دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

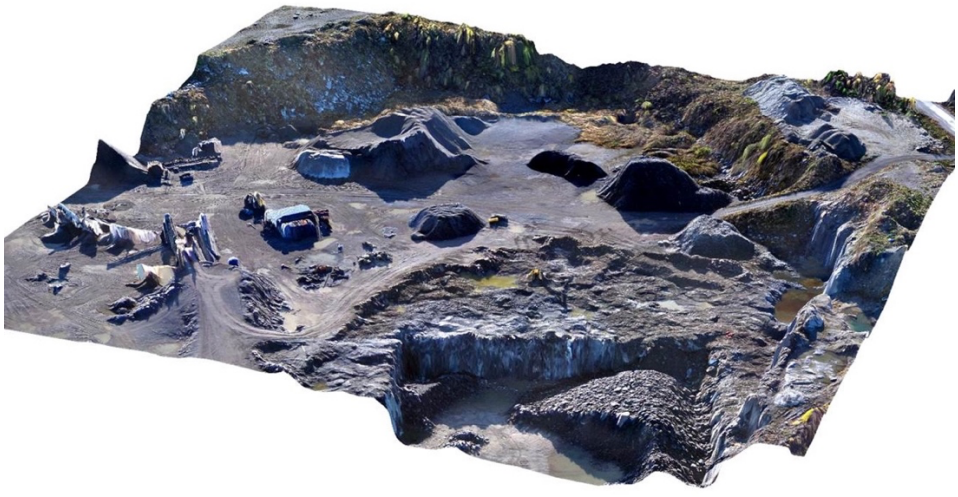
درس بنیادی کامپوزیت سه بعدی

استاد درس جناب آقای دکتر جوانمردی

(تکلیف دوم)

محسن عبادپور | ۴۰۰۱۳۱۰۸۰ | m.ebadpour@aut.ac.ir

نیمسال دوم سال تحصیلی ۱۴۰۲-۱۴۰۱



فهرست پاسخ‌ها

بخش اول: تطبیق ابر نقاط با روش ICP ۳

الف) بارگذاری مجموعه داده ۳

ب) فرآیند تطبیق ۳

ج) ایجاد و بروزرسانی نقشه ٤

د) خروجی و نتایج ٤

بخش اول: تطبیق ابر نقاط با روش NDT ۷

الف) بارگذاری مجموعه داده ۷

ب) فرآیند تطبیق و پیاده‌سازی ۷

ج) ایجاد و بروزرسانی نقشه ۸

د) خروجی و نتایج ۹

بخش اول: تطبیق ابر نقاط با روش ICP

الف) بارگذاری مجموعه داده

برای بارگذاری مجموعه داده از کد در اختیار گذاشته شده کمک گرفته شده(vis.py) و هر ابر نقاط را بصورت یک آرایه n*3 تبدیل و نگهداری کرده ام. برای جلوگیری از پردازش های بعدی، این ابر نقاط آماده با استفاده از pickle ذخیره و برای بار های بعدی استفاده می‌شود. در زمان استفاده از ابرنقاط نمونه برداری با حد ۰.۱ متر انجام می شود(voxel).

ب) فرآیند تطبیق

طبق صورت جدید تمرین، برای تطبیق ابر نقاط بصورت نسبی عمل کرده و تبدیل ها را ذخیره می کنیم؛ برای این منظور از پیاده سازی این لینک[۱] استفاده شده است که دو ابر نقاط را دریافت و تبدیلی که ابر نقاط اول را به دوم نگاشت می‌کند را بر میگرداند که این تبدیل 4*4 می باشد و مبتنی بر روش ICP بوده و بصورت تکراری و بر اساس نزدیک ترین همسایه اقدام به محاسبه بهترین تبدیل می کند. برای بدست آوردن نقشه و لیست تبدیل ها، تابع slam_icp پیاده سازی شده است که لیست کل ابر نقاط را ورودی می گیرد. این تابع ابر نقاط را پیمایش کرده و هر ابر نقاط را با یک ابر نقاط قبلی خود تطبیق داده و ماتریس تبدیل T را بدست می آورد.

