UDACITY | DiDi

# Udacity-Didi Self-Driving Car Challenge

## Team Tea - 4th place

Andres Torrubia, Ali Aliev

# Agenda

- Team
- Challenge
- Pipeline overview
- Segmentation
- Bbox regression
- Fusion & filtering
- Results
- Conclusion

# Team

Andres Torrubia → Devised and implemented deep learning architecture (segmentation + localization).

Ali Aliev → Implemented sensor fusion, point cloud processing, visualization, ROS pipeline.

*Merged 2 weeks before the final deadline.*

# Challenge: conditions

Detect *vehicle* and *pedestrian*

- Detection can be done separately for a vehicle and a pedestrian, i.e. 2 models
- One obstacle at once

**Input**:

- Lidar @10Hz
- Radar @20Hz
- Camera @24Hz
- GPS @10Hz

**Output**:

- Vehicle: *pose* [x, y, z, azimuth (yaw)], *bounding box* [length, width, height]
- Pedestrian: *pose* [x, y, z], *cylinder* [radius, height]

# Challenge: evaluation

- Score is Intersection Over Union
  - Vehicles and pedestrians together
- Submit XML tracklets
  - Pose and bbox for each *camera* frame

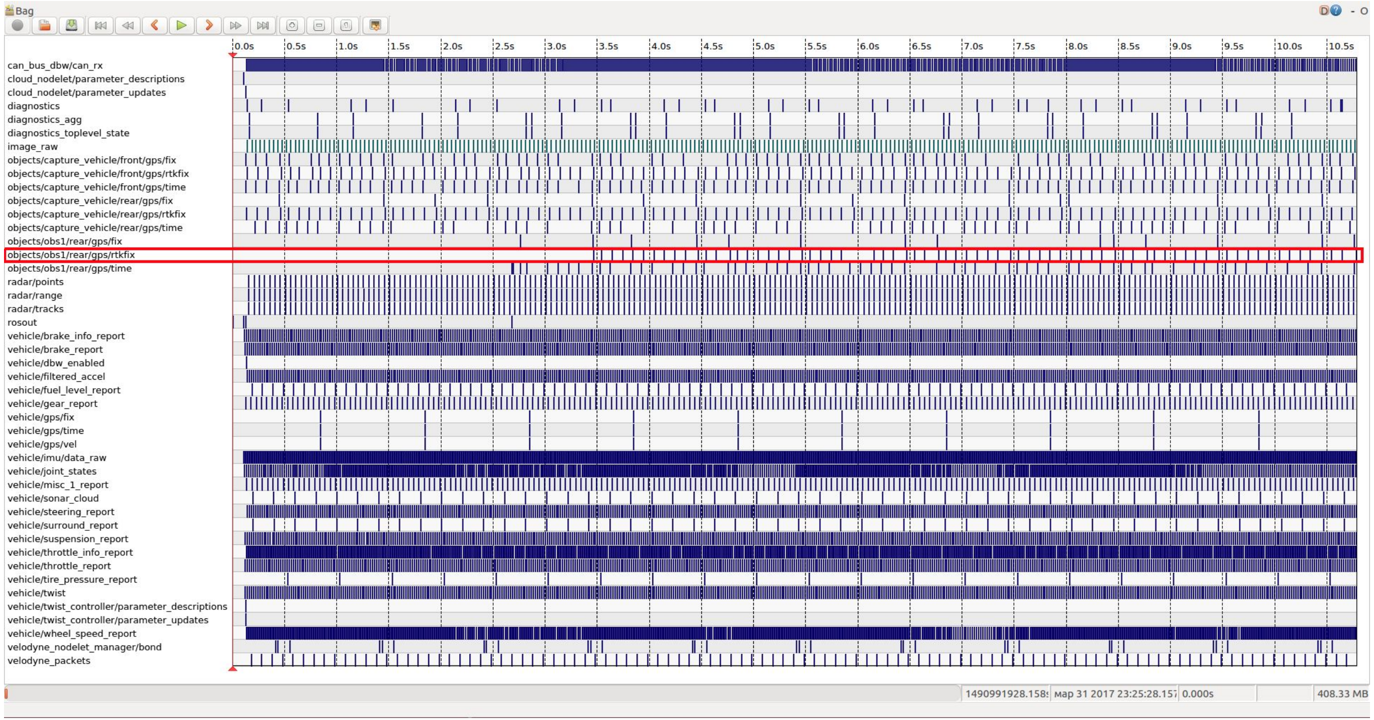
- Pipeline must be implemented as a *ROS** node
- Pipeline rate must be at least *10fps* on Titan X & Core i7


```xml
<tracklets class_id="0" tracking_level="0" version="0">
        <count>92</count>
        <item_version>1</item_version>
        <item class_id="1" tracking_level="0" version="1">
                <objectType>Car</objectType>
                <h>1.682660</h>
                <w>2.104239</w>
                <l>4.554860</l>
                <first_frame>15</first_frame>
                <poses class_id="2" tracking_level="0" version="0">
                        <count>3</count>
                        <item_version>2</item_version>
                        <!-- frame 15 -->
                        <item class_id="3" tracking_level="0" version="2">
                                <tx>28.818745</tx>
                                <ty>23.988316</ty>
                                <tz>-0.456823</tz>
                                <rx>0.000000</rx>
                                <ry>0.000000</ry>
                                <rz>0.542737</rz>
                                <state>0</state>
                                <occlusion>-1</occlusion>
                                <occlusion_kf>-1</occlusion_kf>
                                <truncation>-1</truncation>
                                <amt_occlusion>0.0</amt_occlusion>
                                <amt_occlusion_kf>-1</amt_occlusion_kf>
                                <amt_border_l>0.0</amt_border_l>
                                <amt_border_r>0.0</amt_border_r>
                                <amt_border_kf>-1</amt_border_kf>
                        </item>
                        <!-- frame 16 -->
                        <item>
```


