Trent Ziemer

Dr. Newman

EECS 397

October 20, 2016

PS6: LIDAR Transforms

**1. Overview and Results**

I decided to take the scan data from each callback and process it into a list of points from that scan that are definitely part of the block. Those are then added to a global list of points. When the entire block has been scanned, all the points are analyzed (again, all during a scan callback) for various parameters.

The results from one scan are below:

< x, y, z>

Number of block scanned points total:1134.

Mins are: -0.210001 -0.664065 0.101793.

Maxs are: 0.195664 -0.2581 0.171775.

Lens are: 0.405665, 0.405965, 0.0699824.

Halfway points are: -0.00716833, -0.461082, 0.136784.

Bulk averages are: -0.000134869, -0.443653, 0.142938.

Corner 1:-0.210001, -0.664065, 0.142938.

Corner 2:-0.210001, -0.2581, 0.142938.

Corner 3:0.195664, -0.664065, 0.142938.

Corner 4:0.195664, -0.2581, 0.142938.

Line-by-line, the above shows the number of points included in the block scan, the minimum values (always <x, y, z> order), the maximum values, the differences between extrema values, the centroid of the top of the block, the mean location of the points, and the location of the four corners of the top of the block.

If we assume that we know that the object is a perfect prism block then the dimensions of the block are approximately the length of the x and y scans times the z halfway point (of the data set). This is 0.405x0.405x0.137. Then we can also find that the center of the block is at approximately <-0.00716833, -0.461082, 0.684>.

**2. Method**

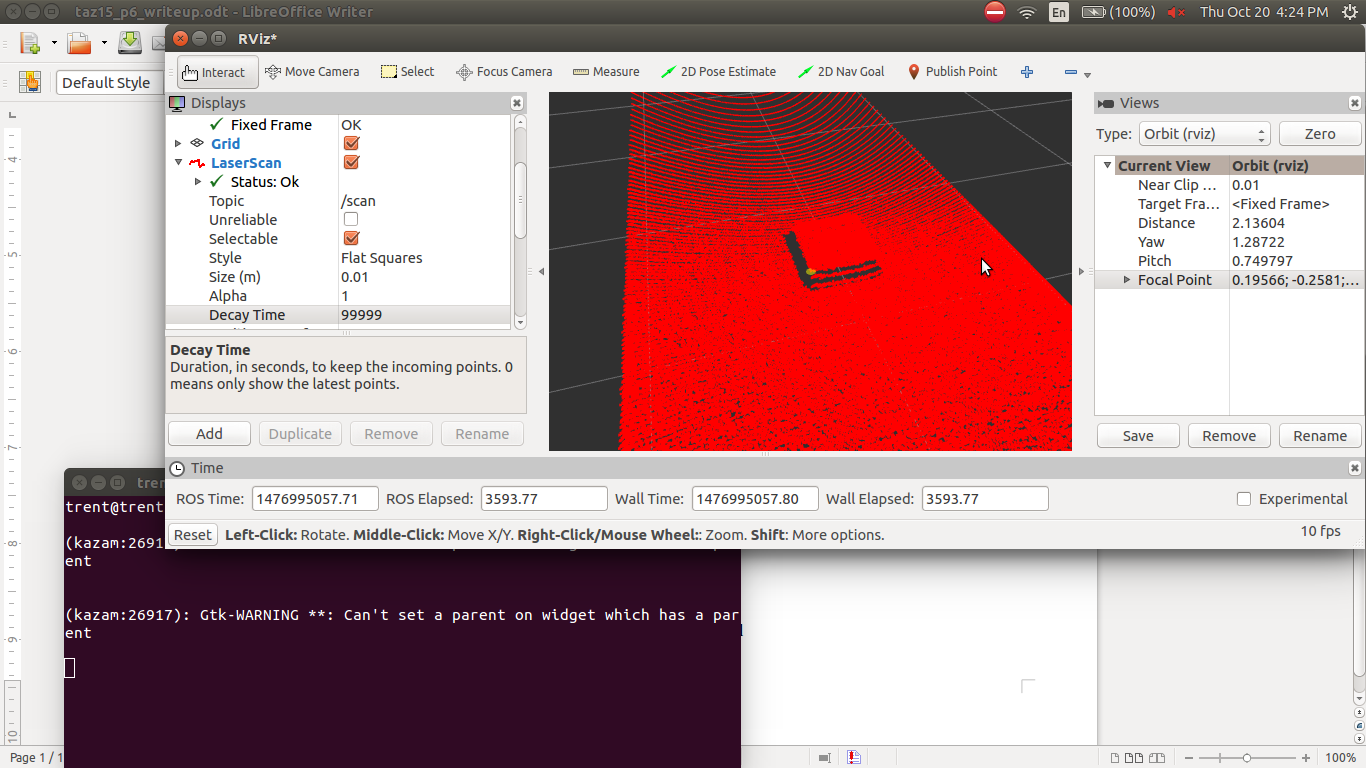
As you can see in the source code, each line scan, as a vector of 3D points, is first cut up into slices in the Z-axis. This categorizes each line scan in a series of slices, and we know that the slice at the bottom is the floor, while the next slice is the top of the block (empirical testing). If the slice has fewer than 5 points, we throw it out because it's definitely noise (and not the block).

