tysik / obstacle_detector

A ROS package for 2D obstacle detection based on laser range data.

29 commits **2** branches **0** releases **2** contributors

Branch: **master** New pull request Create new file Upload files Find file Clone or download

tysik Merge pull request #6 from milesial/master ··· Latest commit `bd59d40` on May 1

icons/classes	Initialized repository.	a year ag
include/obstacle_detector	Extractor: obstacles are now correctly transformed	4 months ag
launch	Turned off obstacle publisher	a year ag
msg	Initialized repository.	a year ag
resources	Put obstacle publisher to front	a year ag
src	Removed unused variable	4 months ag
.gitignore	Initialized repository.	a year ag
CMakeLists.txt	Added destructors and new params	a year ag
README.md	Added frames in obstacle publisher panel	a year ag
nodelet_plugins.xml	Removed scan_rectifier in favor of using laser_geometry in scans_merger	a year ag

package.xml	Added destructors and new params	a year ag
rviz_plugins.xml	Initialized repository.	a year ag

README.md

The obstacle_detector package

The `obstacle_detector` package provides utilities to detect and track obstacles from data provided by 2D laser scanners. Detected obstacles come in a form of line segments or circles. The package was designed for a robot equipped with two laser scanners therefore it contains several additional utilities. The working principles of the method are described in an article provided in the `resources` folder.

The package requires Armadillo C++ library for compilation and runtime.

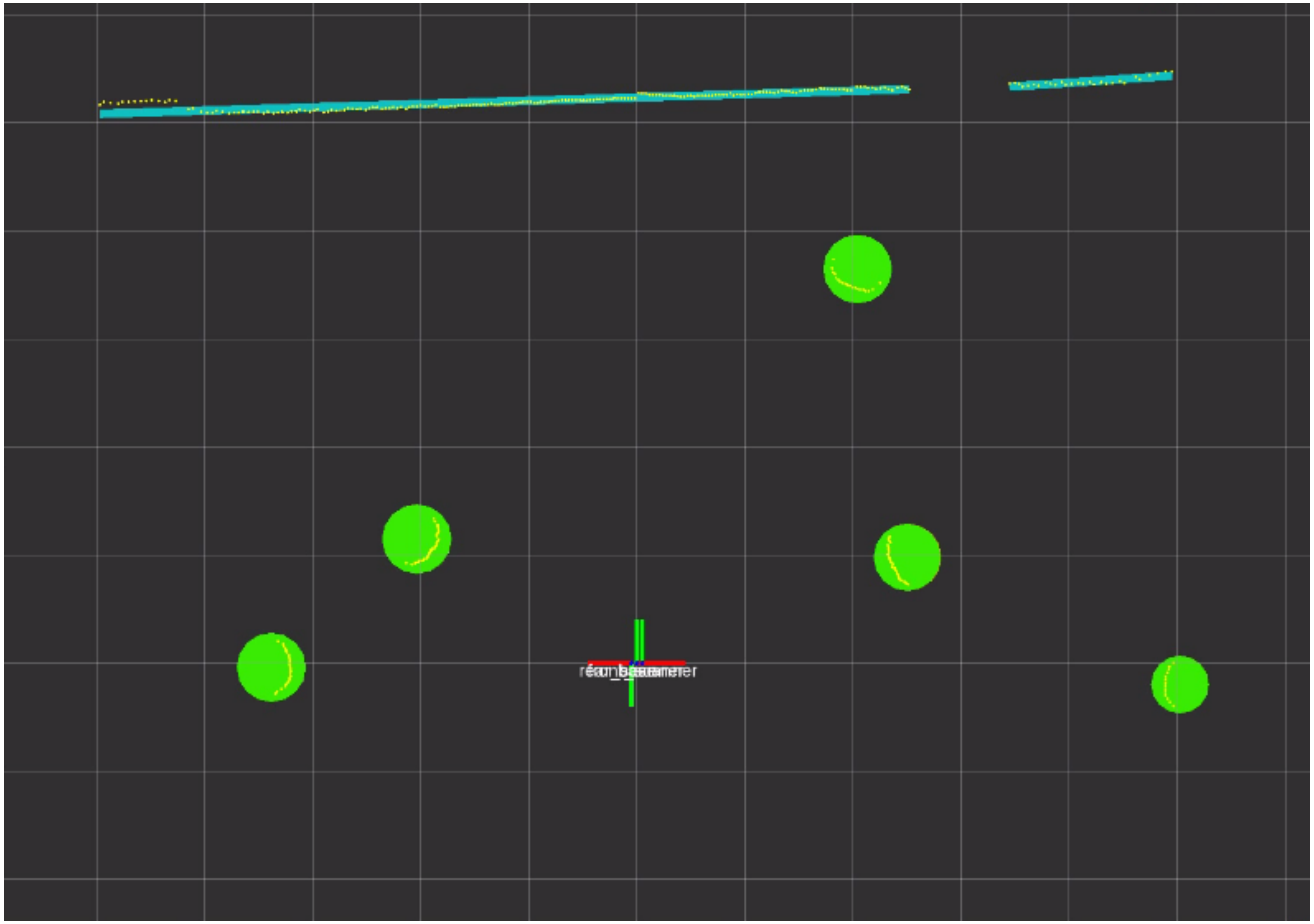


Fig. 1. Visual example of obstacle detector output.

