Linear Obstacle Creation and its Projection

Jose Ramirez

UC, San Diego

March 21st, 2016

jlr020@ucsd.edu

Abstract

This report explains the process of creating an obstacle via the android library, its translation via a ROS TF message, and the conversion from TF message into a drawn obstacle via openCV. This project is part of the lab's current goals in object detection and collision avoidance. The projection of obstacles will help MURO labs better implement and test algorithms that deal with unmanned exploration, by creating a diverse set of scenarios.

Big Picture

The MURO lab is currently venturing into projects that revolve around object detection and collision avoidance. This has opened new research opportunities for our lab group. One of these new areas, is that of Digital Terrain Design. We are utilizing different technologies and tools, such as openCV, android, and a projector, in order to create a diverse set of test scenarios for our algorithms. Currently, we have set the foundations of this Nobel approach. We have developed a process to create a series of linear obstacles. First via the android application, where a series of coordinates (utilizing ROS TF messages) are created. These points are then transferred as a ROS message from the android application to a ROS node that proceeds to draw the coordinates utilizing openCV [3]. This methodology allows us to create linear obstacles that will then be projected for real life testing, as well as set the foundation for more complex set of obstacles, such as convex hull and non-linear (Details in Conclusion & Future Directions).

1.0 Creation of Obstacles via the Android Library

Aaron Ma developed an android application that assists with the visualization of user defined input. The app is a great tool for simulating different scenarios in obstacle creation. The user is able to create a series of data points that serve as TF messages. These coordinates are stored into a finite multidimensional array that can be listened to via a ROS subscriber. From there, the node listening to the coordinates is able to utilize them as a parameter for its functions. In this case, I utilize them as the perimeter of the

obstacles that will be created.

1.1 TF Message

ROS [1] is the heart of communications for the network; it works by using a system known as a Publisher/Subscriber model. In this system, there are executable files of code called nodes which can publish (send data) and subscribe (extract data) to a topic (communication bus), which serves as a data center and intermediary step for node communication. The topic in particular that we utilized in this project is the tf library. This message allows to track the 3D coordinate frames that change over time [2]. A crucial part of utilizing the TF messages, is the transform broadcaster and listener. The broadcaster task is to message every time an update is hear about the transform. They send these messages periodically and a listening module is in charge of receiving the information. This allow us to continue in creating data points that result in the creation of multiple obstacles. The information that is tracked from each data point is their x and y coordinates, which is sent from the android application to a ROS node that will proceed to draw the obstacles in openCV.

1.2 OpenCV

OpenCV is a powerful library of programming functions mainly aimed at real-time computer vision [3]. It contains the drawing functions, which work with matrices of arbitrary depth. They also include an RGB value. Currently, the drawing function that I utilized for the drawing of obstacles is the line function. This function allows us to draw different line segments connecting the TF coordinate points. The function also boasts a respectable amount of customization, with the parameters of the function allowing to change the shape, color, and size of the line. While currently the project is limited to the line function, there is the opportunity to utilize some of the other openCV function in order to elaborate a more complex set of obstacles.

My Contributions & Methodology

This quarter I spent the majority of my time integrating the work that Aaron Ma did with the Android applet and Android studio, into openCV. First I had to set up the node that would listen to the TF messages being created by the Android applet. After confirming that the coordinates were being sent and received by the node I created, by utilizing the command ROS echo, I proceeded to formulating an algorithm that would take these coordinates and use openCV to draw them. The method I deployed was that every time that a TF coordinate was being created by the applet, the node would store these coordinate into an array and proceed to draw a new line with the new points. It would continue to update as long as new TF points were created. In order to create the obstacle, I decided on the line drawing function, due to the parameters that we were using (2-D coordinates). The first step for the algorithm to work is to create a class to store the obstacle lines. The variables of this class were a set of x and y coordinates. From there, the creation of an array with this new variable was needed to store the TF messages being broadcasted by the android applet. Next as previously mentioned, the points would be used to drawn via the draw line function, and updating every time a new TF message was

created and broadcasted. For more details, please refer to the code section.

Tips

While openCV is a very powerful tool, it is also very sensitive to its syntaxes. Make sure to look into the flow of several tutorials in order to get a feeling of how to utilize this library best. Also, look into RGB values, in order to get acquainted with the different colors and their respective code number. Finally, make sure that your node is receiving the correct TF values. You can verify with the ROS echo command. Also, make sure that the table is connected to the ROS address you are currently using.

