Spring 2026: Human-Robot Collaboration and Companionship Lab Report

Jonathan Distler, Cornell MAE '27

***Abstract*—This report outlines my contributions to the HRC^2 lab as part of a research-for-credit initiative. My main contributions were the implementation of robot localization in parallel with a NAV2 stack native to the Stretch RE1, as well as a PDDL visualization tool showing dependencies between agents and their objectives. The goal of localization and the NAV2 stack was to have a robot test-bed that could carry out tasks after a brief planning phase with a human; with the system understanding its location in the environment, it could then create a plan to navigate around obstacles from point-to-point and carry outs its delegated tasks. The goal of the visualization tool was to help the human build a mental model of the tasks and their dependencies in tandem with the robot, such that the planning was more equitable and would lead to more efficient tasks.**

I. Introduction

Human-Robot Interactions involve understanding the complex interactions between humans and robots in shared environments, given unique constraints placed on both the human and robot agents (e.g. humans limited load bearing capabilities and robots limited manipulation capabilities). Having a mutual understanding of goals, roles, and task decompositions is crucial in successful teams. As is the case with human-human teams, communication plays a vital role in shaping each agent's mental models with respect to plans, goals, and constraints. In human-human teams, successful communication can be accomplished through verbal and non-verbal communication (e.g. eye contact) and the ongoing dialogue is a prerequisite for restrategizing throughout a task, and the idea is that invoking these dialogues with robots will lead to more successful teaming and performance among human-robot collaborators.

Given the prevalence of artificial intelligence (AI) across many domains of society, previous researchers have attempted to leverage Large Language Models (LLMs) to solve long-horizon tasks, often involving multiple agents [1]–[3]. Many studies have used LLMs as a stand-in for dialogue on behalf of robots in order to facilitate mutual understanding among agents [4], [5].

This project aims to use LLMs to mediate a long horizon task between a human and robot; delegating tasks among agents based on constraints (including willingness to complete a task on each agent's end) and goals for the task. The LLM will produce a plan for each agent, which will be converted in to a PDDL. The PDDL will then be solved for each agent, showing the step by step tasks for each agent, at which point each agent will attempt to complete their respective tasks.

This report was produced by Jonathan Distler
Written May 8, 2025

II. Research

A key component of this study was understanding the collaboration between robots and humans, and the development of a shared mental model among agents. This is accomplished via the delegation of tasks between each agent based on their respective physical constraints and environmental constraints, and in the case of the human agent, their respective wants. The interplay between each agent with their communication and updated and shared body of knowledge was an important aspect, as was translating those tasks into successfully completing a task.

The goal of the project was to take a natural language description and pass the description to an LLM. The LLM would then break the problem down into subgoals; for each subgoal, the agent (one of either a human or robot) is given the provided subgoal and turns it into a relevant PDDL with the help of a LLM. Although not implemented in our current project, with the newly devised PDDL setup (domain and problem file), a PDDL algorithm (e.g. fast-forward heuristic) would solve the PDDL problem and pass the output to the main agent to execute.

My work was to contribute to the robot platform that would take a PDDL and navigate to different locations, avoiding the error propagation and lack of localization that was prevalent in the hard-coded versions of past. This was successfully accomplished, but was not implemented with the rest of the stack, which included an aruco-alignment and grasp phase.

A. Localization

My first project that I worked on was the implementation of a localization method. First, I created a map of the room. After some trial and error, I started the `offline_mapping` mode for Stretch RE1, instantiated the robot's teleoperation, and then switched the robot to position mode so that the stretch driver (the internal mechanics that translate desired actions into movements) could properly control the robot.

bash

```
ros2 launch stretch_NAV2 offline_mapping.launch.py

ros2 run stretch_core keyboard_teleop

ros2 service call /switch_to_position_mode \
std_srvs/srv/Trigger {}
```

Then, I drove the robot around the room around twice. This particular step was relatively time intensive, and in the even the room's layout changed, then the localization wouldn't necessarily work as well, if at all - due to its internal SLAM localization running underneath the hood [8]. After I felt that the robot had covered much of the room, I saved the robot's

map with the following, which creates a folder to save the maps and then saves the map made (using LiDAR). After a lot of testing, I found it helpful to have barrier added to the room (and staying in place for all future iterations) to help ground the robot; in the example below 1 I used a board laid perpendicular to the room's back wall so that I could easily instantiate a starting location with a high level of confidence, and so that the SLAM architecture has a very obvious feature to begin detection around [9].

bash

```
source /opt/ros/humble/setup.bash
source ~/ament_ws/install/setup.bash
mkdir -p /home/hello-robot/stretch_user/maps

ros2 run NAV2_map_server map_saver_cli \
-f /home/hello-robot/stretch_user/maps/updated_lab_map
```