1. The nodes and nodelets

The package provides separate nodes/nodelets to perform separate tasks. In general solution the data is processed in a following manner:

`two laser scans -> scans merger -> merged scan or pcl -> obstacle extractor -> obstacles -> obstacle tracker -> refined obstacles`

For some scenarios the pure obstacle extraction directly from a laser scan (without tracking) might be sufficient.

The nodes are configurable with the use of ROS parameter server. All of the nodes provide a private `params` service, which allows the process to get the latest parameters from the parameter server.

All of the nodes can be in either active or sleep mode, triggered by setting the appropriate variable in the parameter server and calling `params` service. In the sleep mode, any subscribers or publishers are shut down and the node does nothing until activated again.

For the ease of use it is recomended to use appropriate Rviz panels provided for the nodes with the package. The Rviz panels communicate via parameter server and service-client calls, therefore the names of the nodes must be preserved unchanged (cf. launch files for examples).

1.1. The scans_merger node

This node converts two messages of type `sensor_msgs/LaserScan` from topics `front_scan` and `rear_scan` into a single laser scan of the same type, published under topic `scan` and/or a point cloud of type `sensor_msgs/PointCloud` , published under topic `pcl` . The difference between both is that the resulting laser scan divides the area into finite number of circular sectors and put one point (or actually one range value) in each section occupied by some measured points, whereas the resulting point cloud simply copies all of the points obtained from sensors.

The input laser scans are firstly rectified to incorporate the motion of the scanner in time (see `laser_geometry` package). Next, two PCLs obtained from the previous step are synchronized and transformed into the target coordinate frame at the current point in time.