We then add those scan points that are the top of the block to the global list of points at the top of the block. Each scan only send this function a single line, so we have to accumulate all the points from each line into a (noisy) 2D plane indicating the top of the block surface. Then the function latches through a state machine sequence that only proceeds to the last if statement once the scan callback has determined that we have finished scanning one block and then also finished scanning another block. This is necessary because the first block scan might be only partial.

In the last if statement, we know that we have a full picture of the top of the block scan. We compute various statistics here using the g\_block\_points vector of 3D vectors and then print them. The program will print out the most recent scan of the top of the block every approximately 5 seconds.

**3. Picture of an Identified Corner #4**

This picture shoes the scans, including the block, and the focal point (yellow circle) centered on one of the corners. This matches the corner #4 from the printout in .1, and, as you can see, is actually at one of the corners of the block in the picture.



**4. Source Code**

*// TRENT ZIEMER - Copied from WSN code 10/20/2016*

*#include <math.h>*

*#include <stdlib.h>*

*#include <string>*

*#include <vector>*

*#include <ros/ros.h> //ALWAYS need to include this*

*#include <tf/transform\_listener.h>*

*#include <xform\_utils/xform\_utils.h>*

*#include <sensor\_msgs/LaserScan.h>*

*using namespace std;*

*//these are globals*

*tf::TransformListener \*g\_listener\_ptr; //a transform listener*

*XformUtils xformUtils; //instantiate an object of XformUtils*

*vector <Eigen::Vector3d> g\_pt\_vecs\_wrt\_lidar\_frame; //will hold 3-D points in LIDAR frame*

*vector <Eigen::Vector3d> g\_pt\_vecs\_wrt\_world\_frame; //will hold 3\_D points in world frame*

*// Global record of all block scan points in the current sweep (gets reset each time we calculate a full block)*

*vector<Eigen::Vector3d> g\_block\_points;*

*// State machine booleans that latch the scans to the SECOND block we find.*

*bool block\_detected;*

*bool block\_finished;*

*bool wait\_for\_block\_again;*

*bool block\_detected\_again;*

*bool block\_finished\_again;*

*// Function to return the mean value from a vector of points*

*double find\_avg(vector<double> list\_of\_pts)*

*{*

*double avg = 0;*

*for (int i = 0; i < list\_of\_pts.size(); i++)*

*{*

*avg = avg + list\_of\_pts[i];*

*}*

*avg = avg/((float)list\_of\_pts.size());*

*return avg;*

*}*

*// Function to return the smallest value from a vector of points*

*double find\_min(vector<double> list\_of\_pts)*

*{*

*double minimum\_value = 99999;*

*for (int i = 0; i < list\_of\_pts.size(); i++)*

*{*

*if (list\_of\_pts[i] < minimum\_value)*

*{*

*minimum\_value = list\_of\_pts[i];*

*}*

*}*

*return minimum\_value;*

*}*

*// Function to return the largest value from a vector of points*

*double find\_max(vector<double> list\_of\_pts)*

*{*

*double maximum\_value = -99999;*

*for (int i = 0; i < list\_of\_pts.size(); i++)*

*{*

*if (list\_of\_pts[i] > maximum\_value)*

*{*

*maximum\_value = list\_of\_pts[i];*

*}*

*}*

*return maximum\_value;*

*}*

*void scanCallback(const sensor\_msgs::LaserScan::ConstPtr& scan\_in) {*

*//if here, then a new LIDAR scan has been received*

*// get the transform from LIDAR frame to world frame*

*tf::StampedTransform stfLidar2World;*

*//specialized for lidar\_wobbler; more generally, use scan\_in->header.frame\_id*

