Final Project Report

Christian Lee: chle8404@colorado.edu
Sean Shi: sesh9096@colorado.edu
Nicolas Perrault: Nicolas.Perrault@colorado.edu

Github repository: https://github.com/nipe1783/CSCI_3302_Final_Project

Mapping	Tier	Points -- 12
Manual	1	6
Autonomous	2	12
SLAM (Manual/Autonomous)	3	18

Localization		12
WeBots GPS/Compass	1	8
Odometry	2	12
SLAM / MCL	3	18

Computer Vision		18
WeBots recognition API	1	6
Color blob detection (HW3) (Recognition on Camera but use only color data, not recognition ID to identify which block obtained from webots API)	2	18
Machine / Deep Learning or any kind of object localization	3	28

Planning for Navigation		14
Teleoperation	1	5
A*	2	8
RRT	3	14
RRT w/ Path Smoothing	4	18

Manipulation		24
Trajectory: Hardcoding in Joint Space	1	12
Teleoperation in Cartesian Space (requires IK)	2	24
IK	2	24
Autonomous: Task-Level Planning + Obstacle Avoidance (IK + Hardcoded Waypoints)	3	30

Total Points (in %)	100
Objects	20
Completing Tiers	80
Bonus Objects Points (4 pts each object)	8
Bonus Tier Points	32
Maximum Points	**140**

Deliverables/Implementation Plan

Cubes acquired: 9 Yellow

Note: LiDAR sensor is translated 0.5 meters up to prevent it from hitting the ground all the time.

- **Computer Vision System – Lead: Nicolas Perrault, Other contributors: Sean Shi**

For computer vision, we used a combination of color filtering and blob detection using the open-cv library. We then used the centers of blobs to filter the webots recognition api objects and calculated their positions in the world. Calculation of object positions from webot's recognition api was done by Nicolas perrault and color blob detection and adding of points to the goal queue were implemented by Sean Shi. See computer_vision.py. We did encounter some issues with computer vision early on in the lab. We were not sure how to convert the blob locations into environment 3D locations, and after 2 weeks of asking on Piazza the requirements were finally changed so the Webots location API was allowed.

- **Mapping - Lead: Nicolas Perrault, Other Contributors: Sean Shi**

Autonomous LiDAR mapping. Implementing the same method used in Lab 5 to generate a map based off of lidar and robot positions. We did face some challenges with this as originally our robots lidar was tilted forward due to the cart being there. We(Nicolas Perrault) fixed this by moving the sensor up. Another issue we encountered was with translating the Lab 5 code to work with this project. Due to the different coordinate axes being used we had to do some translation in order to properly map the environment. Lastly, the environment not being square provided some additional confusion as originally our map was appearing slanted due to the environment being rectangular.

Another problem we encountered was that the robot had no way to explore or keep track of points it had seen. Our (Sean Shi's) solution was another map we called seen which using cv2.line and the existing lidar mapping, would draw all points which the robot had seen and were within lidar distance. Then the robot would always map points which it had seen before and during exploration when generating a path with the modified RRT*, it would return that the position was a goal if it was unseen.

- **Localization - Lead: Nicolas Perrault, Other contributors: Christian Lee**

We used odometry from last lab combined with the webots compass and gps to correct for errors in odometry due to the unmaneuverable nature of the robot. We decided to have the origin in the upper left corner with x being down and y being right. Webots compass and gps and coordinate axes were created by Nicolas Perrault. Condition checking for odometry was added by Nicolas Perrault and Christian Lee. Using the odometer without the GPS system proved to be very difficult as error would accumulate making the other tasks perform worse.

- **Manipulation - Lead: Sean Shi, Other contributors: Nicolas Perrault**

We implemented inverse kinematics with arm using ikpy. To do so, We (Sean Shi) first pruned unnecessary links from the given *robot_urdf.urdf* file and the resulting file is in *arm.urdf*. Some implementation challenges we faced for manipulation were that the initial positions of some joints could not be zero so an initial position had to be passed to the inverse_kinematics

function. An additional issue we encountered was that the gripper would often reach a position, but be at an angle which would not allow it to pick up cubes. This issue was solved by Sean Shi with the addition of 2 rotation matrices to calculate the position the hand would need to be at.

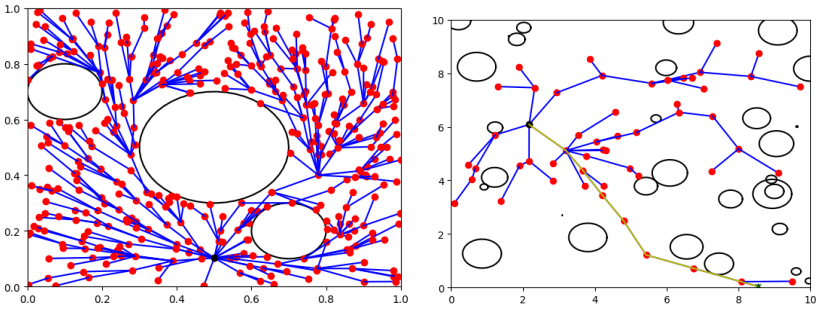
The biggest challenge we faced was that there appeared to be some discrepancies in the position of the objects as calculated by the camera and arm. This led to the arm going to a position which was slightly off from the position of the cube. This issue was made worse by the high need for precision because the cubes were only slightly smaller than the gripper and if the position was incorrect, the gripper would push rather than pick up cubes. This produced errors which were tested and accounted for by Nicolas Perrault.

- **Planning for Navigation - Lead: Sean Shi**

Navigation was implemented using an adaptation of RRT*, adapted from Homework 2 RRT, see planning.py. We used the RRT* algorithm for exploration where the robot would keep track of places it had seen and if no objects were found, would randomly grow a tree with RRT* until an unseen point in the map was reached. When an object was discovered, it would be added to the list of objects(*goal_queue*) and if not already navigating, the robot choose the nearest object from this queue and use RRT* to generate a path to it.

One challenge we encountered while working on navigation was that the robot would be unable to perceive changes in its environment which would potentially invalidate the current path. A potential solution we tried was using a LiDAR sensor to detect if the robot was too close to an obstacle. However, this was ineffective as the sensor was unable to check all possible collisions and would continue to show collisions even after a path was recalculated. Our solution for this problem was to use convolution to draw out all points the robot would occupy on the path and compare it to the map to detect whether new obstacles would cause collisions.

Below are 2 examples of RRT* for directed and undirected searching.



- **State System - Lead: Nicolas Perrault, Other contributors: Sean Shi**

Due to the high precision necessary for picking up the cubes a state system was crucial. In order for the manipulation to work properly we found it most efficient for the robot to approach each cube the same way in a head on manner. Our state system was also used for determining when to search for or navigate towards a goal. The first draft of this system was created by Sean Shi. Refining this state system composed the majority of my time (Nicolas Perrault) as getting each method working together took a lot of debugging and simulating. Our greatest challenge was consistency, due to the randomness of the robot, it would sometimes miss a cube. This required a lot of precision and the best trial is shown in the video below.

Demo

---

https://www.youtube.com/watch?v=xDbsR8DpBQw&ab_channel=NicPerrault