map. All further pose estimates made by either source were transformed into the map frame and used to update the filtered pose. Unfortunately, due to a bug in the robot_pose_ekf package discovered by the authors [37], the output of the filter could not be extracted with respect to the map frame. Instead, a layer of indirection needed to be introduced which transformed the filtered pose into the map frame and re-sent it.

3.3.2 Planning And Execution

All of the tasks required that the robots navigate to specific locations. In order to achieve this capability, two functionalities needed to be implemented. First, the robots needed to be able to plan where they were going to drive in order to reach their locations. Once they had a plan, the robots needed to be able to execute them.

The navigation planning is powered by A*[38]. This algorithm uses a Cartesian distance heuristic to estimate the total cost of a path from a certain point. The algorithm also keeps track of the accumulated cost associated with following a path. It is guaranteed to find an optimal solution and it also works relatively quickly on maps of all sizes. The ability to adjust the costs of each location also makes it very easy to implement obstacle avoidance.

Once the desired path is found, the robots needed to be able to execute it. This was accomplished by creating a parametrized ROS node. The parameters to the node determined the top linear and angular speed of the robot. The node then calculated velocities for the robot using a saturated piecewise proportional controller. For angular errors greater than ten degrees, linear velocity was set to zero. Once within ten degrees of angular error, the linear velocity would then be set by the saturated proportional controller. Errors were determined using the filtered pose and waypoints along the desired path.

With these two functionalities in place, the robots chosen to be used to test our framework were equipped to handle the tasks that would be given to them. Tasks requiring navigation to a location would first provide the goal to A*. The algorithm would then find an optimal path for the robot to follow, which would be given to the executor, which calculated errors in poses and determined appropriate velocities to reduce those errors to zero. Actuation based on the velocities was left to specialized low level control assumed to already be on the robots.

3.4 Task Selection And Implementation

Given the generalizable nature of the framework, robots using it could be expected to perform many different kinds of tasks. For testing purposes, three simple tasks were chosen for implementation on the selected robots. These tasks were considered to be representative of those required for a typical HSMRS scenario, and could use different criteria to determine the optimal allocation of work.

3.4.1 Go-To

The Go-To task orders a robot to move to a specified location. A task of this type is necessary for any application involving semi-autonomous mobile robots, so it was an obvious choice to include it for testing. A robot's utility for this task is generated by determining the time required for the robot to travel the Euclidean distance from its current location to the location specified by the task. (Robots keep track of their maximum speed using their AgentState object.) Once a robot obtains the task, it determines and follows a path to the goal using the navigation stack discussed in Section 3.3.

3.4.2 Search

The Search task orders a robot or a group of robots to search a specified area for a specified AR tag, and to alert the supervisor upon either finding it or traversing the entire area. This task was also chosen for its similarity to tasks required for a variety of different scenarios; for example, a team of robots deployed in a disaster zone may need to search for survivors, or a team of exploratory robots may need to search for samples of interesting materials to analyze.

Robots search an area by traversing it in a serpentine pattern from one corner to another. If a tag matching the given ID is spotted, the robot will attempt to approach it, and it alerts the supervisor that it has found the tag once it gets close enough. If the robot completes its search pattern, it informs the supervisor that the search was completed without locating the tag. This process is shown graphically in Figure 13. In the figure, the yellow circles represent user specified vertices for the search area, the blue circles represent waypoints generated by the algorithm, and the green block represents the object being searched for. As implemented, a robot's utility is calculated based on the time required to travel the Euclidean distance to the starting point of the search pattern, but a robot's aptitude for searching or time required to complete the search pattern could be factored in, as well. The supervisor can allow an arbitrary number of robots to participate in a single Search task at once.

3.4.3 Follow-Tag

The Follow-Tag task orders a robot to drive toward a specified AR tag and remain 0.5 meters away from it until the supervisor gives the order to stop. This task was chosen mainly as an early means of testing AR tag detection software, but it does have some parallels with real-world applications (e.g. a robot at a construction site could follow a worker carrying heavy tools). The closest robot to the specified tag will be preferred for this task, with no regard for how fast the robot moves, and any robot that does not have the tag in its field of view during bidding will not attempt to bid at all. To perform the task, the robot simply drives forward until the tag is 0.5 meters away and steers left and right based on the tag's position in the robot's field of view.

3.5 Test Scenario

All of the aforementioned elements were necessary in order to conduct the test scenario. The goal of the test scenario was to be general enough to be representative of a wide range of applications of HSMRS

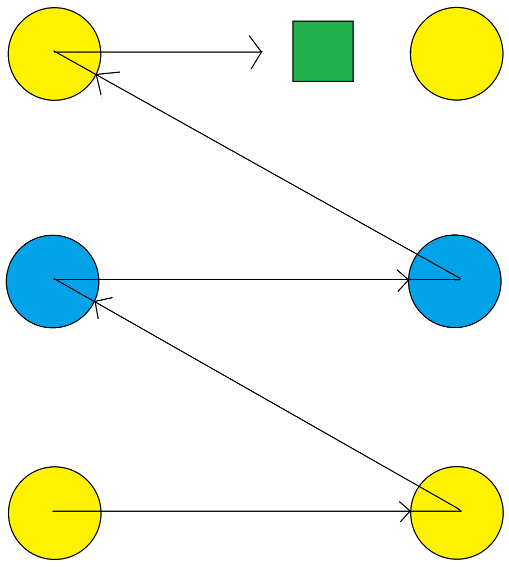


Figure 13: A graphical representation of a robot's path during a Search task

systems. From the review of areas of application from Section 1.2.3, it was determined that a simple search and discover mission would be sufficient.

This search and discover mission was conducted by a team of two robots, Husky and Hermes. These robots localized using the method described in Section 3.3.1 in a flat and open $144m^2$ rectangular area. A single human supervisor dispensed Go-To tasks and Search tasks in order to have the robots explore the area. The robots searched for specific AR tags which were miniaturized and hidden to simulate objects of interest.

Unit tests were also conducted to test specific functionalities of the framework. These were performed using a team of two Turtlebots. All of the basic functionalities were tested including all communications with the GUI, localization and navigation, direct and autonomous task allocation, role assignment, continuous autonomous task allocation, interruptibility of tasks, and tele-operation.

4 Results and Discussion

The test scenario was performed in order to verify that the framework met all of the design requirements set forth in Section 2.1. Table 8 summarizes each requirement and whether or not it was met by the framework. A similar table shows the results regarding the desired capabilities of the framework Explanations of the failures are provided in the following sections.

ID	Summary	Results	Verification
R.01	The system shall incorporate at least two robotic agents.	The system was able to accommodate two and three robots, while larger numbers were untested.	Passed
R.02	The system shall be able to represent different robot configurations and capabilities that are comparable to those of existing field robots currently used in the target scenarios.	Heterogeneous robots were abstracted into Robot entities which were easily manipulated by the system. The Heterogeneous properties were represented in the AttributeWeights which allowed the autonomous bidding process to choose the most fit robots for tasks.	Passed
R.03	The system shall be able to utilize the different robot configurations and capabilities specified in R.02	The AttributeWeights entity allowed the robots to assert their strong capabilities while bidding for tasks, allowing them to be utilized for the tasks that they were good for.	Passed
R.04	The system shall send relevant information about the agents in the robot team to the supervisor UI.	Using the data received from the robots, the supervisor was able to teleoperate the robots, give high level commands to the robots, and help the robots when they encountered problems. Tracking progress worked generally however occasional threading errors would occur on the GUI.	Passed
R.05	The supervisor UI shall update its representation of the team and task status at a frequency suitable for effective supervision.	The UI was capable of updating all of its components however due to large amounts of latency(¿10s) it was not suitable for effective supervision.	Failed
R.06	Components of the system shall be able to send data to other components at a rate suitable for the coordination and supervision of the entire team.	Large amounts of lag(¿10s) in the system often interfered with the communications between other robots.	Failed
R.07	The system shall provide robots with tasks to complete.	User generated tasks could be either directly assigned to the robots or autonomously delegated.	Passed

R.08	The system agents shall be able to accomplish tasks relevant to the aforementioned scenarios.	Using the navigation stack, the robots were able to accomplish Go To tasks as well as Search tasks. The Follow tag tasks similarly worked as expected.	Passed
R.09	The supervisor shall be able to utilize all capabilities of the system using the user interface.	The user was able to send commands to the robots using keyboard keystrokes which could be interpreted by the robot being controlled. Tasks could be generated and assigned from the GUI and Roles could be created and assigned.	Passed
R.10	The user interface shall require only a single device to interact with the user.	The user interface consisted of only a single computer window which could be run on a desktop of laptop computer running Ubuntu and ROS	Passed
R.11	The device running the user interface shall be mobile.	Using a laptop, the GUI was able to be moved. The supervisor was not able to perform tasks while using the interface, however.	Passed

Table 8: Summary of the requirements, results associated with them, and their verification status.