*g\_listener\_ptr->lookupTransform("world", "lidar\_link", ros::Time(0), stfLidar2World);*

*//extract transform from transformStamped:*

*tf::Transform tf = xformUtils.get\_tf\_from\_stamped\_tf(stfLidar2World);*

*//stfLidar2World is only the pose of the LIDAR at the LAST ping...*

*//better would be to consider separate transforms for each ping*

*//using the above transform for all points is adequate approx if LIDAR is wobbling slowly enough*

*Eigen::Affine3d affine\_tf,affine\_tf\_inv; //can use an Eigen type "affine" object for transformations*

*//convert transform to Eigen::Affine3d*

*affine\_tf = xformUtils.transformTFToAffine3d(tf); //can use this to transform points to world frame*

*affine\_tf\_inv = affine\_tf.inverse();*

*vector <float> ranges = scan\_in->ranges; //extract all the radius values from scan*

*int npts = ranges.size(); //see how many pings there are in the scan; expect 181 for wobbler model*

*g\_pt\_vecs\_wrt\_lidar\_frame.clear();*

*g\_pt\_vecs\_wrt\_world\_frame.clear();*

*//ROS\_INFO("received %d ranges: ", npts);*

*double start\_ang = scan\_in->angle\_min; //get start and end angles from scan message*

*double end\_ang = scan\_in->angle\_max; //should be -90 deg to +90 deg*

*double d\_ang = (end\_ang - start\_ang) / (npts - 1); //samples are at this angular increment*

*//ROS\_INFO("d\_ang = %f", d\_ang);*

*Eigen::Vector3d vec; //var to hold one point at a time*

*vec[2] = 0.0; //all pings in the LIDAR frame are in x-y plane, so z-component is 0*

*double ang;*

*for (int i = 0; i < npts; i++) {*

*if (ranges[i] < 5.0) { //only transform points within 5m*

*//if range is too long, LIDAR is nearly parallel to the ground plane, so skip this ping*

*ang = start\_ang + i\*d\_ang; //polar angle of this ping*

*vec[0] = ranges[i] \* cos(ang); //convert polar coords to Cartesian coords*

*vec[1] = ranges[i] \* sin(ang);*

*g\_pt\_vecs\_wrt\_lidar\_frame.push\_back(vec); //save the valid 3d points*

*}*

*}*

*int npts3d = g\_pt\_vecs\_wrt\_lidar\_frame.size(); //this many points got converted*

*//ROS\_INFO("computed %d 3-D pts w/rt LIDAR frame", npts3d);*

*g\_pt\_vecs\_wrt\_world\_frame.resize(npts3d);*

*//transform the points to world frame:*

*//do this one point at a time; alternatively, could have listed all points*

*//as column vectors in a single matrix, then do a single multiply to convert the*

*//entire matrix of points to the world frame*

*for (int i = 0; i < npts3d; i++) {*

*g\_pt\_vecs\_wrt\_world\_frame[i] = affine\_tf \* g\_pt\_vecs\_wrt\_lidar\_frame[i];*

*}*

*//the points in g\_pt\_vecs\_wrt\_world\_frame are now in Cartesian coordinates*

*// points in this frame are easier to interpret*

*//can now analyze these points to interpret shape of objects on the ground plane*

*//but for this example, simply display the z values w/rt world frame:*

*// <---------THIS IS WHERE TRENTS CODE STARTS AND OLD CODE ENDS. ---------------------->*

*// An array of point slices in the Z direction.*

*vector<vector<Eigen::Vector3d> > array\_of\_point\_slices;*

*// A single slice of the array of points slices. This represents a single scan of the*

