TurtleBot: Gesture Recognition

Vertically Integrated Project: Robosense

Spring 2014

Weiyu Liu

Professor: Fumin Zhang

Graduate Student Advisor: Carol Young

Introduction

The main research objective for this semester was to get familiar with Robot Operating System (ROS) and add basic gesture recognition ability to TurtleBot. Since TurtleBot and many other robot platforms use ROS as their frameworks, to comprehend this system became the first step to allow for further development. Once this was accomplished, implementing basic gesture recognition in TurtleBot by using its Kinect camera helped to explore and improve human-robot interaction. Both of these two short-term tasks prepared for achieving a boarder goal: to have TurtleBot gather various gesture information from users and act correspondingly. In order to finish all these tasks, I referred to tutorials available in ROS official websites, integrated existing libraries and packages, and wrote my own source codes for basic application of gesture recognition.

Background

Important Concepts:

For this project, TurtleBot was used. TurtleBot combines popular off-the-shelf robot components, the iRobot Create, Yujin Robot's Kobuki and Microsoft's Kinect into an integrated development platform. Behind the TurtleBot, ROS acts as its operating system. ROS is an open-source framework, which contains collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. It provides the services of all operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management.

Preparation Work:

As ROS runs on Unix-based platforms, to install Ubuntu operating system on my computer became necessary. Different versions of ROS were chosen to suit specific demands, like to install “fuerte” for TurtleBot simulator or to install “groovy” for using gesture recognition packages. Once the installation was completed, I started reading tutorials for ROS on its official websites. The tutorials has became a important reference to understand different abilities and tools provided by ROS.

Results

Plan:

To achieve the two tasks and the overall goal, four basic steps were involved:

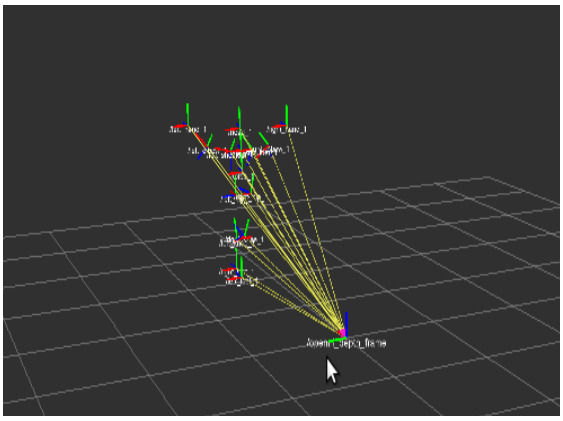
1. To be familiar with ROS commands and tools.
2. To test the existing gesture recognition package, and confirm its output by using visualization tools. To find out useful transformation information for this project.
3. To integrate gesture recognition package in my own source code to control Turtlesim.
4. To print detailed transformation information to console for debugging and finally to develop a successful Turtlesim program with gesture recognition ability.

Accomplishments:

In the first step, I went through the tutorials available online to be familiar with ROS. Because of the different versions of ROS, there are also two kinds of build systems called “rosbuild” and “catkin”, which correspond to two different sets of commands in shell. Since I used a more recent version of ROS “groovy”, “catkin” is the only option. In order to compile and run ROS programs, a workspace named “catkin_ws” was first being set up. All the files in this workspace have to be sourced in the shell environment as ROS relies on the notion of combining spaces using the shell environment. After all this, different commands for creating ROS packages, searching ROS packages, and compiling ROS packages were learnt. By applying commands like $rosnode and $rostopic, the basic concepts of nodes and message communication in ROS were understood. A node is a process that performs computation; Nodes are combined together into a graph and communicate with one another. For example, in a robot system, one node controls a laser range-finder, one node controls the robot's wheel motors, and one node performs localization. ROS also includes three types of message passing interface for the communication of nodes, which are synchronous communication over services, an asynchronous streaming of data over topics, and a storage of data on the parameter server. Finally, it was learnt that ROS accepts both C++ and Python for its source code. Since I was familiar with C++, I used C++ for all of my later codings.