*Robot Operating System

# Challenge: data

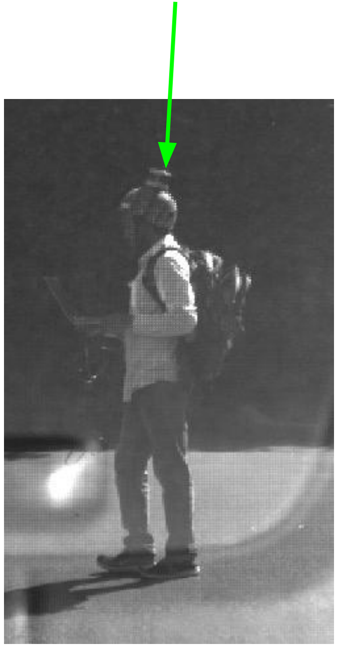
## ROSbag file



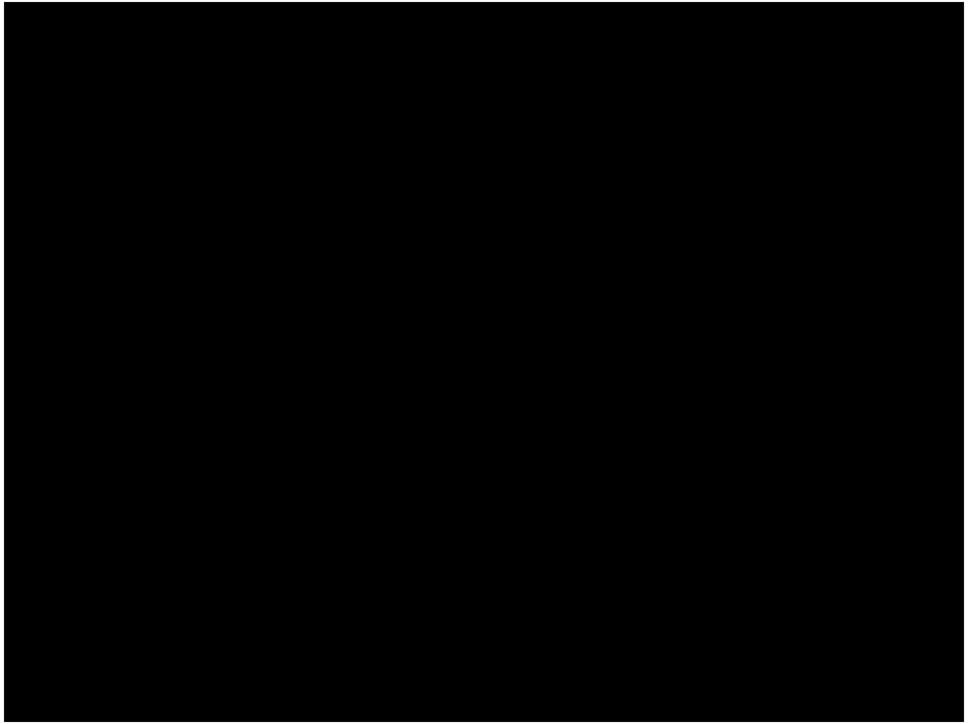
# Challenge: ground truth mining

Orientation and position from the front & rear GPS pair

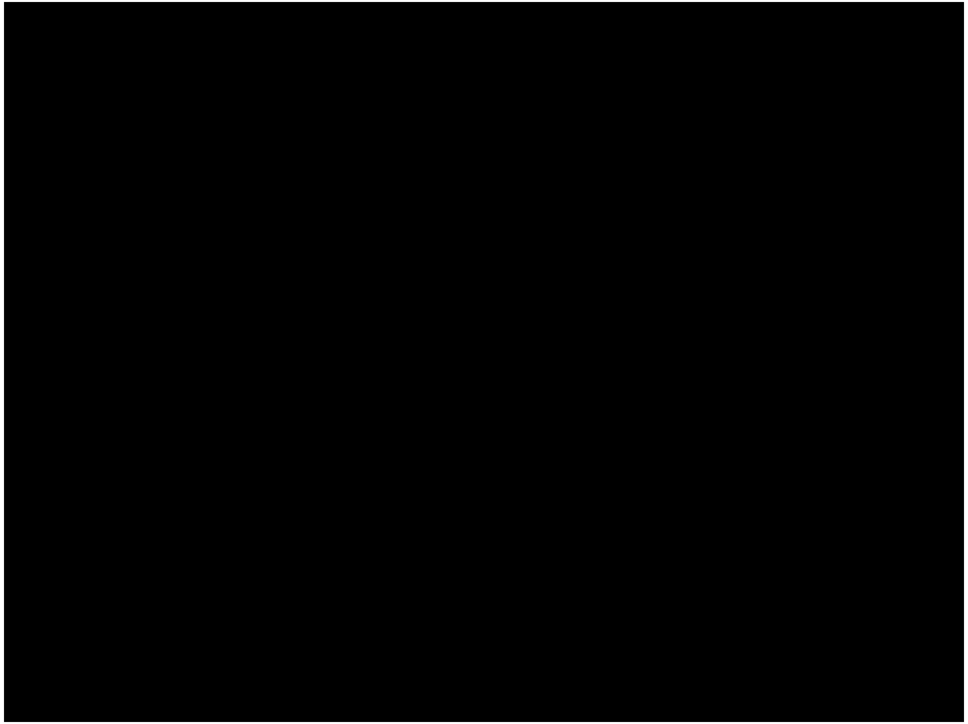
GPS for position (no orientation)



# Challenge: input example



# Challenge: output example



# Challenge: timeline & prizes

**Registration** → **Round 1** → **Round 2** → **Final**

| **Registration** | **Round 1** | **Round 2** | **Final** |
|---|---|---|---|
| March 8 - March 21 | March 22 - May 31<br>Vehicle detection | June 5 - July 3<br>Vehicle + pedestrian detection | August 31<br>Presentation & Chance to run on a self-driving car |

