Collaborative Robotics Final Group Project Report

Bianca Yu, Connor Tingley, Mateo Massey, Abhyudit Manhas

CS 339R: Winter Quarter 2023

Website link:

https://sites.google.com/stanford.edu/me-326-final-project-team-2/home

Contents:

Problem Statement

Robots outperform humans on tasks that involve repetitive, precise, and dangerous work. They can achieve consistent accuracy and speed without breaks or rest. As a result, industrial robots thrive in static environments, and are often used for assembly line work, welding, and material handling. They operate at high forces and torques, so they can usually only work separately from humans for safety.

On the other hand, humans are more creative and flexible. People are better equipped to handle unstructured tasks in dynamic situations and environments. As a result, humans outperform robots on tasks that require critical thinking, communication, and physical dexterity.

Many tasks and environments require a combination of repetitive and creative elements, creating a need for human-robot teams. Safe interaction is then an issue of paramount importance, since collaborative robots work in close proximity to humans. Collaborative robotics has the potential to address several challenges, including an aging workforce, search and rescue, and specialized agile manufacturing.

This project involves the collaboration of two robots to gather resources without digital communication. During runtime, a configuration file will describe the number and configuration of the blocks, including the designated block colors for each team, and the number of blocks, by color, per station, but without specifying their location. The arena bounds will be described by a tag located at the arena center. The robots must collaborate to determine when and where to gather each resource, despite the absence of digital communication.

The rules for the task are as follows: Each block present at a station will earn 25 points, but every additional block at a station will deduct 2 points. Only one block color is allowed per station, and teams have the freedom to choose which resource type to use. A block can only be counted at one station. During gametime, only the designated team can add or remove blocks from a station. Each team has their own blocks, and moving the other team's blocks into a station will deduct 5 points, as well as additional points for any extra blocks. The game time is limited to

10 minutes, and both teams work simultaneously. Finally, teams cannot preemptively agree on strategy, such as going clockwise, to ensure fair play.

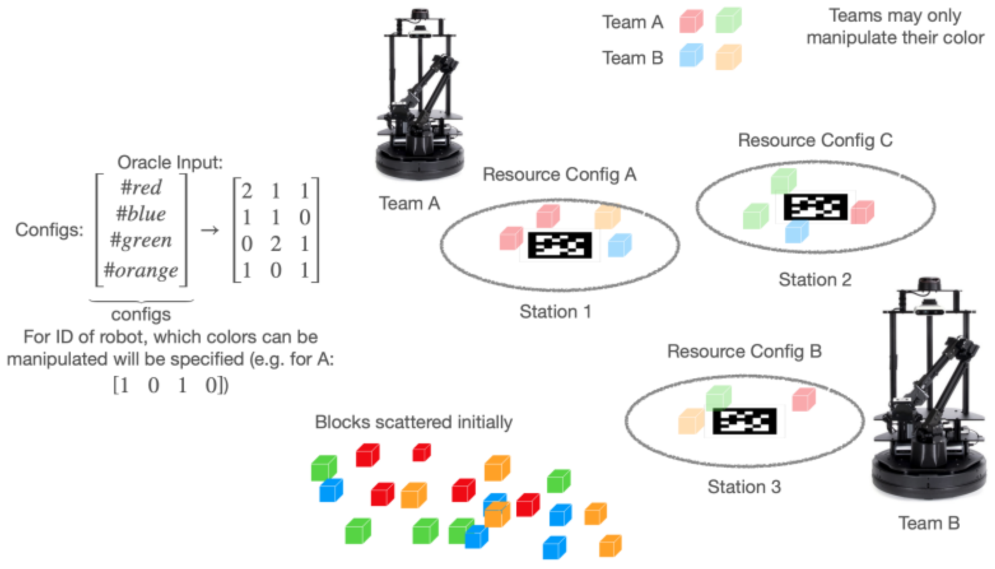
Team A
Team B
Teams may only manipulate their color
Resource Config C
Oracle Input:
Resource Config A
Configs:
#red
#blue
#green
#orange
configs
Team A
Station 2
Station 1
For ID of robot, which colors can be manipulated will be specified (e.g. for A: [1 0 1 0])
Resource Config B
Blocks scattered initially
Station 3
Team B

Fig 1. Game Mechanics for Collaborative Resource Gathering by Two Robots

Overview

Since there is no digital communication allowed, our strategy needed to be compatible with a wide variety of collaboration strategies used by the other team's Locobot. In order to collaborate we attempted to deduce with the other robot's world model through observation of their block placements.

