Final Project Report: Mobile Pick and Place

Xiaoyang Liu (xiaoyan1) **Juncheng Zhang (junchen1)**

Karthik Ramachandran (kramacha) **Sumit Saxena (sumits1)** **Yihao Qian (yihaoq)**

Adviser: Dr Matthew Travers
Carnegie Mellon University Robotics Institute

1 Problem Definition

The Biorobotics lab's hexapod (snake monster) is a great robot for navigating complex terrains but it lacks vision. The goal of our project was to integrate vision into the snake monster system and develop capability to enable it to map the environment, localize itself, detect and locate specific objects in the environment, plan paths and execute them.

Below are the detailed specifications required for each of the sub tasks:

1.1 Robot Localization and Mapping

The robot should be able to map the surrounding environment and localize itself using the point-cloud data from Kinect

1.2 Locate the object of interest

The robot should be able to locate the object of interest (a cup in our case) using the kinect sensor. The assumption is that the robot can identify the object by scanning the environment from its starting position.

1.3 Locate the final destination for the object

The robot has to move the identified object from the identified location to the destination location. This requires that the robot be able to locate the final destination to go to. An AprilTag was used to specify the destination location in our case.

1.4 Plan paths

The robot should be able to plan paths to go from the starting location to the object location, and from the object location to the destination location.

1.5 Execute planned paths

The robot should be able to execute the planned paths to go from starting location to the object location, and from the object location to the destination location.

1.6 Grasp and carry the object

The robot should be able to grasp and carry the object of interest from the identified location to the destination location.

1.7 Release the object at the destination

The robot should be able to release the object at the destination location.

2 Related work

A* is a popular algorithm used for static and dynamic planning. In [1], the authors present a graph-based planning that works dynamically at each step, optimizing between anytime and incremental planning approaches. In its current manifestation, this project uses a simple static planner to plan actions towards achieving the goal of grasping an object and moving it to a target location but it can be easily be extended to incorporate a more sophisticated planner like [1] that can dynamically plan around obstacles and moving targets.

Autonomous robot working in real environment must build a map to navigate in large, unknown spaces, and to localize itself, which is known as Simultaneous localization and mapping(SLAM). RGB-D cameras (such as the Microsoft Kinect) are novel sensing systems that capture RGB images along with per-pixel depth information, which could be very useful for 3D SLAM applications. One of the state-of-art methods for RGB-D SLAM is [2], which is a graph-based SLAM system with a memory management approach[3] within real-time constraint. In order to decrease the accumulated error, it also uses the appearance-based global loop closure detector[4] for the better pose estimation and mapping result.

3 Approach

3.1 Hardware Setup

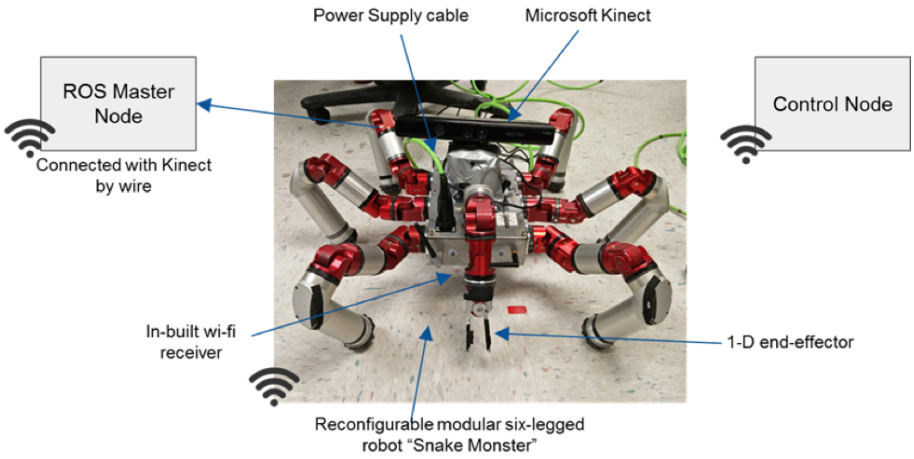

Figure 1: Hardware Setup of Our Snake Monster System with Vision Integrated

The final hardware setup for our system includes:

1. Reconfigurable modular six-legged robot "Snake Monster": It has an in-built wi-fi receiver and receives commands from the Control Node
2. Microsoft Kinect: It is mounted over the Snake Monster facing the front
3. ROS Master Node: This is a laptop connected directly with the Kinect on the robot and runs ROS. It processes data from the Kinect, performs SLAM, does object detection and publishes the robot's localization/pose information and the detected object/AprilTag's location information
4. Control Node: It subscribes to the data published by the ROS master node and determines the control needed to perform the required actions. It runs a MATLAB program which executes high-level controls and interfaces with the Hebi API, which generates low-level controls to control each of the modules on the robot to execute the required motion

4 System Design

The overall system design could be illustrated by the following graph.