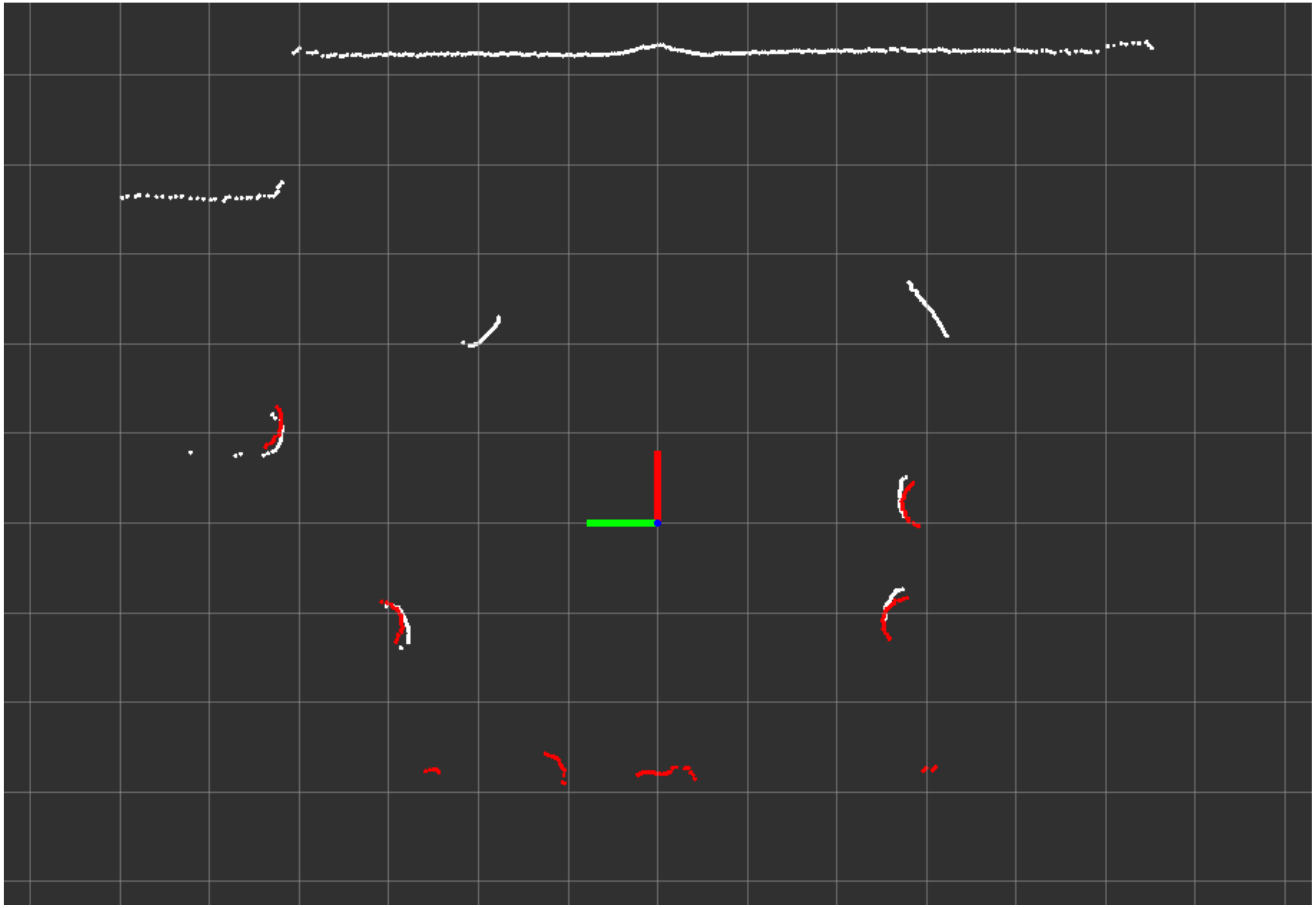


Fig. 2. Visual example of `scans_merger` output.

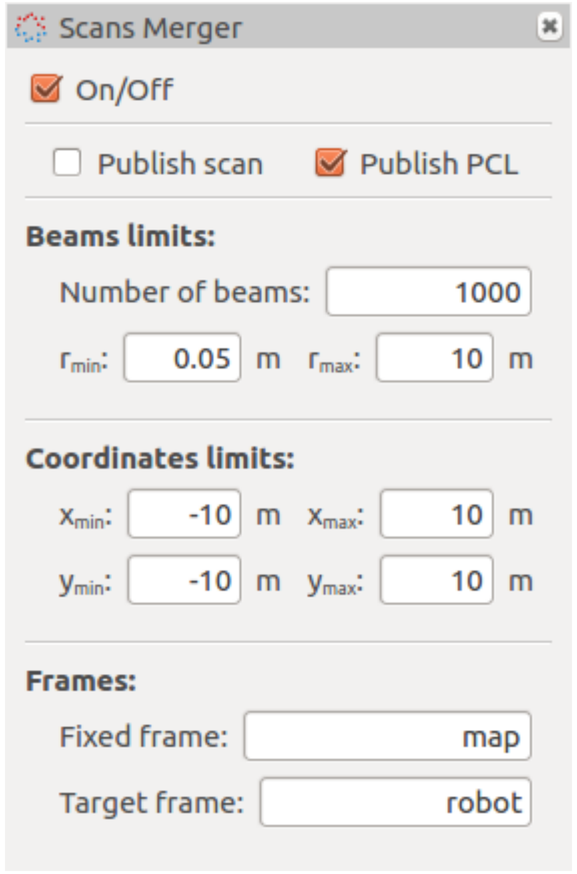
The resulting messages contain geometric data described with respect to a specific coordinate frame (e.g. `robot`). Assuming that the coordinate frames attached to two laser scanners are called `front_scanner` and `rear_scanner`, both transformation from `robot` frame to `front_scanner` frame and from `robot` frame to `rear_scanner` frame must be provided. The node allows to artificially restrict measured points to some rectangular region around the `robot` frame as well as to limit the resulting laser scan range. The points falling behind this region will be discarded.

Even if only one laser scanner is used, the node can be useful for simple data pre-processing, e.g. rectification, range restriction or recalculation of points to a different coordinate frame. The node uses the following set of local parameters:

- `~active` (`bool`, default: `true`) - active/sleep mode,
- `~publish_scan` (`bool`, default: `false`) - publish the merged laser scan message,
- `~publish_pcl` (`bool`, default: `true`) - publish the merged point cloud message,
- `~ranges_num` (`int`, default: `1000`) - number of ranges (circular sectors) contained in the 360 deg laser scan message,
- `~min_scanner_range` (`double`, default: `0.05`) - minimal allowable range value for produced laser scan message,
- `~max_scanner_range` (`double`, default: `10.0`) - maximal allowable range value for produced laser scan message,
- `~min_x_range` (`double`, default: `-10.0`) - limitation for points coordinates (points with coordinates behind these limitations will be discarded),
- `~max_x_range` (`double`, default: `10.0`) - as above,
- `~min_y_range` (`double`, default: `-10.0`) - as above,
- `~max_y_range` (`double`, default: `10.0`) - as above,
- `~fixed_frame_id` (`string`, default: `map`) - name of the fixed coordinate frame used for scan rectification in time,

- `~target_frame_id` (`string` , default: `robot`) - name of the coordinate frame used as the origin for the produced laser scan or point cloud.

The package comes with Rviz panel for this node.



