ESE-615 - Autonomous Racing **(Due: 02/25/20)**

Scan Matching Lab

Instructor: Rahul Mangharam *Name:* Student name(s), *PennID:* NetId(s)

Learning and Outcome:
This section of the class deals with the problem of localization in Robotics and provides an introduction to localization and why is it important in the autonomy stack. Through the lab, one of the most fundamental algorithms of localization, scan matching, is implemented. It uses the Iterative Closest Point algorithm which has been introduced in class. You can take reference from the Andre Censi PLICP paper which is referenced in the pinned piazza post. By the end of this lab you will have a certain level of knowledge and expertise in localization of a robot given a mapped environment and how it is important in path planning and trajectory tracking.
Following are the key topics you should have some level of understanding in once you complete this lab.:

1. Localization
2. Odometry Estimation
3. Convex optimization
4. C++ OOP
5. Quadratic Programming

Evaluation of results:
You will be evaluated on the implementation of the scan matching algorithm and the analysis of the performance. We will evaluate how much the odometry estimates published by the scan matching algorithm deviate from the true odometry published by the vesc in the simulator. You can check this on your own by using the ROS tool called plotjuggler[Linked].

An important part of your task will be to measure and analyze the performance of your implementation. You can measure and plots results such as metric score versus iteration, mean-squared-error in odometry versus iterations until solution, etc. A thorough analysis of the different factors that affect performance and explanation for limitations is highly recommended.

The assignment is individual but you are encouraged to discuss without your teams(and everyone else)

Deliverable:

1. Answers to the theoretical questions in a separate pdf file. You can give handwritten answers converted to pdf files (the answers should be clear)
2. A ROS Package by the name of : student_name_scan_matching_lab
3. the ROS Package should have the following files
   (a) All the existing files of the skeleton with all the functions specified in this handout filled in.
   (b) A short video of the simulation demonstrating the odometry estimation
   (c) A PDF writeup describing your approach and the roadblocks you encountered in the development.
   (d) Writeup should include plots of measurements of performance and analysis of results
   (e) Any other helper function files that you use.
   (f) A README with any other dependencies your submission requires.

Problem : Lab Assignment (points)

Introduction:
The provided skeleton consists of three main program files and their respective header files which are needed to run the scan_matching package. You have to work on only two of these program files. Correspond.cpp and transform.cpp. You will not necessarily have to change anything anywhere unless explicitly required by your code. In which case, you should specify the change made in your readme.

1 Theoretical Questions (20)

1. $M_i = \begin{pmatrix} 1 & 0 & p_{i0} & -p_{i1} \\ 0 & 1 & p_{i1} & p_{i0} \end{pmatrix}$

1. Show that $B_i := M_i^T M_i$ is symmetric.
2. Demonstrate that B_i is positive semi-definite

2. The following is the optimization problem

$x^* = \text{argmin}_{x \in \mathbb{R}^4} \sum_{i=1}^{n} \|M_i x - \pi_i\|_2^2$
s.t. $x_3^2 + x_4^2 = 1$

a. Find the matrices M, W and g which give you the formulation

$x^* = \text{argmin}_{x \in \mathbb{R}^4} x^T M x + g^T x$
s.t. $x^T W x = 1$

b. Show that M and W are positive semi definite.

2 Skeleton code

. The package contains the following program files :

1. **scan_match.cpp**
   Contains the main function and the driver code for running the scan matching algorith. the node built is through this function and is called scan_matcher. You can run the node using the command:

   $rosrun < package - name > scan\_matcher$

2. **correspond.cpp and correspond.h**
   Contains the function definitions(in the correspond.h header file), the various structs needed for calculations (in the correspond.h header file) and the implementation of the functions needed for performing fast correspondence search and naive correspondence search.

   In correspond.cpp you have to fill in the function : getCorrespondence()

   Input:

   (a) old_points: vector of struct points containing the old points of the scan
   (b) trans_points: vector of struct points containing the new points transformed in the previous scans frame
   (c) jump_Table: jump table created from the helper function

(d) c : vector of struct correspondences as a references. Being a reference this will also be the output.

Output :

(a) c: vector of struct correspondences as a reference will be changed in place.

Helper Functions:

(a) computeJump: computes the jump table based on the previous table and current scan points

3. **transform.cpp and transform.h**
   Contains the function definitions, function implementations and structs needed to perform transformation of points and updating the transforms by closed form ICP optimization technique taught in class.
   You have to fill in the function: updateTransform()

   Input:

   (a) corresponds: vector of struct correspondence containing corresponding points
   (b) curr_trans: contains the transform found by the previous optimization step. Being a reference, you will update this with the new transform.