در این صورت اگر بخواهیم مختصات نقاط در مختصات مرجع را بدست آوریم کافی است بترتیب تبدیل ها را از انتها به اول روی ابرنقاط مفروض اعمال نماییم تا نقاط به مختصات مرجع در آیند و معنی دار شوند. برای مثال اگر بخواهیم

[۱] https://github.com/ClayFlannigan/icp

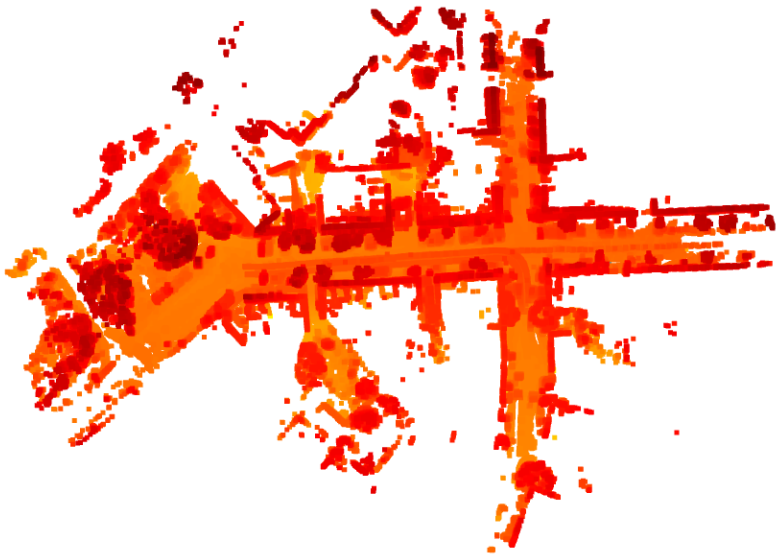
مختصات ابر نقاط پنجم را بدست آوریم کافی است روی آن تبدیل پنج به چهار را اعمال نماییم که در آخرین مرحله حساب کرده ایم؛ سپس بروی نتیجه‌ی آن، تبدیل چهار به سه را انجام دهیم که در مرحله قبلی حساب کرده ایم و ... .

ج) ایجاد و بروزرسانی نقشه

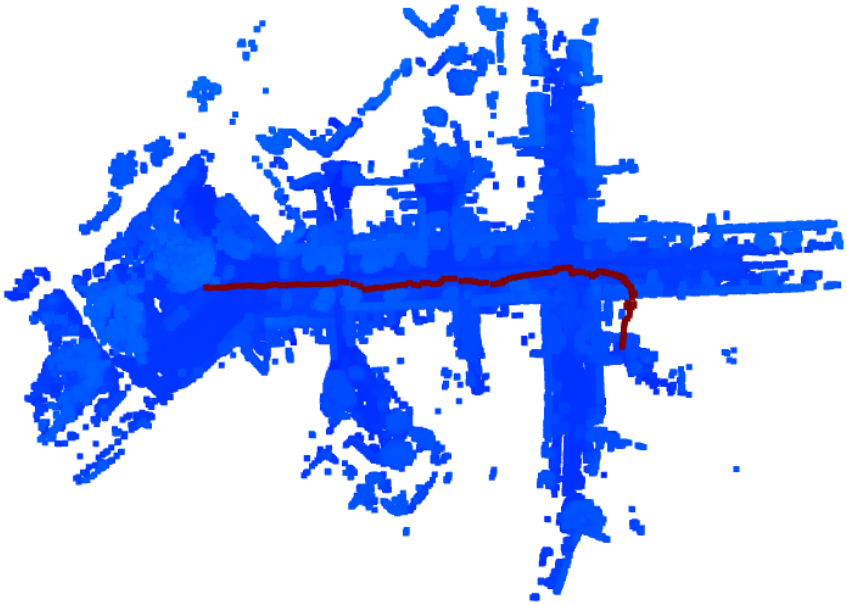
بنده در پیاده‌سازی خود از همه‌ی ابرنقاط برای تولید نقشه استفاده کرده ام. برای اینکه تعداد نقاط نقشه بصورت نمایی زیاد نشده و سنجش وجود یا عدم وجود نقطه‌ی یکسان منطقی شود، ابر نقاط حاضر را با تقریب ۰.۱ رند کرده ام(np.round). در گام نخست همه‌ی نقاط از ابر نقاط اول به نقشه افزوده می شود؛ در گام های بعدی ابر نقاط ابتدا رند و به نقشه افزوده می شود(np.concatenate)؛ حال در نقشه‌ی جدید نقاط تکراری حذف می شود(np.unique). دلیل اینکه در زمان افزودن وجود نقطه تکراری بررسی نمیشود، بحث مرتبه اجرایی و پیاده سازی بهینه توابع از کتابخانه numpy می باشد که ترجیح داده شده است از آن استفاده شود.