Scans Merger

[✓] On/Off

[ ] Publish scan [✓] Publish PCL

Beams limits:

Number of beams: 1000

r_{min}: 0.05 m r_{max}: 10 m

Coordinates limits:

x_{min}: -10 m x_{max}: 10 m

y_{min}: -10 m y_{max}: 10 m

Frames:

Fixed frame: map

Target frame: robot

Fig. 3. Rviz panel for the `scans_merger` node.

1.2. The obstacle_extractor node

This node converts messages of type `sensor_msgs/LaserScan` from topic `scan` or messages of type `sensor_msgs/PointCloud` from topic `pcl` into obstacles which are published as messages of custom type `obstacles_detector/Obstacles` under topic `raw_obstacles` . The PCL message must be ordered in the angular fashion, because the algorithm exploits the polar nature of laser scanners.

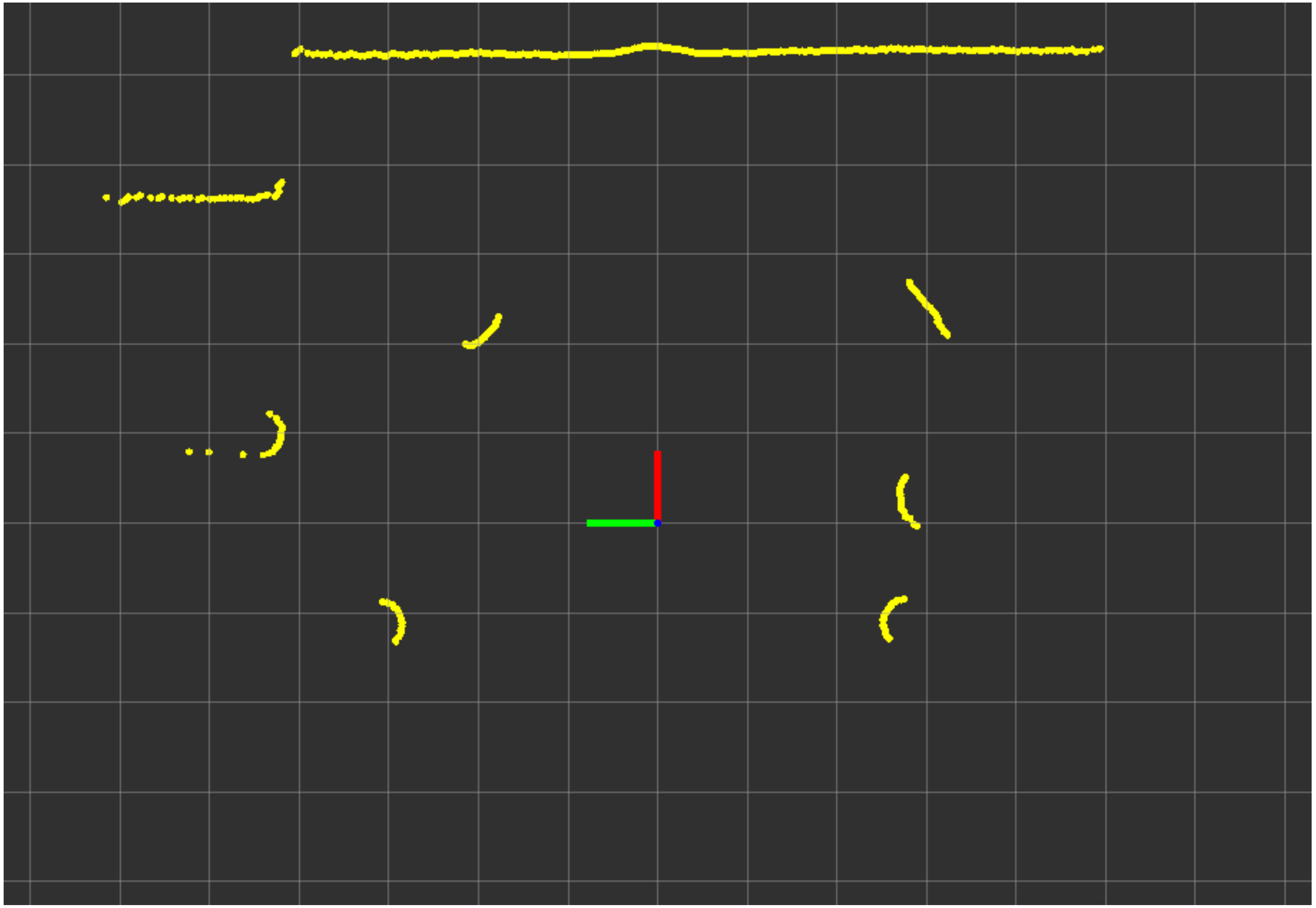


Fig. 4. Visual example of `obstacle_extractor` output.

