SLAM with Moving Object Removal

Yu Ting Kevin Lai and Shih Ho Hsieh

***Abstract*—In this project, we present a method to incorporate Simultaneous Localization And Mapping (SLAM) with an algorithm to remove moving objects. The purpose is to construct the static map of the real world, whereas SLAM would contain the trajectory of a moving object, which will form a path along the trajectory of that object. Therefore, our main goal is to remove the moving object prior to constructing the map such that the outcome would not contain the dynamic changes, but only static objects in the map.**

***Index Terms*—SLAM, moving object removal.**

I. Introduction

SLAM is a technique to construct the map of the real world environment. The map is useful for localization and tracking in self-driving cars and other autonomous robots. However, the environment is constantly changing, and most of the state-of-the-art algorithms [1] [2] cannot handle this situation perfectly. For example, if a car is approaching on the opposite lane, SLAM will accidently contain the velodyne point cloud of the car consecutively during, which will result in a unexpected path in the constructed map as shown in Fig. 1. The white curved path in the green box indicate the moving bicycle captured by the scan and was accidently put into the map. Hence, one need to update the existing maps accordingly for more accurate results. Therefore, we believe that the constructed maps should only contains static objects. We remove the moving objects from each scan in order to avoid the undesired path in the constructed map. The results show that we can successfully remove moving objects using the NCTU dataset.

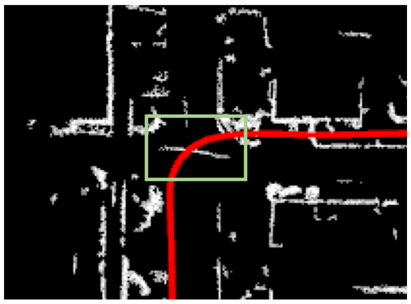


Fig. 1: The unexpected path in the green box of the entire map.

II. Related Work

SLAM technique has been developed for several years, and the methods varied differently. Cvii et al. [3] used stereo visual SLAM to construct the map, while Deschaud [4] proposed a scan-to-model matching framework that produces low-drift SLAM results. However, The method proposed by Zhang et al. [1] achieved the most promising results with low-drift and low-computational complexity. One program performs odometry and estimate the velocity of the lidar, while the other do fine matching and registration of the point cloud. Hence, in our project, we adopt the algorithm of [1] and made several modifications.

Y. T. K. Lai and S. H. Hsieh were with the Department of Electrical and Control Engineering, National Chiao Tung University, Taiwan.

III. The Proposed Method

We proposed an algorithm in Algorithm 1 and 2 to remove the moving objects and are described in the following subsections. The full codes are available at our Github page https://github.com/xlab905/velodyne_slam.

A. Transform Point Cloud Coordinate

Before processing each scan, we first transform the coordinate system of each point from Cartesian coordinate to polar coordinate. The polar coordinate is used because when encountering a moving object in consecutive scans, we can inspect the differences under polar coordinate easier. Algorithm 1 illustrates our transformation procedure with radius initialization. We define theta as the horizontal angle ranging from 0 to 2π of each point, and phi as vertial angle default ranging from $-\pi/2$ to $\pi/2$ of each point. The theta and phi angle division of the lidar scan is shown in Fig. 2.

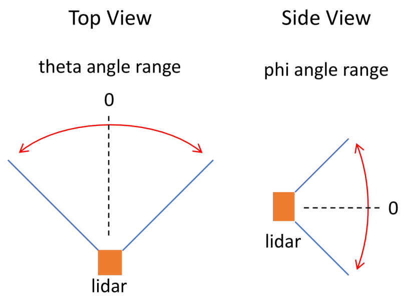


Fig. 2: The theta and phi division of the lidar scan.

First, we take the current scan of Lidar point cloud as the input point cloud. Then calculate the theta and phi angle of each point in the input point cloud. We then find the maximum and minimum bound of theta and phi angle such that we can calculate the index number of the polar coordinate, and store in the $Radius$ data structure, a two-dimensional array, with every element initialized as 0. Note that in $getPhi$ function, we set a $phiRegion$ parameter, which is to constrain the field of view of the vertical range so that we only consider the moving objects in that particular region of interest. This operation can also speed up the calculation because the moving objects in real world mostly appear near the horizon (0 degree). However, $getTheta$ function does not need this extra parameter.

