Heterogenous Multi-Robot Testing and Exploration

Jackson Parker

Abstract

The Heterogeneous Autonomous Robotic Exploration system (HARE) is meant as a proof of concept for cooperative heterogeneous systems that divide tasks based on the differentiable capabilities of the robots. This research project ended up down an interesting path due to early design choices, leading to the creation of a custom ROS package for simple testing of systems like this.

I. Introduction

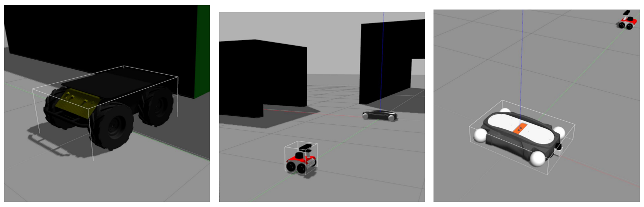

(a) Husky bot (b) Rosbot (c) Youbot

Fig. 1: Three commonly used robots with very different shapes, sizes, and capabilities.

The study of multi-robot systems and autonomous swarms is becoming more and more of a hot topic in the world today. Being able to parallelize a large task or simply cover more ground, these systems have the capability to perform a lot of interesting and tedious tasks to the common individual. More often than not, these applications are inherently homogeneous, meaning each robot is the same as this could be cheaper and build in more of a fault tolerant attitude. One downside to homogeneity in the context of multi-robot systems is that the overall system would be limited based on the capabilities of that specific type of robot. This brings to mind the idea of utilizing heterogeneous robots to accomplish singular tasks. A lot of research has gone into heterogeneous multi-robot systems in the past few years, but a common trend amongst most of them is that the model is built for the specific platforms that are to be used. For instance if a researcher utilized a UGV and UAV as their heterogeneity agent makeup. This proposed research aims to tackle that problem and to encourage interest and experimentation in the field of heterogeneous multi-robot systems. Upon iterations, this research will ultimately be meant to provide the groundwork for a heterogeneous model that allows for the insertion of a robots with different skill sets with the purpose of enhancing the overall capability of the system. This framework would allow for a variety of models to be built on top of it, with a variety of hardware. Researchers wanting to build a specific system could allocate funds and resources to build each member to accomplish a portion of the overall task, possibly allowing for more cost efficiency and remove unnecessary redundancy. The main challenge in this research will be finding the optimal way to reassign tasks and have a variety of systems work together to accomplish a global task. With primary informatic clashes would occur at the sensor level with varying sensor models as well as at the control level relating to movement capabilities. What was not necessarily expected out of this research was a structured and easy to understand testing package for heterogeneous systems developed in ROS and simulated with GAZEBO [1] . Due to encountered limitations with initially chosen software and technology, the development of this framework is necessary to fully test this concept before taking the step towards hardware.

II. Methodology

A. Testing Framework

After exploring a set of options to test this idea, the best place to go was to the drawing board. Taking a step back it was decided that pushing to get straight into hardware testing was a poor decision and the underlying concept of heterogeneous task reallocation based on capabilities needed to be tested at the lowest of levels. To do this, the creation of a testing framework in roscpp was necessary. The primary components of this system include the physical entities, motion model, robot addition, robot.run() and the simulated verbose map,. The system has each robot running individually, only communicating with neighbors through subscription and publication to topics that are necessary to implement cooperation. This allows for testing of a completely decentralized algorithm in a centralized system, giving algorithmic and cooperation designs the primary focus while abstracting out complicated sensor models. This system started to take shape this over the course of the past few months when trying to use third party multimaster implementations in Gazebo, which turned out to necessitate the experiment be running on multiple computers.

The major components of the testing framework that I designed include an Entity class with Neighbor and Robot derived class, a map class that holds the obstacle information and can be queried based on a sensing radius, and the main loop. The use of C++ templates and proper organization according to ROS best practices, this testing framework can be utilized for far more than this project alone, and was all written from scratch [1].

1) Entities: The Entity class is the base physical structure for this testing package. This could represent a static or one of the two derived classes: Neighbor and Robot. Entities hold major variables like state information, initial position and odometry. The Neighbor and Robot classes are very similar to each other as they both represent robots, the Neighbor class is just used to hold the information for the Robot's neighboring robots as they do not need to keep track of their entire path and known map, like they are doing for themselves.

2) Motion Model: The motion model is pretty simple, but is by far the hardest to get right for all robot systems. This portion would often necessitate the user to implement some helper methods to properly used the simple motion primitives that each Robot has as a few class functions like goLeft(), goRight, go(float3), etc. Regardless, the main way to ensure that control is occurring properly. One feature that would allow for easier handling of this would be the yaml config files that are briefly discussed in the next section.

