Heriot-Watt University

Robotic Systems Science

B31YS

Husky Project

Author:	*Student ID:*
Sean Katagiri	H00230479
Joshua Roe	H00216377

December 28, 2018

Contents

1 Abstract **2**

2 Introduction **2**

3 Implementation **3**
3.1 Navigation - Mapping 3
3.2 Navigation - move_base 4
3.3 Camera 5
3.4 Camera - object detection 7
3.5 Object Manipulation 8

4 Discussions and Next Steps **9**

5 Annex **10**
5.1 Changes to husky onboard software 10
5.2 Faults found 10

List of Figures

1 A flawed map produced by gmapping (left) compared to map produced by hector_mapping (right) 3
2 move_base plotting a course to a goal set by full_demo.py. . . . 4
3 An example of a good disparity image gained in simulation with a bumblebee camera (left), versus poor disparity image gained from bumblebee camera on the husky (right). 5
4 Astra camera mounted on top of bumblebee camera. 6
5 find_object_3d GUI showing image feed and objects detected, and corresponding transforms created. Marker object (left) and object for manipulation (right) 7
6 Visualisation of path produced by moveit_planner. The orange highlighted section represents the desired position and orientation of the joints once the plan is executed 8

1 Abstract

The objective of the project was to implement ROS packages to make the Husky A200 with dual ur5 manipulators, to allow the Husky to locate a predefined object in the environment, navigate to the object and pick up the object with the manipulators.

The result proved to be very promising, with the Husky being able to navigate to the object in an unknown environment, obtain the object with the ur5 manipulators with Robotiq grippers, and return to the starting position. This was done with the combination of hector_mapping for SLAM, move_base for navigation and moveit_planner for motion planning of the dual ur5 manipulators. However, moveit_planner encountered an issue in forming a valid plan to a target joint end point frequently, which was not resolved in the end.

2 Introduction

In real world applications, the robot is required to map and localise itself within the environment, and in many cases also required to detect objects within the environment for manipulation of some form.

For this project, we were tasked with setting up the husky in a way that allows us to achieve this goal of performing SLAM, navigation and manipulation of an object within the environment. Since it will be one of the first time the Husky with a dual ur5 setup will be used, we will be testing various packages to integrate into the system to see which is the most suitable for the Husky.

3 Implementation

3.1 Navigation - Mapping

The Husky came equipped with a 16 channel Velodyne LIDAR sensor, which provides a 360° pointcloud reading with a $-15°/25°$ vertical FOV, as well as a SICK 2D LIDAR sensor. Although the performance of the Velodyne sensor is impressive, for mapping purposes in the Husky project it was deemed unnecessary to implement, and thus was decided that we would only be using the SICK sensor. One of the main issues we found with the Husky was that both sensor readings from the SICK and Velodyne sensors were being published to the same `/scan` topic, causing any nodes subscribing to the topic producing unreliable outputs. To resolve this we edited the launch file for the Velodyne node, adding a remap from `/scan` to `/scan_velodyne` allowing us to view the two sensor readings separately.
We initially tested mapping by attempting to make use of the `gmapping` package. However, we encountered a problem with the generated map where even when the gmapping node was using the filtered odometry data (which accounted for wheel slip on the Husky), the Husky wasn't being localised properly within the mapped environment causing each scan of the environment to result in an inaccurate overlapped map. Because of this issue, we instead opted to make use of the `hector_mapping` package. Since it only uses the scan data as opposed to using odometry as well as scan it worked perfectly within the testing environment, managing to perform SLAM without any issues provided that the `odom_frame` was set to `odom` as a roslaunch argument for `hector_mapping/launch/mapping_default.launch`.

