
 **srpatel2000** fixed formatting issueLatest commit eec6afd 13 seconds agoHistory

4 contributors

datmo_ros2

Introduction

DISCLAIMER: This package was originally built in ROS1 by Kostas Konstantinidis. For information on that please check out this repo: <https://github.com/kostaskonkk/datmo>.

We are students at UCSD who were tasked with using 2D LiDAR perception to deal with the assessment of a vehicle's surroundings for our DSC 190: Introduction to Robotics Course. Our goal was to use the preexisting DATMO package and rebuild it in ROS2 in order to make the package more timeless and better organized. In this repo, you will find our code which uses the same logic as the original DATMO package as seen below.

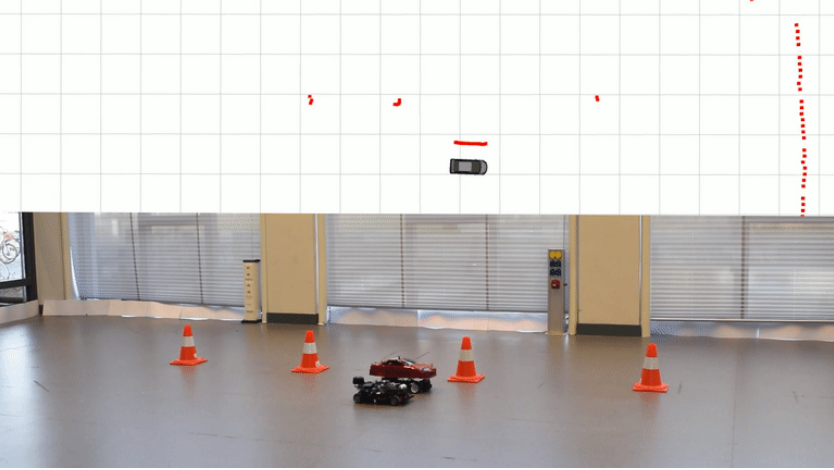
This package is not currently finished and is a work in progress. This is because our goal was simply to output this algorithm using a ROS1 bridge as well as attempt to reduce the amount of errors in the pre-existing ROS2 translated code of DATMO. In the future if this code is going to be used, there is much debugging to be done, however most of the ROS1 code is translated to ROS2.

Begin original explanation written by Kostas Konstantinidis:

Detection and Tracking of Moving Objects with 2D LIDAR

=====

This package aims to provide Detection and Tracking of Moving Objects capabilities to robotic platforms that are equipped with a 2D LIDAR sensor and publish 'sensor_msgs/LaseScan' ROS messages. Such a scenario would be the one visualized below, in which the black scaled car is equipped with a LIDAR sensor and it needs to track the motion of the red vehicle through the LIDAR measurements.



The output of this package is visualized below and it can be observed that it estimates position, velocity, orientation and dimensions of the red vehicle.



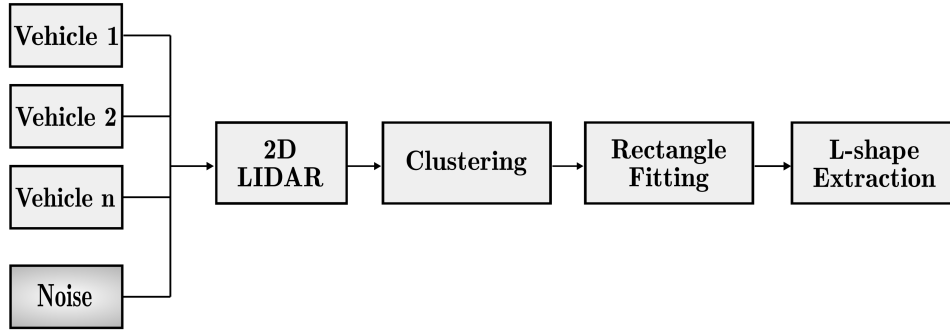
It specializes in tracking of rectangle shaped objects and therefore it is most used in vehicle tracking. The workflow of this package is inspired by the one presented in Kim et al., 2018 [1].

Overview

Below you can read a synopsis of its function and operation. A more in depth explanation of this package's inner workings is given in this [paper](#).