Algorithm 1 Radius Initialization

```
$InputCloud \leftarrow CurrentScan$
for $point$ in $InputCloud$ do
    $\theta \leftarrow getTheta(point)$;                ▷ Get theta angle
    $\phi \leftarrow getPhi(point)$;                    ▷ Get phi angle
    Find theta max and min bound
    Find phi max and min bound
end for
Calculate $theta$ and $phi$ indice number
for $theta$ in *theta indice number* do
    for $phi$ in *phi indice number* do
        $Radius[phi][theta] \leftarrow 0$               ▷ Initialization
    end for
end for
return $Radius$
```

B. Outlier Measurement

Once we get the polar coordinate representation with initialization, we again take the current lidar point cloud as input point cloud. Then get the theta and phi angle of each point according to the resolution parameter calculated previously. Followed by calculating the squared distance ($calcSquaredPointDistance()$) of each indice and store in $Radius$ variable. Then, we remove the edge of $Radius$ with $smoothRadiusScan()$ function, which will be elaborated in section III-C. We further use the data pre-computed by [1], that is, corner features and surface features. Also, theta and phi indexes are computed with the calculation of the squared point distance. Subsequently, we compare the calculated distance with previous scan, if the difference exceed certain threshold, here we called reserveDistance, we consider this point as a moving point and mark as an outlier point to be removed. Next, we use point cloud library (PCL) to perform the filtering to filter out the outlier points. The same procedure is also applied on processing surface features. After precessing this scan, we reset the $Radius$ variable and continues to process the next scan.

C. Smooth Radius

The $Radius$ variable is smoothed by calculating the gradient between previous point and next point. We can calculate the difference of phi angle and theta angle respectively. The center point of the loop is set to $i+1$ for phi, and $j+1$ for theta. After that, the gradient is computed using weighted squared sum of $diffPhi$ and $diffTheta$. We set the $phiDiffFactor$ be always less than 1 because theta angle resolution is more compact so we emphasize more on theta angle. Then we loop over all computed gradients to check if it exceed the threshold, which we called gradThreshold. If true, then this corresponding indice will not be processed in the following codes in Algorithm 2.

Algorithm 2 Moving Object Removal algorithm

```
$InputCloud \leftarrow CurrentScan$
for $point$ in $InputCloud$ do                          ▷ For every point
    Get theta and phi indice
    $Radius \leftarrow calcSquaredPointDistance(point)$
end for
$smoothRadiusScan()$;
for $point$ in $CornerFeaturePoints$ do
    Get theta and phi indice
    $r \leftarrow calcSquaredPointDistance(point)$
    if $r - Radius[phi][theta] <$ reserveDistance then
        Mark as outlier point
    end if
end for
Remove the outliers
for $point$ in $SurfaceFeaturePoints$ do
    Get theta and phi indice
    $r \leftarrow calcSquaredPointDistance(point)$
    if $r - Radius[phi][theta] <$ reserveDistance then
        Mark as outlier point
    end if
end for
Remove the outliers
Reset $Radius$ variable
```

Algorithm 3 Smooth Radius

```
for $i$ in *phi indice number* do
    for $j$ in *theta indice number* do
        $diffPhi \leftarrow$ Calculate phi diff of $i$ and $i+2$ indices
        $thetaPhi \leftarrow$ Calculate theta diff of $i$ and $i+2$ indices
        $grad \leftarrow$ phiDiffFactor $* diffPhi^2 + diffTheta^2$
    end for
end for
for $i$ in *phi indice number* do
    for $j$ in *theta indice number* do
        if $grad >$ gradThreshold then
            Neglect this indice
        end if
    end for
end for
return $Radius$
```

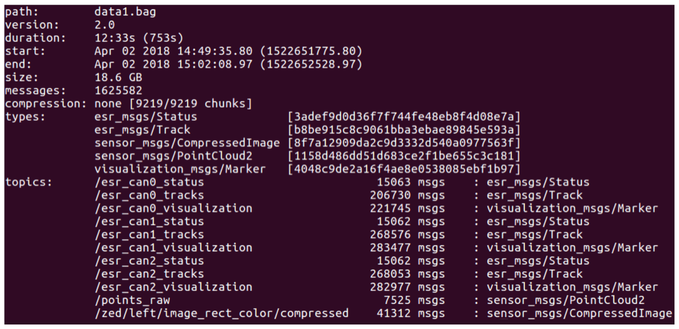
IV. Experiments

A. Dataset & Configuration

For our experiments, we use the datasets provided by the TAs. These datasets would start from one place in the NCTU campus and drive until the car covers the whole campus, then it will end at the same place in the start. The published topics are shown in Fig. 3. There are several parameters to be tuned. First, we set $phiRegion$ mentioned above to 0.5 radians, i.e., the range of phi angle is ± 30 degrees. Moreover, $phiDiffFactor$ set to 0.1. For threshol parameters, reserveDistance threshold in Algorithm 2 is set to 0.2 and gradThreshold in Algorithm 3 is set to 20.