Once I comprehended the basics of ROS, I tested the kinect camera on TurtleBot by using a drive program called “openni_kinect” and display the video in a ROS visualization software called “rviz”. In “rviz”, two parameters called “fixed frame” and “topic” has to be set up. By

setting different parameters, the point cloud image or the RGB image can be shown. After this, I further tested the gesture recognition package “openni_tracker”. This package broadcasts the OpenNI skeleton frames using transform information. I learnt that transform (tf) is a package that lets the user keep track of multiple coordinate frames over time. tf maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc between any two coordinate frames at any desired point in time. So in the “openni_tracker” program, the user's pose will be published as a set of transforms, like /head, / torso, and /left_elbow. In order to confirm that whether the “openni_tracker” recognizes different joints of user’s body correctly, I was attempted to visualize the tf information. It was found that there were only two ways to accomplish this. One is to use a software NiUserTracker provided by Nite company. Another approach is to use a tf visualization tool built in “rviz”. Failing to find the first software, I turned to use rviz. The result is shown in the figure below.



After testing “openni_kinect” and “openni_tracker”, the two gesture recognition packages I would integrate to my program, I started to plan the use of the tf information to control robot. Then I decided to use a simple ROS package called “turtlesim” to try out. Package “turtlesim" is a simple GUI application which contains a simulated turtle in the background moving around with controlled linear velocity and angular velocity. In order to use tf information provided by “openni_tracker” to control the movement the simulated turtle, I wrote a tf listener program which use tf information as inputs and determines the velocity of simulated turtle accordingly. In this program, I first calculated the displacement between head and torso. The code is shown below.

```
listener.lookupTransform("torso_1", "head_1", ros::Time(0), transform);
```

Then by using code `transform().getRotation()` and `transform().getOrigin()`, I was able to get the displacement in x, y, and z dimensions. However, I was not able to understand the difference between these two methods until later. Instead, in the first trial, I simply used `transform().getRotation().x()` to control the linear velocity of simulated turtle. Since the

user may have small unintentionally movements, thresholds were being set up to filter out negligible fluctuations in data. In first trail, the user was able to control the acceleration of the simulated turtle by lean forward and backward.

The next step was to determine both linear and angular velocity by tf. There were two problems ahead: one was to determine the correspondence between three dimensions of tf and the user's movement towards different directions, another was to understand the data putted out by method `transform().getRotation()` and `transform().getOrigin()`. The information from website which shows the the unit of tf is in meters helped to make sense of the data. After that, I planed to compare the data closely with the kinect video to find out the pattern. At first it was found that the displacement between head and torso is too small to have a clear pattern as it was constantly disturbed by unwanted fluctuations of data. I solved it by replacing head with left hand which can moves far away from torso in different angles. After this change, I compared the data with the camera video again and found that `transform().getOrigin()` actually provides correct displacements between left hand and torso. In the same time, it was discovered that x direction in tf corresponds to the movement of left hand to the left side of torse, y direction corresponds to the movement up, and z direction corresponds to the movement forward. By solving both problems, I was able to use tf in x and y directions to determine the angular velocity and linear velocity. In order to allow the program to work with different user with different body dimensions, I applied the algorithm that will constantly record the maximum and minimum displacement and use these two values to scale the velocity of simulated turtle. A portion of the code is shown below.

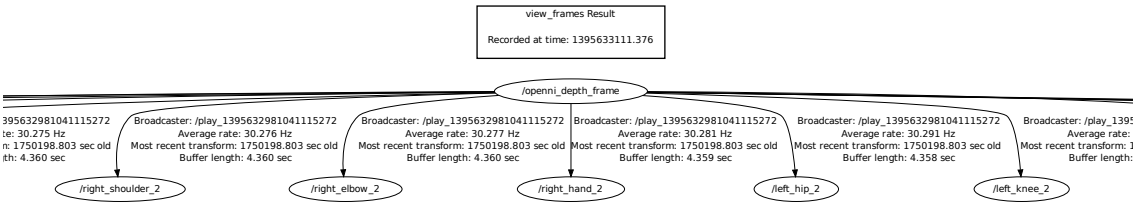
```
y = transform.getOrigin().y(); // stores tf in y direction to y
if (y > max_y) {  // actively determine the maximum of y and record it.
   max_y = y;
}
if (y < min_y) {  // actively determine the maximum of y and record it.
   min_y = y;
}
// below use max and min y to scale linear velocity
double tempy = (y - min_y) / (max_y - min_y) * (MAX_LINEAR_SPEED - 0);
if (tempy < 0.2) {
   vel_msg.linear = 0;
} else {
   vel_msg.linear = tempy;
}
```