The input points are firstly grouped into subsets and marked as visible or not (if a group is in front of neighbouring groups, it is visible. Otherwise it is assumed to be occluded). The algorithm extracts segments from each points subset. Next, the segments are checked for possible merging between each other. The circular obstacles are then extracted from segments and also merged if possible. Resulting set of obstacles can be transformed to a dedicated coordinate frame.

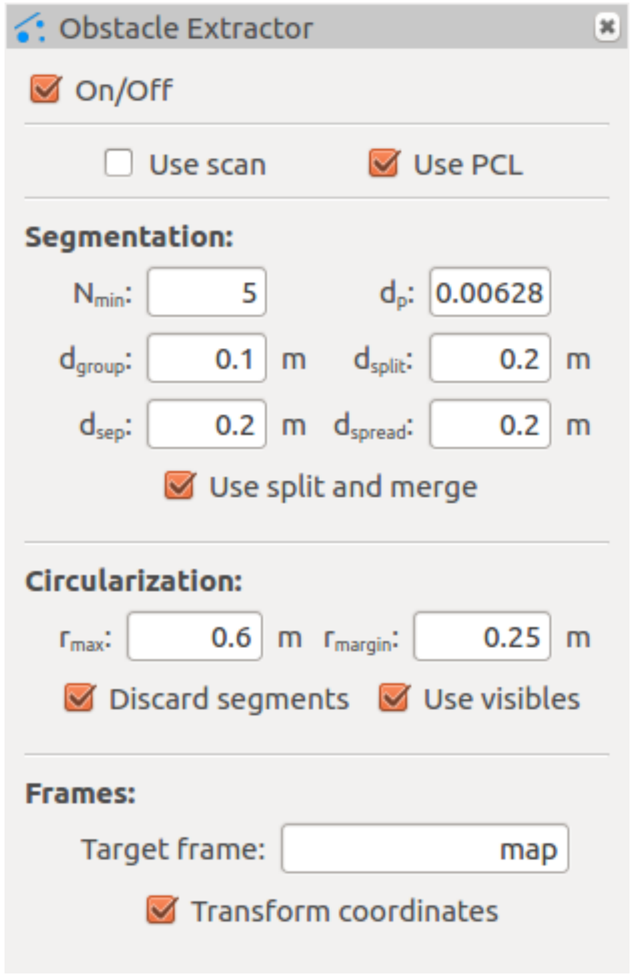
The node is configurable with the following set of local parameters:

- `~active` (`bool` , default: `true`) - active/sleep mode,
- `~use_scan` (`bool` , default: `false`) - use laser scan messages,
- `~use_pcl` (`bool` , default: `true`) - use point cloud messages (if both scan and pcl are chosen, scans will have priority),
- `~use_split_and_merge` (`bool` , default: `true`) - choose wether to use Iterative End Point Fit (false) or Split And Merge (true) algorithm to detect segments,
- `~circles_from_visibles` (`bool` , default: `true`) - detect circular obstacles only from fully visible (not occluded) segments,
- `~discard_converted_segments` (`bool` , default: `true`) - do not publish segments, from which the circles were spawned,
- `~transform_coordinates` (`bool` , default: `true`) - transform the coordinates of obstacles to a frame described with `frame_id` parameter,
- `~min_group_points` (`int` , default: `5`) - minimum number of points comprising a group to be further processed,
- `~max_group_distance` (`double` , default: `0.1`) - if the distance between two points is greater than this value, start a new group,
- `~distance_proportion` (`double` , default: `0.00628`) - enlarge the allowable distance between points proportionally to the

range of point (use scan angle increment in radians),

- `~max_split_distance` (`double` , default: `0.2`) - if a point in group lays further from a leading line than this value, split the group,
- `~max_merge_separation` (`double` , default: `0.2`) - if distance between obstacles is smaller than this value, consider merging them,
- `~max_merge_spread` (`double` , default: `0.2`) - merge two segments if all of their extreme points lay closer to the leading line than this value,
- `~max_circle_radius` (`double` , default: `0.6`) - if a circle would have greater radius than this value, skip it,
- `~radius_enlargement` (`double` , default: `0.25`) - artificially enlarge the circles radius by this value,
- `~frame_id` (`string` , default: `map`) - name of the coordinate frame used as origin for produced obstacles (used only if `transform_coordinates` flag is set to true).

The package comes with Rviz panel for this node.


Obstacle Extractor

[✓] On/Off

[ ] Use scan [✓] Use PCL

Segmentation:

N_{min}: 5 d_p: 0.00628

d_{group}: 0.1 m d_{split}: 0.2 m

d_{sep}: 0.2 m d_{spread}: 0.2 m

[✓] Use split and merge

Circularization:

r_{max}: 0.6 m r_{margin}: 0.25 m

[✓] Discard segments [✓] Use visibles

Frames:

Target frame: map

[✓] Transform coordinates

Fig. 5. Rviz panel for the `obstacle_extractor` node.