د) خروجی و نتایج

برای تولید نقشه طبق توضیحات ارائه شده در بخش (ج) اقدام کرده و آن را با کتابخانه opend3d مصور کرده ام که در زیر قابل مشاهده است؛ این نقشه حاصل از همه‌ی ابر نقاط می باشد:



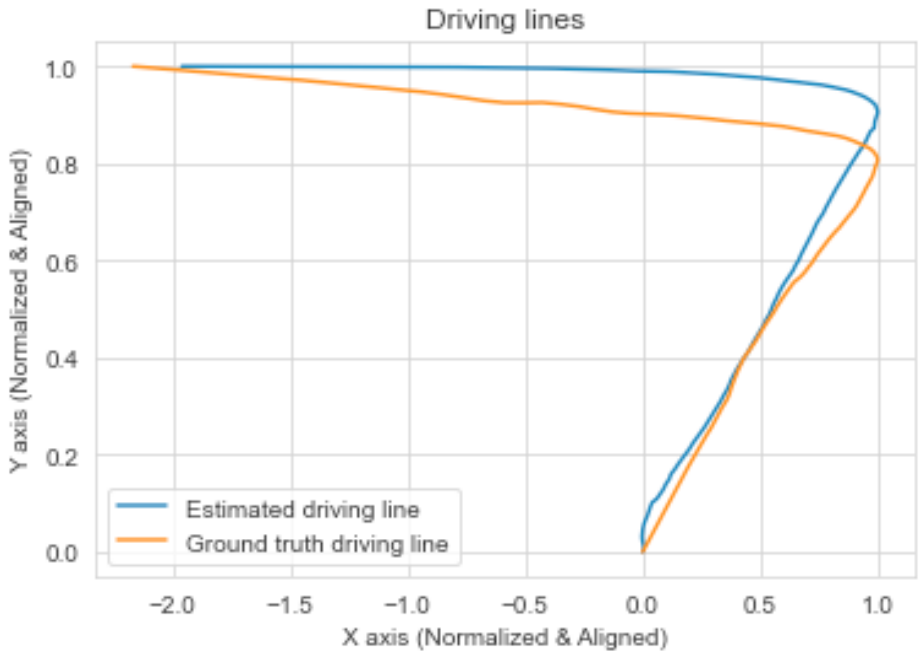
حال برای اینکه بتوانم مسیر حرکت در نقشه را مصور کنم، یک نقطه جدید با z بسیار بالا در نظر گرفته(0,0,100) و تبدیل های متوالی را روی آن انجام و مختصات متناظر را به ازای قرار گیری در هر ابر نقاط بدست می آوریم؛ از جایی که z آن بسیار متفاوت است، در زمان مصور سازی رنگ آن با رنگ نقشه تفاوت فاحش پیدا کرده و مسیر حرکت نمایان می شود. مسیر حرکتی در زیر قابل مشاهده است:



همانطور که قابل مشاهده است مسیر کلی حرکت خودرو کاملا درست بدست آورده شده است. دلیل اندکی خطا در انتهای مسیر میتواند دلایلی متعددی نظیر انباشتگی خطا در تبدیل های متوالی، عدم قرار گیری مختصات لایدار در وسط خودرو(بر خلاف فرض ما برای رسم مسیر، واقعا در (0,0) نباشد!) و ... باشد.

