Transform Library in Distributed Robotics

Jose Ramirez
MURO LAB Student Researcher
Email: jlr020@ucsd.edu

Abstract:

The introduction of the tf library allows to track the 3D coordinate frames that change over time. The capabilities of the library allow the user to track different components of the robot, such as the head frame. The tf library can operate in a distributed system, therefore all the information about the coordinate frames of a robot is available to all ROS components on any computer in the system. My goal is to create a tf transform message between the turtlebots and the webcameras, in order to better track their coordinates.

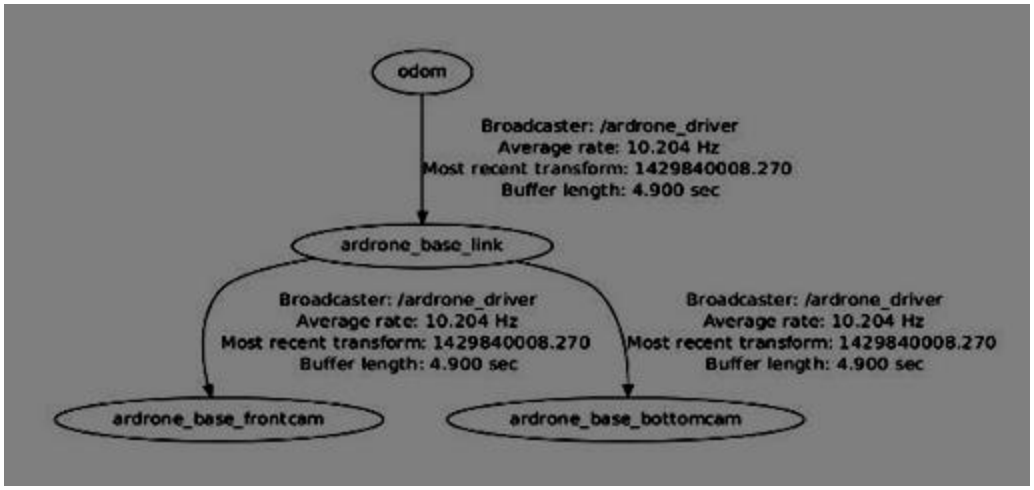
Big Picture

The tf library will improve the tracking of the turtlebots via the webcams by now tracking multiple coordinate frames over time. One of the data types that tf specializes on, is describing poses, which is one of the main types of messages that we use in the turtlebots code. The tf library also comes with a lot of tools to help us visualize and debug these coordinates. A need that the team is currently trying to accomplish is the visualization of the node tree, which the tf library will help us achieve. Overall the tf coordinates system will allow us to have a flexible relative coordinate system.

Introduction

View_Frames

This command creates a diagram of the frames being broadcast by tf over ROS. It proceeds to create and draw a tree of how the frames are connected. This tool is useful, since it allow us to keep track on how the different nodes in our project interact. Here is an example of the command being used, while I run the two cameras:



Transform Broadcasting & Listening

The broadcaster task is to message every time an update is heard about a transform. They send this messages periodically and a listening module is in charge of receiving the information. A key factor between these two nodes, is the frequency at which the tf coordinates are being transmitted. The higher the frequency, the higher the accuracy, however the bandwidth cost will increase as well. Right now the current plan is having the turtlebots sending the tf coordinates into the web cameras.

My contributions

I was tasked with creating a tf transform message between the turtlebots and the cameras. I first started by getting familiar with the library and the current code the MURO team uses to deploy the turtlebots and the web cameras (uvc camera, camera localization). After localizing these files, I proceeded to create a launch file and CMAKElist text file to run them simultaneously. Next I created a broadcaster file c++ file. This file will send tf coordinates from the turtle bots into the camera. Currently the broadcaster runs without any errors, but I still need to test it with a listener c++ file in order to see if the coordinates are being transmitted properly. The main objective of this task, is being able to visualize the different tf coordinates between the different nodes, because of this I tested the tf library by utilizing their tools such as view_frames, and printed the link between the cameras (please refer to figure 1). I have started the tutorials to develop a listener and the actual transform message, which I will be developing during the break.

Tips

Spend time reviewing the ROS terminology. This is very important when designing your transform message file and the CMAKE list. Also, make sure to review the tf coordinate wiki at: http://wiki.ros.org/tf. Make sure to go over the example, since they will get you familiarize with the syntax.