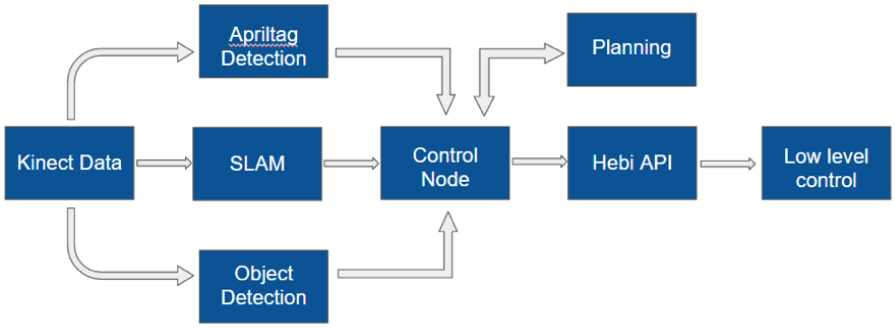

Figure 2: Overall System Design

Our system reads the data from the kinect for object detection and SLAM, and sends the obtained localization data, map information, and the detected object location to the main control node. After receiving all these processed data, the control node is able to know the location of the robot, the goal location(cup or AprilTag), and the obstacles around the robot, so that the planning algorithm could be implemented to figure out what the next movement the robot should conduct. The actual command will be converted to low level control by Hebi API for the robot to execute.

5 Robot Localization and Mapping

For robot localization and mapping, we tried to compare different methods to find the most appropriate one for our use case. We implemented two different methods, including RTAB-MAP (Real-Time Appearance-Based Mapping) and Hector SLAM.

RTAB_MAP is a RGB-D Graph SLAM approach based on a global Bayesian loop closure detector. It uses feature-based matching method to localize and map in 3D environment. This method doesn't need IMU information, but it requires relatively high computation capability since it processes 3D Pointcloud data. However, it can provide fantastic result using the RGB-D sensor, such as the kinect. One screenshot of running this method to map a room is shown in Figure 3:

In terms of Hector SLAM, it is a 2D SLAM method based on scan-to-map matching method using the scan data. In order to use this method for our case, we need to first transform the pointcloud data to the laser scan data. But because of the limited range of kinect and the noisy data from it, the performance of Hector SLAM using the kinect is not satisfactory. Consequently, we finally decided to use RTAB_MAP for our SLAM task.

The result of our SLAM method when we first initialize the map is shown in Figure 4:

As we can see from Figure 4, there are several mapped points and a red arrow representing the pose of the robot in rviz. The map and the pose of the robot will keep being updated when the robot conducts its task.

6 Object Identification

6.1 Object Detection

The cup detection subsystem is used to detect the cup position in the global coordinate. The input of the subsystem are RGB image and point cloud captured by kinect v1 sensor. You may find how cup detection sub-system works in Figure 5.

As you can see in Figure 6(1), The sub-system first applies superpixel algorithm on the raw-image. This process gives us a preliminary segmentation of the image based on color (in Figure 6(2)). Then