In order to build a working prototype in a short span of time, we first aimed to construct the minimum viable product (MVP) in Gazebo. Given the constraints laid out in the task, the MVP for this project involves a Drive Controller (to move the robot), an Arm Controller (to pick up and drop off blocks), a Path Planner (to avoid obstacles), and an Odometry module (to determine the Locobot's position). This simulation MVP was then translated into the physical world.

By week seven, we achieved the MVP and moved on to task collaboration in the physical world. By the end of week nine, we were able to run the entire task on the physical robot, including scanning the environment for a depth reading, selecting an accurate block target, and following a pre-planned trajectory to the block.

The main challenges we faced in Gazebo were unliftable blocks, slow simulation time, and multi-robot simulation. Despite our best efforts with the Arm Controller module, it was not possible to pick up the blocks. In the end, it turned out that the mass and inertia were set to infeasible constraints in the block urdf files. In order to solve this issue, we decreased the block weight from 0.5 kg to 0.005 kg. The slow simulation time was solved by relying on native Ubuntu computers. Finally, during the last week, we were able to initialize multiple robots in the simulation with individual rostopics and nodes running our system. However, full setup was unsuccessful as robot state and joint state information was undetectable by Gazebo, likely due to robot description discrepancies.

On the physical Locobot, we faced noisy odometry readings, goal overshooting, and gripper issues. If there were multiple Locobots present on the field, the odometry rostopic would provide a noisy reading. It turned out that we combined the rostopics for all three Locobots into one channel, and after isolating each, we were able to reduce much of the noise. However, the camera system continued to jump between Locobots while generating pose, so we ended up using only one Locobot at a time. For the second challenge, our Locobot would overshoot the goal position, causing it to run into the target block. This was caused by the proportional control system. In order to fix overshooting, we optimized our gain values. Finally, we encountered many gripper issues while working with the Locobot. We discovered that MoveIt was the main culprit, and set about integrating the Interbotix gripper module.

Methods

Level of Autonomy

Mukherjee et al. [1] introduces the following levels of interaction between humans and robots: fully programmed, coexistence, assistance, cooperation, collaboration, and full autonomy.

Autonomy refers to the robot's ability to make independent decisions based on its surroundings, the state of the task, and the state of the human partner. In this task, we are interested in robot-robot collaboration, which could then be extended to human-robot collaboration in future work.

Various aspects of autonomy were addressed in our system design — perception, planning, and control. Perception of the state of the robot's immediate surroundings utilized RGB, depth camera data, and Locobot pose in order to populate an internal occupancy grid (see the "Occupancy Grid" section below for more information), to represent spatial information about objects in the environment. This occupancy grid informs a path planning module (see "Path Planner" below) that creates a collision-free path towards a goal location using A* or Rapidly-exploring Random Tree algorithms. Finally, control of robot motion and arm movement when picking up blocks were designed to ensure that goal locations were reached and that blocks could be retrieved (see "Drive Controller" and "Arm Controller").

Approach to Collaboration

Bauer et al. [2] surveyed the communication methods used to form joint consensus between humans and robots, including gestures, actions, haptic signals, speech, and physiological signals. However, since we cannot assume a common strategy with the other robot, we cannot use these communication protocols. Bauer et al. also suggests proactive task execution to communicate intention. Legible motion [3] and emphasized actions like withdrawing a cup in a cup stacking task can convey meaning to a teammate. For this task, proactive task execution is the pattern of blocks placed by our teammate in each station. From the placement of blocks by the other team, we can extrapolate the probability or their belief and act accordingly, choosing the move with the best expected value for the time invested. This approach is described in further detail in the "Station Tracker Module" section. According to Tabrez et al., a shared mental model like this one can improve fluent behavior, adaptability, and trust between teammates [4].

Key Algorithms for Collaboration

A*

The A* algorithm finds the shortest path between two points in a grid. From the grid, it tracks two lists of nodes: an open list containing unexplored nodes, and a closed list containing explored nodes. For any given node, the cost is the sum of the arrival cost and an estimate of the cost to reach the destination.