3) Adding Robots: This framework allows for easy addition of robots by simply adding the details for the new robot in the xml robots launch file. To add any additional information yaml files can be used to set rosparams and can be queried in any rosnode as long as the yaml file was specified in one of the launch files as a rosparam. Utilization of the yaml files is essential to adding and sharing information like hardware characteristics before Robots enter the main loop. So all in all, By configuring the launch file with the proper robots, one can edit the one robot launch file to have each robot launch any set of nodes that would exist in each robot's namespace.

One thing that was essential to streamline and allow for flexibility was the instantiation of robots via partitioned launch files. By defining separate roslaunch files instead of one, it is easy to see where to put new robots as well as ensure that one point of failure is taken care of. The one robots launch file launches nodes that are to be in every robot, who are defined by namespaces in the context of ROS. The robots launch file is where robot descriptions are set along with locations of urdf files containing meshes and other simulation specific data for the individual robot. The launch file can be called by a users custom launch file; in our case this was the hare sim launch file.

As can be seen in the figure at the top of this document tests were conducted with a Husky bot, rosbot, and youbot. The Husky bot is more of a heavy duty autonomous vehicle with a wide range of terrain capabilities. The rosbot is a small robot that is generally used for testing and eduction. Finally, the youbot which may be the most interesting of the bunch as it has holonomic capabilities due to having the well know swiss wheels.

4) Robot.run(): Due to this being a decentralized testing system, even though it is actually centralized, all robots must be able to operate autonomously. This is accomplished by all robots running the same main loop and keeping their neighbors aware of where they were in the operational lifetime as well as if help is needed from a fellow agent. This loop can be seen below in 2 after the first direction assignment has been determined and is located solely in the method robot.run().

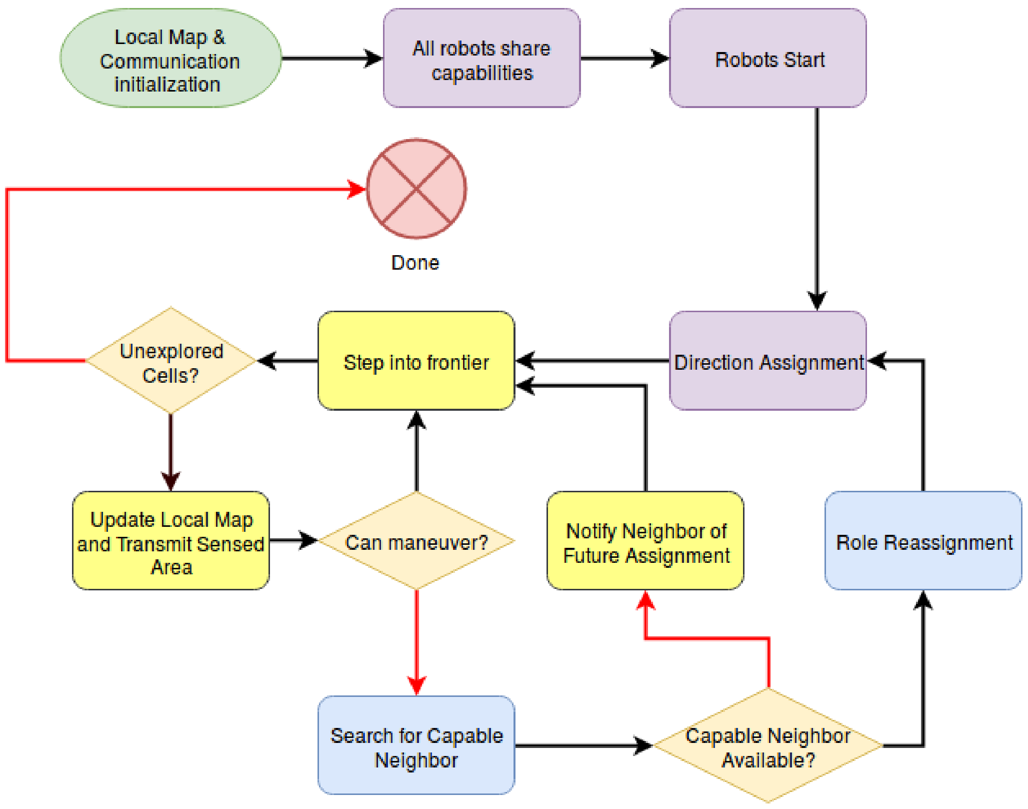
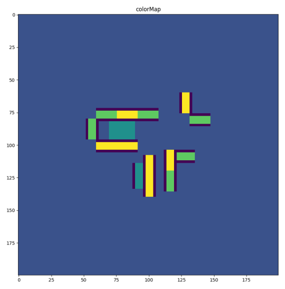
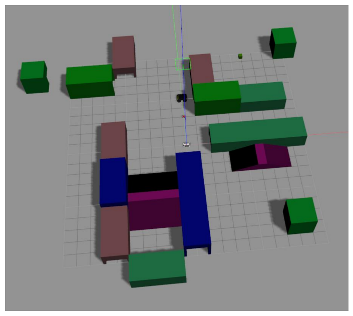