1.3. The obstacle_tracker node

This node tracks and filters the circular obstacles with the use of Kalman filter. It subscribes to messages of custom type `obstacle_detector/Obstacles` from topic `raw_obstacles` and publishes messages of the same type under topic `tracked_obstacles`. The tracking algorithm is applied only to the circular obstacles, hence the segments list in the published message is simply a copy of the original segments. The tracked obstacles are supplemented with additional information on their velocity.

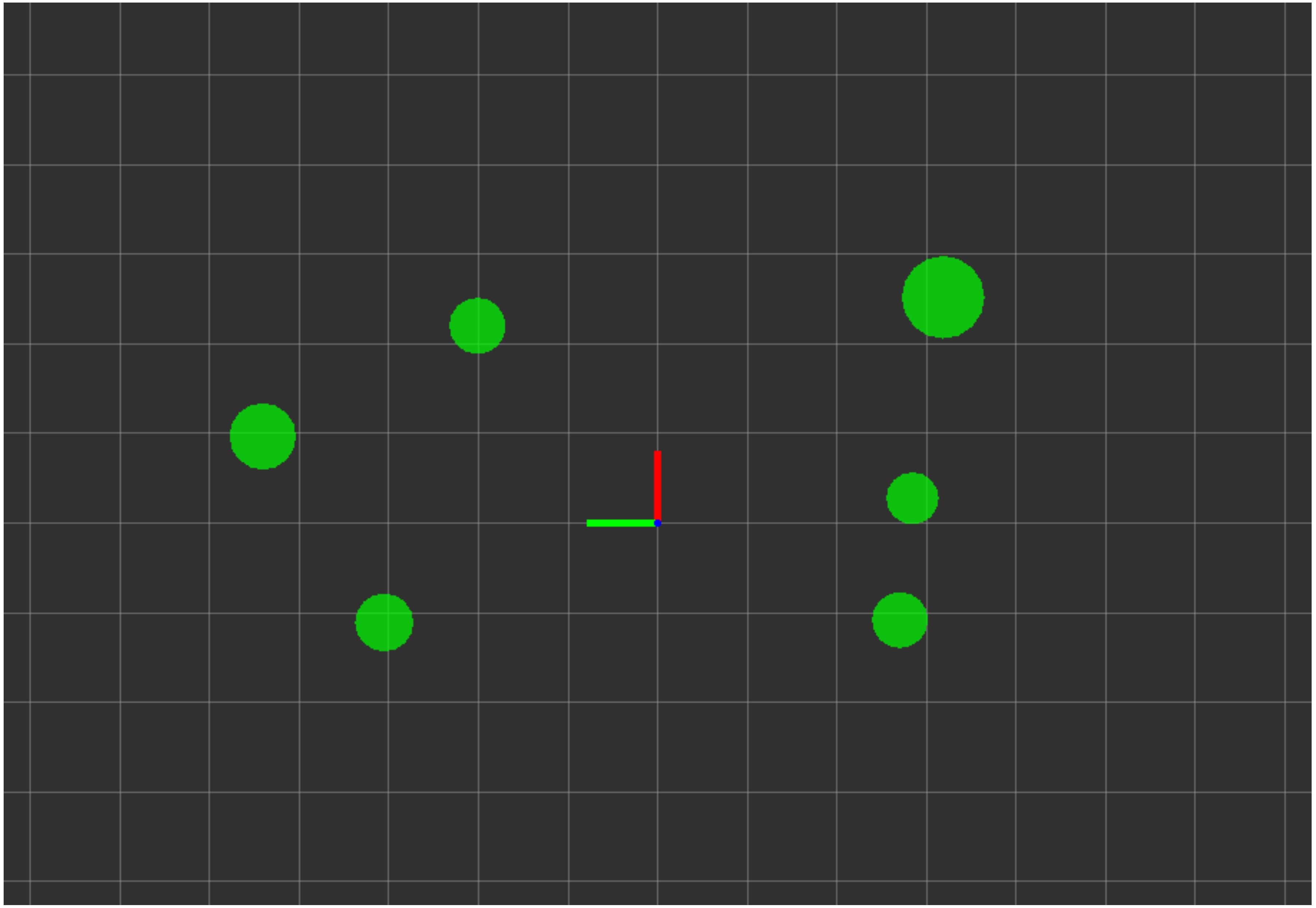
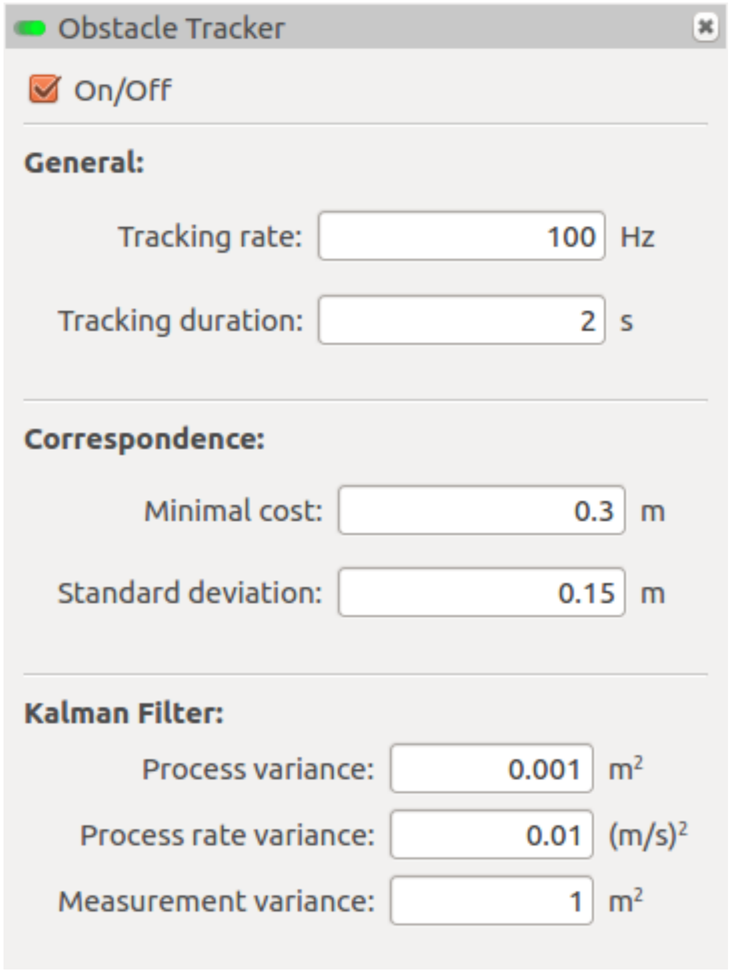