Entity	Capability	Result
Activities	Be abstracted and encapsulated.	The Task entity succeeded at abstracting and encapsulating similar activities through parametrization.
	Have a unique measure of the capabilities needed to be completed.	Using the AttributeWeights entity, Tasks could prefer some attributes over others to influence which robot should claim the Task.
	Have an order to be completed in relative to other activities.	The Prerequisite attribute of Tasks was designed to handle this capability, but ultimately failed. This will be discussed in detail in Section 4.1.2.
	Be bundled with other activities to create more complicated activities.	Meta tasks, which are tasks composed of a hierarchy of other tasks were designed and implemented but not tested due to time constraints.
Robots	Take responsibility for completing an activity.	As part of the Task assignment workflow, robots claimed tasks that they were either directly assigned or won after bidding.

	Ask for help if it cannot complete an activity autonomously.	Using the Turtlebots, collisions with obstacles in the environment were detected and used to alert the user to a problem with the robot.
	Communicate regular status updates to other actors in the system.	As tested, the robots only communicated when a task was either claimed or completed, and nothing else. Regular updates were not tested due to system latency of 10 seconds and greater.
	Be able to act autonomously or be controlled by a human.	All robots were able to interpret keystrokes made by the user on the GUI and use these commands to perform tele-operation tasks.
Humans	Assign responsibility to a robot to complete an activity.	The user was able to directly assign a task to a robot while creating the task or after creating the task. Additionally, the user was able to remove robots from tasks they had claimed.
	Directly control one or more robots.	The user could control robots directly through keystroke commands which were interpreted by the robots.
	Establish situational awareness for whatever task is being focused on.	Using the camera view and the overhead map view, the user could gain all of the necessary information about the environment.
	Project an intuition about future states of the environment to increase the efficiency of the system.	Using Roles, the human could alter the outcomes of future autonomous auctions by prohibiting robots fro certain tasks based on their judgment of future states of the environment.

Table 9: Summary of the desired capabilities and the results associated with them.

4.1 Tasks

4.1.1 Strengths

The Tasks chosen for testing were analogous to tasks which would be required in real applications, and they had clear requirements for the robots that would undertake them. Setting up each Robot's AgentState was simple because the number of attributes required was small. The chosen Tasks are also a solid basis for future work due to their broad applicability.

4.1.2 Shortcomings

The amount of information required for Prerequisites to have access to in order to function is unrealistic given the rest of the framework. Because a Prerequisite is an abstract type, it can represent any number of different conditions required for a task to be started, but that information has to come from outside of the Prerequisite. This means that the Prerequisite would need a generic method of accessing any information held by any framework class, which is problematic and impractical at best from a design perspective.

The framework does not specify a standard structure or method to handle the execution of Tasks, but it does prescribe that a Task's progress be monitored (through the Progress class). This is effectively a gap in the framework which developers would need to find their own way to fill. The development team created another type of object called Behaviors to deal with this problem, but the Behavior pattern only works in this situation because of the unified navigation stack used across all of the test robots.

None of the Tasks used for testing made use of the UtilityHelper's default utility calculation. This is because auctions for Tasks were won by the highest bidder, and the utilities for the Tasks used were based on time to travel to a target, in which a lower score is better. This meant that the UtilityHelper needed to have a separate case for calculating the utility of each type of Task.

4.2 Robots

4.2.1 Strengths

The Robot abstraction in the Framework made including new robots into the system very straightforward. A lot of code was able to be recycled when developing for new robots. This meant that most of the development time for each robot was limited to the low level control of each robot. There was little need to think about the interaction with the rest of the framework, as this was encapsulated within the Robot code. The strengths of the used robots were that they could be acquired quickly and cheaply, and represent a range of heterogeneous platforms.

4.2.2 Shortcomings

None of the robots used for testing were particularly well-suited for it. While the degree of heterogenity between the robots used did make for realistic testing, having a set of similar, highly-capable robots would make the capabilities and constraints of the framework more clear. (For example, using underpowered computers makes it difficult to tell if the source of performance issues is the code or the hardware that the code is running on.) Some of the robots also took an inordinate amount of work to make operational due to a series of unforseen problems, such as Hermes's computer malfunction and Husky's motor controller failure.

4.3 Humans/GUI

4.3.1 Strengths

The Human entity of the framework was found to be well designed and did not have any obvious shortcomings. The role of the human supervisor was well defined as an actor who could perform tasks but mostly created and delegated tasks while monitoring the state of the system. The designed GUI, shown in Figure 7, was strong in monitoring the system, with separate views concurrently showing the status of all tasks, robots, and providing situational awareness. Accessing other functionalities of the GUI was fairly straightforward. Creating and deleting tasks could be done with a click of the labeled button. All of the different manipulations that were possible for tasks and robots were available through a standard right click interface. Tele-operation of mobile robots through use of the arrow keys was intuitive and worked without a problem.

4.3.2 Shortcomings

While the elements of the GUI that were created for this project met all of the specified requirements, there were some features which would have made the interface more user friendly and quicker. A more diverse suite of modules for situational awareness could have been created. One example of a desired module which would have increased situational awareness drastically was a multi-camera view. This view could have supported multiple image streams coming from the system whose sources could have been selected through a right click menu. This would have allowed the supervisor to monitor multiple cameras on the same robot or camera outputs from multiple robots.

When new robots were added to the system, they communicated their existence to the human through a registration ROS message. ROS messages are sent once and receive no reply. This means that if a robot sends a registration message and for some reason it is not received correctly, the robot has no idea that it is not recognized by the rest of the system. This could have been fixed by using a ROS service instead of a message to perform registration. In addition, registration was done using a colon delimited String message. Using a custom service with dedicated containers for data would have been much more effective and easy to use. Finally, the registration only accepts a single source for image streams. This limits the GUI to only be able to see one camera per robot. This diminishes the potential for situational awareness and could be fixed by replacing the image source field with an array of image sources.

Another shortcoming of the GUI was that creating tasks was tedious at times. For search tasks, the user needed to select four cells to be the vertices of the search area, click on the new task button, and enter in all of the information. This could have been greatly sped up if the overhead map had a right click option which created a task from the selected cells without needing to go through the new task view. The user would only need to select the cells and then confirm that the task was correct before sending it to the system.

4.4 Implementation

4.4.1 Strengths

The implementation of the HSMRS framework was based on the Robot Operating System(ROS). ROS is an open sourced project which serves to support modular code for robotics applications. It was chosen for this application because it provided a full communication protocol based on TCP/IP and it established many design practices which typically accelerate the development of large projects. A ROS package was created which centralized all of the important aspects of the framework, including the abstract classes, simple concrete implementations of the classes, and their representations as ROS messages. This was a strength of the implementation because it enabled easy portability and re-usability of the framework.

The modular nature of the ROS packages also helped with parallel development. The message publishing/subscribing system provided a clear interface for inter-process communication. This allowed code developed separately to easily integrate. It also helped in that third party modules were able to be used with some configuration. An example of this is the ar_track_alvar package which was used detect AR tags used for localization.

4.4.2 Shortcomings

ROS's architecture poses a serious problem for the scalability of any system built on it. When a message is published in ROS, it is broadcast to all nodes connected to the master server, whether or not a given node is subscribed to the message's topic. The direct result of this lack of scope in publishing is that a substantial portion of the traffic ROS generates is superfluous. In systems with larger numbers of nodes, this quickly leads to network congestion and high processing overhead for each machine running a node. In future work, this could possibly be avoided by not depending on ROS for inter-robot communication, and instead keeping a separate ROS master on each robot.

5 Conclusion

This project involved the creation and testing of a framework for human supervision of multi-robot systems. A review of literature in this field was conducted, and the best practices were used as the basis of this framework. The main entities of the system were abstracted and encapsulated to ensure portability and reusability. The sequences of interactions between these entities was spelled out as part of the framework to ensure clarity and regularity throughout implementations.