At each iteration, the node from the open list with the lowest cost is added to the closed list. Then, it calculates the costs for the neighboring nodes. If a neighboring node is not in either list, it is added to the open list. If it is already in the open list, then its cost is updated if it the new path is cheaper than the previous path. If it is already in the closed list, the node is reopened if the new path is cheaper than the previous path.

There are two optimality conditions: the destination node is added to the closed list or the open list is empty. If the open list is empty, there is no path to the destination. We can reconstruct the final path by tracing each parent node, starting at the destination node.

Rapidly-exploring Random Tree

Rapidly-exploring Random Tree (RRT) is also used to generate a probabilistic roadmap from an occupancy grid. First, it initializes a tree data structure with a single node, representing the starting position of the robot. Next, it takes a random sample from the occupancy grid within the given bounds. The nearest node in the tree to this randomly sampled node is found by calculating the cost between the random node and all existing nodes in the tree. The nearest node is connected to the random node via a trajectory. If this trajectory collides with an obstacle, then we skip to the next iteration by sampling a new random node. If not, we can select a new node by moving an incremental set distance away from the nearest node along this trajectory. This new node is added to the tree. We then repeat this process, sampling a new random node.

This process continues until a termination condition is met. If the new node is within a set radius of the goal node, then we add the goal node to the tree and the process is complete. If we reach the maximum number of tree nodes, we will terminate without finding a solution. We can extract the potential target path by traversing the tree from the goal node to the root node. We are guaranteed to avoid obstacles along this path, but it is not necessarily the optimal path. RRT is a probabilistically complete algorithm, meaning that as the max number of nodes approaches infinity, the probability of finding a collision-free path between the starting and goal configurations approaches 1.

System Architecture

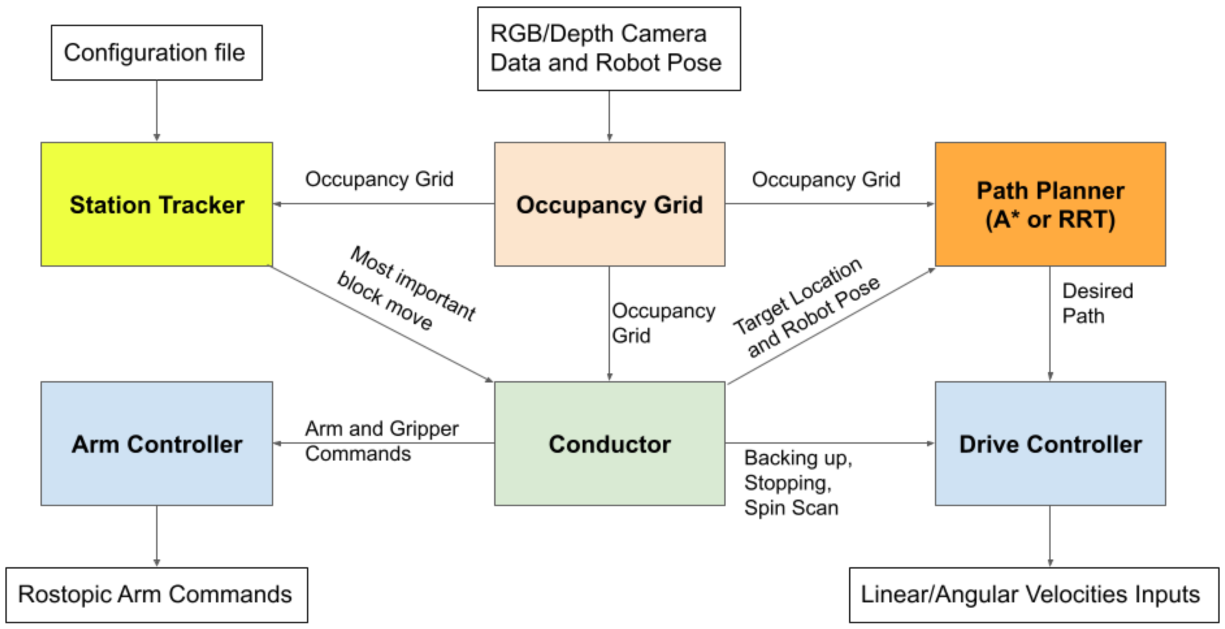

Fig. 2: Overview of System Architecture

Conductor Module

The conductor module is a state machine that orchestrates the other systems and decides what the next task is. The first input is the occupancy grid, which represents the environment and identifies the locations of the blocks and other obstacles. The second input is the highest priority block move. The module then passes the relevant information to each subsystem. The conductor also orchestrates the state of the robot, telling the drive controller when and where to drive and the arm controller when and where to pick and place.