Fig. 6. Visual example of `obstacle_tracker` output.

The node works in a synchronous manner with the default rate of 100 Hz. If detected obstacles are published less often, the tracker will supersample them and smoothen their position and radius. The following set of local parameters can be used to tune the node:

- `~active` (`bool` , default: `true`) - active/sleep mode,
- `~copy_segments` (`bool` , default: `true`) - copy detected segments into tracked obstacles message,
- `~loop_rate` (`double` , default: `100.0`) - the main loop rate in Hz,
- `~tracking_duration` (`double` , default: `2.0`) - the duration of obstacle tracking in the case of lack of incomming data (after this time from the last corresponding measurement the tracked obstacle will be removed from the list),
- `~min_correspondence_cost` (`double` , default `0.3`) - a threshold for correspondence test,
- `~std_correspondence_dev` (`double` , default `0.15`) - (experimental) standard deviation of the position ellipse in the correspondence test,
- `~process_variance` (`double` , default `0.01`) - variance of obstacles position and radius (parameter of Kalman Filter),
- `~process_rate_variance` (`double` , default `0.1`) - variance of rate of change of obstacles values (parameter of Kalman Filter),
- `~measurement_variance` (`double` , default `1.0`) - variance of measured obstacles values (parameter of Kalman Filter),
- `~frame_id` (`string` , default: `map`) - name of the coordinate frame in which the obstacles are described,

The package comes with Rviz panel for this node.


Obstacle Tracker

☑ On/Off

General:

Tracking rate:	100	Hz
Tracking duration:	2	s

Correspondence:

Minimal cost:	0.3	m
Standard deviation:	0.15	m

Kalman Filter:

Process variance:	0.001	m^2
Process rate variance:	0.01	$(m/s)^2$
Measurement variance:	1	m^2

Fig. 7. Rviz panel for the `obstacle_tracker` node.