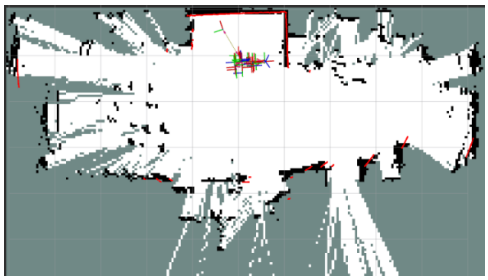


Fig. 1. LiDAR Map Representation of the Room

Then, to obtain the locations of other relevant positions throughout the room, I first initialized a starting location. First I placed the robot in a relatively open space in the room, near the aforementioned border, and measured its location from two easy to detect locations within the room (the back wall and the horizontal board). Then, I launched the *Stretch NAV2* package along with the saved map, *updated_lab_map.yaml*. In the RVIZ window that opens up alongside the navigation stack, I clicked "2D Pose Estimate" and then using the native measurement tool (alongside the known position of the robot in regard to the map) I placed the cursor at the location of the robot in the map's coordinate system.

bash

```
ros2 launch stretch_NAV2 navigation.launch.py \
map:=/home/hello-robot/stretch_user/maps/updated_lab_map.yaml
```

Next, I teleoperated the robot around the room to strategic locations and then used the following to determine its position in the map's frame, making note of the linear coordinates and the angular rotation of the robot, but not the covariance matrix. After that was done, the localization was complete.

bash

```
ros2 run teleop_twist_keyboard \
teleop_twist_keyboard --ros-args \
-r cmd_vel:=/stretch/cmd_vel

ros2 topic echo /amcl_pose
```

B. NAV2 Stack

For the NAV2 stack, I created a *.yaml* file named *locations* that stored the linear and angular positions of the for each

```
back_table:
  x: -1.6921562670651404
  y: -3.9190102024565774
  z: 0.0
  qx: 0.0
  qy: 0.0
  qz: 0.026681022308769006
  qw: 0.9996439981556229
```

Fig. 2. Example Object Position in .yaml File

intended location from the previous `amcl_poses topic` (Figure 2).

Then, I went into the `setup.py` file for the package and changed the `entry_points` to reflect the addition of waypoints:

setup.py

```
setup(
    name='stretch_init_pose',
    version='0.0.0',
    packages=find_packages(exclude=['test']),

    data_files=[
        ('share/ament_index/resource_index/packages',
         ['resource/' + package_name]),

        ('share/' + package_name, ['package.xml']),

        (os.path.join('share', package_name, 'config'),
         glob('config/*.yaml')),
    ],

    install_requires=['setuptools', 'pyyaml'],
    zip_safe=True,

    tests_require=['pytest'],

    entry_points={
        'console_scripts': [
        'initial_pose = stretch_init_pose.initial_pose_publisher:main',
        'goto_location = stretch_init_pose.goto_location:main',
        ],
    },
)
```

The pipeline is now as follows: first, the robot is homed to calibrate it. Then, the NAV2 stack is started, and after a short delay the robot is instantiated within the map. Finally, the robot autonomously navigates to a target location (e.g., *table_with_robot*).

bash

```
stretch_robot_home.py

ros2 launch stretch_NAV2 navigation.launch.py \
map:=/home/hello-robot/stretch_user/maps/finalized_map.yaml

ros2 run stretch_init_pose initial_pose

ros2 run stretch_init_pose goto_location table_with_robot
```

Ideally the robot will navigate from its start location to its goal position and maintain some level of localization. However, if the room is too cluttered, the robot may not be able to compute a valid navigation path to the goal (in this case, the */local_costmap/costmap* topic should indicate no valid information). Additionally, the robot has a tolerance for reaching a goal location, defined as a roughly 0.25 m radius around the target position (configured in *NAV2_params.yaml*). Reducing this tolerance further caused the robot to oscillate

around the goal, which is undesirable when integrated with a action-server architecture for executing full task sequences.

C. PDDL Visualization for Delegated Tasks

As a tertiary task, I worked on a visualization tool that would show the human the delegated tasks between them and the robot, ideally helping to build their body of knowledge on the dependencies and orderings of a PDDL output, which isn't necessarily temporally consistent, meaning it doesn't preserve the idea that the human agent and the robot can work at the same time in tasks that aren't contingent on one another (e.g. washing lettuce and cutting ham). The goal of this was to implement with the user-interface along with the LLM chatbot to help visualize the sequence of events, with dependencies highlighted in dotted arrows and each agent's action color-coated (Figure 3) [10].

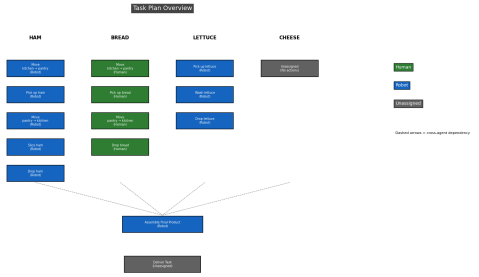


Fig. 3. Example PDDL Graph Showing Human and Robot Actions

III. Outcomes

As a whole, I was able to accurately localize the robot, get it to autonomously navigate from place-to-place with a high level of success, and then I was also able to create a real-time graph that shows delegated tasks between human and robot agents. I had also worked on trying to create a controller architecture that called different action servers (e.g. aruco-alignment, grasp, drop) that correlate with different actions in the domain.pddl. However, there were some issues with the stretch driver, as the NAV2 stack used a separate driver from the aruco-alignment driver (which included the RealSense camera), and that was causing issues when stringing together multiple tasks. However, this was solved by Timothy. Additionally, the graph has yet to be implemented in the UI for the project, along with the chat-bot, however, I hope to implement this at some point over the summer.