The testing of the framework was carried out in the form of a test scenario. To prepare for this test scenario, a graphical user interface was designed and implemented, two robots were repaired and reprogrammed, and all three robots were programmed to comply with the framework. Other utilities were developed to aid in the execution of the framework as well.

After the testing, it was found that most of the requirements that were established for the project had been met. Those which had been failed did so as a result of high latency in the network being used. The desired capabilities for the system which were based on the requirements had similar results, with most being met, some left untested, and others not met. Overall, the framework proved to be a powerful resource for developing multi-robot systems. Due to the reusability of the code written for the framework, most of the development time spent by the team was on the low level control of specific robots.

In the future, further work can be conducted to refine the framework and make it conform to all of the requirements. The Prerequisite abstraction can be re-imagined to make it more feasible and easier to implement. The implementation of the framework could be improved by moving to a new communication platform which, unlike ROS, minimizes network traffic to increase communication speeds. Also, a more capable and diverse team of robots can be chosen to carry out a more elaborate test scenario to test the unverified requirements.

References

[1] Paul Scerri, David Pynadath, Lewis Johnson, Paul Rosenbloom, Mei Si, Nathan Schurr, and Milind Tambe. 2001. A prototype infrastructure for distributed robot-agent-person teams. In-Proceedings of the second international joint conference on Autonomous agents and multiagent systems (AAMAS '03). ACM, New York, NY, USA, 433-440. DOI=10.1145/860575.860645 http://doi.acm.org/10.1145/860575.860645

[2] Chipalkatty, Rahul. "Human-in-the-loop control for cooperative human-robot tasks." (2012).

[3] Pedro U. Lima, L. M. (2005). Multi-Robot Systems. Studies in Computational Intelligence Volume 8, 1-64.

[4] Takayama, Leila, Wendy Ju, and Clifford Nass. "Beyond dirty, dangerous and dull: what everyday people think robots should do." Proceedings of the 3rd ACM/IEEE international conference on Human robot interaction. ACM, 2008.

[5] Iocchi, Luca, et al. "Distributed coordination in heterogeneous multi-robot systems." Autonomous Robots 15.2 (2003): 155-168.

[6] RUNSWift Naos 2010, Wikimedia Commons, [online] 2010, http://en.wikipedia.org/wiki/File:RUNSWift_Naos_2010.jpg (accessed April 29, 2015)

[7] Simmons, Reid, et al. "Coordination for multi-robot exploration and mapping."AAAI/IAAI. 2000.

[8] Gerkey, Brian P., and Maja J. Matarić. "A formal analysis and taxonomy of task allocation in multi-robot systems." The International Journal of Robotics Research 23.9 (2004): 939-954.

[9] Chen, Jessie YC, Michael J. Barnes, and Michelle Harper-Sciarini. "Supervisory control of multiple robots: Human-performance issues and user-interface design." Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 41.4 (2011): 435-454.

[10] Endsley, Mica R. "Design and evaluation for situation awareness enhancement." Proceedings of the Human Factors and Ergonomics Society Annual Meeting. Vol. 32. No. 2. SAGE Publications, 1988.

[11] Endsley, Mica R., and Daniel J. Garland, eds. Situation awareness analysis and measurement. CRC Press, 2000.

[12] Kruijff, Geert-Jan M., et al. "Experience in System Design for Human-Robot Teaming in Urban Search and Rescue." Field and Service Robotics. Springer Berlin Heidelberg, 2014.

[13] Goodrich, Michael A., Jacob W. Crandall, and Jeffrey L. Stimpson. "Neglect tolerant teaming: Issues and dilemmas." Proceedings of the 2003 AAAI Spring Symposium on Human Interaction with Autonomous Systems in Complex Environments. 2003.

[14] Unhelkar, Vaibhav V., Ho Chit Siu, and Julie A. Shah. "Comparative performance of human and mobile robotic assistants in collaborative fetch-and-deliver tasks." Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction. ACM, 2014.

[15] United States. Bureau of Labor Statistics. "National Census of Fatal Occupational Injuries in 2013" Washington: GPO, 2014. Print.

[16] C. Balaguer and M. Abderrahim, Robotics and Automation in Construction, InTech, 2008.

[17] About SpaceX, SpaceX, [online] 2013, http://www.spacex.com/about (accessed September 29, 2014)

[18] Mission, Planetary Resources, [online] 2013, http://www.planetaryresources.com/mission (accessed September 29, 2014)

[19] About Mars One, Mars One, [onine] 2012, http://www.mars-one.com/about-mars-one (accessed September 29, 2014)

[20] Deep Space Industries, [online] 2013, http://deepspaceindustries.com/ (accessed September 29, 2014) About Us, Harvest Automation, [online], 2014, http://www.harvestai.com/company/about-us (accessed October 15, 2014) Company

[21] R.H. Miller, Human productivity in space and systems costs, Acta Astronautica, Volume 36, Issues 8–12, October–December 1995, Pages 581-587, ISSN 0094-5765, http://dx.doi.org/10.1016/0094-5765(95)00145-X. (http://www.sciencedirect.com/science/article/pii/009457659500145X)

[22] Crawford, Ian A. "Dispelling the myth of robotic efficiency: why human space exploration will tell us more about the Solar System than will robotic exploration alone." arXiv preprint arXiv:1203.6250 (2012).

[23] Nagatani, Keiji, et al. "Emergency response to the nuclear accident at the Fukushima Daiichi Nuclear Power Plants using mobile rescue robots." Journal of Field Robotics 30.1 (2013): 44-63.

[24] Shah, Julie A., Patrick R. Conrad, and Brian C. Williams. "Fast Distributed Multi-agent Plan Execution with Dynamic Task Assignment and Scheduling."ICAPS. 2009.

[25] Nagatani, Keiji, Satoshi Tachibana, Makoto Sofue, and Yutaka Tanaka. "Improvement of odometry for omnidirectional vehicle using optical flow information." IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2000.

[26] Guizzo, Erico. "Three engineers, hundreds of robots, one warehouse."Spectrum, IEEE 45.7 (2008): 26-34.

[27] About Us, Harvest Automation, [online], 2014, http://www.harvestai.com/company/about-us (accessed October 15, 2014) Company

[28] Petersen, Kirstin, Radhika Nagpal, and Justin Werfel. "Termes: An autonomous robotic system for three-dimensional collective construction." Proc. Robotics: Science & Systems VII (2011).

[29] Micire, Mark, et al. "Multi-touch interaction for robot control." Proceedings of the 14th international conference on Intelligent user interfaces. ACM, 2009.

[30] Fong, Terrence, et al. "A personal user interface for collaborative human-robot exploration." 6th International Symposium on Artificial Intelligence, Robotics, and Automation in Space (iSAIRAS). No. LSRO2-CONF-2001-001. 2001.

[31] B Glass, G Briggs, J Jasper, and K Snook. Evaluation of human vs. teleoperated robotic performance in field geology tasks at a mars analog site. In Proc. 7th i-SAIRAS, volume 3. Citeseer, 2003.

[32] Keyes, Brenden, et al. "Improving human-robot interaction through interface evolution." (2010).

[33] Lee, Woohun, et al. "Design guidelines for map-based human–robot interfaces: A colocated workspace perspective." International journal of industrial ergonomics 37.7 (2007): 589-604.

[34] Murphy, Robin R. "Human-robot interaction in rescue robotics." Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 34.2 (2004): 138-153.

[35] Wang, Jijun, Huadong Wang, and Michael Lewis. "Assessing cooperation in human control of heterogeneous robots." Human-Robot Interaction (HRI), 2008 3rd ACM/IEEE International Conference on. IEEE, 2008.

[36] ar_track_alvar, ROS Wiki, [online], 2015, http://wiki.ros.org/ar_track_alvar Wiki

[37] Issue #313, robot_pose_ekf github, [online], 2015 https://github.com/ros-planning/navigation/issues/313

[38] Delling, D.; Sanders, P.; Schultes, D.; Wagner, D. (2009). "Engineering route planning algorithms". Algorithmics of Large and Complex Networks: Design, Analysis, and Simulation. Springer. pp. 117–139.

A Systems Engineering Methodology

The goal of this project was to implement a framework for human-robot team coordination. To accomplish this goal, a systems engineering methodology was used. To begin, an analysis was performed to determine the identity, impact, and importance of this project's stakeholders. Second, a needs analysis was performed to identify the needs and goals of the stakeholders. An abstract view of the system's interactions was illustrated using context diagrams. Once finished, requirements for the project were formulated. A gap analysis followed and helped identify the technologies that need to be developed in order for the project to succeed. These technologies were then compared to one another in a trade study. Use cases were presented to portray how the finished system will be used. Finally, key performance parameters were identified in order to quantify the performance of the system.