1st place -- $100k

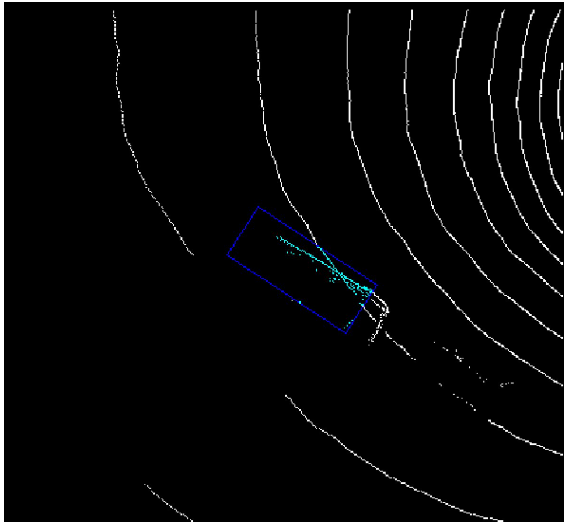
2nd place -- $3k

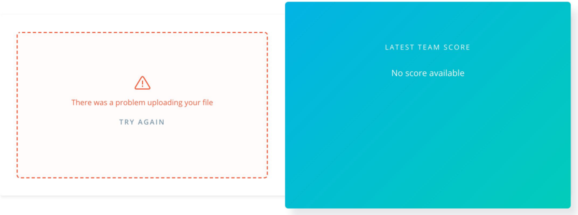
3rd place -- $1.5k

Top 5 -- Airfare and hotel accommodation for two representatives from each team to attend the award ceremony in Silicon Valley

# Challenge: issues

- Bad ground truth
  - GPS misalignment/lags, inaccurate sensors calibration
- Inconsistent datasets
  - Dataset release 1 is incompatible with the release 2 & 3
- Poor support
  - Evaluation server crashed twice in 7 hours before the deadline

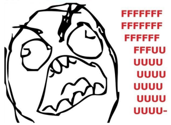



3:00AM MSK


4:00AM MSK


8:00AM MSK



# Pipeline: related work

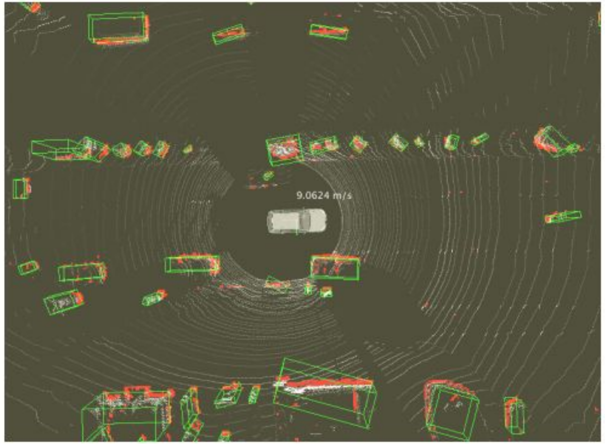
## Traditional pipeline

*Remove ground plane* => *extract clusters* => *extract features* => *classify* => *fit bbox*

- *Remove ground plane*
  - plane fitting / RANSAC
- *extract clusters*
  - DBSCAN
  - Euclidian
- *extract features*
  - Object level features
  - Point level features
  - Keypoints
- *classify*
  - SVM
  - NN
  - etc
- *fit bbox*
  - SVD
  - ML?

\+ Source code (PCL), decent research done

\- Expensive feature engineering, low accuracy



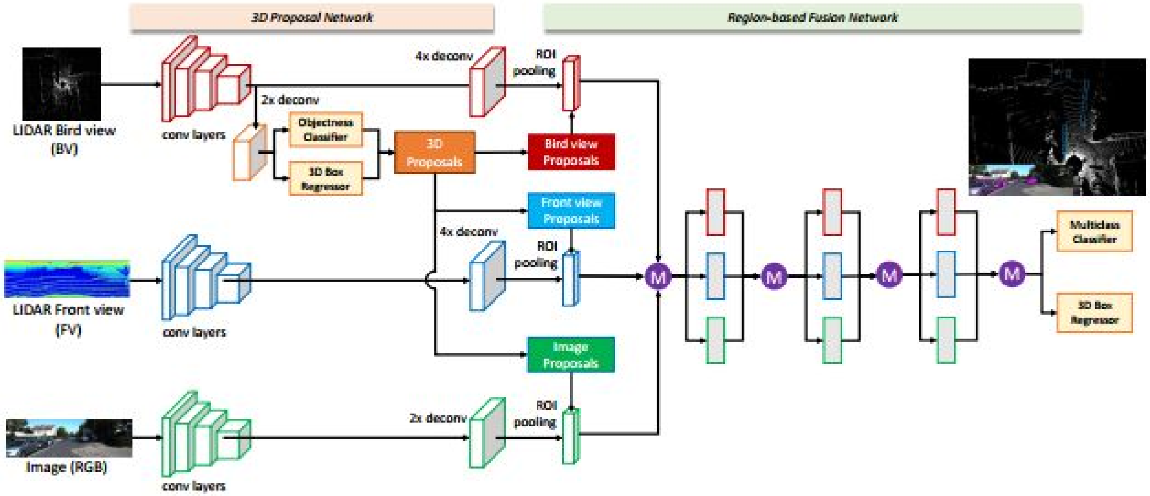

Himmelsbach et al, “LIDAR-based 3D Object Perception”