   Output

   (a) curr_trans: to be updated in place as the new transform using the previous transform.

   Helper Functions

   (a) transformPoints() : you can use this function apply any transform to the set of points. Input the required transform and the points to be transformed.
   (b) get_cubic_root: use this function to find the cubic roots given the 4 input coefficients of a cubic polynomial.
   (c) greatest_real_root() : use this function to find the greatest real root of a quartic polynomial using the 5 coefficients of the quartic polynomial as the input.

4. **visualization.cpp and visualization.h**
   You will not necessarily have to make any changes to these files. If you correctly fill in the previous files you will be able to visualize the odometry computed through the ICP scan matching algorithm.

Comments:

- We understand that the performance of the scan matching will not be ideal, the report should reflect an analysis of the performance and explanation of the limitations.
- The initial implementation uses the naive correspondence search. This can be compared to the performance after implementing the improved correspondence search.

Data Structures:

The skeleton uses a few data structures which are defined in the different program files. You are free to use the data structures provided or make new ones of your own. The functions however should be implemented as defined.

1. Point
   The struct the radial distance and angular distance of a point from the car frame. In this struct there are few functions implemented which can be used to derive other information from the points:

   - distToPoint(point P) : find the distance to another point
   - distToPoint2(point P) : find the square of the distance to another point
   - radialGap(point P) : find the radial gap to another point

- getx() : get the x coordinate of the point
- gety() : get the y coordinate of the point
- wrapTheta() : wrap theta around 2pi during rotation
- rotate(phi) : rotate the point by angle phi
- translate(x,y) : translate a point by distance x and y
- getVector(): get the vector (in x and y)

2. Correspondence
   This struct stores the correspondence values which you find through the fast search algorithm. It contains the transformed points: P, original points : Po , first best point: Pj1, second best point pj2.

   - getNormal() Get normal of the correspondence
   - getPiGeo() get correspondence point as a geometry message
   - getPiVec() get correspondence point as a vector message

3. Transform
   The struct stores the transform which you calculate through optimization at every step. it contains the x translation, y translation and the theta rotation.

   - apply(point P): apply the transform to a provided point
   - getMatrix() : get the transformation matrix from the found transform

Complete the programming exercises that follow on the next page in order to implement the algorithm.

3 Algorithm Implementation (40)

In order to implement the scan matching algorithm you have to solve the following optimization problem:

$$\min_{\boldsymbol{x}} \boldsymbol{x}^{\mathrm{T}}\mathbf{M}\boldsymbol{x} + \boldsymbol{g}^{\mathrm{T}}\boldsymbol{x}$$

$$\underbrace{\boldsymbol{x}^{\mathrm{T}}\left(\sum_i \mathbf{M}_i^{\mathrm{T}}\mathbf{C}_i\mathbf{M}_i\right)}_{\mathbf{M}}\boldsymbol{x} + \underbrace{\left(\sum_i -2\boldsymbol{\pi}_i^{\mathrm{T}}\mathbf{C}_i\mathbf{M}_i\right)}_{\boldsymbol{g}}\boldsymbol{x}$$

$$\text{subject to } \boldsymbol{x}^{\mathrm{T}}\mathbf{W}\boldsymbol{x} = 1$$

You must implement the following steps to obtain the solution to the optimization problem:

First, define the following variables in the code.

a. $\boldsymbol{x} = [x_1, x_2, x_3, x_4] = [t_x, t_y, \cos\theta, \sin\theta]$ is the optimization variable which contains the position and orientation transforms to be optimized.

b. $\mathbf{W} = \begin{bmatrix} \mathbf{0}_{2\times2} & \mathbf{0}_{2\times2} \\ \mathbf{0}_{2\times2} & \mathbf{I}_{2\times2} \end{bmatrix}$

W is the weight matrix which is needed for applying the constraint :
$x_3^2 + x_4^2 = 1$

c. $\mathrm{M}_i = \begin{bmatrix} 1 & 0 & p_{i0} & -p_{i1} \\ 0 & 1 & p_{i1} & p_{i0} \end{bmatrix}$

d.$\boldsymbol{p}_i = (p_{i0}, p_{i1})$ These are the points in the scans

e.$\mathbf{C}_i = w_i\boldsymbol{n}_i\boldsymbol{n}_i^{\mathrm{T}}$, where $\boldsymbol{n}_i \in \mathbb{R}^2$ is the versor normal to the line.