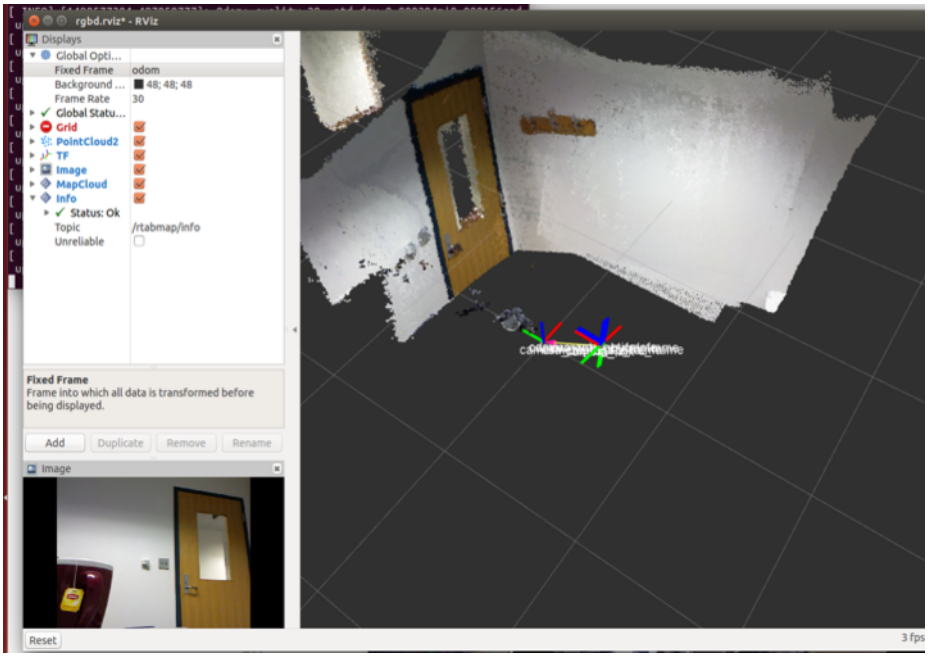

Figure 3: Result of RTAB_MAP

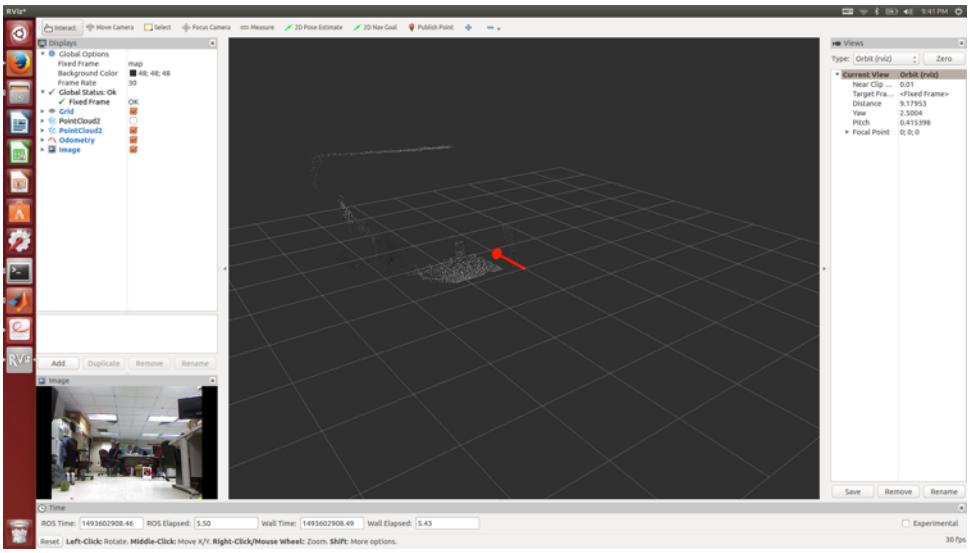

Figure 4: Result of the Initial Map

the sub-system applies K-means algorithm on the segmentation image. After this process, we may get several potential objects in the image. Based on our prior knowledge, we know the cup has high possibility that it would occur at the bottom of the image, and also we will choose the cluster which has the smallest area to simplify the problem (Figure 6(3)). So we just pick the ROI in the bottom of the image as our guess of the position of the cup (Figure 6(4)). Then the algorithm would compute the global position of cup based on the bounding box position in the image and the point-cloud it received.