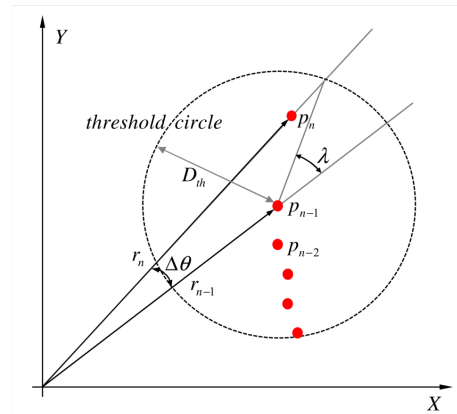
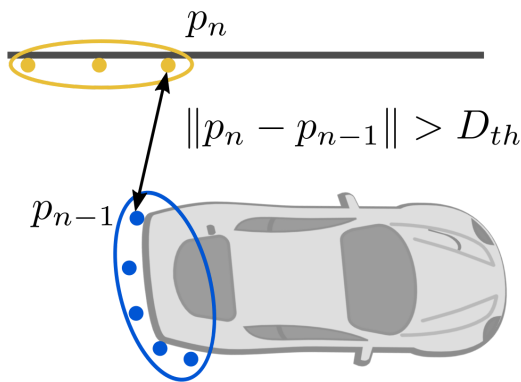
Detection

The detection part of the system is visualized in the following flowchart:



Clustering

In the clustering step the raw LIDAR measurements are divided to groups/clusters. In this way, the different objects in the environment are differentiated. A simple way to do this is by separating clusters, based on the inbetween euclidean distance of LIDAR measurements. Therefore, if the distance of two consecutive LIDAR measurements is greater than a predefined threshold distance the two points are divided in two separate clusters.

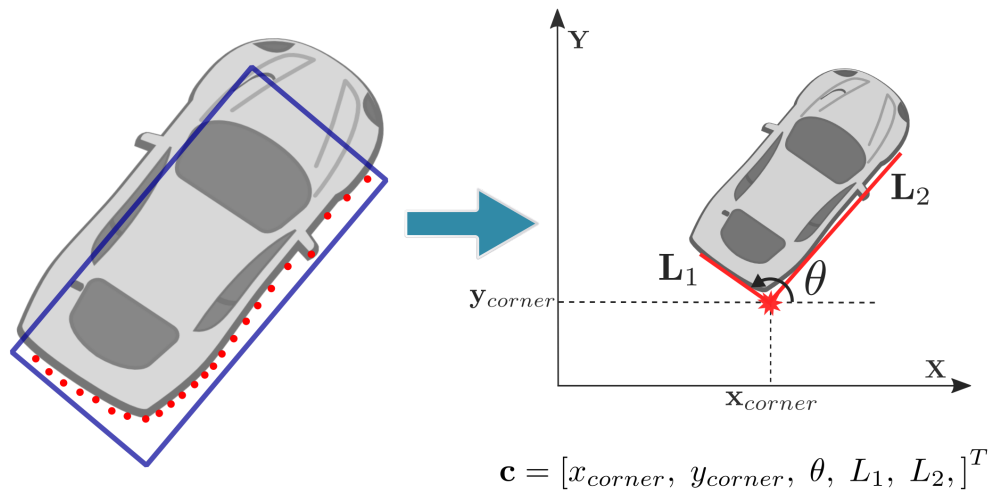


$$D_{th} = \min(r_n, r_{n-1}) \frac{\sin \Delta\theta}{\sin(\lambda - \Delta\theta)}$$

However, since LIDAR measurements become less dense as the distance from the sensor increases, objects For this reason, the threshold distance should be adapted in a way that it increases in relation with the range distance. In this system, this is achieved by using the Adaptive Breakpoint Detector algorithm. Its operation is visualized in the right side of the above figure and the equation that it uses is given below it.

