Final Project

Python Applications for Robotics

****

August 23, 2022

Student:

Yoseph A Kebede

Instructor:

Z. kootbally

Course code:

ENPM809E

Contents

Introduction

This project aims to simulate the delivery of a product to a desired destination via a pre-programmed robot that has been configured to arrive at predetermined location to receive as well as deliver parts. Thus, although not autonomous, the robot will be able to use proportional control to arrive to its destination by navigating to each part in its queue. Hence, using the Gazebo environment in ROS, a selected turtle Bot model is first placed at the center of a work cell containing four cameras in the four corners of the surrounding, each viewing a single part placed within its vicinity. Similarly, there are four designated drop locations found in between these cameras which - depending on the design of the iteration - is where the robot will be dropping off the parts in any given order. Therefore, the major steps in the mission can be summarized as below:

1. identify part to be delivered.
    (a) part type.
    (b) number of parts.
    (c) drop location.
2. search for camera near part.
3. move robot to part
4. attach part to robot
5. carrying item to drop location.
6. drop item by detaching from robot.
7. return to start location, if complete. If not, repeat above steps until all parts are delivered.

To assist in this project, a bot_controller_class has been provided with fundamental methods acting as the building blocks to communicate instructions to and listen results from the robot. These are subscribers, publishers, proportional controls, navigators taking in goal coordinates, frame transformers, and part attach/detachers to name main ones. Therefore, taking in all of these inputs in hand, one must construct a functionality instructing the robot to deliver all of the desired parts to their intended location in their sequence, returning to the start position at the end.

Approach

As described in the previous section, the first half of the challenge has already been solved, that is all of the parts loaded in the environment have a child frame created from their respective parent camera frames, named as camera_***#***_assembly_pump_***color***_0_frame, where "#" represent the number distinguishing each camera while "color" is a visual identifier for the four-color parts described above.

Therefore, the solution begins by trying to match the parts available in the queue to the cameras recording them, so that the part frame of reference can be obtained interms of the world frame. This is completed by subscribing to the topics /odom, all four cameras in /logical_camera/, and /part_infos. Although the node would have received continuous messages published in the rate designed, the rospy.wait_for_message() function is called for the cameras and /part_infos topic to save the very first message published, since the part name and pose will not change until end of mission. Finally, these values are stored into the bot_controller as a dictionary for camera and list for the parts.

The program then loops through all the parts in the list. In each iteration, the camera parts dictionary keys, which have been assigned the part names, will be searched and compared to the part being looped through. If no match found, loop skips to next item in the list. If not, then the part name is used to instantiate an object from the Parts class that contains type, frame, and pose information. Next, using the get_transform() pre-designed method, the parts location will be transferred to the /odom frame with the new pose being saved as the destination spots for the robot. Later, using the built in go_to_goal() method, the robot is commanded to arrive at part pose through a control loop proportional drive system which applies linear and angular velocity based on how distant or veered the robot is from the destination. Following arrival, the attach_goal() method is called to delete, spawn and relink part to robot which is immediately followed by the go_to_goal() method guiding the robot to the drop off location pulled from the parts list. Finally, the detach() method is called to detach part from robot. Here, program checks if there are other parts remaining in queue. If not, the program terminates after driving the robot to the initial location. Summarized pseudocode for this process is shown in Figure 1a below with a breakdown of actions in each iteration. Additionally, Figure 1b shows block cell drawing where the process begins and ends in the multiple iterations of the parts queue list..

```
Delivery():
    ➢ For i in Queue (parts)
        o Iterate through one part from a parts list
        o Find a matching camera with selected part from cameras list
        o Transform part frame from camera frame to /odom frame
        o Obtain goal coordinates from part in /odom frame
        o Command robot to goal location
        o Attach part to robot
        o Command robot to drop location
        o Detach part from robot
        o Command robot to start position
    ➢ Repeat above steps for all parts
```