B. Remove Moving Object

Our developed alrogithm can successfully remove the moving objects. Fig. 4 shows the SLAM result before adopting moving object removal algorithm. One car was entering the parking lot straight from bottom-left corner and turn left on



```
path:        data1.bag
version:     2.0
duration:    12:33s (753s)
start:       Apr 02 2018 14:49:35.80 (1522651775.80)
end:         Apr 02 2018 15:02:08.97 (1522652528.97)
size:        18.6 GB
messages:    1625582
compression: none [9219/9219 chunks]
types:       esr_msgs/Status             [3adef9d0d36f7f744fe48eb8f4d08e7a]
             esr_msgs/Track              [b8be915c8c9061bba3ebae89845e593a]
             sensor_msgs/CompressedImage [8f7a12909da2c9d3332d540a0977563f]
             sensor_msgs/PointCloud2     [1158d486dd51d683ce2f1be655c3c181]
             visualization_msgs/Marker   [4048c9de2a16f4ae8e0538085ebf1b97]
topics:      /esr_can0_status                         15063 msgs    : esr_msgs/Status
             /esr_can0_tracks                        206730 msgs    : esr_msgs/Track
             /esr_can0_visualization                 221745 msgs    : visualization_msgs/Marker
             /esr_can1_status                         15062 msgs    : esr_msgs/Status
             /esr_can1_tracks                        268576 msgs    : esr_msgs/Track
             /esr_can1_visualization                 283477 msgs    : visualization_msgs/Marker
             /esr_can2_status                         15062 msgs    : esr_msgs/Status
             /esr_can2_tracks                        268053 msgs    : esr_msgs/Track
             /esr_can2_visualization                 282977 msgs    : visualization_msgs/Marker
             /points_raw                               7525 msgs    : sensor_msgs/PointCloud2
             /zed/left/image_rect_color/compressed    41312 msgs    : sensor_msgs/CompressedImage
```

Fig. 3: The published topics in the datasets.

the bottom-right corner then gradually moved to the top-right corner, while one pedestrian was walking from the middle to the right. We can clearly see the moving paths of both the car and the pedestrian in the map. However, in Fig. 5 we construct the map with moving object removal algorithm and only reserve static objects in the map. For real-time demo video, please referred to https://goo.gl/pobWUe.

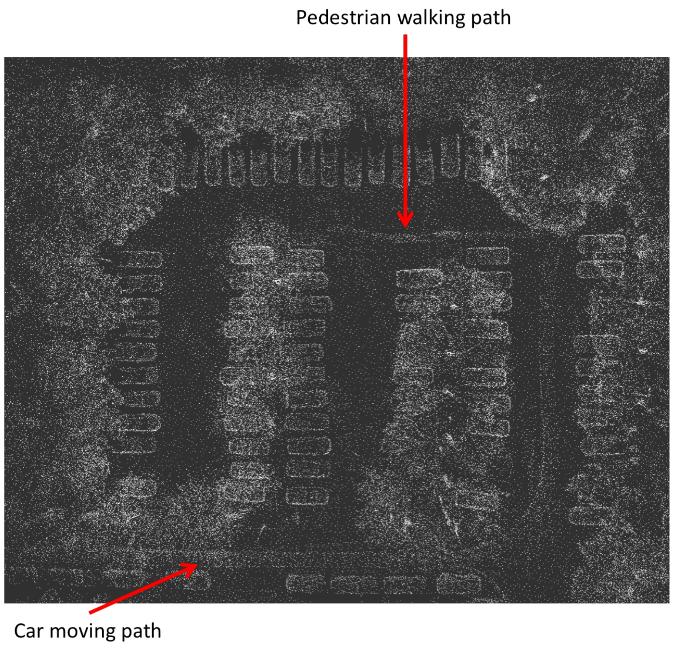


Fig. 4: The map before moving object removal. Note that there are moving paths of a car and a pedestrian.