Rectangle Fitting and L-shape extraction

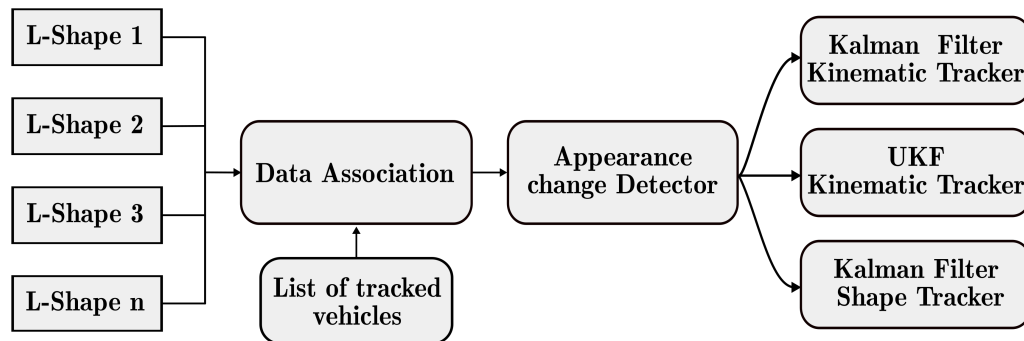
In this step, rectangles are fitted onto the extracted clusters are fitted with rectangles. This is done to increase the tracking accuracy and shape estimation of rectangular objects. The algorithm used for rectangle fitting is the Search-Based Rectangle Fitting algorithm developed by Zhang et al., 2017 [2].



After rectangle fitting, L-shapes are extracted from all the rectangles. L-shapes represent the corner of the rectangle closer to the sensor and its two adjacent sides. Therefore, every L-shape contains five measurements, the position of the corner point, the orientation of the rectangle (theta) and the length of its sides (L_1 , L_2).

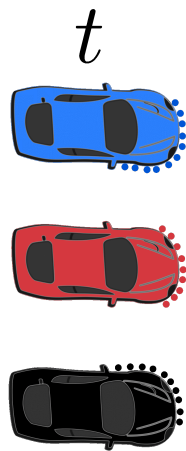
Tracking

The tracking part of the system is visualized in the following flowchart:



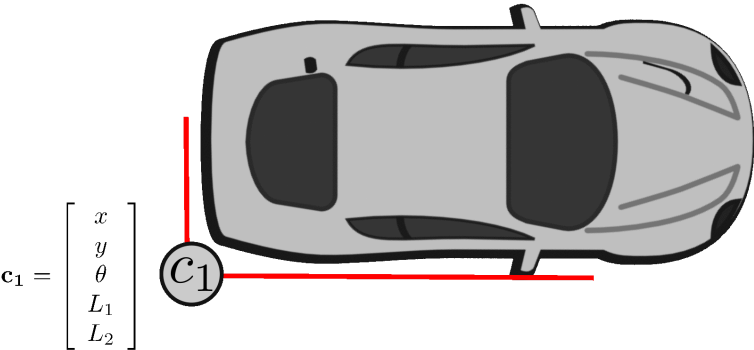
Data Association

The clusters are tracked between time frames by a Nearest Neighbour data association scheme, with a Euclidean distance criterion.



⌘ **Apperance Change Detector**

In cases that the closest corner point of a tracked vehicle changes between measurements, this is detected by comparing the Mahalanobis distance of the four corner points of the vehicle with that of the new L-shape.

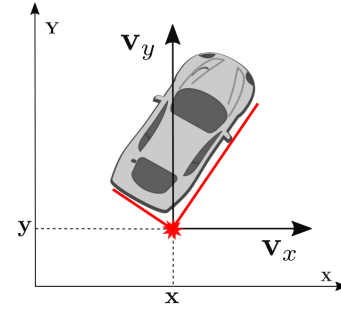


⌘ **Kinematic and Shape Trackers**

The motion of the detected vehicles is tracked based on two kinematic trackers. A Kalman Filter with a Constant Velocity Model and an Unscented Kalman Filter with a Coordinated-Turn model.

