Team Members:

- Nathan Kochera (nako8494@colorado.edu)
- Hailey Kellackey (hailey.kellackey@colorado.edu)
- Josh Truong (jotr4800@colorado.edu)

Hephaestus Final Project Report

- Robot Manipulations:
    - Added two cameras to the robot, one on each side and a multi sense camera on top. More details in the readme
- Github link:
    - https://github.com/josh-truong/Hephaestus
- Number of cubes collection automatically: 0
- Video Link:
    - https://drive.google.com/file/d/1vZqwvpRtVIE6nupG80o1Q6wOBq9yKZ0P/view?usp=sharing

Deliverables:

Mapping - Team Lead: Josh

Final Tier: Autonomous Mapping

We completed autonomous mapping using an algorithm which was similar to RRT with added object detection. The mapping portion of our robot begins with generating a random “goal” point. RRT, then, is run to find a path between the robot’s current position and and this goal. The robot begins to follow this generated path, scanning ahead of it for obstacles. If one is reached, the robot stops, marks that location on the map, and recalculates the path using RRT. Most of the difficulties in implementing this portion of the project came from not hitting objects in the early stage of mapping. Because the mapping is not yet completed in this stage, the robot does not know where obstacles are and is prone to running into them. We fixed this problem of obstacle avoidance by

adding a lidar sensor to the robot and also by having it reverse its direction when it got too close to obstacles. Map filtering was also difficult because they tend to accumulate bogging down rrt when we convolve the map.

Localization - Team Lead: Josh

Final Tier: WeBots GPS/Compass

We implemented odometry for our localization tier. We did not have time to implement SLAM. Our implementation of odometry very closely resembled what we did in our labs. The localization class includes all of the functions involved with computing the robot's location in world coordinates. The update_odometry() function calculates the change in the robot's location after each time step.

Computer Vision - Team Lead: Hailey

Final Tier: Color Blob Detection

Our initial goal for computer vision was color blob detection. We completed this goal and chose not to go any further due to time constraints. The implementation of this portion of the code was quite straightforward. We implemented it in a way which was very similar to homework 3. It was, however, difficult to take the next step, from blob detection to world coordinates.

Planning for Navigation - Team Lead: Nathan

Final Tier: RRT with Path Smoothing

We implemented RRT with path smoothing for our robot's navigation system. Once computer vision locates all of the blocks, we run the RRT with path smoothing algorithm to plan a path from the current location to the goal point. The code for our planning implementation is contained in the Planning class. This class contains functions which can perform all of the tasks associated with planning. This includes the RRT algorithm, a path smoothing function, as well as a host of functions we used to implement RRT.

The RRT algorithm is also used for our autonomous mapping. The path smoothing uses splprev(a method of b spline).

Manipulation - Team Lead: Nathan

Final Tier: Inverse Kinematics

We implemented inverse kinematics, as we did not have time to implement task-level planning. The implementation for this portion of the controller is found in the manipulation.py file. This class contains all of the functions we created to handle certain common actions the robot might need to perform. We have moveArmToCube(), grabCube(), and setDrivingState() among others. All of these functions use an API we found, ikpy.chain, to perform the bulk of the calculations. The function, ikpy.chain.inverse_kinematics(...) in particular was used quite frequently. This program receives a goal position and orientation and calculates the joint positions required to reach that goal.

Overarching Functionality

This is where most of our issues arose. Each individual segment of our code is working correctly, as you can see in our video, however, we had a tough time fitting these pieces together. We successfully completed autonomous mapping. We tried to identify landmarks from our map to establish waypoints through the aisles to find cubes, but we were unable to make this work correctly. Due to time constraints, we hardcoded values for the waypoints and continued with blob detection. We then manually navigate to a cube and pick it up.