Figure 1a. Pseudocode for turtlebot part delivery

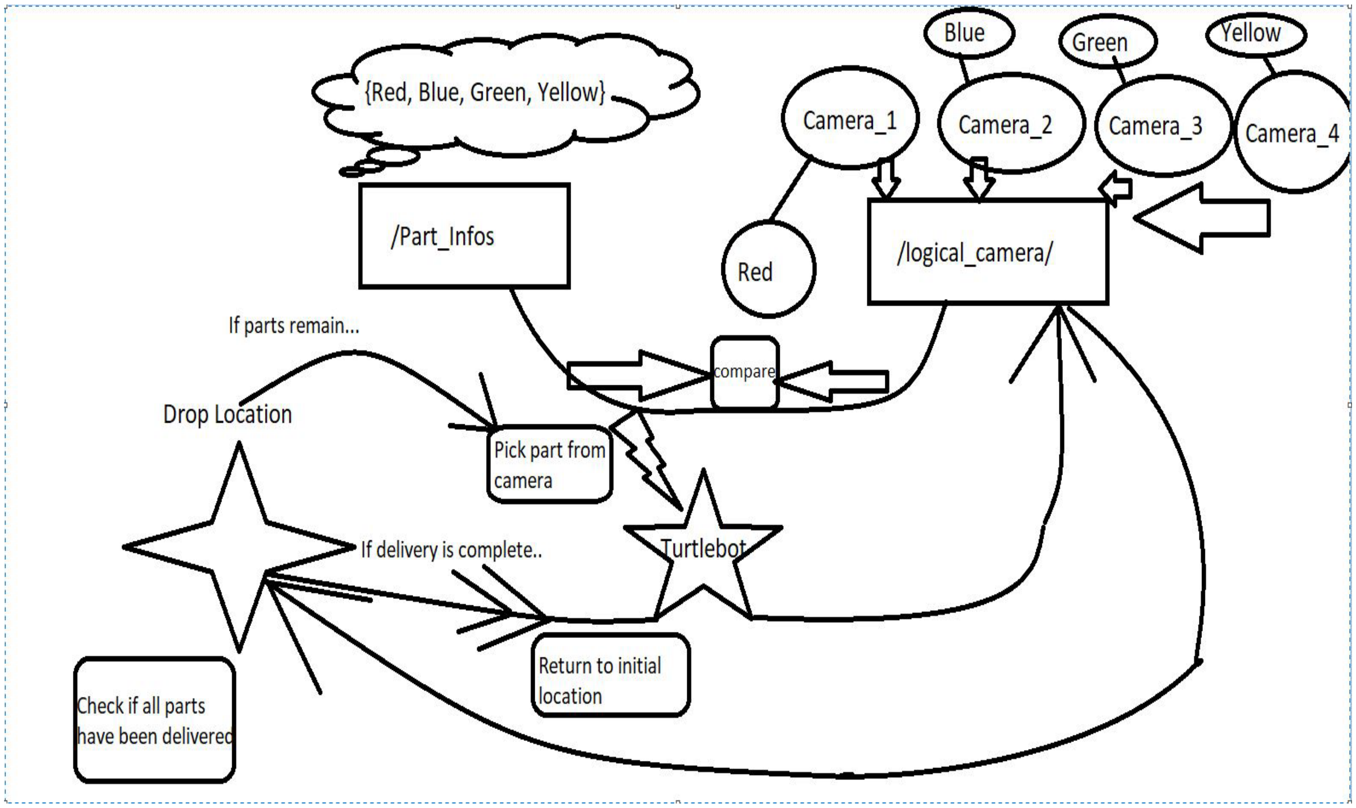

Figure 1b. Flow diagram for robot delivery

Challenges

While working on this projects, there were couple obstacles that had to be overcome. To begin with, figuring out on how to correctly extract variables from ROS messages published on any topics has been difficult. Since these values were being used in the bot_controller_class script for comparision and sending back signals for the robot to listen too, figuring out how to call a value for assignment took numerous help website searches and lecture note navigation. Furthermore, because ROS melodic was ran on the system while VSC has Python 3 interpreter set up, some imports such as tf module and calls of the bot_controller_class from the bot_controller node were not compiling. It is until much later that changing the interpreter of the node to Python 2 was learnt to do the trick. Finally, the get_transform() method had the source and target frames switched in its body, which resulted in the robot arriving to a different destination every time program was run. At the end, through iterative method of elimination debug attempts, fixing the function finally did the trick. All in all, the learning curve for writing an independent function that could ingratiate all the tools (methods) provided for this project was not significant. However, looking on where and how to fix them has been an annoyance until inevitably resolving them all.

Resources

Class lectures 9, 10, and 11.

Course Feedback

I enjoyed the class. It's practical, engaging, and intuitive. Furthermore, they start of simple, but slowly build off to a more advanced class, thus integrating beginner and advanced level programming all together with well taught lectures. Professor welcomed all questions and made sure every doubt was answered in or outside class. The one area I would suggest as an improvement is for the department to set up virtual machines for future students so that simulations that require high computing power to be conducted effortlessly when completing homeworks or project submissions. If this system is set up, just like other classes such as CAD classes in Engineering, I believe that more students will register to become better robot programmers.