Local Map & Communication initialization
All robots share capabilities
Robots Start
Done
Unexplored Cells?
Step into frontier
Direction Assignment
Update Local Map and Transmit Sensed Area
Can maneuver?
Notify Neighbor of Future Assignment
Role Reassignment
Search for Capable Neighbor
Capable Neighbor Available?

Fig. 2: High level block diagram for HARE

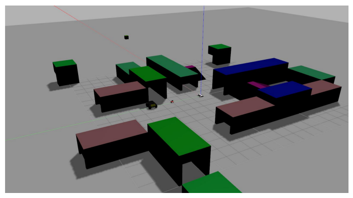
5) Simulated Map: The simulated map is essential to testing as it allows for user defined obstacles as well as the potential to expand into randomly generated maps. This map is a static NxN array of map nodes which hold information like cell terrain, if the cell had been explored, edge characteristics, and more. The usefulness of a map like this comes from the ability to self define edge cases as well as not have to worry about a complicated mapping and sensing scheme. This could obviously not be used in the real world, but is great for testing. Even though the entire map exists in the software, robots cannot simply access this map, they can only access a certain radius around them called their sensed region. Each robot will update its known map whenever new cells are sensed and will also transmit those cells to their neighbors to maximize exploration and global coverage. The first map that this system utilizes can be seen below in 2D in 3a as well as 3D in 3b and 3c.



(a) 2D view of HARE test map



(b) Hare Map angle a



(c) Hare Map angle b

B. Algorithms

Even though the algorithms in this system became less of a focus, the usage of a custom weighted Depth First Search was designed as well as a heuristic based path planning approach called Greedy Best First.

1) DFS: Psuedo code for the designed "DFS" algorithm can be seen below:

Algorithm 1 Custom MRS Weighted DFS

```
1: Let: $map = \{cell_{\text{x,y,ownerID}} \in R^2\}$
2: Let: $path = \{cells \in map_{\text{visited}} | length \leq maxPathLength\}$
3: Let: $weights = \{neighbor, groupCentroid, xy\}$

4: procedure OVERALL STEP FUNCTION
5:     recvUpdates:
6:     $neighborInfo \leftarrow robot_i.update$
7:     $map_{\text{x,y}} \leftarrow robot_i.id$
8:     checkForStopCriteria:
9:     if $length(group) == length(Robots)$ && $self.cell == leader.cell$ then return
10:    refineGroup:
11:    computeOptions:
12:    move:
13:    $checkFreedom()$
14:    $checkForPause()$
15:    $sortDirections()$
16:    for $rankedDirections \in possibleDirections$ do
17:        if $self.backTrack$ then break
18:        if $cell_{\text{t+1}}.ownerID! = self.id$ then
19:            $group.add(cell_{\text{t+1}}.ownerID)$
20:            $move()$
21:    if $hasNotMoved$ then
22:        $backTrack()$
23:        $self.backTrack = false$
24:    checkForSuccessCriteria:
25:    if $length(group) == length(Robots) \&\& euclideanDistance(robot_i, self) == 0$ then
26:        $success = true$
27:        $exit()$
```

2) Greedy Best First Search: In addition to DFS, Greedy Best First Search algorithm was implemented to derive a possible path to a certain location in the map [2]. In 4 one can see the path from one point to another being highlighted. This algorithm is not the best for path planning but was thought to be sufficient for the design with this test, not to mention much of the focus had to be shifted the design of the testing package after losing time to technology option rabbit-holes.