Occupancy Grid Module

The occupancy grid module allows the Locobot to track observed objects in a world space grid. The main advantage of using an occupancy grid is that it enables negative sensing, which means that measurements, where a block is not seen in a location, can lower the belief that a block is there. This would be much harder to implement if the detection happened in camera space. Additionally, the module can remember the state of blocks near the station, allowing computation of what blocks need to go where even if the station is not in line of sight. By taking the position-weighted average of neighboring cells, the module can get a good idea of where the center of a block is within a cell by fitting Gaussian distribution with OpenCV blob detection. This allows us to reduce the cell size for faster computation without losing accuracy when we go to pick up the block.

The module takes RGB, depth camera data, and Locobot pose as inputs and outputs the occupancy grid. The transformation between camera UV space and world space is done using vectorized numpy equations (no python for-loops) for speed. The occupancy grid classifies objects by color and by height. If an object falls into the red, yellow, green, or blue HSV ranges and is cube height (1-2 cm tall) it is classified as a cube. The height requirement was important for differentiating between the cubes and the colored tape on the floor of the lab. Objects taller than 3cm are classified as obstacles to be avoided. The grid encompasses the playable area, with a grid resolution of approximately 1 cm. This resolution can be decreased for smaller blocks, but this will incur more computation time.

Path Planner Module

The Path Planner module utilizes the A-Star algorithm or the Rapidly-exploring Random Tree (RRT) algorithm in combination with the Occupancy Grid to plan the optimal path to the target location. The main goal of this module is to avoid any obstacles that may be present.

The module takes in the target location, robot pose, and occupancy grid as inputs and outputs the desired path. Both A-Star and RRT use the occupancy grid to compute the shortest path to the target location while planning around potential obstacles. It defines safety zones to ensure that the bot does not collide with blocks or the environment during its travel. It then ensures that the planned trajectory does not intersect any of these zones. The planner also takes into account the robot's current position and heading, and will continuously replan the path every few seconds to account for any moving obstacles (such as other robots) or errors in path following.

Station Tracker Module

The Station Tracker module is responsible for keeping track of the resources in each station, estimating the other robot’s belief, and determining the next resource that needs to be moved. By analyzing the occupancy grid, we can determine the count of each resource in every station. From these counts, our belief of their belief (a meta-belief if you will) can be estimated. The closer a station is to being in a particular state, the higher the likelihood that the other team has been placing blocks there, so our system will assign it a higher probability. The system will compute the expected score gain of each move with these probabilities and choose the move with the largest expected value per second. In other words, the move with the largest expected score gain (conditioned on the meta-belief) divided by the amount of time that it would take to complete the move. This way moves that require less effort but yield the same amount of points are prioritized. By approximating the other team’s belief using the board state, this method implicitly takes the initial state of the board into account. The initial state acts like a prior to the meta-belief; if we assume that the other robot is also a competent agent, we can guess that they are more likely to choose a configuration that is closer to being done. If the other robot decides to follow our lead, this prior also helps our robot carry out a solution with fewer moves, as configurations with closer station completions are assigned higher probabilities and thus higher expected scores.

The station tracker module takes in a configuration file, occupancy grid, and station identification (which may be provided by a collaborator) as inputs, and outputs the most important block move - i.e., which block needs to be moved next and where it needs to be moved from and to (outside the field, Station 1, Station 2, etc.). Once the module has determined the highest priority block move, it outputs it to the Conductor module, which then distributes this information to the other subsystems (such as the Path Planner, Drive Controller, and Arm Controller) to carry out the task.

Arm Controller Module

The Arm Controller module receives arm commands from the conductor module and generates rostopic arm commands to be executed. It takes into consideration the kinematics of the robot arm and generates joint angles to achieve the desired end-effector pose.

Drive Controller Module

The Drive Controller module takes in the desired path from the Path Planner and outputs the linear and angular velocities needed to follow that path. It uses the method from Homework One to control the Locobot's movement and maintain this desired trajectory. The control point P is set 10cm in front of the center of the Locobot, such that the arm and gripper can easily place or pick up a block.

