Set up and Foundation of the Husky

Marisa Warner, Jeremy Gottlieb, Gabriel Elkaim
Worcester Polytechnic Institute
University of California, Santa Cruz

***Abstract*—Clearpath's Husky A200 is an unmanned ground vehicle that UCSC's Autonomous Systems Lab acquired to use as a vehicle base for future projects. The purpose of this project is to establish a foundation and documentation for the Husky and set up the sensors on board the Husky. I set up the Point Cloud Library to view the Kinect and LiDAR video streams in Rviz. I then streamlined a way to record and play back the data using a built-in feature in ROS. I also set up how to visualize the Husky in a simulator Gazebo and how to command the Husky. In particular, I devised a system whereby the same code could be copied from programs running within Gazebo to those running onboard the Husky. This will help streamline the development cycle for control software for the Husky. (*Abstract*)**

Keywords—Robotics; Point Cloud Library; Robot Operating System (*key words****)***

I. Introduction

The Autonomous Systems Lab (ASL) at UC Santa Cruz has as its main focus creating low-cost, scalable autonomous vehicles - aerial, ground and water. The ASL obtained from Clearpath Robotics a Husky A200 all-terrain autonomous ground vehicle. The Husky will be used for future projects in the lab such as route planning, terrain estimation, and obstacle avoidance.

The Husky was shipped with multiple sensors, such as Stereo Camera's, Microsoft Kinect and the Hokuyo LiDAR, already attached that needed to be verified and calibrated to be able to work with the required software such as the Robot Operating System (ROS), Point Cloud Library (PCL) and Gazebo simulator.

The purpose of this project was to create and document a system to efficiently calibrate the sensors, access and store sensor data for later use, and test code that will operate on the Husky without it being physically present. The documentation will later help aid future projects by providing a code bank of snippets of basic code to help in development of extensive functions*.

II. Robot Operating System

Robot Operating System, know as ROS, is a software framework system created by Stanford Artificial Intelligence Laboratory known as Willow Garage. The idea of ROS came from the concept of Graph Archetiture. As describe in the *paper* ROS is design to be modular; composed of nodes. Each node has a way of communicating between each other through topics composed of messages. Where there is node that will 'publish' and 'subscribe' to other node and will send data messages to another node. The main concepts documented about ROS are nodes, message, topics, and bags.

As described before, Nodes are consider a software module that communicate within each other like a visualized graph. Nodes communicate with each other by passing messages. A node sends specific messages to another node by publishing related topic. If another node needs that certain data, the node can subscribe to that topic appropriate topic. There may be multiple coexisting publishers and subscribers for one topic, and a node can publish and subscribe to several topics.

A topic is a piece of data that contains messages. Each topic is associated with a certain input or output of a sensor. For example, the topics associate with the stereo camera's are the live feed from each camera in either compressed. Within the topic, is a message that contains the feed's information. Depending on the frequency-publishing rate, a topic can publish a thousand of messages per minute.

Another goal of this project was to find a way to be able to record data from a topic and be able to replay it back in place of the topic. A bag is a file format used for recording and playing back ROS messages from topics. A Bag file is an important mechanism for storing

data, such as sensor data, that can be difficult to collect due to the volume at which messages are being published.

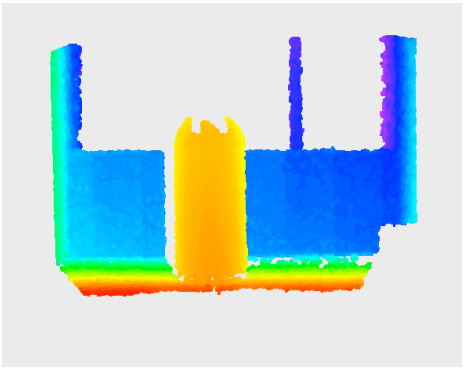
III. Point Cloud Library

The Point Cloud library is an open source project that was developed for image and cloud processing. A Point Cloud is a set of Data points that compose a three dimensional image. The library contains several features such as filtering, surface reconstruction, and model fitting and segmentation. With these features, the user can accurately recognized an object or define a preset image for better recognition. The Point Cloud Library was easily integrated within ROS so that from the sensors, such as the Kinect, the data could be view from a built in visualizer called Rviz.

A. Sensors

The main sensors used on the Husky with the Point Cloud Library are the Microsoft Kinect and the Hokuyo LiDAR. The goal of these sensors is to get the robot a informational depiction of the it's surroundings. The Kinect was set up front and centered of the robot to get a maximum view of the of what could be in front of the robot.

One of the main sensors is the Microsoft Kinect. Within the Kinect is a 3D depth Sensor and RBG camera. Combining the two, with the Point Cloud Library, it would produce a three-dimensional image such in Figure 1. Within Rviz, there are multiple settings which you can view the image. You can change the axis setting that measures depth, along with the different sizes of data points.



(*Figure 1.* The Point Cloud Library integrated with the Kinect. This is a picture of a trashcan two feet away from the window. The Kinect camera is 5 feet away from the trashcan)

The other main sensor on the Husky is the Hokuyo LiDAR. A LiDAR is a remote sensor that measures ranges of variable distances of an environment by analyzing the reflecting light of a laser. With this data, it can generate a lot of elevations that form a point cloud. With this Point Cloud and combine with the Point Cloud Library, it can generate a three dimensional image of its surroundings.