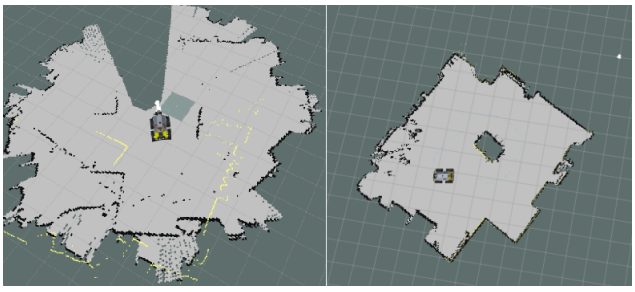

Figure 1: A flawed map produced by gmapping (left) compared to map produced by hector_mapping (right)

3.2 Navigation - move_base

For navigation to a point in the environment, we used the `move_base` node within the `husky_navigation` package. Our python script `full_demo.py` retrieved the goal position from the goal transform published by the object detection node using a tflistener, and the script included an action client of type `MoveBaseAction` that sends a message of type `MoveBaseGoal` with target pose position and orientation to the move_base action server.
Initially we tried to run the code and start the move_base node from our laptops, however that caused many issues with time sync errors, where the node would timeout due to the walltime timestamps of the messages being sent having a difference greater than the threshold. Initially, we corrected the time on the husky manually at every boot, however we quickly realised that a simpler way to solve this issue was to instead of running the code on our laptops, we launched everything including our own python code on the roscore PC on the husky by ssh'ing into the onboard computer.

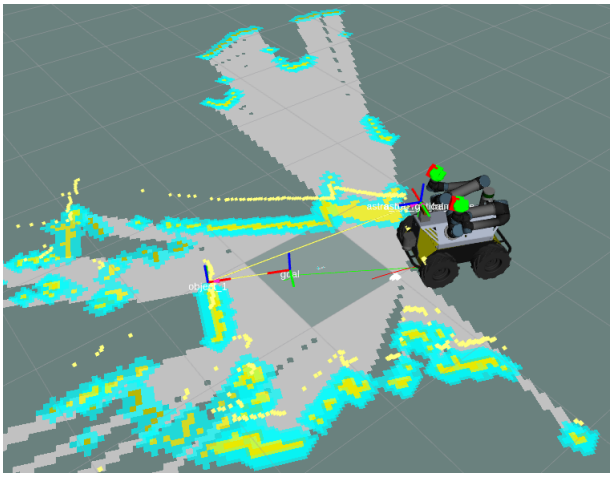

Figure 2: move_base plotting a course to a goal set by full_demo.py.

One aspect of move_base worthy of note is that it will not make the husky rotate if it expects the robot will collide with an object that is too close. On the husky itself, there are bumpers which are attached to the front and the back of the base with the front bumper being in view of the sick sensor, which caused issues resulting in move_base refusing to turn the husky due

to "objects" being perceived too close to the husky. However this issue was resolved by setting the scan topic move_base uses to `scan_filtered` which filters objects too close to the husky from the scan data. This was a feature initially implemented for the velodyne sensor so that the arms would not cause the velodyne sensor to see an "object" practically right on top of the husky. An additional solution was to reduce the radius of the husky footprint in `local_costmap.yaml` within the config for the move_base node.

3.3 Camera

The bumblebee camera was the intended camera of choice as it was already installed on the husky, however attempts to recover a disparity image from the camera in real time proved to be a challenge due to what was assumed to be an incorrect setup of the transforms of the camera itself, resulting in the disparity image gained from it to be completely incorrect. Specifically, the distances to points in the image were inaccurate by a large amount making the disparity image completely useless for object detection in 3D space.