Pitfalls

For some reason, when using openCV the libraries weren't properly set up when I tried to compile my cpp file. I was able to fix this via a run on compile command that aligned the libraries. Also, I encountered some issues with the tablet. Sometimes it would not turn on. After talking to Aaron, it seems that the best course of action is to let it charge for a while, and then try to turn it on while pressing the reset button.

Conclusion & Future Directions

The MURO lab has some more way to go before being proficient in obstacle detection and coillision avoidance, but this quarter we have made a lot of good progress setting the foundations. While for this project, we currently can only create linear obstacles, there is a lot of potential to generate more complex obstacles. The next step will be to implement openCV with Convex Hull. The goal will be to create a obstacle utilizing a the convex hull of a set of data points [5].

References

[1] Wiki.ros.org,. (2015). ROS/Tutorials - ROS Wiki. Retrieved 22 September 2015, from http://wiki.ros.org/ROS/Tutorials

[2] Tf information http://wiki.ros.org/tf

[3] http://opencv.org/

[4] http://docs.opencv.org/2.4/doc/tutorials/core/basic_geometric_drawing/basic_geometric_drawing.html

[5] http://mathworld.wolfram.com/ConvexHull.html

Code

```
#include <opencv2/imgproc/imgproc.hpp>

#include <opencv2/highgui/highgui.hpp>
```

```
#include <iostream>
#include <stdio.h>
#include <ros/ros.h>
#include <geometry_msgs/Twist.h>
#include "geometry_msgs/PoseWithCovarianceStamped.h"
#include <geometry_msgs/PoseStamped.h>
#include <geometry_msgs/Vector3.h>
#include <geometry_msgs/PoseArray.h>
#include <tf/tf.h>
#include <fstream>
#include <math.h>
#include <vector>
#include <stdlib.h>
#include "std_msgs/String.h"
#include <std_msgs/Bool.h>
#include <std_msgs/Float64.h>
#include <sstream>
#include <search.h>
//#include <stdlib.h>
#include <tf2_msgs/TFMessage.h>
using namespace cv;
using namespace std;
#define l 400
int obstacleLineSize=0;
/// Windows names
char square_window[] = "Drawing: line";
/// Create black empty images
Mat square_image = Mat::zeros(1300, 2500, CV_8UC3 );
```

```
class obstacleLine {
public:
float x[2];
float y[2];
        obstacleLine(){
        x[0]=0;
        x[1]=0;
        y[0]=0;
        y[1]=0;
        }
};
void MyLine( Mat img, Point start, Point end )
{
 int thickness = 2;
 int lineType = 8;
 line( img,
   start,
   end,
   Scalar( 255, 255,255 ),
   thickness,
   lineType );
   imshow( square_window, square_image );
}
obstacleLine obstacleLineList[50];
void obstacleCallback2(const tf2_msgs::TFMessage::ConstPtr& posePtr ){

if (posePtr -> transforms[0].transform.rotation.w==301){
obstacleLineSize=0;
```

```
for (int i=0;i< posePtr->transforms[0].transform.rotation.x;i++){
obstacleLineList[i].x[0]=posePtr->transforms[i].transform.translation.x;
obstacleLineList[i].x[1]=posePtr->transforms[i].transform.translation.z;
obstacleLineList[i].y[0]=posePtr->transforms[i].transform.translation.y;
obstacleLineList[i].y[1]=posePtr->transforms[i].transform.rotation.y;
}
MyLine( square_image, Point( obstacleLineList[0].x[0]), Point (obstacleLineList[10].x[1]) );
MyLine( square_image, Point( obstacleLineList[0].y[0]), Point (obstacleLineList[10].y[1]) );


 /// 3. Display your stuff
  waitKey( 0 );
obstacleLineSize=posePtr->transforms[0].transform.rotation.x;
}
}
/// Function headers
void MyLine( Mat img, Point start, Point end );
int main(int argc, char** argv){
 ros::init(argc, argv, "projector_test");
 ros::NodeHandle node;
 ros::Rate loop_rate(5000);
ros::Subscriber obstacle_sub2;
  obstacle_sub2= node.subscribe<tf2_msgs::TFMessage>("/tf", 25,obstacleCallback2);

while(ros::ok)

 {
```

```
  ros::spinOnce();
   loop_rate.sleep();
}
return(0);
}
```