Report for "Robot Programming" and "Least Squares and SLAM"

Pedro Daniel de Cerqueira Gava
pedrodanieldecerqueiragava@gmail.com

July 20, 2014

Chapter 1

Robot Programming

1.1 Introduction

In this chapter its covered the aspects of the implementation of the nodes responsible for transform a input depth map into a virtual laser scan and the normals of its points which will be used for calculate the transformation between scans. In order to perform this task the project is divided in three modules: virtual scans; normals pack; align pack; All of it develop using the libraries of ROS(Robot Operational System).

1.2 Virtual scan

This part of the project deals with the transformation of a line of the depth image into a simulated laser scan. This process is started receiving the input image and transforming it to a format that is possible to work with OpenCV, once that it comes as a *const sensor_msgs::ImageConstPtr* type.

Using *cv_bridge*, the image is transformed to an OpenCV "handable" with enconding 16UC1 and putting the depth values in meters. The scan is basically the depth values of the middle line of the image, by the way, to pick the correct values, we have to transform these values into values written in respect to the camera frame and not the image frame.

First things first, the scan message that this node will send is composed mainly by the range values and the angles between this values. Also there are the fields for max and minimum range to be considered when reading that message. To give the information about the angle step between beams, one can calculate it as:

$$angle\_step = \frac{horizontal\ angular\ field\ of\ view\ of\ the\ camera}{number\ of\ pixels\ in\ a\ row\ of\ the\ image} \tag{1.1}$$

Range values are obtained through transforming the points of line that is being used as base for the virtual laser scan from the image frame to the camera frame, but to do that, one only need to compute $y^{cameraframe}$, because the matrix of the transformation from the image frame to the camera frame is given as:

$$M = \begin{pmatrix} 0 & 0 & 1 & \alpha \\ -1 & 0 & 0 & \beta \\ 0 & -1 & 0 & \gamma \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{1.2}$$

Where α, β, γ are considered zero in this case to make computation easy and the angle of the beam is given in respect of the x axis of the laser frame, in this case the x axis of the camera frame. There for we compute:

$$y^{camera\ frame} = -\frac{(principal\ point_x - x^{image\ frame})depth_{(x,y)^{image\ frame}}}{focal\ lenght} \tag{1.3}$$

$$x^{camera\ frame} = depth \tag{1.4}$$

Where $x^{image\ frame}$ and $y^{image\ frame}$ are pixels and $x^{camera\ frame}$ and $y^{camera\ frame}$ are in meters. Now we compute the range of the beam of a point $(x,y)^{camera\ frame}$ as:

$$beam\ range = \sqrt{(x^{camera\ frame})^2 + (y^{camera\ frame})^2} \tag{1.5}$$

Max and minimum in is according to the programmer, in this project 13 and 0.04 were used respectively.

Also, it is necessary to set the maximum and minimum laser angle, this is, whats the angle interval of the scan:

$$maximum\ angle = \frac{0.5.horizoltal\ field\ of\ view.\pi}{180} \tag{1.6}$$

$$minimum\ angle = -maximum\ angle \tag{1.7}$$

The times between each beam is considered zero because it is considered that all the ranges are coming at the same time.

This whole process make possible to create a virtual laser scan with number of ranges equal to the width of the image, and once the message is publish the next step is to compute the normals of that set of ranges.

1.3 Normals computation

This part shows the computation done in order to obtain the normals of the points obtained from the laser.

First, the points are computed using the following equations:

$$x_i^{camra\ frame} = range_i.cos(angle\ max - i.angle\ increment) \tag{1.8}$$

$$y_i^{camra\ frame} = range_i.sin(angle\ max - i.angle\ increment) \tag{1.9}$$

Where the index i denotes the i-th beam to be used, starting from the beam with maximum angle.

And than the scan is written in respect to the a fixed point in the world to allow comparison between two sets of points in the next chapter. This transformation is done through */tf::TransformListener::transformPoint*.

The normals are computed using eigenvalues and eigenvector obtained from the covariance matrix of the point and its neighbors as follows:

$$C = \sum_{i=1}^{k} (c - p_i)(c - p_i)^T \tag{1.10}$$

Where C is the covariance matrix, k is the number of neighbors of the point which the normal is being computed and c is the centroid of that set of points.

Taking the eigenvectors corresponding to the lowest eigenvalue of a C_i we obtain the normal of point p_i.

The points and their respective normals are published in order to be used by the node presented in the next chapter.

Chapter 2

Least Squares and point Alignment

2.1 Introduction

This chapter will show the equations and decision made in order to calculate approximately the transformation needed to align two set of points which are coming already with their normals. The method used to calculate it is the least squares approach where the parameter to set small as possible is the computed error between two sets. Also, here we're not dealing with a set that has all points with correspondence in the other set of points, and for that we need the normals. We will use the normals and Cartesian distance to create this correlations.

2.2 Definitions

First of all we shall decide the appearance of the state representing this transformation, and in this case we will use a 3D vector which represents a transformation matrix, $v = [x, y, \theta]^T$.

Next, the increment is defined as:

$$v' = v + \Delta v \tag{2.1}$$

And fixing θ with,

$$\theta' = atan2(sin(v'_x), cos(v'_y)); \tag{2.2}$$

The error is computed with a $\boxminus$:

$$e_i = p_i \boxminus p_j \tag{2.3}$$

$$e_i = X * p_i - p_j \tag{2.4}$$

Where p_i and p_j represent respectively the point from the set that is being used as reference and the point of the set who is being aligned to the previous one. With increment and $\boxminus$ defined now we can define explicitly the Jacobian:

$$Jacobian = \begin{pmatrix} 1 & 0 & -sin\theta p_{i,x} - cos\theta p_{i,y} \\ 0 & 1 & cos\theta p_{i,x} - sin\theta p_{i,y} \end{pmatrix} \tag{2.5}$$

One can pay attention that the last column represents the derivative of the rotation matrix of the actual state estimation multiplied by the point from the reference set p_i.

The algorithm is done in way that it minimizes the error of the transformation iterating through the estimations, in other words, every time the the least square part finishes with an approximation, the algorithm uses this as a new guess to obtain a refined estimation. In this project, the number of iterations is according to the number of measurements in the measurement matrix Z.

In order to obtain the measurement matrix Z, the node responsible for the point alignment receives the message coming from the normal package with the set of points and their normals and use them to find correlations between the points of both sets. The structure of Z is as follows:

$$Z = \begin{pmatrix} p_{1,reference\ set} & p_{2,reference\ set} & \cdots & p_{n,reference\ set} \\ p_{1,set\ to\ be\ aligned} & p_{2,set\ to\ be\ aligned} & \cdots & p_{n,set\ to\ be\ aligned} \end{pmatrix} \tag{2.6}$$

The correlations are find first looking the distance between points from both sets, once two poinst of each set have euclidean distance lower than a threshold, it is verified if their normals have an angle lower than a certain threshold.

Another thing is that the sequence of points are being taken sequentially, the transform resulting from it can be turned into non presentable values due to its littleness, in those cases the program will consider it as zero.

The H and b are computed using and identity matrix of Ω.

$$H = Jacobian^T * \Omega * Jacobian \tag{2.7}$$

$$b = Jacobian^T * \Omega * e \tag{2.8}$$