# Pipeline: related work

## State of the art

*Rasterize point cloud* => *detect objects on the image* => *regress bbox*

- \+ Accurate
- \- (potentially) slow

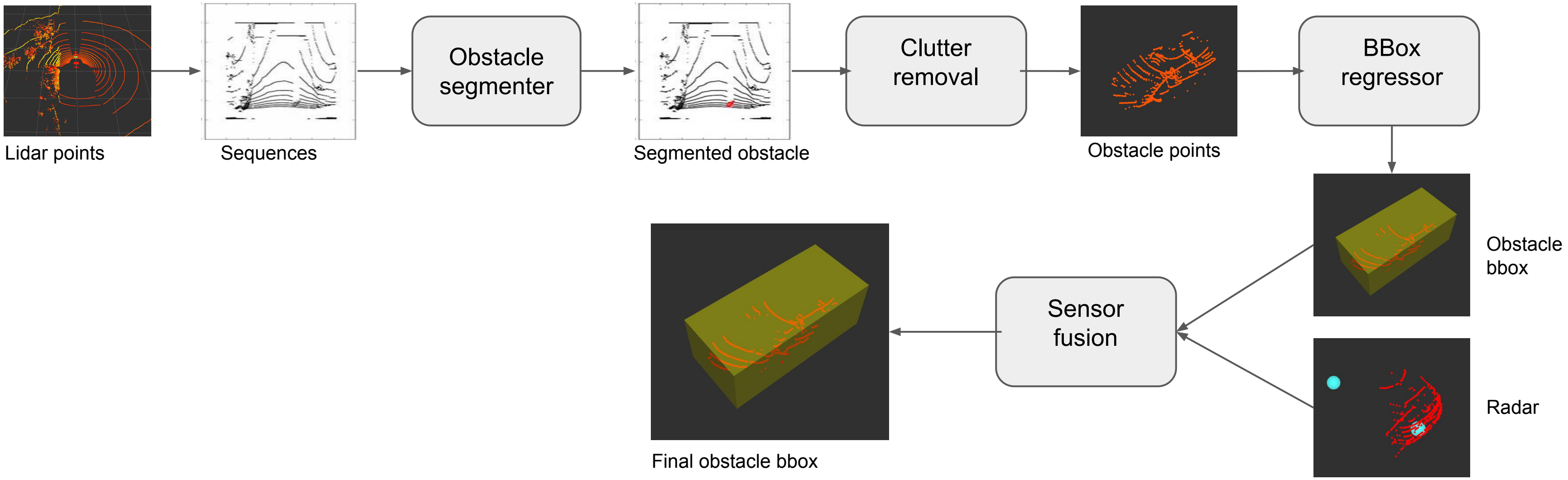



Chen et al, "Multi-View Object Detection for Autonomous Driving"

# Pipeline: our architecture

We wanted to do something *different*, *original* and *new*.

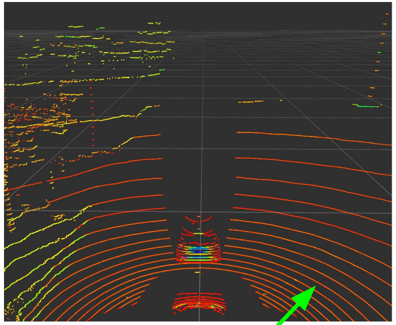
- Preserve original data format, i.e. 3D point cloud
- Create a fast & robust pipeline