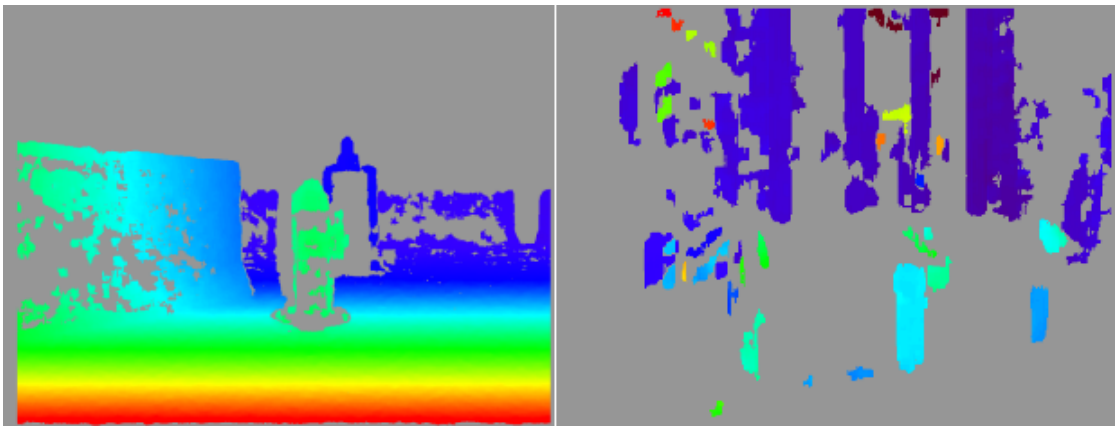

Figure 3: An example of a good disparity image gained in simulation with a bumblebee camera (left), versus poor disparity image gained from bumblebee camera on the husky (right).

After further testing with the bumblebee, it was decided that rather than trying to work with the bumblebee the simpler solution was to install the Orbbec Astra 3D camera due to its RGBD output, which provided a direct depth im-

age without having to first retrieve then convert the disparity image ourselves. The Astra camera was tied securely on top of the bumblebee, and required transforms were added by using the `astra_launch/launch/astra.launch` file, which was also installed onto the husky. With this new installation, we were able to make use of object detection packages without any further issues caused by the camera itself.
A more general issue concerning the cameras was that the image feed gained from the husky over a wifi had too much lag to be of any use in realtime, requiring a LAN connection in order to make use of the image feed if object detection was to be performed on a secondary PC outside of the husky. Again, this issue was resolved by running the `find_object_2d` package from the onboard PC within the husky.

Figure 4: Astra camera mounted on top of bumblebee camera.

3.4 Camera - object detection

Our approach to object detection was to use two steps in completing the task:

- Detect a large marker object within the environment which can be found from a long distance
- Detect object for manipulation on top of marker object once husky and dual ur5 arms are close enough to carry out manipulation task.

Part of the reason for this approach was due to the features of the object for manipulation being too small to be detected and matched with the predefined object from afar. By taking this two step approach we were able to eliminate concerns regarding the number of the features required for the object. The find_object_2d package was implemented to achieve this goal, more specifically making use of `find_object_2d/launch/find_object_3d.launch` for detection in 3D space.
Once an object is detected, a static transform is created in relation to the pose of the object produced by find_object_3d as a goal reference point for move_base and moveit_planner.

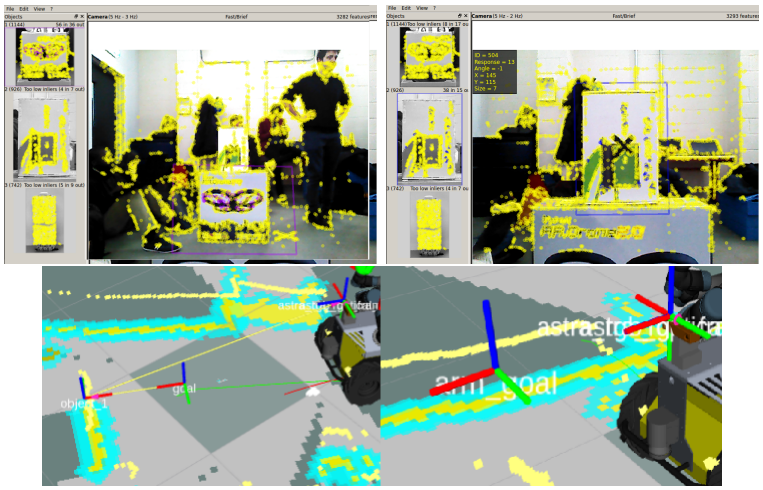

Figure 5: find_object_3d GUI showing image feed and objects detected, and corresponding transforms created. Marker object (left) and object for manipulation (right)