Constant Velocity Model - Kalman Filter

$$\underbrace{\begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix}}_{x_t} = \underbrace{\begin{bmatrix} x + v_x T \\ y + v_y T \\ v_x \\ v_y \end{bmatrix}}_{Ax_{t-1}}$$



Coordinated Turn Model - Unscented Kalman Filter

$$\underbrace{\begin{bmatrix} x \\ y \\ v_x \\ v_y \\ \omega \end{bmatrix}}_{x_t} = \underbrace{\begin{bmatrix} x + \frac{v_x}{\omega} \sin(\omega T) - \frac{v_y}{\omega} (1 - \cos(\omega T)) \\ y + \frac{v_x}{\omega} (1 - \cos(\omega T)) + \frac{v_y}{\omega} \sin(\omega T) \\ v_x \cos(\omega T) - v_y \sin(\omega T) \\ v_x \sin(\omega T) + v_y \cos(\omega T) \\ \omega \end{bmatrix}}_{f(x_{t-1})}$$

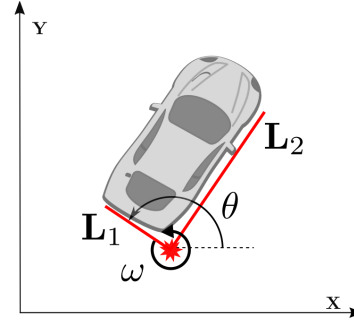
$$z = \begin{bmatrix} x & y \end{bmatrix}^T$$

The shape and orientation of the detected vehicles are tracked by a Kalman Filter that contains two models. The first model is a Constant Shape model and it indicates that the shape of the detected vehicle remains constant. The second model is a Constant Turn Rate model that indicates that the turn rate of the detected vehicle remains constant, while its orientation depends on the turn rate.

Kalman Filter with Constant Shape & Constant Turn Rate Models

$$\underbrace{\begin{bmatrix} L_1 \\ L_2 \\ \theta \\ \omega \end{bmatrix}}_{x_t} = \underbrace{\begin{bmatrix} L_1 \\ L_2 \\ \theta + \omega T \\ \omega \end{bmatrix}}_{Ax_{t-1}}$$

$$z = \begin{bmatrix} L_1 & L_2 & \theta \end{bmatrix}^T$$



Below you can find a video of a presentation of mine, in which I explain some early features of this package.



🔗 Installation and use

This ROS package can be installed in the following way:

1. First you should navigate to the source folder of your catkin_ws. For example `cd ~/catkin_ws/src.`
2. Run

```
git clone git@github.com:kostaskonkk/datmo.git
cd ..
catkin_make
```

The datmo package should be now installed to your computer and you will be able to use it after sourcing your workspace.

```
source devel/setup.bash
```

You can run a demonstration of the DATMO package by running:

```
roslaunch datmo example.launch bag:=overtakes
```

You can run it by typing:

```
roslaunch datmo datmo.launch
```

ROS API

Subscribed Topics

`scan(sensor_msgs/LaserScan)` - This topic should be created by your LIDAR sensor.

Published Topics

This node can publish a variety of topics but the final configuration depends on the user. By default the majority of the topics are disabled and they should be enabled through the launch file configuration.

datmo/marker_array (visualization_msgs/MarkerArray) - In this topic a variety of Rviz markers are published, which can facilitate in understanding the inner workings of the program.

datmo/box_kf (datmo/TrackArray) - In this topic the output of a Kalman Filter with a Constant Velocity model, which tracks the center of the box that surrounds the clusters is published.

Note: In case that the `marker_array` topic is published from a robot and visualized in computer, which has a different version of ROS installed (kinetic, melodic, ...), the `msgs` will not be published and the `datmo` node will crash with an `md5sum` error. To mitigate this, you should install on your robot the `visualization_msgs` package of the ROS installation that runs on your computer.

Custom Messages

This package uses two custom msg types `datmo/Track` and `datmo/TrackArray` to facilitate the publishing of its results. To my knowledge, at the time of development, there was no standard ROS messages that accomplish the same task.

The `datmo/Track` message has the following structure:

`int32 id` - object ID, so it is possible to differentiate between different objects during tracking

`float32 length` - Estimated length of the object

`float32 width` - Estimated width of the object

`nav_msgs/Odometry odom` - Estimated pose of the object

The `datmo/TrackArray` message is an array that contains multiple `datmo/Track` messages, with the goal of efficient publishing.