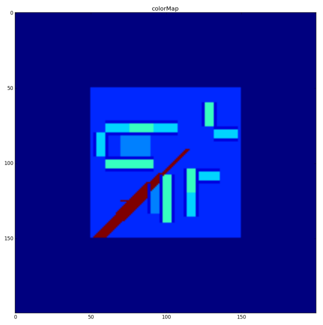


Fig. 4: Greedy Best First Path Illuminated

3) Role Switching: As stated, the purpose of this research is to prove that subtask division can be accomplished based on knowledge of robot capabilities. To test this at the lowest level, each robot was given a set of terrains that they could maneuver through, with no one being able to maneuver through walls. Terrain classification was set in the following scheme and was recorded on the map nodes of the fullMap, which as previously mentioned is hidden from view until the robots sense the cells in question.

wall	open cell	ramp	small arch	large arch
-1	0	1	2	3

Once a robot comes to an obstacle or region that it cannot maneuver around, it will evaluate which team members are capable of doing so a request the closest candidate to switch observable areas.

III. Discussion

Below one can see the general framework of executables that this testing package generates to emulate and decentralized system of 3 heterogeneous robots. Each robot is in its own namespace, allowing it to have the same internal nodes as other robots as well as abstract away information that is only pertinent to itself.

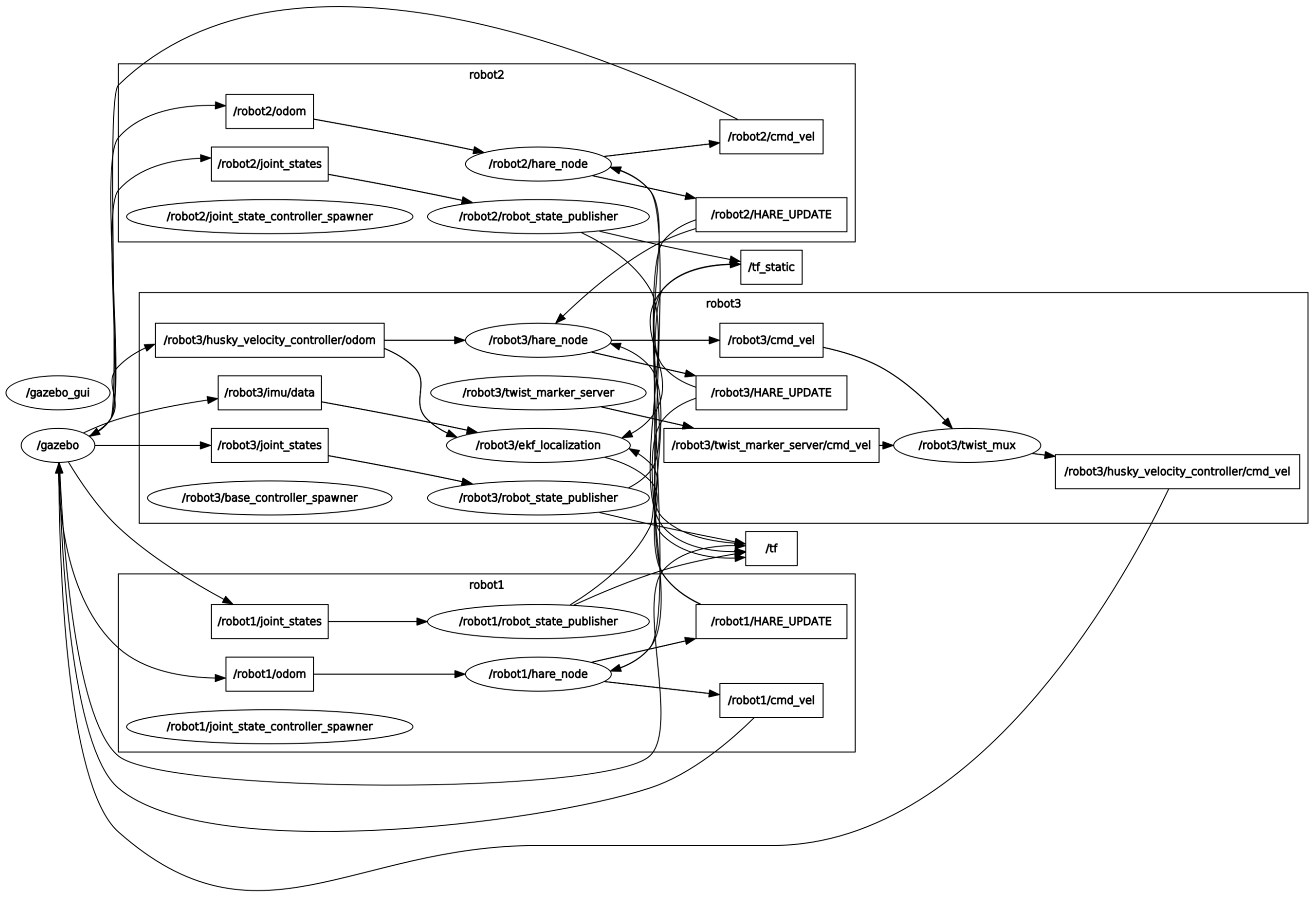


Fig. 5: RQT Graph of Peer-to-Peer ROS node organization

A. Conclusion

Over the course of this semester, even though a lot of problems were encountered by this project, the learning experience was invaluable. Understanding a system like ROS, especially being forced to understand it at a ”deep C” level, opens a lot of doors for my future research and ability to simulate ideas in a near real world environment.