Dependencies and Development Environment

We created a launch file that included the arm controller, namespace, and conductor. The conductor can be run with "run_on_robot=true". If false, it is run on Gazebo. We used the commands "catkin_create_pkg me326_final_project std_msgs roscpp rospy", "catkin build", and "source devel/setup.bash" to make the package accessible in the workspace.

GitHub Repository

All code for this project can be found here:
https://github.com/ConnorTingley/ME326-FinalProject

Results

Video Demo and Description

https://www.youtube.com/watch?v=F3Mx3NWDFKo

This video depicts the collaborative capabilities of our system. The robot begins by scanning the environment to learn the positions of the blocks, the status of the stations, and potential obstacles in the space. None of the stations have cubes at the start, so the station tracker has no preference for which station is assigned to which location. Given no other information, the shortest and highest point-gaining move is chosen, and the robot moves the green block to the nearest station. While the robot is making this move, the collaborator moves the blue block to another station. The move is recognized by the station tracker, which increases the probability that the collaborator has chosen this station to be either the red-blue station or the blue-yellow station. Though the system cannot be sure which of these options the collaborator believes, this move increased the probability of the blue-red station assignment, which bumps the expected value of moving the red block there. The human collaborator then decides to disagree with the robot and move the blue and yellow blocks to the station that the robot has already assigned. Luckily, our system is a flexible collaborator that changes its beliefs based on the actions of others, even if the actions are sub-optimal. At this point, the system has strong confidence that the blue-red station is correct and that the station that now has 3 cubes is not the green station. It decides to move the green cube to the nearest empty station, completing the resource goal.

Multi-robot simulation

We attempted to spawn multiple robots in simulation that could each run our system modules and collaborate to accomplish their block-sorting tasks. This involved creating new files for 1) launching each robot's namespaces, 2) setting up rviz robot descriptions, and 3) launching each robot in Gazebo. The interbotix many_xslocobots example was used as a model to structure our own code, however it quickly became apparent that launching multiple robots in the same simulation was much more complex than expected, and more time would be required to fully understand the framework for fully defining and controlling each robot independently, nonetheless adapt this framework for our applications.

The current state of the many_robots.launch file first defines multiple variables (most importantly, robot_name_1 and robot_name_2), launches an empty Gazebo world, launches each robot into Gazebo with a unique starting pose and position, launches rviz, then launches a conductor node in each robot's namespace.

We were able to launch two robots with their own rostopics and Conductor nodes running in respective namespaces, visible in Gazebo with distinct starting positions (Fig. 3). Unfortunately, errors with the setup of robot description files likely contributed to undefined robot and joint states which prevented control of the simulated robots. This was evident by the white “ghost” robots in Gazebo, and unresolved, missing reference frames that inform robot’s joint states.