A.1 Stakeholder Analysis

Although this project is not sponsored by a specific sponsor or client, there are several stakeholders that must be considered. They generally fall into three categories: WPI as the facilitating organization, impacted communities or players, and legal requirements or compliance standards represented by proxy.

As a Major Qualifying Project at WPI, the WPI faculty advisor and administration are high priority stakeholders. Secondary advisors include other professors, other project teams, involved students, and faculty that are directly impacted by the project or its development along with proxy organizations that mandate applicable standards or regulations. Finally, the input of influential organizations and communities was considered, along with the impact on proximal organizations that were influential but not necessary to the design of the system. The identified stakeholders are summarized in Table 10.

ID	**Title**	**Description**	**Influence**	**Priority (1 >5)**
SH.01	Development Team	The team completing this project.	Primary, Direct	1
SH.02	MQP Faculty Advisors	Professor Padir is the robotics faculty advisor. Professor Chernova is the Computer Science faculty advisor.	Primary, Direct	1
SH.03	WPI Administration	Including the registrar's office and other offices and persons involved with MQP assessment, qualification, grading, and credit. Although they will not be contacted directly and will have minimal interaction, their requirements are critical to project success.	Primary, Direct	1
SH.04	Technical Experts	Other WPI professors and experts outside WPI will be consulted. While their input is highly valuable, it could be dismissed. These include:The WPI DRC TeamRoboticists working in relevant areas of researchEngineers and consultants familiar with the subsystems used	Secondary, Direct	3

SH.05	WPI Facilities	Some logistics of the project, including shipping and transportation, will require the assistance of WPI facilities.	Secondary, Direct	2
SH.06	Worcester parks and recreation	Institute park is a potential testing ground for this system. If used, the system and test procedures must comply with applicable park and recreation laws.	Secondary, Direct	3
SH.07	Other MQP teams	Other MQP teams compete for resources, including project space, WPI facilities, and advising time with professors.	Primary, Direct	3
SH.08	WPI Facilitation Faculty and Departments	Namely Tracey Coetzee, who will process parts orders, provide budgeting information, etc.	Secondary, Direct	2
SH.09	Regulatory Agencies	Government agencies that regulate technology used in the system or procedures/practices/activities involved in its testing or operation. These include:OSHA: Sets standards for safe design and operation of equipment, working conditions and environments, and occupational procedures.FCC: Regulates radio communication and sets standards for radio systems.FAA: Regulates UAVs.	Secondary, Proxy	4
SH.10	Previous MQP teams	The teams the managed the subsystems (robots) used in the project.	Secondary, Direct	2

SH.11	Relevant technical communities	Communities, associations, etc. that are interested in this project, but not directly involved. These include:ROS Community: The developers of the Open-Source Robotic Operating System.Various Robotic conferences.NASA Centennial Challenge teams.	Secondary, Direct	3
SH.12	WPI Community	The system will be tested in proximity to faculty and students of WPI that are not involved with the project.	Secondary, Direct	3
SH.13	Researchers active in the same field	Researchers at WPI and elsewhere that are actively researching, developing, or previously contributed work in the areas of heterogeneous multi-robot systems, human-robot interaction, search-and-rescue, disaster-response, or sample-return robotic systems. These researchers could contribute valuable technical experience, theoretical models, algorithms, etc. but any of their work used for this project could require permission and must be properly licensed and/or acknowledged.	Secondary, Direct	3
SH.14	Potential Sponsors	Companies, organizations, or individuals who contribute resources directly to the project. Their terms of contribution could include additional systems requirements that must be met. Potential sponsors include:NASA, Caterpillar Corp, DARPA, and NSF	Primary, Direct	3

SH.15	Affected Workers	Any group or class of personnel whose occupation is influenced or drastically altered by the deployment of this system.	Secondary, Proxy	4
SH.16	Displaced Workers	Any group or class or personnel whose occupation is potentially deprecated or obviated by the deployment or development of this system. These are the only identified negative stakeholders.	Secondary, Proxy	4
SH.17	ABET	The Accreditation Board for Engineering and Technology.	Secondary, Proxy	4

Table 10: Stakeholder Table

The influence of each stakeholder is visualized in the onion diagram shown in A.1. Each layer contains a category of stakeholders. Stakeholders in outer layers have less influence, and a lower priority will be placed on satisfying their needs.

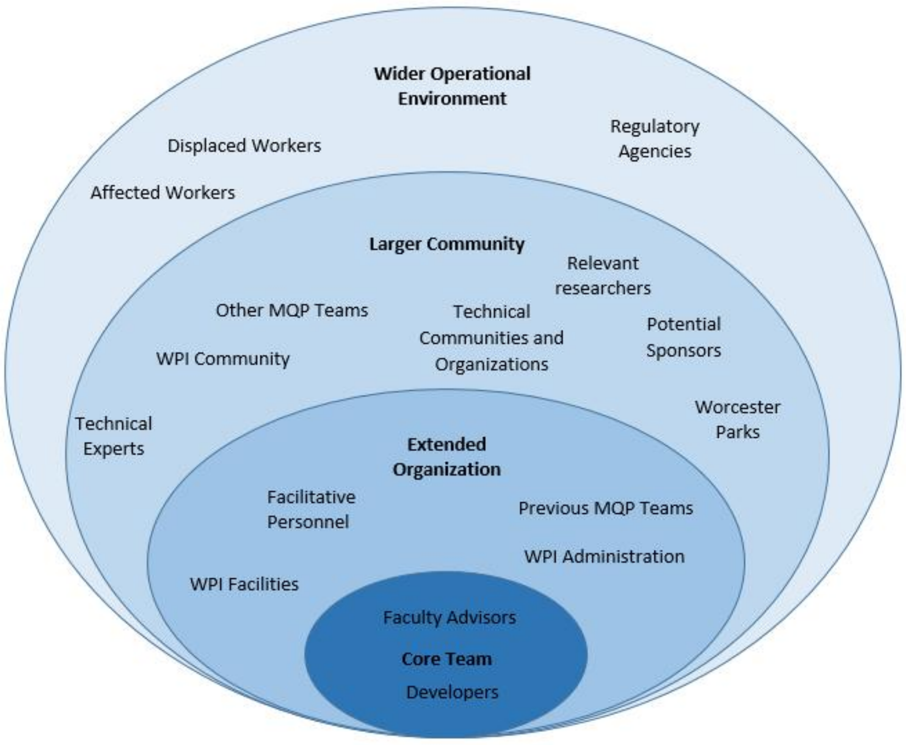


Figure 14: An onion diagram of stakeholders

A.2 Needs Analysis

The stakeholders of this project generally have expectations and/or goals for the outcomes of the project. These fall into the category of needs. The stakeholders' needs for this project were elicited using two main techniques. First, the team interviewed stakeholders with whom its members have personal connections. Second, the needs of regulatory agencies and other proxy stakeholders were ascertained by examining the regulations and documentation which relate to the project. The project needs are summarized in Table 11.

ID	Title	Description	Validation	Priority
N.01	Demonstration of HSHMRS	The project should succeed in demonstrating human supervision of heterogeneous multi-robot systems	SH.02	1
N.02	Demonstration of competence	The project should demonstrate the developer's competence in robotics engineering and computer science and meet the expectations of an MQP.	SH.01, SH.03, SH.17	1
N.03	Documentation	The system should be well documented in the case of re-use or expansion	SH.04, SH.07	2
N.04	Transportation	The system should be transportable by facilities personnel	SH.05	4
N.05	Safety	The system should be safely and acceptably operable in and around the WPI community	SH.05, SH.12	2
N.06	Compliance with WPI rules	The system should comply with all rules in place for the campus	SH.05	2
N.07	Compliance with park rules	The system should comply with all applicable parks and recreation regulations	SH.06	3
N.08	Non-interference	The system should not interfere with the processes or systems of other MQP teams.	SH.07	4
N.09	Budget	The system budget should fall within an acceptable limit	SH.08	1
N.10	Procurement	The procurement for the system should be acceptable by relevant processing procedures	SH.08	3

N.11	Compliance with government regulations	The system should comply with all applicable OSHA, FCC, FAA standards and regulations	SH.09	3
N.12	Incorporation of past MQPs	The system should properly and respectfully incorporate subsystems of previous MQP teams	SH.10	3
N.13	Contribution	The system should significantly contribute to the scientific or technical knowledge of involved communities	SH.11	4
N.14	Non-compete	The system should not infringe on or hinder the research of others	SH.13	2
N.15	Academic honesty	The system should not involve plagiarism	SH.13	1
N.16	Compliance with IP laws	The system should comply with all applicable IP laws	SH.13	1
N.17	Applications to space exploration	The system should demonstrate amenability to applications in space exploration	SH.14	4
N.18	Outdoor operation	The system should be able to operate in outdoor unstructured environments	SH.14	2
N.19	Applications to Mining	The system shall demonstrate amenability to applications in mining	SH.14	4
N.20	Applications in Defense	The system should demonstrate capabilities that have potential value to defense	SH.14	4
N.21	Transferability	The system should be transferable to existing working environments and processes	SH.15	5

Table 11: Needs Analysis

A.3 Context Diagrams

In order to clearly define the boundaries of the system and how it interacts with other systems, context diagrams were created. Each diagram depicts the system's interactions with other systems belonging to a specific context. The figures below, A.3 and A.3, show the multi-robot system interacting with systems in the context of human-robot interactions and in the context of its operating environment.