3.5 Object Manipulation

For object manipulation, the dual ur5 arms were controlled using `moveit_planner`. The goal position for the joint end point is gained from the goal transform produced by object detection using a tflistener, and is sent to the moveit_planner for path planning. In our implementation a confirmation check was used to ensure the arms did not move in a way that would harm persons, the environment or the hardware of the husky. This was achieved by first making the planner visualise the plan for the arms first in the moveit rviz console, and ask for user input to confirm the plan is safe to perform. As well as being useful during testing and tuning to understand the path planning, it also adds an additional safety mechanism for when using the husky.

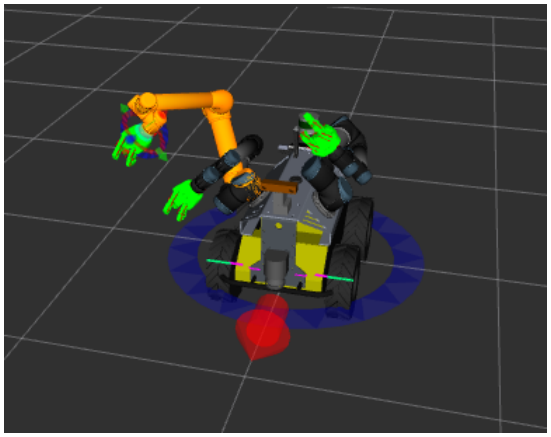

Figure 6: Visualisation of path produced by moveit_planner. The orange highlighted section represents the desired position and orientation of the joints once the plan is executed

While testing we discovered that the planner would often fail to achieve a valid plan for the dual ur5 arm even when the goal position is detected and set properly, requiring multiple attempts of planning and occasionally requiring a slight change in the orientation of the target object or the husky in relation to the target object. Although the issue was improved by changing the parameter for the goal distance of the husky itself in relation to the marker object, the planner would still fail to create a valid plan occasionally. This issue was never fully resolved.

4 Discussions and Next Steps

While making use of the Astra camera proved to be very effective, it would be a worthwhile investment to troubleshoot with the bumblebee camera, to truly make use of the husky setup in it's original form without having to apply makeshift fixes.

It is also worth noting that we were able to make use of the dual ur5 arms individually, however a method for controlling both arms simultaneously by sending goal points was not found during this project. Further research is required to make maneuvers possible with simultaenous use of the arms, as that would open up many more possibilities for tasks the husky could perform.

Although the project did not achieve the goal of placing the object down on the ground once returning to the start position with the object (due to moveit_planner failing to find a valid path to place the object), the project was a success overall as it managed to achieve the main goals of detecting and navigating to a given object, and picking it up using the dual ur5 arms.

5 Annex

Link to video: https://www.youtube.com/watch?v=YRMUL0r2nTc
Link to github: https://github.com/Joshua-Roe/Husky-Project

5.1 Changes to husky onboard software

Installs

- hector_mapping
- astra_driver
- find_object_2d
- chrony – for time sync
- move_base – upgraded to fix bugs
- costmap_2d – upgraded to fix bugs

Changes to config

- velodyne model changed to account for height extension.
- velodyne_nodelet_manager – now publishing to scan_velodyne
- astra_launch – changed namespace from camera to astra
- astra_frames – transforms configured for mounting on top of bumblebee2, now compatable with PTU
- find_object_2d – added seperate launch file for use with astra

5.2 Faults found

- husky turning on spot should be restricted due to heavy payload, as cuases damage/loosening of wheel axels
- husky time drift not calibrated causing errors to begin within a few days of time sync chrony configuration in progress to act as time server
- bumblebee camera to configured correctly, possibly a namespace issue

- imu does not appear to be calibrated, may be causing gmapping/odomery issues