برای مقایسه‌ی نتیجه‌ی بدست آمده، مسیر طی شده را با مسیر حقیقی (ground truth) رسم می کنیم. برای اینکه مسیر و جهت طی شده در دو حالت در یک پایه‌ی مقایسه‌ی یکسان قرار گیرد، هر دو مسیر بین ۱/۰ نرمال و aling شده اند تا تفاوت های بین شان بهتر آشکار شود(به همین جهت توپولوژی رسم شده با مسیر فوق در نقشه ممکن است

اندکی متفاوت بنظر برسد اما دلیل این امر صرفا نرمال سازی و آوردنشان در یک رنج است). نمودار زیر مسیر های حرکتی را نمایش می‌دهد:



همانطور که قابل مشاهده است با وجود انباشتگی‌های خطا و خطاهای ناشی از محاسبه و رند سازی نقشه و ابر نقاط، مسیر حرکتی حاصل با مسیر حرکتی ground truth تطبیق معنایی تقریبا کامل داشته و یکسان استخراج شده است.

بخش اول: تطبیق ابر نقاط با روش NDT

الف) بارگذاری مجموعه داده

در این قسمت نیز از پردازش حاصل از قسمت ICP و فایل ذخیره شده‌ی pikle قسمت قبل استفاده شده است. فقط قابل ذکر است که بر خلاف ICP، در زمان استفاده از ابرنقاط نمونه برداری جدا انجام نشده است.

ب) فرآیند تطبیق و پیاده‌سازی

کلیات روند پیاده سازی برای بدست آوردن تبدیل های بین ابرنقاط متوالی بصورت نسبی همانند روش ICP می باشد و تغییری در فرآیند تطبیق با آن وجود ندارد. برای پیاده‌سازی و یافتن ماتریس تبدیل بین دو ابر نقاط با روش NDT هیچ کتابخانه یا پکیج مناسبی برای زبان پایتون وجود نداشت لذا با ماجرا های بسیار چالشی اقدام به استفاده از کتابخانه pcl زبان برنامه نویسی C++ کردم که نصب و راه اندازی اولیه‌ی آن حدود دو روز زمان گرفت.

بنده به زبان C++ سورس کدی را پیاده سازی کردم که بتواند دو ابرنقاط را از فایل های مربوطه بخواند و سپس روش NDT را برای یافتن تبدیل T بین آن دو اعمال نموده و در نهایت ماتریس تبدیل حاصل را در یک فایل خروجی دهد.

این سورس کد را build و make کرده و در نهایت فایل اجرایی را در کنار فایل های مربوط به ابر نقاط (pcdها) run کرده و لیست ماتریس های تبدیل ایجاد شدند؛ فرآیند نام گذاری فایل ها کاملا شفاف بود و امکان بازیابی اینکه کدام فایل تبدیل با چه نامی مربوط به چه دو ابرنقاط متوالی است وجود داشت. پس از بدست آوردن ماتریس های تبدیل، همگی آنها با زبان برنامه نویسی python خوانده و مورد استفاده قرار گرفت.

لازم به اشاره است که tuning مناسب پارامتر های این روش زمان بسیار زیادی گرفت چرا که زمان پردازش برای بدست آوردن هر تبدیل بالا بوده و انتظار نسبی برای ملاحظه نتیجه بیشتر بود. پارامتر هایی که نیاز به تنظیم داشت عبارت بود از leaf size، epsilon، step size، resolution و max iteration .

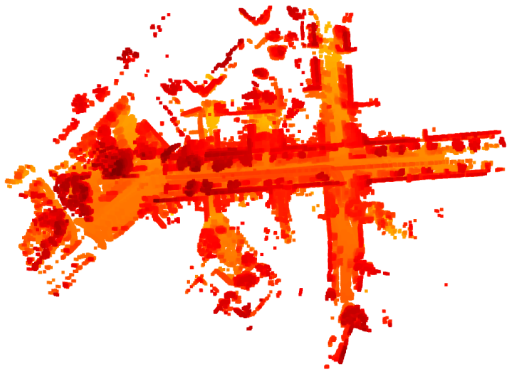
این پارامتر ها بترتیب برابر با ۱، ۰.۰۰۰۰۱، ۰.۰۵، ۱ و ۱۰۰۰۰۰۰۰ در نهایت تنظیم شدند؛ در قسمت تولید نقشه یک نمونه از نتایج نامطلوب حاصل از tuning آورده شده است. نکته ای وجود دارد عدم وجود داکیومنت کامل و مرجع در استفاده از کتابخانه مذکور است. بنده پس از مطالعه سورس تابع پیاده سازی شده متوجه شدم که بایستی چگونه و با چه ارتباطی این پارامتر ها تنظیم شوند. هر گونه تبدیلی که بدست می آوردم بردار انتقال همگی در حد چند سانتی‌متر و نزدیک به صفر بود و این باعث میشد که ابر نقاط متوالی را بصورت پیشفرض کاملا منطبق روی هم بدست آورد و در نتیجه اسکن های مختلف از جاهای مختلف نقشه روی هم رجیستر شود که غلط است.