*// LIDAR scanner between two Z bounds (upper and lower).*

*vector<Eigen::Vector3d> point\_slice;*

*// Z slice parameters*

*double delta\_z = 0.1;*

*double upper\_z;*

*// Divide the scan into slices (or statifications) in the Z direction.*

*for(upper\_z = delta\_z; upper\_z < 1; upper\_z = upper\_z + delta\_z)*

*{*

*// Iterate over all points to put it into one of the slices.*

*for (int i = 0; i < npts3d; i++)*

*{*

*// Load the vector.*

*vec = g\_pt\_vecs\_wrt\_world\_frame[i]; //consider the i'th point*

*// Add the point vector to the slice if its between the bounds.*

*if(vec[2] < upper\_z && vec[2] > upper\_z-delta\_z)*

*{*

*point\_slice.push\_back(vec);*

*}*

*}*

*// BUT, don't add the slice if it has too few points in it.*

*// We don't want noise getting in the way.*

*if (point\_slice.size() > 5)*

*{*

*array\_of\_point\_slices.push\_back(point\_slice);*

*}*

*point\_slice.clear();*

*}*

*// If the number of dense (has more than a few points) slices is more than one, then*

*// we know that the first one (i = 0) is the floor plane, while the second one (i = 1)*

*// is the scan of the block.*

*// Thus, we want to load this line scan of points into the full list of*

*// all points that we know are aprt of the block scan (global across these scan callbacks).*

*if (array\_of\_point\_slices.size() > 1)*

*{*

*for (int n = 0; n < array\_of\_point\_slices[1].size(); n++)*

*{*

*g\_block\_points.push\_back(array\_of\_point\_slices[1][n]);*

*}*

*}*

*// The following series of "if" statements are the formation of*

*// a rudimentary state machine, that will latch onto the SECOND full block we see*

*// (because the first one seen might/will be a partial scan only)*

*// Wait around until we see the start of the block.*

*if (array\_of\_point\_slices.size() > 1)*

*{*

*block\_detected = true;*

*//cout << "Block detected" << endl;*

*}*

*// Wait until we see the end of the block.*

*if (block\_detected == true)*

*{*

*if(array\_of\_point\_slices.size() == 1)*

*{*

*block\_finished = true;*

*block\_detected = false;*

*//cout << "Block finished" << endl;*

*}*

*}*

*// Once the first partial block started to scan, wait until we finish scanning it.*

*if (block\_finished == true)*

*{*

*wait\_for\_block\_again = true;*

*block\_finished = false;*

*//cout << "Block finished" << endl;*

*}*

*// Now that we are done with the first (propably partial) block scan, we*

*// are going to wait until we see the block again to start scanning.*

*if (wait\_for\_block\_again == true)*

*{*

*if (array\_of\_point\_slices.size() > 1)*

*{*

*block\_detected\_again = true;*

*wait\_for\_block\_again = false;*

*//cout << "Block detected again" << endl;*

*}*

*}*

*// Once we have hit the block again, wait until we STOP seeing the block again to know*

*// that we are done with the full scan.*

*if (block\_detected\_again == true)*

*{*

*if (array\_of\_point\_slices.size() == 1)*

*{*

*block\_detected\_again = false;*

*block\_finished\_again = true;*

*//cout << "Block finished again" << endl;*

*}*

*}*

*// If we finally have a full block scan done, so proceed with computing things*

*// about the full scan.*

*if (block\_finished\_again == true)*

*{*

*block\_finished\_again = false;*

*// Declare various statistic variables.*

*double com\_x, com\_y, com\_z;*

*double min\_x, min\_y, min\_z;*

*double max\_x, max\_y, max\_z;*

*double len\_x, len\_y, len\_z;*

*double avg\_x, avg\_y, avg\_z;*

*double cor1\_x, cor1\_y, cor1\_z;*

*double cor2\_x, cor2\_y, cor2\_z;*

*double cor3\_x, cor3\_y, cor3\_z;*

*double cor4\_x, cor4\_y, cor4\_z;*

*vector<double> point\_vector;*

*// Compute statistics for the X coordinates of the block points.*

*for (int i = 0; i < g\_block\_points.size(); i++)*

*{*

*point\_vector.push\_back(g\_block\_points[i][0]);*

*}*

*min\_x = find\_min(point\_vector);*

*max\_x = find\_max(point\_vector);*

*avg\_x = find\_avg(point\_vector);*

*point\_vector.clear();*

*// Compute statistics for the Y coordinates of the block points.*

*for (int i = 0; i < g\_block\_points.size(); i++)*

*{*

*point\_vector.push\_back(g\_block\_points[i][1]);*

*}*

*min\_y = find\_min(point\_vector);*

*max\_y = find\_max(point\_vector);*

*avg\_y = find\_avg(point\_vector);*

*point\_vector.clear();*

*// Compute statistics for the Z coordinates of the block points.*

*for (int i = 0; i < g\_block\_points.size(); i++)*

*{*

*point\_vector.push\_back(g\_block\_points[i][2]);*

*}*

*min\_z = find\_min(point\_vector);*

*max\_z = find\_max(point\_vector);*

*avg\_z = find\_avg(point\_vector);*

*point\_vector.clear();*

*// Compute the lengths of the scan area. Only the x and y are actually for the block itself,*