Pitfalls

I encountered a big issue when working on this project. When trying to test the launch file, it would not localize the camera node. Part of the problem was due to issues between my github branch and the master branch. I worked with Evan in helping set up and clean my branch. Now I'm happy to report that the issue has been resolved. The other part of the issue was a syntax error. Make sure to digest the example code, in order to determine the right components.

Conclusion & Future Directions

This project will facilitate the visualization of the intercommunication between our different projects. We already utilize a significant amount of nodes at the same time, and with the addition of the quad copter division we expect to use even more. My next steps are to create a listener c++ file that hears the tf coordinates of the turtlebot. Thanks to Evan, the issue that was preventing to progress in my task has been dealt with. I plan to finish this project by the start of the winter quarter. In the future we will eventually convert all our system in to tf, so that we can see the entire relationship of our project.

References

[1] Tf information http://wiki.ros.org/tf
[2] Further Tf information:
http://ap.isr.uc.pt/events/iros2012tutorial/download/Presentations/IROS2012-Hands-on-2-tf.pdf
[3] Tf paper:
The transform library
Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on Open-Source Software workshop
2013
April
Pages 1-6
Foote, Tully
[4] Tf tutorials: http://wiki.ros.org/tf/Tutorials/Introduction%20to%20tf

Code:

1) Tf_broadcaster.cpp

```
include  <ros/ros.h>
#include  <tf/transform_broadcaster.h>
#include  <turtlesim/Pose.h>

std::string  turtle_name;



void poseCallback(const  turtlesim::PoseConstPtr&  msg){
  static tf::TransformBroadcaster  br;
  tf::Transform  transform;
  transform.setOrigin(  tf::Vector3(msg->x,  msg->y,  0.0) );
  tf::Quaternion  q;
  q.setRPY(0,  0, msg->theta);
  transform.setRotation(q);
  br.sendTransform(tf::StampedTransform(transform,  ros::Time::now(),  "world",  turtle_name));
}

int  main(int  argc, char**  argv){
  ros::init(argc,  argv,  "my_tf_broadcaster");
  if  (argc  != 2){ROS_ERROR("need  turtle  name  as argument");  return  -1;};
  turtle_name  = argv[1];

  ros::NodeHandle  node;
  ros::Subscriber  sub = node.subscribe(turtle_name+"/pose",  10, &poseCallback);

  ros::spin();
  return  0;
};
```

2) CMAKEList.txt

```
cmake_minimum_required(VERSION  2.8.3)
project(turtlebot_camera_localization)

find_package(catkin  REQUIRED  COMPONENTS
  roscpp
  std_msgs
  image_transport
  cv_bridge
  tf
  turtlesim
)
```

```
find_package(OpenCV  REQUIRED)

include_directories(
  include
  ${catkin_INCLUDE_DIRS}
  ${OpenCV_INCLUDE_DIRS}
  /home/kliu/aruco-1.2.4/src
#  /home/evan/OpenCV/aruco-1.3.0/src
)

catkin_package(
#  INCLUDE_DIRS
#  LIBRARIES
  CATKIN_DEPENDS  roscpp cv_bridge  opencv2  std_msgs  image_transport  tf
#  DEPENDS
)

add_executable(camera_localization  src/main.cpp)
target_link_libraries(camera_localization
  ${OpenCV_LIBS}
  ${catkin_LIBRARIES}
<<<<<<< HEAD
  /home/evangravelle/catkin_ws/src/Aruco/build/src/libaruco.so
#  /home/evan/OpenCV/aruco-1.3.0/build/src/libaruco.so
)

add_executable(tf_broadcaster  src/tf_broadcaster.cpp)
target_link_libraries(tf_broadcaster  ${catkin_LIBRARIES})
=======
  /home/kliu/aruco-1.2.4/build/src/libaruco.so
#  /home/evan/OpenCV/aruco-1.3.0/build/src/libaruco.so
)

add_executable(turtlebot_camera_localization  src/tf_broadcaster.cpp)
target_link_libraries(turtlebot_camera_localization  ${catkin_LIBRARIES})
>>>>>>> Test
```

3) tf_broadcaster.launch

```
<launch>

  <!-- Camera -->
  <include  file="$(find  turtlebot_camera_localization)/launch/uvcCameraLaunch.launch"/>

  <!-- Localization  -->
  <node name="localization"  pkg="turtlebot_camera_localization"  type="camera_localization"/>

  <node pkg="turtlebot_camera_localization"  type="tf_broadcaster"
        args="/turtle1"  name="odom_quat"  /> </launch>
```