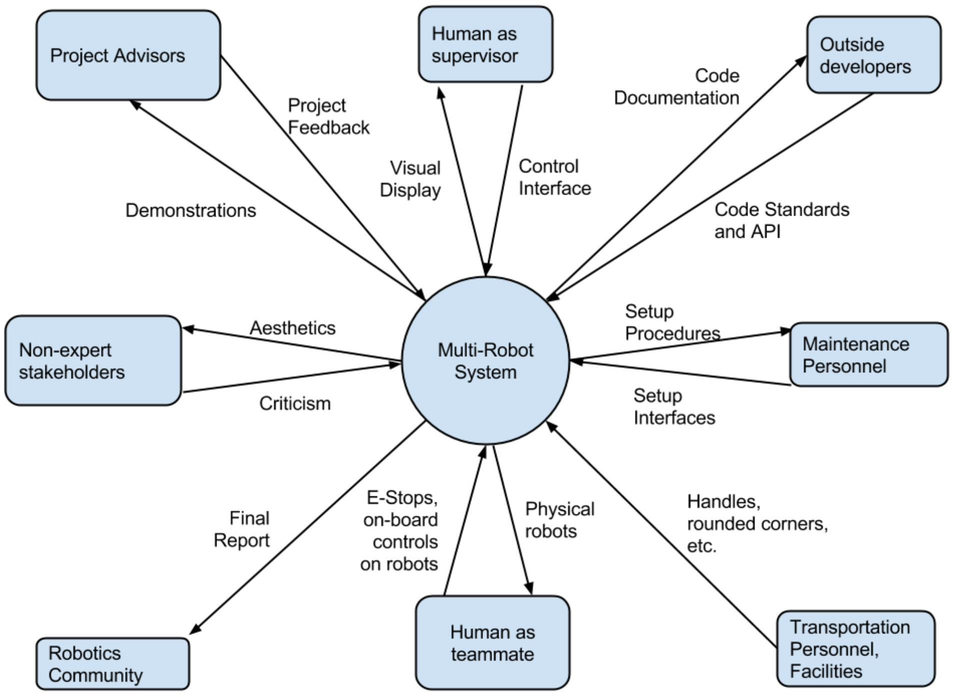


Figure 15: A context diagram showing interactions between the system and humans.

A.4 Gap Analysis

In order to determine which technologies needed to be developed and which technologies are available to be adopted, a gap analysis was conducted. This gap analysis focuses on several desired capabilities of the final product. If a capability can be acquired without conflicting with another capability using a currently available technology, then there is no gap. If there is no such technology, then one must be developed in order to attain the desired set of capabilities. These gaps between available and unavailable technologies must be understood before system development begins in order to determine the feasibility of the project.

A.4.1 Existing Technology

Existing robotic technology is sufficient to meet most of the requirements of the system. Mobile robots are currently deployed in search and rescue operations, military, law enforcement, medical care, and more, and have demonstrated manipulation capabilities suitable for these tasks. Multi-robot systems have also been demonstrated. Homogeneous "swarm" robots are well established in warehouse automation [26] and some other niche applications [27]. There is research on heterogeneous multi-robot systems, but they are not yet prevalent in application. Simple taxonomies for categorization robots and tasks have previously been developed [8] and more complex categorization methods have been implemented [5], as have algorithms for distributing tasks to robots [5, 28]. A variety of interfaces for tele-operating field

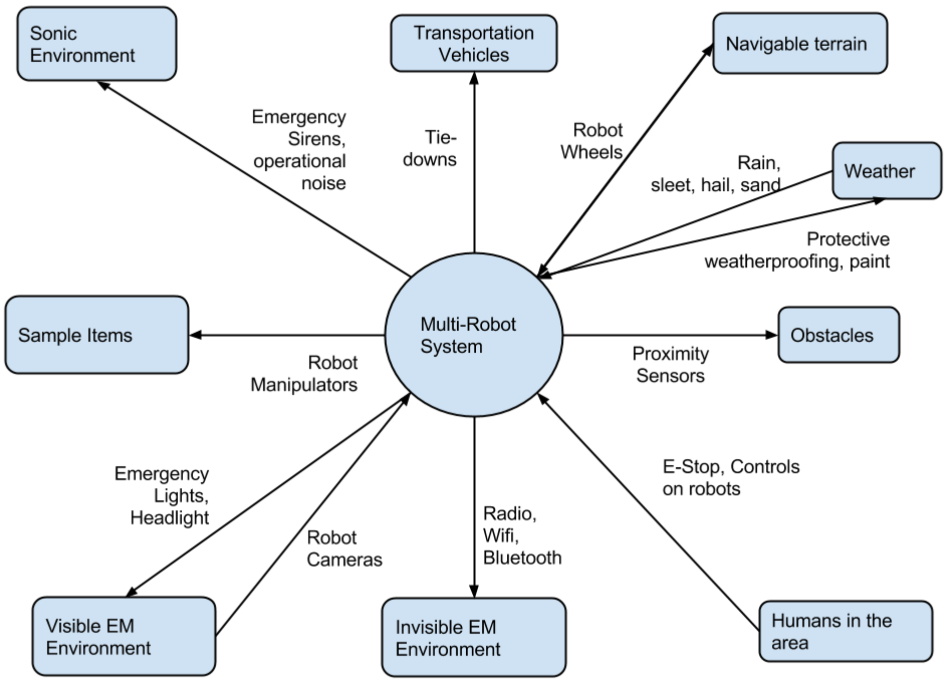


Figure 16: A context diagram showing interactions between the system and it operating environment.

robots are in use, including mobile control stations, and more advanced overhead interfaces are in active area of research [29]. Furthermore, COTS network components capable of transmitting information between and among multirobot systems and such interfaces are widely available.

A.4.2 Desired Technology

Heterogeneous multi-robot systems (HMRS) have not yet been deployed with human supervision. To do so, two primary areas need to be developed. The first is coordination algorithms for HMRS. The second is interface designs for effective human control thereof. Here, "coordination" refers to task identification, assignment, execution, by and to agents in the system. "Effective" human control is measurable by objective team performance metrics in achieving tasks during testing and by measures of the user's situational awareness - subjective and objective - while supervising those tasks.

A.4.3 Gap

There is a need for algorithms to coordinate HMRS and interfaces for humans to supervise them. All require subcomponents of such a system exist, but their working combination has yet to be demonstrated in a realistic field scenario.

A.5 Trade Study

Many of the requirements for this project were met using a variety of different technologies, methods, processes, and solutions. In order to determine which solution fits the best, trade studies were conducted. In Table 12 technologies for localization, navigation, and user interfaces were enumerated, and those for localization were examined and compared to one another. Additionally, locations for the system to be tested were also compared to find the best place to test the system.