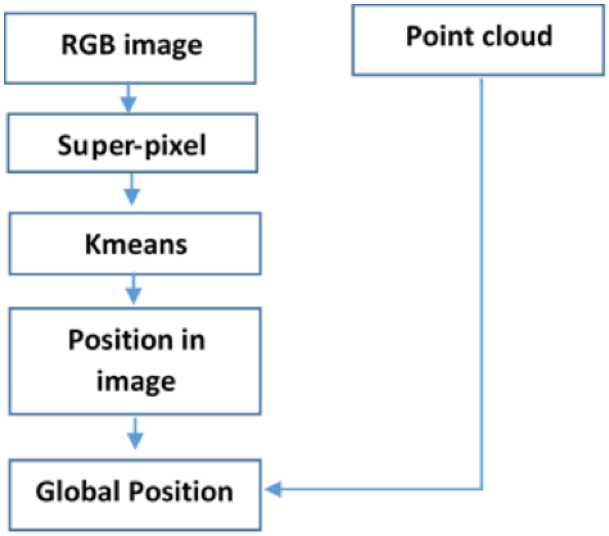

Figure 5: Object Detection Sub-system

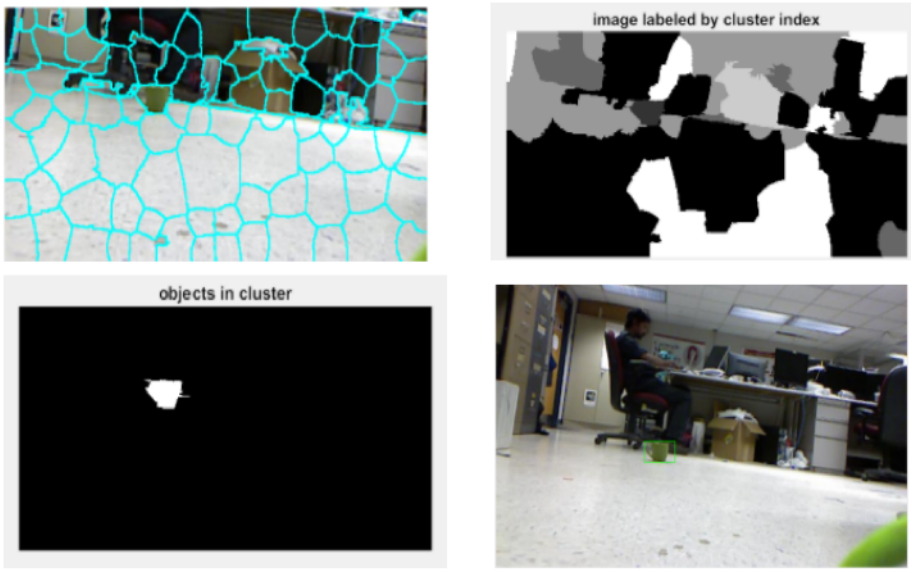

Figure 6: Object Detection Based On Clustering

6.2 April-tag Detection

We used aprital_ros package to detect the AprilTag from the image subscribed from kinect, and to publish the relative pose of the AprilTag with respect to the robot frame. The global location of the AprilTag could be estimated based on the current location of the robot and the relative pose of theAprilTag. Figure 7 shows a screenshot of a properly detected AprilTag.

7 Path Planning

For this project a simple A* based static planner was considered. The environment had two goals namely, an object location to grasp the object and a goal location to drop the object. Task accomplishment was modelled as a set of actions to be executed. At a high level the following tasks were defined for the A* planner: Object Detection, Tag Detection, Translation, Orientation and