B. Future Work

Primary future focus will be to get the testing framework wrapped up for release as well as applying that completely finalized framework to extensive heterogeneous multi-robot testing. Even though the testing framework was not the overall goal of this research, it turned out to be one of the strong suits of the project. Future work would certainly include iteration on this framework to ensure that it is as easy as possible to randomly generate maps as well as add any number of any kind of robot. The benefit of doing this on Gazebo is the ability to emulate robotic agents much closer to the real world representation than Unity, ARGoS, or any other open-source platform capable of multi-robot simulation.

Another step would of course be iteration of the tested cooperation model on hardware. This would require much more extraneous simulation to ensure that all edge cases are covered. To properly do this, the testing framework would have to be iterated as described above. Due to Gazebo not inherently supporting multimaster network architectures, there is the possibility that a plugin or new simulation package would need to be developed.

C. Eventual Rejuvenation of Initial Goals

What started as the basis of this research, was in the definition of attributes to generalize robot software and hardware capabilities to optimally cooperate with other heterogeneous robots. As this research was meant to strictly provide a basis for heterogeneous task division and capability utilization, the model built on top of it could take many forms. Fully fleshing out this system would be as simple as having a reasonable usage for extensive robot capability information, really meaning that a complex enough task would have to be derived and tested once the testing framework has been completely solidified and ready for release. Before problems arose that required focus to be redirected towards the testing framework that was implemented. To illustrate the thought that went into attribute listing, the appendix has a set of tables that show where this research is eventually headed.

IV. Using The Code From This Paper

The code written for this paper is available at https://github.com/uga-ssrl/hare. The README.md must be read carefully in order to launch successfully, as with any other ROS package.

Updates to this repository will occur on a weekly basis for the foreseeable future.

V. Attribute Listing

Keep in mind that these were not used during this project, just designed before changes and pivots were necessary. These attributes are to be used as descriptors of a robots capabilities and are shared with all neighboring robots. Many of them have associated numerical values, enumerations, or tuples to further detail extent of related capability. All action nodes in the behavior graph, along with some directive nodes, have requirements and target values associated with certain attributes to direct traversal. As long as requirements are fulfilled, the euclidean distance away from that set of target values, along with some other specific values like location, would be used to determine the cost of the individual robot taking that path forward in the behavior graph. If necessary, neighboring robots would communicated their cost to one another as well as take into account what the global objective requires to determine if one or both can take the evaluated path.

Dimensionality	0, 2D, 3D
FOV	angle
Global Positioning	Boolean: Good if within sensor error margin in meters
Non-spatial	sensors num, telemetry type
Visible Distance	In meters

Dimensionality	directional proximity, 2D, 3D
Allowable Altitude Range	relative to lowest known ground 0
Terrain Compatibility	submersible, non-submersible, ground, aerial, amphibious, submersible amphibious, full
Turn Radius	[0 (holonomic) , (nonholonomic))
Moving appendages	num, dexterity levels, branch/joint specifiers
Motor Current Requirement	Current (minimum)

Duplex	half or full
Transmission FOV	angle
Receiver FOV	angle
Transmitter Count	num, type specifiers
Receiver Count	num, type specifiers
Transmission S/N ratio	num
S/N receiving threshold	num
Channel	Capacitynum
Range	distance
Bandwidth	num

Capacity	Watt hour
Chargeable	no,externally, solar
Power profile/average draw	Voltage (low, medium, charged)

Number of Cores	num
Microcontroller(s)	num, function specifiers
ASIC(s)	num, function specifiers
GPU(s)	num, function specifiers
FPGA(s)	num, function specifiers

REFERENCES

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

[2] (2014, May) Introduction to the a* algorithm. [Online]. Available: https://www.redblobgames.com/pathfinding/a-star/introduction.html#greedy-best-first

Jackson Parker Currently a Masters Student in the ECE Department at UGA, working as a systems engineer in the Small Satellite Research Lab