(Figure 2. A picture of the LiDAR scan of the hall way containing the trash can.

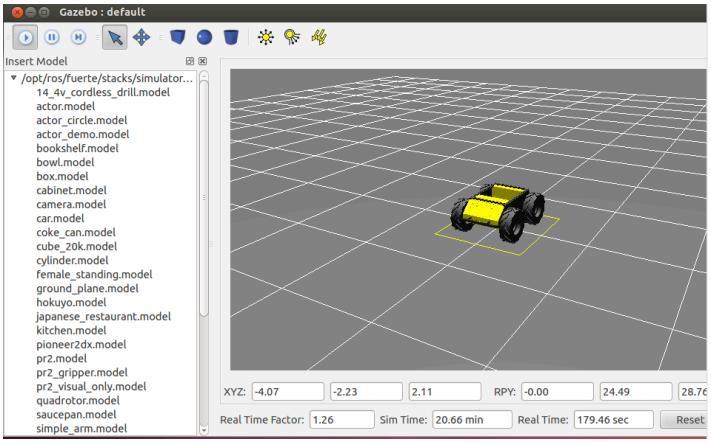
With the LiDAR connected with ROS and the Point Cloud Library, it was possibly to view the image in Rviz. For example in Figure 2, it is an image of the front perimeter of the husky. You can see the outline of the walls of the Hall was the front outline of the trashcan put in front of the Husky.

IV. Gazebo

Gazebo is a multi-robot simulator used to simulate outdoor environments. Gazebo is capable of simulating multiple robots, sensors and object in a three dimensional world. You can get realistic sensor feedback while having generating physical interaction with objects. The main part of this project was to be able to create a clean transition between developing code that can be used on the simulator and the final code on the Husky platform. Within ROS, Clearpath Robotics has set up launch files of the husky and certain objects. For example, to go to the launch directory where the launch files for the Clearpath Husky and Husky environment you enter the command:

Cd opt/ros/fuerte/stacks/husky_simulator/husky_gazebo/launch

In figure # is an example what will come up when these launch files are ran.



(Figure 3. The Gazebo version Clearpath environment and Husky)

Also, within the *husky_simulator* stack, the user can manually manipulate the simulated Husky through initializing the teleop program. This program allows the user to click and move the husky.

V. Simulation Code

One of the main goals that ASL wanted to accomplish was be able to find a way to develop and test the code for the husky with out needing to have the Husky physically present. This would provide multiple projects to be able to work on the Husky without being restrained by waiting for the husky to be come available.

As explained before, Gazebo is a simulated environment that allows the user to manipulate simulated object to real life parameters. The user can control the Husky in Gazebo through the manipulation of the Twist messages. Twist messages are of type geometry_msgs. Geometry messages are common geometry primitives such as points, vectors, and poses. The geometry_msg/Twist can be formatted by linear and angular inputs.

The code i8n Appendix 1, is as snippet of how as were able to move the Husky in a continuous square. A Node was created to publish the Geometry Twist Messages to the topic /cmd_Vel. The topic /cmd_Vel will then publish the motors on the Husky to either turn right or left at a certain angle.

We took this code from the simulator and place it directly onto the Husky's hard drive to the node auto_drive. Auto_drive.py is a node that allows the user to switch between autonomous mode and teleop mode by pressing the deadman switch. Once autonomous mode is initiated it will run through the sequence of the husky driving in a square.

VI. Acknowledgment

I would like to acknowledge the Summer Undergraduate Research Fellowship program at University of California, Santa Cruz and the Autonomous System Lab for allowing me to participate in this program.

REFERENCES

Appendix of Code

```
#!/usr/bin/env python

""" Example code of how to move a robot around the shape of a square. """

# We always import roslib, and load the manifest to handle dependencies
import roslib; roslib.load_manifest('mini_max_tutorials')
import rospy
# recall: robots generally take base movement commands on a topic
#  called "cmd_vel" using a message type "geometry_msgs/Twist" from geometry_msgs.msg
 import Twist

class square:
   """ This example is in the form of a class. """

   def __init__(self):
      """ This is the constructor of our class. """
      # register this function to be called on shutdown
      rospy.on_shutdown(self.cleanup)

      # publish to cmd_vel
      self.pub = rospy.Publisher('cmd_vel', Twist)
      # give our node/publisher a bit of time to connect
      rospy.sleep(1)

      # use a rate to make sure the bot keeps moving
      r = rospy.Rate(5.0)

      # go forever!
      while not rospy.is_shutdown():
         # create a Twist message, fill it in to drive forward
         twist = Twist()
         twist.linear.x = 0.15
         for i in range(10):       # 10*5hz = 2sec
            self.pub.publish(twist)
            r.sleep()
         # create a twist message, fill it in to turn
         twist = Twist()
         twist.angular.z = 1.57/2    # 45 deg/s * 2sec = 90 degrees
         for i in range(10):       # 10*5hz = 2sec
            self.pub.publish(twist)
            r.sleep()

   def cleanup(self):
      # stop the robot!
      twist = Twist()
      self.pub.publish(twist)
```

```
if __name__=="__main__":
    rospy.init_node('square')
square()
```