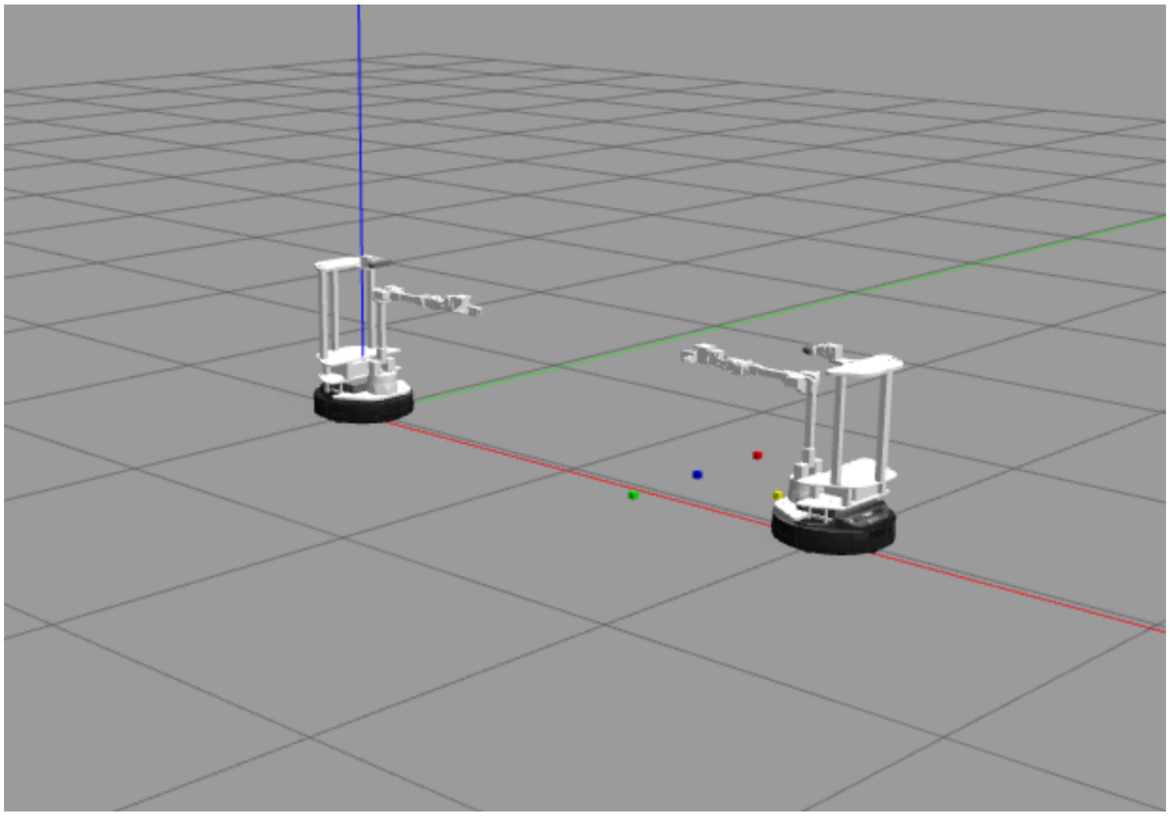

Fig 3: Multiple Locobots (with undefined descriptions) simulated in Gazebo task environment

Weekly Project Milestones

During the second and third weeks of the quarter, our team completed a variety of tasks to set up the necessary infrastructure for our project. We started by choosing the course laptop as the remote computer, because our remote Ubuntu computers could not access the ARMLab 5G network. We then proceeded to download Linux, ROS, vim, net-tools, terminator, python3, scipy, the Locobot build, and python_catkin.

To ensure smooth communication between our robot and the computer, we added Locobot addresses to our ~/.bashrc ROS URI and successfully SSH’d into our robot. Next, we built the course workspace with catkin and ran the environment. By the end of week three, we made the launch shell scripts executable and ran the full A to B example with RViz. The culmination of our efforts was when we were finally able to move the real robot, marking a significant milestone in our project.

During week four, we set up a group Github account to collaborate more efficiently on the project. We then pulled the most recent version of the course repository to our original folder to ensure we were working with the latest updates. Next, we successfully spawned blocks in the Gazebo simulation using spawn_cube.launch.

By week five, using our block computer vision script from homework two, we were able to target a block and get its position through the depth reading. We ran an example that used MoveIt to move the arm down and point the camera at the spawned blocks in Gazebo. We also started working on the occupancy grid, mapping the block color pixels to a corresponding depth using matching_ptcld_serv and pix_to_point_cpp. At first, we assumed that there was only one block of the given color in the field of view. During week six, we started to think about how to deal with adding many blocks to the occupancy grid. Finally, we were able to move the robot towards a specific block in the Gazebo simulation. However, it was far more difficult to pick it up.

At this point, our efforts began to focus on running our scripts on the physical Locobot. During week six, we started working on retrieving the odometry of the Locobot from the motion capture system. We decided to use MoveIt to control the physical Locobot's arm, gripper, and camera angle. During this process, we troubleshooted many MoveIt errors by checking the published Locobot topics. We created an arm controller module that could control the pickup and drop-off functions of the robot's arm.

During week 7, we set the maximum velocities (~0.3 m/s) in the code and ensured that the arm returned to sleep pose before killing the code. Next, we developed a station tracker module that estimates the probability associated with the other team's Locobot station beliefs. These probabilities are used to determine the best block move. To integrate our new modules into the current path planning system, we created a launch file that included the arm controller, namespace, and conductor.

Weeks 8 and 9 were spent improving the modules and debugging the physical robot. One significant accomplishment during week nine was visualizing multiple robots at once with our custom many_robots.launch. This was a crucial step in our project as it could allow us to simulate collaboration between multiple robots running our algorithm in the future.