Finally, the whole program was completed. It will use the kinect camera on TurtleBot to capture visual informations. These information will then be processed by "openni_kinect" and "openni_tracker" which will track the movement of user and output tf of every body parts. My program will then use the tf to control the simulated turtle in the "turtlesim" program. The program will first take about five seconds to calibrate for different users and will then constantly computes the velocity of the simulated turtle. Finally, user will be able to accelerate the motion

of the simulated turtle by moving up his right hand and turn the turtle by moving his right hand to the right or left side of body. (Although the tf of left_hand is used, due to mirror effect of camera, user will use right hand to control the simulated turtle).

Problems and Lessons:

Several problems were encountered. First, in order to use tf I tried to undertand the relations between all them. By using command tools $rosrun tf view_frames and $evince frames.pdf, a tree structure of the tf will be printed. This helped me to find the orders of all tf. The result is shown below. The parent frame is /openni_depth_frame which is used as the source frame in “rviz”.



Another problem was to printing data to console. Since the normal c++ style “cout” did not work, I had to import another ROS library called <ros/console.h> to print data to console. The advantage of this package is that I can assign different verbosity levels to messages.

Future Work

Turtlesim

One potential improvement of the project is for “turtlesim”. As the user is able to drive the simulated turtle by his body movement, there should be many ways to develop games of it. One possibility is to design a labyrinth game that the user has to move his body precisely to drive the turtle to the destination. In order to design a game, works related to improving the graphic interface of “turtlesim” are involved.

TurtleBot

Another approach is to move the gesture recognition program to TurtleBot. This requires knowledge of controlling the movement of TurtleBot. Another issue involved is that the kinect camera is installed on the TurtleBot so the perspective of camera may change during the

movement of TurtleBot. This may cause the failure to track a user continuously. However, one advantage of the "openni_tracker" is that it also provides information about the orientation of a user. This information presumably can be pulled out by using `transform().getOrigin()`. By using these information, programs can be developed to move the TurtleBot and make it always face to user.

After successful transferring the gesture recognition to TurtleBot, a broader vision is to make it understand a series of meaningful gestures. Different patterns of gestures can corresponds to different command for TurtleBot. One possible application is that the robot can move through office spaces and hallways using directions given by user.

Another possible development is to focus on the human-robot interaction. A program can be designed to have the robot dance with user. VIP team member Nikki Gantos has been developing algorithm for obstacle avoidance. This can be combined to gesture recognition. First, TurtleBot should be able to detect the human as an obstacle and successfully maneuver around them. Then, The TurtleBot should be able to give different dance move by recognizing different dance move by user.

Conclusion

In all, this project is a attempt to understand basic gesture recognition concepts and applications. It prepares for further researches into understanding meaningful gestures and human-robot interactions. In this project, ROS has been systematically studied and exercised; a lot of other robot basics, such as visualization and motion control have been learnt by actually applying them to the project. Finally, a gesture controlled turtle simulation has been developed to show all these concepts in action. However, this is just a starting point for all future works which will give the TurtleBot more robust gesture recognition ability. Such work includes interpreting series of meaningful gestures to commands and bridging this research with obstacle avoidance to develop a dance robot which will allows for further understanding on human-robot interaction.

Reference

Link to ROS tutorials: http://wiki.ros.org/ROS/Tutorials
Link to Trial 1, 2, 3, 4: https://www.youtube.com/watch?v=2SDc4AZhwik
https://www.youtube.com/watch?v=R4yYbU695nM
https://www.youtube.com/watch?v=X5JFaCkdmuE
https://www.youtube.com/watch?v=thgQvbI67LU