پس از مطالعه سورس کد متوجه شدم ماتریسی که به عنوان خروجی برمیگرداند با پارامتر leaf size که به عنوان فیلتر اولیه استفاده می شود ارتباط و نسبت دارد و بایستی این پارامتر برابر با ۱ تنظیم شود تا خروجی ماتریس تبدیل را بصورت مستقیم بتوانیم برای تولید نقشه استفاده کنیم و همچنین در صورت نمونه بردار دقت الگوریتم دچار چالش میشود. پس از دانستن این دو نکته پارامتر های مناسب را تعیین و اقدام به محاسبه ماتریس های تبدیل کردم. لازم به ذکر است که سورس C++ و فایل اجرایی خروجی گرفته شده ضمیمه شده است.

ج) ایجاد و بروزرسانی نقشه

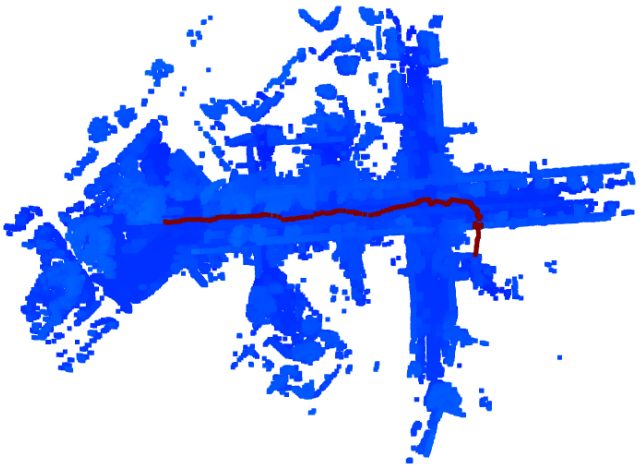
فرآیند تولید و محاسبه نقشه بر اساس ماتریس های تبدیل حاصل از الگوریتم NDT دقیقا مشابه با توضیحات قسمت ICP بوده و هیچ تفاوتی بین آن دو وجود نداشت. در قسمت قبل چالش های پیاده سازی و تعیین پارامتر این الگوریتم بیان شده و فرآیند حل آن نیز توضیح داده شد. در زیر دو تصویر نقشه حاصل از الگوریتم NDT آورده شده است که اولی حاصل از tuning نامطلوب بود که ماتریس های تبدیل نزدیک به ماتریس های همانی حاصل شده و اسکن های مختلف دقیقا روی هم رجیستر می شدند.

تصویر دوم نیز نقشه تولید شده نهایی پس از tuning مناسب پارامتر ها می باشد. همانطور که مشاهده می شود نقشه اصلی تقریبا مشابه با نقشه‌ی ICP حاصل شده است و این شباهت حاکی از صحت پیاده سازی ها می باشد.



د) خروجی و نتایج

حال همانند قسمت مربوط به ICP برای اینکه بتوانم مسیر حرکت در نقشه را مصور کنم، یک نقطه جدید با z بسیار بالا در نظر گرفته(0,0,100) و تبدیل های متوالی را روی آن انجام و مختصات متناظر را به ازای قرار گیری در هر ابر نقاط بدست می آوریم؛ از جایی که z آن بسیار متفاوت است، در زمان مصور سازی رنگ آن با رنگ نقشه تفاوت فاحش پیدا کرده و مسیر حرکت نمایان می شود. مسیر حرکتی در تصویر زیری قابل مشاهده است.