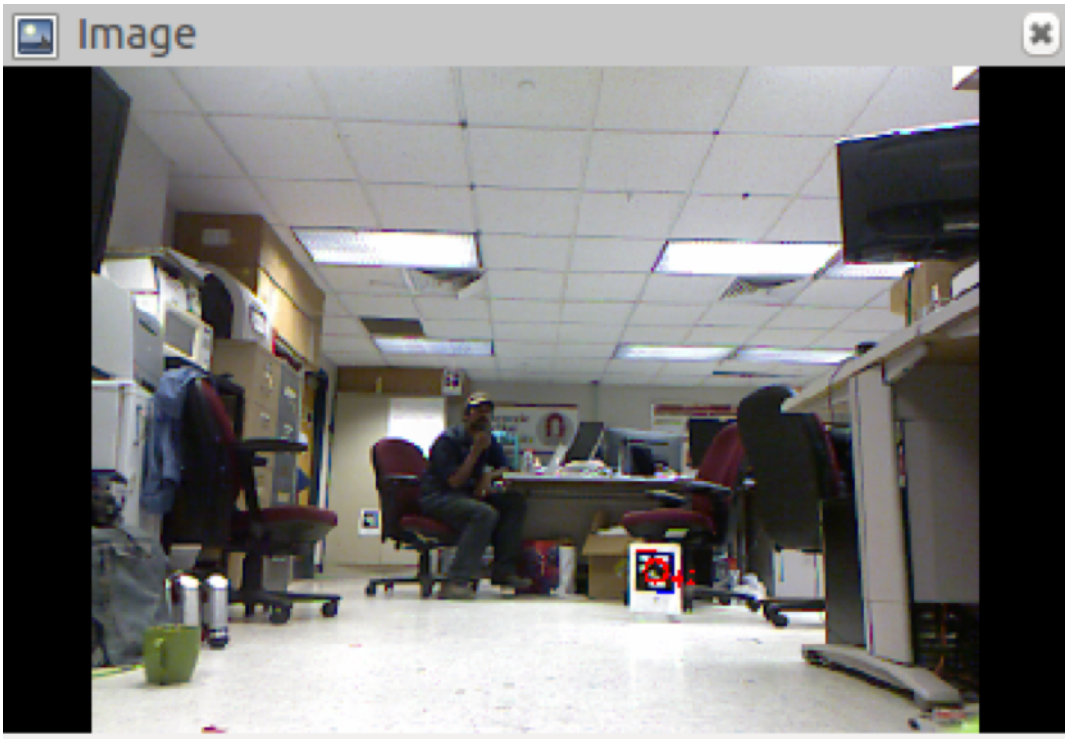

Figure 7: Result of Detected AprilTag

Stabilization. While working with the snake monster, we realized that introducing a stabilization task before switching between two tasks was key to achieving the goals.

8 Motion Control

The controls for the robot were implemented using MATLAB to interface with the HEBI API provided for the robot. The Hebi API provides low-level controls for each of the HEBI modules present on the robot. Using the robot's localization information, object/destination's location information, and the generated plan, we generate high-level controls in a MATLAB program which interfaces with the HEBI API to generate low-level controls. The API provides controls for maneuvering the robot along 4 axis using velocity controls. The robot is controlled using a state diagram corresponding to the actions provided by the planner and executing each of the actions while maintaining state

9 Grasping

The Snake Monster has a 2-DOF arm with a 1-DOF end-effector. Grasping was achieved by controlling the angle of the arm and the opening of the end-effector. HEBI API provides a MATLAB interface, using which we gave commands to the arm to move to required orientations and open/close the end-effector for grasping/releasing the object. We implemented manual grasping but could not achieve autonomous grasping of the object because of the following reasons:

1. To be able to grasp the object, the robot has to be at a distance of about 10 cm from the object and in this position, the kinect is just about 25-40 cm away from the object. Since Kinect can only sense objects at least 80 cm away from it, we are unable to detect the object when the robot is close enough to be able to grasp the object
2. Given 1, the only way we could still achieve autonomous grasping was making the robot reach a precise location close to the object and then execute a pre-decided grasp. But, in our experiments, we were not able to move the robot to precise locations. It always reaches the desired locations with inaccuracies of about 10-15 cm.

As such, it was not possible for us to implement autonomous grasping. So, when the robot reaches close to the object and stops, the robot's arm moves to a predefined orientation and opens the end-effector. We place the object (cup's handle) in between the fingers of the end-effector, the end-effector is closed after some time and the object is grasped. Then, the robot carries the object to the destination location, moves the arm to a predefined orientation and opens the end-effector to release the object.

10 Division of Work

Table 1: Division of Work

SNO	Sub Task	Individual
1	Robot localization and mapping	Juncheng, Yihao, Sumit
2	Detect and locate object	Xiaoyang, Yihao
3	Path Planning	Karthik, Juncheng
4	Motion Control	Karthik, Sumit
5	Grasping	Sumit, Yihao, Xiaoyang

11 Videos

Figure 8: View from Front

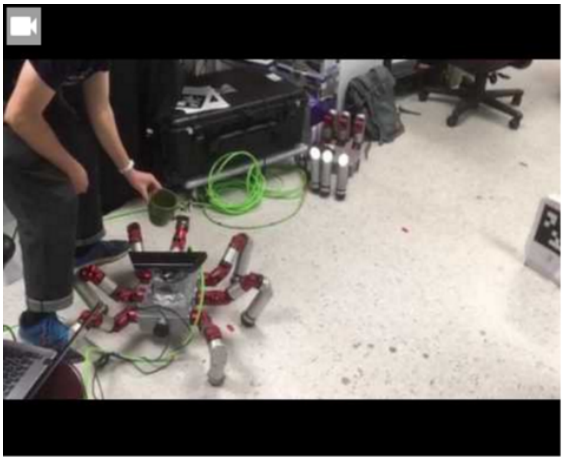

Figure 9: View from Behind

References

[1] http://www.cs.cmu.edu/ ggordon/likhachev-etal.anytime-dstar.pdf

[2] M. Labbé and F. Michaud, “Online Global Loop Closure Detection for Large-Scale Multi-Session Graph-Based SLAM,” in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2014

[3] M. Labbé and F. Michaud, “Memory management for real-time appearance-based loop closure detection,” in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2011, pp. 1271–1276.

[4] M. Labbé and F. Michaud, “Appearance-Based Loop Closure Detection for Online Large-Scale and Long-Term Operation,” in IEEE Transactions on Robotics, vol. 29, no. 3, pp. 734-745, 2013.