Item	**Description**	**Candidates**	**Assessment Criteria**
Localization System	Determines spatial locations of robotic agents in operational area	GPS, Vicon localization, Fiducials (April tags), SLAM,	Position accuracy, Update frequency, Operational environment, Cost, Schedule risk, Option confidence (TRL)
Navigation System	Generates paths for the robots to take through the environments (ROS)	move_base package, nav_core package, Custom package	Schedule risk (ease of integration), Growth Potential/expandability, reliability, compatibility
UI Device	Physical device with which the user operates the system via a UI	Google Glass, Tablet, Smartphone, Laptop	Compatibility with existing systems (ROS), Cost, Portability
Testing Environment	Physical Location and setup that will serve as a testing area for the system	WPI Recreational Center Robotics Pits, Institute Park, Salisbury Park, Elm Park	Representative, Accuracy, Accessibility and restrictions, Proximity to developers, Cost, and transportation logistics

Table 12: Trade Study

A.5.1 Trade Study - Localization System

The localization subsystem is responsible for determining the spatial positions and orientations of every agent of the mutli-robot system and reporting this information to the rest of the system. As with all other subsystems, the localization system must be operable in the target situations and environments (or, in for analogous testing, the localization system must perform comparably to a system that would be used by the system in real-world deployment.)

Several alternative solutions are listed in part 1, including standard Global Positioning System (GPS) modules, the Vicon motion capture system, April Tag Visual Fiducials, and SLAM algorithms using LIDAR on individual robots.

The features of three of the selected solutions are listed below.

GPS (Global Positioning System Modules)

- Accuracy: +- 2m
- Update Frequency: 1Hz
- Cost: $100

- Technology readiness level(TRL): 9
- Operating Environment: Outdoor with no setup
- Integration (Schedule Risk): ROS GPS integration loosely supported

Vicon MoCap System:

- Accuracy: 1mm
- Update Frequency: 50Hz
- Cost: $12,500
- TRL: 9
- Operating Environment: Indoor with excessive setup and installation
- Integration (Schedule Risk): ROS Vicon integration widely supported

April Tags

- Accuracy: high (depends on range to tag)
- Update frequency: 1Hz
- Cost: $0 (cost of printing the tags)
- TRL: 7
- Operating Environment: indoor or outdoor with some setup
- Integration: ROS April Tags packages experimental

The solutions were considered according to several criteria shown in Table 13

	Position Accuracy	**Update Frequency**	**Operating environment**	**Cost**	**Schedule Risk**	**TRL**	**Total**	**Weight (%)**
Position Accuracy	1	1/2	1/6	1/7	1/9	6	7.921	0.076
Update Frequency	2	1	1/6	1/7	1/9	6	9.421	0.09
Operating Environment	6	6	1	2	1/8	7	20.125	0.193
Cost	7	7	1/2	1	1/5	8	23.7	0.227
Schedule Risk	9	9	8	5	1	9	41	0.393

TRL	1/3	1/3	1/8	1/7	1/9	1	2.045	0.02
TOTAL							104.21	1

Table 13: Selection criteria weighting tool.

Scale [0-9]

1	Equal importance
3	Moderate importance of one over another
5	Strong or essential importance
7	Very strong or demonstrated importance
9	Extreme importance
2,4,6,8	Intermediate values

Although position accuracy and update frequency were two most important technical parameters, extremely high performance in these is not required for the operation of the system in the target scenarios. Operability within the target scenarios was critical, and the cost and schedule risk (difficulty of integrating the localization system with the rest of the multi-robot system) were the key restrictive factors. Technology Readiness Level was considered as a gauge for the viability of CoTS solutions, which were preferable over custom solutions for cost, risk, and developer resources. The results of the localization trade study are summarized in Table 14.

Alternative Solution	**Positional Accuracy**	**Update Frequency**	**Operating Environ-ments**	**Cost**	**Schedule Risk**	**TRL**	**TOTAL**
Weight	0.076	0.09	0.193	0.227	0.393	0.02	
GPS	1	3	3	3	3	4	2.865
Vicon	4	4	2	1	2	4	2.164
April Tags	3	3	3	4	3	3	3.224
SLAM	2	3	4	1	1	3	2.62

Table 14: Localization trade study

A.5.1.1 Selection

April Tags had the highest weighted score of all alternative localization solutions considered.They had high scores in almost all criteria, where all other alternatives had at-least one low score in at-least one heavily weighted criteria. For example, the Vicon system, although the most accurate of all the systems, was also the most expensive, and therefore not a viable solution.

It is important to note that many of the applications where a HSMRS would be used would not be as resource-constrained as this project. Additionally, because many of the identified applications require that the robots be robust and precise, the lower quality options presented in this trade study would not be acceptable. The weights and results of the trade study vary from application to application and are not constant for all projects utilizing a HSMRS.

A.5.1.2 Sensitivity Analysis

Position Accuracy and Update Frequency were similar for every alternative, and were ranked relatively low. TRL was ranked the lowest. Operating Environment, Cost, and Schedule Risk were the three most important factors, with Schedule Risk as the most important. Ultimately, because localization was not a focus of this project, it was less important how much a solution exceeds the localization requirements than that it meets all of the requirements and is quickly integratable with the rest of the system, without going over budget. The April Tags do not have the best performance, but they are operable in the target environments, are low cost, and are relatively compatible with the rest of the system. No other alternative met all of these necessary conditions without going over budget.

A.6 Use Cases

A.6.1 Sample Return Mission

A.6.1.1 Description

The system is deployed within five minutes by a single human user in an outdoor environment to complete a sample-return mission resembling the NASA Centennial Sample Return Robot Challenge (SRR), but without the non-terrestrial constraints. Adapted from the NASA challenge description: the robot-human team will locate and collect a set of specific samples from a large open outdoor area and return the samples to a prespecified starting zone. The exploration/collection area will include rolling terrain, granular medium, soft soils, and a variety of rocks and immovable obstacles (trees, large rocks, water hazards, etc.) The human-robotic team will collect at-least two samples over 30 minutes of operation after deployment.

Actors:

- The human supervisor/operator
- Other human bystanders in the area
- Transportation crew
- Robot team

Assumptions:

- The human-robot team has access to aerial/geographical/topographical maps with appropriate resolution, including the location of the starting position(s) and pre-cached sample(s).
- The samples will be easily distinguished from other materials present at the site since the sophisticated scientific instrumentation for sample identification is outside the scope of the project.

- Samples will be easily collectable by human and robotic agents since manipulation is not a part of the project.
- The robotic systems may have access to any localization technology that provides resolution comparable to that achievable in a disaster scenarios, since multi-robot SLAM has previously been demonstrated, and is likewise outside the scope of the project.
- There is suitable communication bandwidth for all data flowing through the system, since data compression and operation over degraded communication links are likewise previously demonstrated in the DARPA Robotic Challenge (DRC) and are outside the scope of this project.
- The robotic agents can have different starting locations.
- The human can assist the robotic agents in the search for and recovery of samples, forming a hybrid team.
- All components of the system are prepared for generalized deployment (batteries charged, com channels set, etc.).

Steps:

- (Trigger) the sample return challenge analog scenario commences.
- The operator, transportation crew, and three robotic agents arrive on scene.
- The operator and transportation crew move the robotic agents to their starting location(s).
- The operator and crew set up any supplementary systems, such as a localization “stand-in” system. This setup time is not included in total deployment time.
- The supervisor activates the system using a mobile user interface (UI) using a single command, completing the deployment phase.
- The human selects search areas on a “map view” representation of the environment based on the current state of the system (robotic agents) within the environment.
- The system automatically mobilizes the robotic team to search the specified areas and recover samples, intelligently assigning different robotic agents to tasks for which they are most suited based on the state of the system and their individual capabilities and configurations.
- The fastest robot in the team autonomously navigates to the farthest area from the other two robots.
- The most rugged robot is sent to the search area which is most likely to involve difficult terrain, such as soft ground.
- The “fast” robot finds a sample and attempts to recover it, but fails in this task because its manipulator has insufficient power to lift the sample.

- The third robot, which has a more powerful manipulator, is sent to recover the sample instead, while the fast robot continues searching the remaining areas for more samples.
- The fast robot gets stuck in mud. The operator takes direct control of this robot to free the robot to recover mobility.
- The "power" robot and "fast" robot find and collect both samples, returning them to the drop-off location.
- The freed "rugged" robot autonomously navigates back to the drop off location.
- The operator disengages the system using another single command, concluding the sample-return phase.
- The user and transportation crew take down all supplementary systems.
- The user and transportation crew pack up the robots.

A.6.1.2 Variations

This use case has two notable variations. If the user was unable to free the "fast" robot in step 7e, or if the robot failed in some other way, the "rugged" robot would have taken over its responsibilities and completed the search. If the user chooses to search some area themselves in step 6, the robotic system will avoid sending robotic agents to search that area, avoiding redundancy and maximizing agent resource utilization.