1.4. The obstacle_publisher node

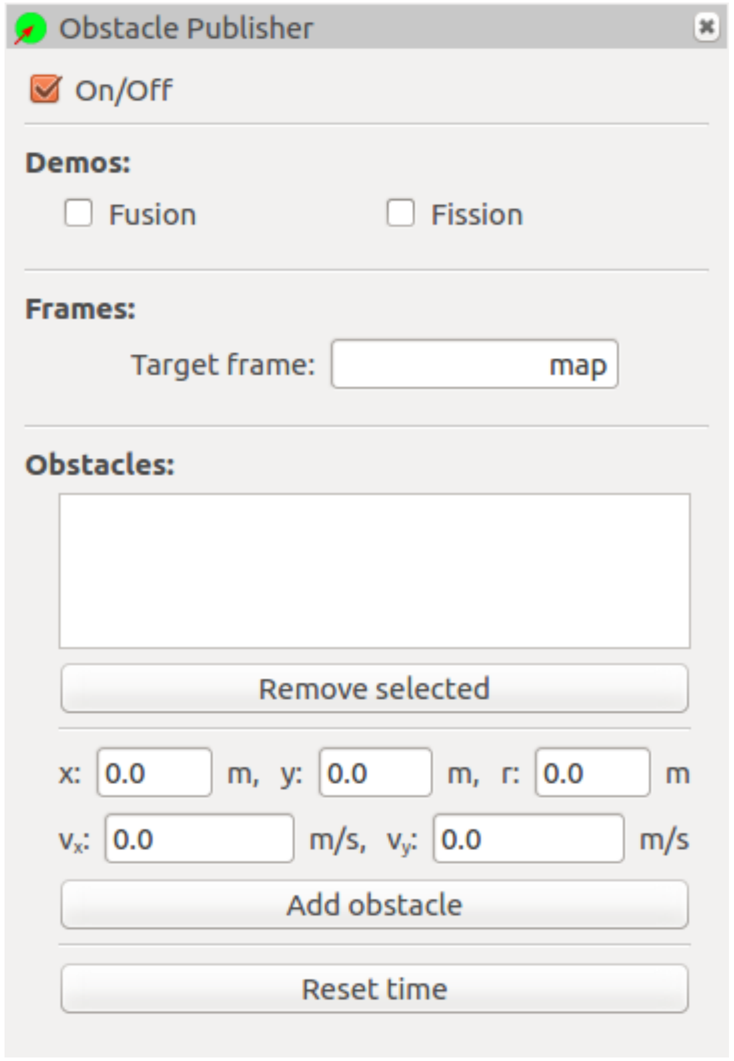
The auxiliary node which allows to publish a set of virtual, circular obstacles in the form of message of type `obstacles_detector/Obstacles` under topic `obstacles`. The node is mostly used for off-line tests. The following parameters are used to configure the node:

- `~active` (`bool` , default: `true`) - active/sleep mode,
- `~reset` (`bool` , default: `false`) - reset time for obstacles motion calculation (used by dedicated Rviz panel),
- `~fusion_example` (`bool` , default: `false`) - produce obstacles showing fusion,
- `~fission_example` (`bool` , default: `false`) - produce obstacles showing fission,
- `~radius_margin` (`double` , default: `0.25`) - artificially enlarge the circles radius by this value in meters,
- `~loop_rate` (`double` , default: `10.0`) - the main loop rate in Hz,
- `~frame_id` (`string` , default: `map`) - name of the coordinate frame in which the obstacles are described.

The following parameters are used to provide the node with a set of obstacles:

- `~x_vector` (`std::vector<double>` , default: `[]`) - the array of x-coordinates of obstacles center points,
- `~y_vector` (`std::vector<double>` , default: `[]`) - the array of y-coordinates of obstacles center points,
- `~r_vector` (`std::vector<double>` , default: `[]`) - the array of obstacles radii,
- `~x_vector` (`std::vector<double>` , default: `[]`) - the array of velocities of obstacles center points in x direction,
- `~y_vector` (`std::vector<double>` , default: `[]`) - the array of velocities of obstacles center points in y direction.

The package comes with Rviz panel for this node.


Obstacle Publisher

☑ On/Off

Demos:

☐ Fusion ☐ Fission

Frames:

Target frame: map

Obstacles:

x: 0.0 m, y: 0.0 m, r: 0.0 m

v_x: 0.0 m/s, v_y: 0.0 m/s

Fig. 8. Rviz panel for the `obstacle_publisher` node.

2. The messages

The package provides three custom message types. All of their numerical values are provided in SI units.

- `CircleObstacle`
  - `geometry_msgs/Point center` - center of circular obstacle,
  - `geometry_msgs/Vector3 velocity` - linear velocity of circular obstacle,
  - `float64 radius` - radius of circular obstacle with added safety margin,
  - `float64 true_radius` - measured radius of obstacle without the safety margin.
- `SegmentObstacle`
  - `geometry_msgs/Point first_point` - first point of the segment (in counter-clockwise direction),
  - `geometry_msgs/Point last_point` - end point of the segment.
- `Obstacles`
  - `Header header`
  - `obstacle_detector/SegmentObstacle[] segments`
  - `obstacle_detector/CircleObstacle[] circles`

3. The launch files

Provided launch files are good examples of how to use `obstacle_detector` package. They give a full list of parameters used by each of provided nodes.

- `demo.launch` - Plays a rosbag with recorded scans and starts all of the nodes with Rviz configured with appropriate panels.

- `nodes_example.launch` - Runs all of the nodes with their parameters set to default values.
- `nodelets_example.launch` - Runs all of the nodelets with their parameters set to default values.

4. The displays

For better visual effects, appropriate Rviz display for `Obstacles` messages was prepared. Via its properties, one can change the colors of the obstacles.

Author: *Mateusz Przybyla*