*// the Z is just the range of the z points collected.*

*len\_x = max\_x - min\_x;*

*len\_y = max\_y - min\_y;*

*len\_z = max\_z - min\_z;*

*// Find the "center of mass" of the block by finding the midway points between the extrema points of the scan.*

*com\_x = (max\_x + min\_x)/2;*

*com\_y = (max\_y + min\_y)/2;*

*com\_z = (max\_z + min\_z)/2;*

*// Compute the locations of the four corners of the top of the block*

*cor1\_x = min\_x;*

*cor1\_y = min\_y;*

*cor1\_z = avg\_z;*

*cor2\_x = min\_x;*

*cor2\_y = max\_y;*

*cor2\_z = avg\_z;*

*cor3\_x = max\_x;*

*cor3\_y = min\_y;*

*cor3\_z = avg\_z;*

*cor4\_x = max\_x;*

*cor4\_y = max\_y;*

*cor4\_z = avg\_z;*

*// Print block info for user*

*cout << "< x, y, z>" << endl;*

*cout << "Number of block scanned points total:" << g\_block\_points.size() << "." << endl;*

*cout << "Mins are: " << min\_x << " " << min\_y << " " << min\_z << "." << endl;*

*cout << "Maxs are: " << max\_x << " " << max\_y << " " << max\_z << "." << endl;*

*cout << "Lens are: " << len\_x << ", " << len\_y << ", " << len\_z << "." << endl;*

*cout << "Halfway points are: " << com\_x << ", " << com\_y << ", " << com\_z << "." << endl;*

*cout << "Bulk averages are: " << avg\_x << ", " << avg\_y << ", " << avg\_z << "." << endl;*

*cout << "Corner 1:" << cor1\_x << ", " << cor1\_y << ", " << cor1\_z << "." << endl;*

*cout << "Corner 2:" << cor2\_x << ", " << cor2\_y << ", " << cor2\_z << "." << endl;*

*cout << "Corner 3:" << cor3\_x << ", " << cor3\_y << ", " << cor3\_z << "." << endl;*

*cout << "Corner 4:" << cor4\_x << ", " << cor4\_y << ", " << cor4\_z << "." << endl;*

*cout << endl;*

*// Done with processing the block, so clear the array.*

*g\_block\_points.clear();*

*}*

*}*

*int main(int argc, char\*\* argv) {*

*ros::init(argc, argv, "lidar\_wobbler\_transformer"); //node name*

*ros::NodeHandle nh;*

*// SOME OF MY CODE HERE.*

*// Declare block state machine latching booleans as all false (will be set true in sequence then reset).*

*block\_detected = false;*

*block\_finished = false;*

*wait\_for\_block\_again = false;*

*block\_detected\_again = false;*

*block\_finished\_again = false;*

*g\_listener\_ptr = new tf::TransformListener;*

*tf::StampedTransform stfLidar2World;*

*bool tferr = true;*

*ROS\_INFO("trying to get tf of lidar\_link w/rt world: ");*

*//topic /scan has lidar data in frame\_id: lidar\_link*

*while (tferr) {*

*tferr = false;*

*try {*

*g\_listener\_ptr->lookupTransform("world", "lidar\_link", ros::Time(0), stfLidar2World);*

*} catch (tf::TransformException &exception) {*

*ROS\_WARN("%s; retrying...", exception.what());*

*tferr = true;*

*ros::Duration(0.5).sleep(); // sleep for half a second*

*ros::spinOnce();*

*}*

*}*

*//ROS\_INFO("transform received; ready to process lidar scans");*

*ros::Subscriber lidar\_subscriber = nh.subscribe("/scan", 1, scanCallback);*

*ros::spin(); //let the callback do all the work*

*return 0;*

*}*