Rviz markers

In case that the **pub_markers** flag is set to true, this package publishes visualization messages, which can be displayed in Rviz. The following messages are published:

closest_corner - The closest corner point of surrounding vehicles is visualized with a black rectangle.

bounding_box_center - The center of the bounding box is visualized with a yellow rectangle.

velocities - The velocities of the tracked objects are represented with an arrow.

Parameters

- "lidar_frame" ("string", default: "laser") - Name of the transformation frame (`frame_id`) of the `LaserScan` msgs
- "world_frame" ("string", default: "world") - Name of the world coordinate frame, if it is not available, this value can be set equal to the `lidar_frame`
- "threshold_distance" ("double", default: "0.2") - This value sets the distance that is used by the clustering algorithm
- "euclidean_distance" ("double", default: "0.25") - This value sets the distance that is used by the euclidean distance data association algorithm
- "pub_markers" ("bool", default: "false") - publish of the visualization markers

References

- [1] D. Kim, K. Jo, M. Lee, and M. Sunwoo, "L-shape model switching-based precise motion tracking of moving vehicles using laser scanners," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 2, pp. 598–612, 2018.
- [2] X. Zhang, W. Xu, C. Dong, and J. M. Dolan, "Efficient l-shape fitting for vehicle detection using laser scanners," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 54–59, IEEE, 2017.

🔗 Development timeline

=====

🔗 Week 7: Preparation

- Cloning DATMO from Github into the Jetson
- Installing all the relevant packages on our Jetson
 - Updating package dependencies in a docker container
- Researching a bridge between ROS1 and ROS2 using ROS1_bridge
- Reasoning: the package is built on ROS1 (and our LIDAR is published in ROS2)
- Trying to run DATMO on our robot by reviewing DATMO README and following the steps to successfully launch it
-

=====

🔗 Week 8 Part 1: Theory

- Understanding the logic of the DATMO Package:
 - Reading through the research paper associated with the DATMO package
 - Understanding the theory and logic
 - Figuring out the input and output
 - Identifying issues that can happen in a real race
 - Watching the tutorial presentation of the package

🔗 Week 8 Part 2: Code Testing

- Coding and Robot Testing:
 - Producing Visualization of DATMO using RVIZ
 - Confirming the DATMO outputs are accurate through RVIZ
 - Visualizing real life outputs in the RVIZ
 - Writing documentation for setting up ROS1_bridge and DATMO with our robot (For Team 2 to use)
 - Understanding general output structure of DATMO
 - x, y, z point arguments
 - r, g, b, a color arguments

🔗 Week 8 Part 3: Communication

- Communication with Team 2
 - Connecting with Team 2 to let them know what is the output of the package
 - Sending the DATMO research paper with annotations to better understand package
 - Sending our DATMO documentation for how to run the package on a Jetson using a ROS1_bridge to Team 2

=====

🔗 Week 9: Start Translating DATMO to ROS2

- Researching how to update the DATMO package
 - Searching instructions on how to translate ROS1 DATMO package to a ROS2 version
 - Creating a new ROS2 package using the ROS1 DATMO code
 - Translating and updating the files from the old package to the newly created package

🔗 Week 9: Code Translation

- Researching library changes between ROS1 to ROS2 and their usage in ROS2 packages
 - EX: Replacing the call of the libraries in hpp/cpp files with the updated ROS2 version
- Replacing the old package dependencies of ROS1 with the latest ROS2 packages and also updating the associated code
- Updating depreciated C++ libraries
- Files that were translated:
 - Source files:
 - main.cpp
 - cluster.cpp & cluster.hpp
 - datmo.cpp & datmo.hpp
 - l_shape_tracker.cpp & l_shape_tracker.hpp
 - kalman.cpp & kalman.hpp
 - CMakeLists.txt
 - package.xml

=====

🔗 Week 10 and Finals Week: Debugging Issues (aka Challenges Faced)

- Debugging the updated ROS2 files !!!
 - We are inexperienced in C++ and understanding differences between ROS2 and ROS1
 - Difficulty in interpreting errors
 - ROS2 error? C++ error?
 - Package dependency error?
 - Compilation error?
 - Needed extensive help in debugging
 - Tanmay and Sid were extremely helpful but rightfully not always available