A.6.1.3 Outside Impacts

This use case deals with three potentially problematic externalities. Firstly, the user will require moderate training in the deployment and supervision of the system. Secondly, completion of specific tasks by the system is dependant on robotic technologies outside the scope of this project, such as manipulation hardware and algorithms, GPS, localization and mapping. Thirdly, the system will need to interface with standard internet communication protocols such as wifi, as well as standard power infrastructure for charging.

A.6.2 Adding a new robotic agent to the system

A.6.2.1 Description

The operator(s) need to integrate an additional robotic agent with the heterogeneous robot team.

Actors:

- The operator(s)
- The development team of the new robotic agent

Assumptions:

- The new robotic agent is technologically compatible with the system (communications, operating system, etc.).
- The new robotic agent is suitably semi-autonomous for the target mission.
- The relevant capabilities and specifications of the new robotic agent are available and well-defined.
- The new robotic agent is mobile and otherwise suitable for integration with a team to perform the target tasks, such as the first use case scenario.

Steps:

- The operator(s) evaluate the new robot's capabilities in the context of those capabilities representable within the semi-autonomous heterogeneous robotic agent framework.
- The operator(s) create a new robot configuration file on the main system computer, either through a text editor or a graphical user interface (GUI) robot configuration manager.
- The operator(s) enumerate the capabilities of the new robot according to the standard robot configuration format.
- The operators save the new robot configuration file to the "RobotAgents" system folder.
- The operator(s) execute a configuration verification command to verify the new robot's configuration can be recognized and successfully parsed by the system using a command line interface or the robot configuration GUI.
- The robot is automatically incorporated by the heterogeneous robot team in th next mission

Variations:

- If the file is not recognized or has an invalid format (unparsable), the system will display on error and allow the user(s) to edit the configuration to correct the mistakes.
- If the configuration is missing required fields (such as robot maximum speed), the system will alert the operator(s) to the error and allow them to fill in the missing information.
- The operator(s) may also choose to create a new robot configuration by copying and editing an existing configuration.

Outside Impacts:

- The system will rely on the operating system and software programming language used to implement file creation, editing, deletion, etc.
- The operator(s) will require moderate training to perform the robot agent adding process.

A.7 Key Performance Parameters

Key Performance Parameters are key system capabilities that must be met in order for a system to meet its operational goals. When determining what these parameters are, a method of testing must also be proposed. The system must pass the tests of all parameters in order to be considered operational. Table 15 lists the five key performance parameters for this project.

MOE	Title	Description	Method of Testing
E.1	Multiple Robots	The system should be configurable to incorporate multiple robotic agents during mission operations.	The number of robotic agents successfully integrated during testing operations.
E.2	Heterogeneous Robots	The system should incorporate robots of different capabilities and configurations	The system is configurable to recognize and utilize the different capabilities and configurations of different robots.
E.3	Human Supervision	The system should be controlled at a high level by a single human.	A human operator can remotely assign mission tasks and supervise their execution though high-level control of the robot team during task completion.
E.4	Deployable Outdoors	The system should operate in outdoor environments resembling the target scenarios	Terrain roughness, area covered, severity of environmental conditions
E.5	Rapidly Deployable	The system should be rapidly deployable by a single trained operator.	Time to fully deploy all components of the system

Table 15: Key Performance Parameters

B Source Code

B.1 Agent

```
/**************************************************************************
 *                                          HSMRS Framework - Agent.h
 *
 *      This class represents an Agent. An Agent performs Tasks it is              ***
 *      assigned.
 **************************************************************************/
```

```
#pragma once

#include <string>
#include "Task.h"

class Agent
{
public:

        /**
         * The constructor for the Agent object.
         */
        Agent(void){}

        /**
         * The destructor for the Agent object.
         */
        ~Agent(void){}

        /**
         * Stops execution of the current Task and requests that
         * the Task be returned to the TaskList.
         */
        virtual void cancelTask() = 0;

        /**
         * Asks the Agent to claim a task pointed to by \p task.
         * @param task A pointer to the task object to be claimed.
         */
        virtual void claimTask(Task* task) = 0;

        virtual std::string getName() = 0;
};
```

B.2 AgentState

```
/******************************************************************
 *                                     HSMRS Framework - AgentState.h
 *
 *      This class represents an AgentState. The AgentState    is used to
 *      keep track of certain important qualities of an Agent, called        ***
 *      attributes.
 **************************************************************************/

#pragma once
#include <string>
#include <map>

class AgentState
{
public:
        /**
         * The constructor for the AgentState object.
         */
        AgentState(void){}

        /**
         * The destructor for the AgentState object.
         */
        ~AgentState(void){}

        /**
         * Returns the value of the given attribute.
         * @param name The name of the atttribute
         * @return The value of the attribute with the name \a name
         */
        virtual double getAttribute(std::string name) = 0;

        virtual void setAttribute(std::string attr, double) = 0;

        /**
         * Returns the mapping of attributes to values.
         * @return The mapping of attributes to values.
         */
        virtual std::map<std::string, double> getAttributes() const = 0;
};
```

B.3 AttributeWeights

```
/***************************************************************************
```

```
 *                                        HSMRS Framework - AttributeWeights.h
 *
 *       This class represents an AttributeWeights is a container that          ***
 *       tracks the importance of each attribute (in a set of attributes) to    ***
 *       completing a certain Task. These weights are used in conjunction       ***
 *       with an AgentState to calculate an Agent\'s utility for a Task.        ***
 ***********************************************************************/

#pragma once
#include <string>
#include <map>

class AttributeWeights
{
public:
        /**
         * The constructor for the AttributeWeights object.
         */
        AttributeWeights(void){}

        /**
         * The destructor for the AttributeWeights object.
         */
        ~AttributeWeights(void){}

        /**
         * Returns the weight of the attribute with the given name.
         * @param name The attribute to get the weight of.
         * @return The weight of the attribute with the name \a name.
         */
        virtual double getWeight(std::string name) = 0;

        /**
         * Returns a map of all contained attributes and weights.
         * @return A map, with keys being attribute names and values
         *                    being their respective weights.
         */
        virtual std::map<std::string, double> getWeights() = 0;
};
```

B.4 Human

```
/*************************************************************************
 *                                                  HSMRS Framework - Human.h
 *
 * The Human class represents the human supervisor of the system and    ***
 * their user interface. The Human is capable of manipulating the            ***
 * TaskList, controlling robots, and sending and receiving messages          ***
 * from all parts of the system.
 ************************************************************************/

#pragma once

#include "Task.h"
#include "Role.h"
#include "Robot.h"
#include "Agent.h"

class Human: public Agent
{
public:
        /**
         * The constructor for the Human object.
         */
        Human(void){}

        /**
         * The destructor for the Human object.
         */
        ~Human(void){}

        /*
         * Handles a request from a Robot to have a task added to the
         * TaskList. Operates as a service for Robots to call.
         * @param task A pointer to the requested task.
         */
        virtual void handleTaskRequest(Task* task) = 0;

        /**
```

```
         * Handles a request from a Robot to have the Human's focus
         * set to that Robot.
         * @param task A pointer to the Robot requesting help.
         */
        virtual void handleHelpCall(Robot* source) = 0;

private:
        /**
         * Assigns a given Task to a given Agent
         * @param task The task to be assigned
         * @param agent The Agent to be assigned \a Task
         */
        virtual void assignTask(Task* task, Agent* agent) = 0;

        /**
         * Remove the current task from the provided Agent.
         * @param A pointer to the Agent whose Task is to be removed.
         */
        virtual void unassignTask(Agent* agent);

        /**
         * Assigns a given Role to a given Agent
         * @param role A pointer to the Role to be assigned
         * @param agent A pointer to the Agent to be assigned \a role
         */
        virtual void assignRole(Role* role, Agent* agent) = 0;

        /**
         * Place a Task on all of the Agent's TaskLists to be queued
         * and auctioned.
         * @param task A pointer to the Task to be queued
         */
        virtual void queueTask(Task* task) = 0;

        /**
         * Remove a Task from all of the Agent's TaskLists.
         * @param task A pointer to the Task to be removed.
         */
        virtual void deleteTask(Task* task) = 0;
```

```
};
```