The following steps are now performed to solve this optimization problem :

1. Using lagrange multipliers the solution to this problem can be found and takes the following form :

$\boldsymbol{x}$=-(2 M+2 $\lambda\mathbf{W})^{-\mathrm{T}}\boldsymbol{g}$

here :

$$\mathbf{W} = \begin{bmatrix} \mathbf{0}_{2\times2} & \mathbf{0}_{2\times2} \\ \mathbf{0}_{2\times2} & \mathbf{I}_{2\times2} \end{bmatrix}$$

And M and g are known to us.
We need λ to find the solution to this equation.

2. Find Lambda using some tricks of linear algebra

3.1 We can write the expression of the previous equation in the form :

$$2{*}\mathbf{M} + 2\lambda\mathbf{W} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^{\mathrm{T}} & \mathbf{D} + 2\lambda\mathbf{I} \end{bmatrix}$$

Find A, B and D from the matrix.

3.2 Calculate S by the expression.

$\left(\mathbf{D}-\mathbf{B}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{B}+2\lambda\mathbf{I}\right)\triangleq(\mathbf{S}+2\lambda\mathbf{I})$

3.3 Calculate S_A

$S^A=\det(\mathbf{S})\cdot\mathbf{S}^{-1}$

3.3 Calculate $p(\lambda)$:

$p(\lambda)=\det(\mathbf{S}+2\lambda\mathbf{I})$

3.4 Using the parameters found above you can now solve the following quartic equation which is in λ
Use the helper functions which have been provided in the code. You only need to find the coefficients of the equation using the above formulated expressions:

$$\lambda^2\cdot 4\boldsymbol{g}^{\mathrm{T}}\begin{bmatrix}\mathbf{A}^{-1}\mathbf{B}\mathbf{B}^{\mathrm{T}}\mathbf{A}^{-\mathrm{T}} & -\mathbf{A}^{-1}\mathbf{B}\\ \text{(symm)} & \mathbf{I}\end{bmatrix}\boldsymbol{g}+$$
$$4\boldsymbol{g}^{\mathrm{T}}\begin{bmatrix}\mathbf{A}^{-1}\mathbf{B}\mathbf{S}^{A}\mathbf{B}^{\mathrm{T}}\mathbf{A}^{-\mathrm{T}} & -\mathbf{A}^{-1}\mathbf{B}\mathbf{S}^{A}\\ \text{(symm)} & \mathbf{S}^{A}\end{bmatrix}\boldsymbol{g}+$$
$$\boldsymbol{g}^{\mathrm{T}}\begin{bmatrix}\mathbf{A}^{-1}\mathbf{B}\mathbf{S}^{A^{\mathrm{T}}}\mathbf{S}^{A}\mathbf{B}^{\mathrm{T}}\mathbf{A}^{-\mathrm{T}} & -\mathbf{A}^{-1}\mathbf{B}\mathbf{S}^{A^{\mathrm{T}}}\quad\mathbf{S}^{A}\\ \text{(symm)} & \mathbf{S}^{A^{\mathrm{T}}}\mathbf{S}^{A}\end{bmatrix}\boldsymbol{g}=[p(\lambda)]^2$$

4. Use lambda found from the above steps to find x from the equation

$\boldsymbol{x}$=-(2 M+2 $\lambda\mathbf{W})^{-\mathrm{T}}\boldsymbol{g}$

5. Repeat until convergence. This can be a fixed number of iterations or until the difference in the metric between iterations is below some threshold.

4 Fast Correspondence Search(40)

The pseudo code given in the censi paper Andre Censi PlICP in the appendix is the best resource to get started on the fast correspondence search algorithm. It is intuitive and the slides proivde sufficient understandings for the concepts.
Here are a few pointers to take care of while coding the algorithm.

- Jump table is computed for one scan set and not for each scan point. The scan table implementation is already present in the skeletion.
- Start the correspondence search from the previous best point correspondence you found for each point.
- It is safe to make the assumption that the second best point is adjacent to the best point you have found. Decreasing or increasing the index (while taking care of the edge cases) should work. You are free to implement better ways.

- Make use of early stopping criterion cleverly. It helps in making the search faster.
- For finding the distances between the points, the functions implemented in the Point struct will be useful. Be careful, the point to line metric is not the metric for finding the closest point. It is only the metric for optimization. For correspondence you only search for the closest point by euclidean distance.

5 Running the package

To run the package use the following commands

1. Run the simulator:

   ```
   roslaunch racecar_simulator simulator.launch
   ```

2. Run the scan matching package :

   ```
   rosrun <name_of_package> scan_matcher
   ```

3. In Rviz, add the pose estimate which is published by the name of scan_match_location
4. Use plotjuggler to see the results

   ```
   rosrun plotjuggler PlotJuggler
   ```

   You will have to use the stream topics option on the top left bar.