C. Path Reconstruction

We further investigate on closing the loop when the car finish its journey and drive back to the same place. Fig. 6 shows the loop closure result of the same NCTU campus dataset using. The result, though not perfect, rely on the accuracy of [1] and can reconstruct the driving path of the car.

V. Conclusion

SLAM is critical for self-driving cars, but it cannot construct a reliable map subject to moving object. It will also consider

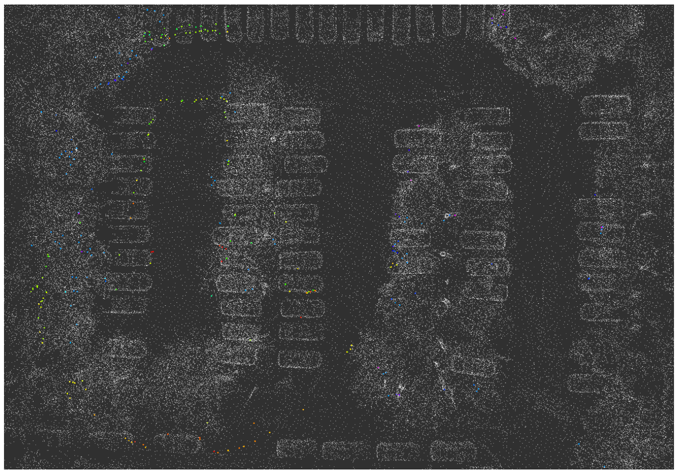


Fig. 5: The map after moving object removal.

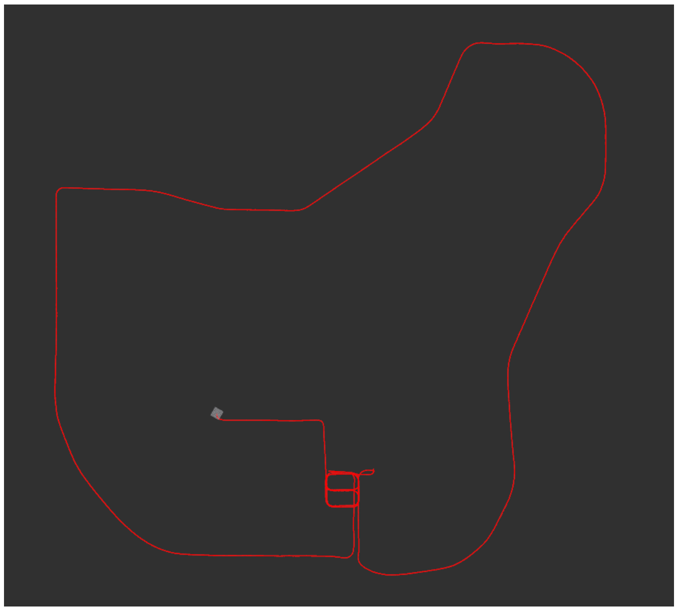


Fig. 6: The path reconstruction result of NCTU campus.

the moving path of a moving object, even using the state-of-the-art algorithms. Thus, in our project, we developed an algorithm aiming at resolving the moving object problem. Our results show that we successfully remove the moving objects and reconstruct the moving path around the entire NCTU campus.

VI. Future Works

Despite the high precision of SLAM results in [1], the precision drops when the car drives in a bigger loop. Therefore, we look forward to combine the loop closure algorithm in [2] with [1] so that the overall result would be more desirable. Moreover, object segments are obtained in [2], in which we can find the object indices more accuracte and further improve the precision of moving object removal.

Acknowledgment

We would like to thank Prof. Chieh-Chih Bob Wang for his advices on our project.

References

[1] J. Zhang and S. Singh, "Loam: Lidar odometry and mapping in real-time," *Robotics: Science and Systems Conference (RSS)*, Jul. 2014.

[2] R. Dubé, A. Cramariuc, D. Dugas, J. I. Nieto, R. Siegwart, and C. Cadena, "Segmap: 3d segment mapping using data-driven descriptors," *CoRR*, vol. abs/1804.09557, 2018. [Online]. Available: http://arxiv.org/abs/1804.09557

[3] I. Cvii, J. esi, I. Markovi, and I. Petrovi, "Soft-slam: Computationally efficient stereo visual slam for autonomous uavs," *Journal of Field Robotics*, vol. 35, no. 4, pp. 578–595, 2017.

[4] J. Deschaud, "IMLS-SLAM: scan-to-model matching based on 3d data," *CoRR*, vol. abs/1802.08633, 2018. [Online]. Available: http://arxiv.org/abs/1802.08633