At the end of last semester, I wrote the following "I hope to implement the FunMap to robustify the pick-and-place tasks, making sure to avoid base contacts, allowing for more interesting human-robot interactions. I also plan on reaching out to my teammates more, so they can better help me understand the software aspect of the work. Similarly, I also plan on trying to stay in the loop with the other projects more to see if I can draw parallels across the research projects, and also improve my knowledge in the field of human-robot collaboration." I was able to robustify the pick-and-place task with the NAV2 stack, which functions very similar to the FunMap [11]. I was also able to help with another project and learn about BORIS software and about codifying qualitative actions for user-studies.

A. Roadblocks

Technically, the biggest issue came from stringing together modular actions like navigation and alignment with a controller architecture. I struggled to implement it modularly. Although I was able to navigate, then align, then grasp it was not in a suitable manner to the PDDL, as they weren't called as individual actions servers, but rather as one action server with all three tasks, which prevented implementation from a true 1:1 relation with the PDDL output which operated one action at a time, and not necessarily in the same order every single time as I had naively assumed.

B. Future Improvements

Going forward, I hope to implement the NAV2 stack with the preexisting architecture of the action servers rather than coding the locations, which is occurring in our current demo. I also hope to implement a more sophisticated mode of failure-detection rather than relying on humans to manually encode a failure; I believe a perception stack could be helpful, and would provide me the opportunity to learn more sophisticated means of computer-vision.

References

[1] H. Wang, K. Kedia, J. Ren, R. Abdullah, A. Bhardwaj, A. Chao, K. Y. Chen, N. Chin, P. Dan, X. Fan, G. Gonzalez-Pumariega, A. Kompella, M. A. Pace, Y. Sharma, X. Sun, N. Sunkara, and S. Choudhury, "MOSAIC: Modular Foundation Models for Assistive and Interactive Cooking," arXiv:2402.18796, 2024.

[2] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, "ProgPrompt: Program Generation for Situated Robot Task Planning using Large Language Models," *Autonomous Robots*, vol. 47, pp. 999–1012, 2023.

[3] D. Bai, I. Singh, D. Traum, and J. Thomason, "TwoStep: Multi-agent Task Planning using Classical Planners and Large Language Models," arXiv:2403.17246, 2024.

[4] J. Ivey, S. Kumar, J. Liu, H. Shen, S. Rakshit, R. Raju, H. Zhang, A. Ananthasubramaniam, J. Kim, B. Yi, D. Wright, A. Israeli, A. G. Møller, L. Zhang, and D. Jurgens, "Real or Robotic? Assessing Whether LLMs Accurately Simulate Qualities of Human Responses in Dialogue," Proc. ACM Conference, 2024. DOI: 10.1145/3610978.3641112. [Online]. Available: https://doi.org/10.1145/3610978.3641112

[5] A. Addlesee, N. Cherakara, N. Nelson, D. H. García, N. Gunson, W. Sieińska, M. Romeo, C. Dondrup, and O. Lemon, "A Multi-party Conversational Social Robot Using LLMs," inCompanion of the 2024 ACM/IEEE International Conference on Human-Robot Interaction (HRI '24), 2024, pp. 1273–1275. doi:10.1145/3610978.3641112.

[6] "Linear Interpolation Formula," Cuemath. Available: https://www.cuemath.com/linear-interpolation-formula/ (accessed Dec. 6, 2025).

[7] stretch_funmap, Hello-Robot Documentation. Available: https://docs.hello-robot.com/0.2/stretch-ros/stretch_funmap/ (accessed Dec. 6, 2025).

[8] Hello Robot, "stretch_NAV2 README," GitHub Repository. Available: https://github.com/hello-robot/stretch_ros2/blob/humble/stretch_NAV2/README.md (accessed Dec. 6, 2025).

[9] MathWorks, "Simultaneous Localization and Mapping (SLAM)." Available: https://www.mathworks.com/discovery/slam.html (accessed Dec. 6, 2025).

[10] Jonathan Distler, "pddl_graph_output.py," GitHub Repository. Available: https://github.com/JonathanDistler/HRC2SP26/blob/main/Resources/PDDL_Resources/pddl_graph_output.py (accessed Dec. 6, 2025).

[11] Hello Robot, "Stretch FUNMAP Documentation." Available: https://docs.hello-robot.com/0.2/stretch-ros/stretch_funmap/ (accessed Dec. 6, 2025).