Future Steps

If we had more time, we could do the following:

1. We could create a wrapper node that acts as an intermediary between the robot and our code. Instead of each module needing to account for the run_on_robot flag, this node could put the information into a standardized format regardless of if the robot is real or in simulation. This node could be run on the robot allowing the use of the Interbotix python package.
2. We could add redundancy to the arm controller module. If both the drive controller and the occupancy grid are accurate, the arm controller can simply move the arm to the target point P to grab or release blocks. However, if one or both of the systems are not accurate, the arm controller would fail. Thus, we can add redundancy by using blob detection directly on the camera feed in addition to the occupancy grid to locate the block and tell the arm where to go in robot-space, bypassing the need for highly-accurate driving and occupancy detection.
3. We could integrate our controller into the multi-robot simulation. By having multiple robots running in parallel, we can test for unaddressed edgecases in our collaborative method.
4. Once the previous steps are implemented, we could run physical tests using two Locobots at once, to test for sim-to-real errors in our collaboration.
5. Incorporate machine learning techniques to optimize the robot's strategy during gameplay

Possible Applications

This task could be extended into a human-robot experiment by making the Locobot collaborate with a human instead. Our collaboration strategy would still be effective, because we mainly observe proactive task execution.

The project could be further developed to include more than two robots, which would require more complex coordination, but would also allow the completion of more significant tasks. Despite being relatively simple when compared to larger lifeforms ants are able to accomplish great feats through collaboration. We could take inspiration from ants and deploy many simple robots instead of creating single complicated machines.

In the real world, this project could be applied to disaster zones where communication is difficult or impossible. Like this task, the robots would work together to search for and rescue victims without digital communication.

Contributions

Bianca Yu

I'm Bianca, a co-term student in Bioengineering. For this project, I worked on our initial testing with the physical locobot, writing a single launch file that could launch either the simulation or real robot, and multi-robot simulation. The main challenge I faced was the multi-robot simulation, a task that seems straightforward but proved to be more complicated under the hood. I attempted to reverse-engineer example code from interbotix, however each file referenced another file which referenced another file… and despite my best efforts (and TA efforts) to adapt that code for our own project, there was not enough time to fully functionalize multi-robot simulation. However, through working through these challenges I gained a deeper appreciation for the complexity behind the ROS network and the information flow that must be orchestrated in order to simulate robot perception and motion.

Connor Tingley

I'm Connor, a co-term in Mechanical Engineering. I worked on designing, implementing, and troubleshooting many of the modules including the conductor, occupancy grid, station tracker, and drive controller. There were many challenges developing such an integrated system of moving parts from scratch, but the hardest part for me was troubleshooting the Interbotix packages to get the system functioning on the real robot. This project taught me about integration testing, creating robust autonomous systems, and designing algorithms that intuit and act on the belief states of collaborators.

Mateo Massey

My name is Mateo and I'm a co-term student in Electrical Engineering. I was responsible for the unit testing module and the Rapidly-exploring Random Tree path planning algorithm. I also set up launch files with the appropriate Locobot flags. In general, most time was spent troubleshooting technical issues with the physical Locobot over the last few weeks of the quarter. I learned a lot about ROS topics, path planning algorithms, and odometry pose transforms.

Abhyudit Manhas

I'm Abhyudit, a Masters student in Mechanical Engineering. I worked on the conductor, arm controller, drive controller, and path planner modules, and was also responsible for implementing the A* planning algorithm. For me, the most difficult part was transferring results (like moving the arm and planning collision-free paths) from simulation to the actual locobot. However, this project gave me a great opportunity to learn more about ROS, MoveIt, and work with real robotic systems.

References

1) D. Mukherjee, K. Gupta, L. H. Chang, and H. Najjaran, "A Survey of Robot Learning Strategies for Human-Robot Collaboration in Industrial Settings," Robotics and Computer-Integrated Manufacturing, Volume 73, 2022, 102231, ISSN 0736-5845.
2) A. Bauer, D. Wollherr, and M. Buss, "Human–robot collaboration: a survey," International Journal of Humanoid Robotics, vol. 5, no. 01, pp. 47–66, nov 2008.
3) A. D. Dragan, S. Bauman, J. Forlizzi, and S. S. Srinivasa, "Effects of Robot Motion on Human-Robot Collaboration," in 2015 10th ACM/IEEE International Conference on Human-Robot Interaction (HRI), 2015, pp. 51–58.
4) A. Tabrez, M. B. Luebbers, and B. Hayes, "A Survey of Mental Modeling Techniques in Human–Robot Teaming," Current Robotics Reports, vol. 1, no. 4, pp. 259–267, Aug 2020.