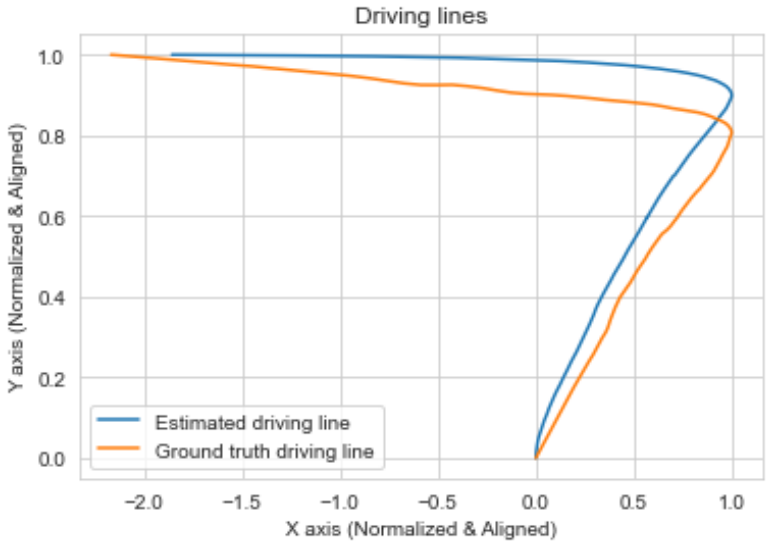


همانطور که قابل مشاهده بوده و مشابه با قسمت ICP می باشد، مسیر کلی حرکت خودرو کاملا درست و هم سو با آن بدست آورده شده است.

دلیل اندکی خطا در انتهای مسیر میتواند دلایلی متعددی نظیر انباشتگی خطا در تبدیل های متوالی، عدم قرار گیری مختصات لایدار در وسط خودرو(بر خلاف فرض ما برای رسم مسیر، واقعا در (0,0) نباشد!) و ... باشد؛ شباهت نوع انحراف در مسیر حرکت دو دو الگوریتم نشان میدهد که طبق چارچوب گفته شده پیاده سازی درست انجام شده است اما شرایط و آگاهی پیش فرض و قبلی برای تخمین مسیر حرکت ناقص است.

همچنین برای مقایسه‌ی نتیجه‌ی بدست آمده، مسیر طی شده را با مسیر حقیقی (ground truth) رسم می کنیم. همانند قسمت ICP برای اینکه مسیر و جهت طی شده در دو حالت در یک پایه‌ی مقایسه‌ی یکسان قرار گیرد، هر دو مسیر بین ۱/۰ نرمال و aling شده اند تا تفاوت های بین شان بهتر آشکار شود(به همین جهت توپولوژی رسم شده با مسیر فوق در نقشه ممکن است اندکی متفاوت بنظر برسد اما دلیل این امر صرفا نرمال سازی و آوردنشان در یک رنج است).

نمودار زیری مسیر های حرکتی را نمایش می‌دهد؛ همانطور که قابل مشاهده است با وجود انباشتگی‌های خطا و خطاهای ناشی از محاسبه و رند سازی نقشه و ابر نقاط و همچنین تاثیر پارامتر های NDT، مسیر حرکتی حاصل با مسیر حرکتی ground truth تطبیق معنایی تقریبا کامل داشته و یکسان استخراج شده است.



همچنین اگر هموار بودن مسیر حرکتی بین هر دو ابر نقاط را در نظر بگیریم، مسیر حرکتی حاصل از NDT نسبت به ICP هموارتر حاصل شده است هر چند بسیار نزدیک بهم نتیجه شده اند.

همچنین در آزمایش دیگری خود ماتریس های تبدیل خصوصا بردار انتقال حاصل از دو الگوریتم را مقایسه میکنیم که به جهت عددی چقدر باهم شباهت یا تفاوت دارند؛ بنده به طور چشمی اکثر بردار های انتقال حاصل از دو الگوریتم یاد شده را ملاحظه کردم که در اکثریت قریب به اتفاق موارد بسیار نزدیک بهم حاصل شده اند؛ این نشان میدهد همگرایی هر دو الگوریتم به درستی انجام شده و هر دو در سطح قابل قبولی توانسته اند به یک state یکسان میل کنند. برای مثال در زیر بردار انتقال مربوط به دو الگوریتم را به ازای ابر نقاط ۴۸ و ۴۷ آورده شده است:

[1.06675034, 0.00944513, 0.02574799] [1.00891, 0.00934935, 0.0108246]