B.5 Prerequisite

```
/*************************************************************************
 *                                       HSMRS Framework - Prerequisite.h
 *
 * The Prerequisite class represents the an abstract prerequisite that  ***
 * can be fulfilled or not
 *************************************************************************/

#pragma once
class Prerequisite
{
public:
        /**
         * The constructor for the Prerequisite class
         */
        Prerequisite(void){}

        /**
         * The destructor for the Prerequisite class
         */
        ~Prerequisite(void){}

        /**
         * Determines if this Prerequisite has been fulfilled.
         * @return True if the Prerequiste has been fulfilled.
         */
        virtual bool isFulfilled() = 0;
};
```

B.6 Progress

```
/*************************************************************************
 *                                           HSMRS Framework - Progress.h
 *
 * The Progress class represents an abstract progress which can have   ***
 * reports on the state of the progress
 *************************************************************************/
```

```
#pragma once
class Progress
{
public:
        /**
         * The constructor for the Progress class
         */
        Progress(void){}

        /**
         * The destructor for the Progress class
         */
        ~Progress(void){}

        /**
         * Determines the state of the Progress and returns it as
         * a decimal percent.
         * @return The state of the Progress as a decimal percent.
         */
        virtual double report() = 0;
};
```

B.7 Robot

```
/**********************************************************************
 *                                        HSMRS Framework - Robot.h
 *
 * The Robot class represents a robotic agent of the system. Robots are ***
 * capable of performing tasks, auctioning tasks, and communicating     ***
 * with other agents.
 *********************************************************************/

#pragma once
#include <string>

#include "AgentState.h"
#include "Task.h"
#include "Agent.h"
#include <hsmrs_framework/BidMsg.h>
```

```
class Robot: public Agent
{
public:
        /**
         * The constructor for the Robot object.
         */
        Robot(void){}

        /**
         * The destructor for the Robot object.
         */
        ~Robot(void){}

        /**
        * Returns the value of the specified attribute from this Robot's AgentState.
        */
        virtual double getAttribute(std::string) = 0;

        /**
        * Returns this Robot's utility for the specified Task.
        */
        virtual double getUtility(Task *task) = 0;

        /**
        * Returns this Robot's AgentState.
        */
        virtual AgentState* getState() = 0;

        /**
        * Checks if this Robot has the given attribute.
        */
        virtual bool hasAttribute(std::string attr) = 0;

        /**
        * Sets this Robot's currently acctive Task.
        */
        virtual void setTask(Task* task) = 0;

        //callbacks
        /**
        * Handles the auctioning of Tasks by sending and receiving bids.
        */
        virtual void handleBids(const hsmrs_framework::BidMsg::ConstPtr& msg) = 0;

        /**
        * Verifies that an Agent claiming a Task has the highest utility for it. If not,
        */
        virtual void verifyTaskClaim() = 0;

        virtual void callForHelp() = 0;

        virtual void sendMessage(std::string message) = 0;
private:
        /**
         * Makes this Robot bid on the given task
         */
        virtual double bid(const hsmrs_framework::BidMsg::ConstPtr& msg) = 0;

        virtual void executeTask() = 0;

        virtual void handleTeleop() = 0;

        virtual void requestTaskForQueue(Task* task) = 0;
};
```

B.8 Role

```
/*******************************************************************
 *                                              HSMRS Framework - Role.h
 *
 * The Role class represents a set of behaviors an Agent should exhibit ***
 * when not assigned a specific Task. The behaviors are represented as  ***
 * Tasks.
 *******************************************************************/

#pragma once
#include <vector>
#include <string>
```

```
class Role
{
public:
        /**
         * The constructor for the Role class
         */
        Role(void){}

        /**
         * The destructor for the Role class
         */
        ~Role(void){}

        /**
         * Adds a Task to the Role
         */
        virtual void addTask(std::string task) = 0;

        /**
        * Removes a Task from the Role
        */
        virtual void removeTask(std::string task) = 0;

        /**
        * Returns all Tasks associated with this Role
        */
        virtual std::vector<std::string> getTasks() = 0;
};
```

B.9 Task

```
/***********************************************************************
 *                                          HSMRS Framework - Task.h
 *
 * The Task class represents a set of actions Agents that have claimed  ***
 * it should perform.
 **********************************************************************/

#ifndef _TASK_H_
 #define _TASK_H_
```

```
#include <hsmrs_framework/TaskMsg.h>

#include <string>
#include <vector>
#include <map>

class Task{
public:
        /**
         * The constructor for the Task class
         */
        Task(void){}

        /**
         * The destructor for the Task class
         */
        ~Task(void){}

        /**
         * Retrieves the type of this Task.
         * @return A string containing the type of this Task.
         */
        virtual std::string getType() = 0;

        /**
         * Retrieves a vector of Agents who have claimed this Task
         * @return A vector of Agents who have claimed this Task
         */
        virtual std::vector<std::string> getOwners() = 0;

        /**
         * Determines the minimum number of Agents that need to claim this
         * Task for the Task to be executed.
         * @return The minimum number of owners for this Task.
         */
        virtual int getMinOwners() = 0;

        /**
```

```
 * Determines the maximum number of Agents that can claim this
 * Task.
 * @return The maximum number of owners for this task.
 */
virtual int getMaxOwners() = 0;

/**
 * Retrieves a mapping between Attributes and their weights.
 * @return A map of strings and doubles
 */
virtual std::map<std::string, double> getAttributeWeights() = 0;

/**
 * Retrieves a vector of subtasks which need to be completed as part of
 * this task
 * @return A vector of Tasks which are a part of this Task.
 */
virtual std::vector<Task*> getSubtasks() = 0;

/**
 * Adds an Agent as an owner to this task, if another owner can be added
 * @param agent The Agent to be added as an owner.
 */
virtual void addOwner(std::string agent) = 0;

/**
 * Removes the given Agent as an owner.
 * @param agent The name of the Agent to be removed as an owner.
 */

virtual void removeOwner(std::string name) = 0;

virtual double getPriority() = 0;

virtual void setPriority(double p) = 0;

virtual bool isReady() = 0;

virtual int getID() = 0;
```

```
        virtual void setProgress(double val) = 0;

        virtual hsmrs_framework::TaskMsg* toMsg() = 0;
};

#endif
```

B.10 TaskList

```
/*********************************************************************
 *                                        HSMRS Framework - TaskList.h
 *
 * The TaskList class represents a container which holds, sorts, and    ***
 * provides Task objects when needed.
 *********************************************************************/

#pragma once

#include <vector>

#include "Task.h"

class TaskList
{
public:
        /**
         * The constructor for the TaskList class
         */
        TaskList(void){}

        /**
         * The destructor for the TaskList class
         */
        ~TaskList(void){}

        /**
         * Adds a Task to the TaskList
         * @param task The Task to be added
         */
        virtual void addTask(Task* task) = 0;

        /**
                 * Retrieves the Tasks in the TaskList
                 * @return A vector of Tasks contained in this TaskList
                 */
        virtual std::vector<Task*> getTasks() = 0;

        /**
         * Determines if the TaskList is empty
         * @return True if the TaskList is empty
         */
        virtual bool isEmpty() = 0;

        /**
         * Retrieves the next task from the TaskList. The
         * next task is one with all of its Prerequisites
         * fulfilled and the highest priority.
         * @return The next Task
         */
        virtual Task* pullNextTask() = 0;

        virtual Task* getTask(int id) = 0;

/**
         * Removes the given task from the TaskList.
         * @return The removed Task
         */
        virtual void removeTask(int id) = 0;

        /**
         * Sets the priority of the given Task to the given priority
         * @param task The Task whose priority will be changed
         * @param priority The new priority for the task.
         */
        virtual void setPriority(int task, double priority) = 0;
private:
        /**
         * Sorts the TaskList in order of decreasing priority
         */
        virtual void sortByPriority() = 0;
};
```

B.11 UtilityHelper

```
/*****************************************************************************
 *                                          HSMRS Framework - UtilityHelper.h
 *
 * The UtilityHelper class represents an object whose sole purpose is    ***
 * to calculate the Utility of an Agent
 *****************************************************************************/

#pragma once

#include "Robot.h"
#include "Task.h"

class UtilityHelper
{
public:
        /**
         * This is the constructor for the UtilityHelper class
         */
        UtilityHelper(void){}

        /**
         * This is the destructor for the UtilityHelper class
         */
        ~UtilityHelper(void){}

        /**
         * Calculates the utility of the given Robot for the given
         * task.
         * @param robot The Robot whose utility is being calculated
         * @param task The Task used to determine the utility.
         */
        virtual double calculate(Robot* robot, Task* task) = 0;
};
```