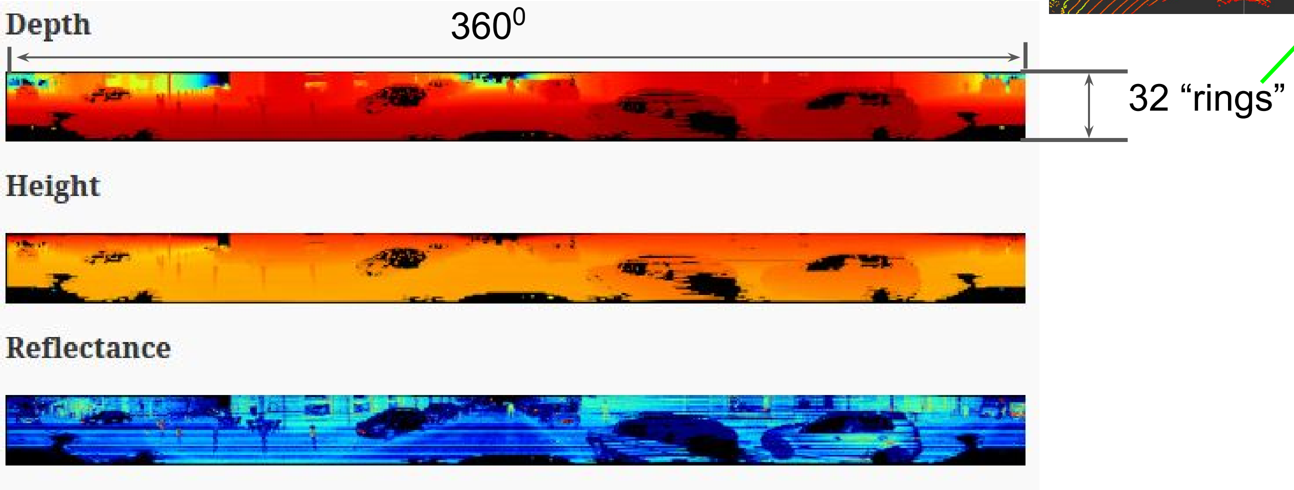


# LIDAR: how it works

Velodyne HDL-32E

- 32 lasers; 7000 pts/sec
- Range 1-100m; accuracy <2cm
- Depth, height, reflectance
- 10Hz





# Obstacle segmenter: point cloud preprocessing

Lidar package
~30k {x, y, z, intensity, ring} points

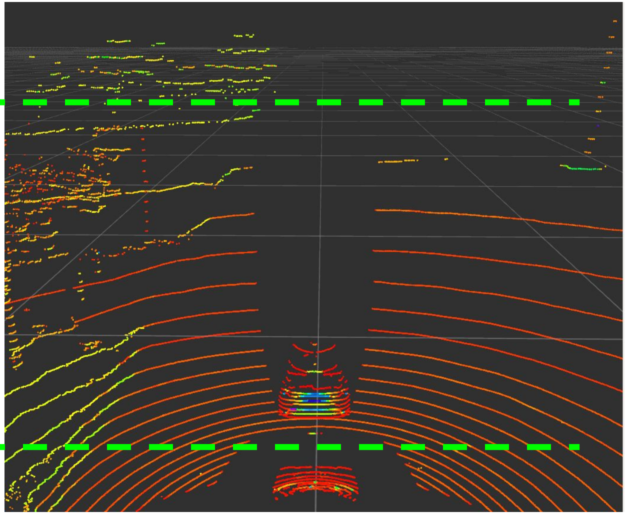
↓

Sampled 2048 points from each ring
16 rings with 2048 {depth, height, intensity} points

↓

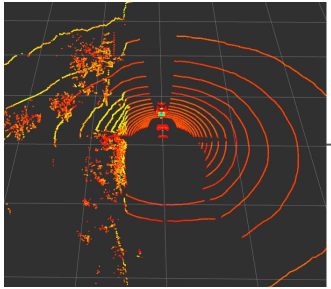
Split rings to 32 sectors
32 sectors with 64 points

16 rings
are enough



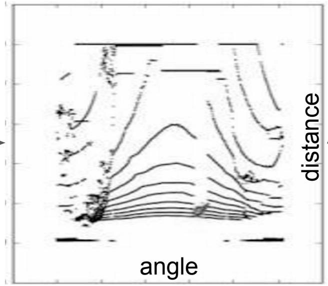
- uniform sampling from -pi to pi
- nearest neighbor interpolation

# Obstacle segmenter: input



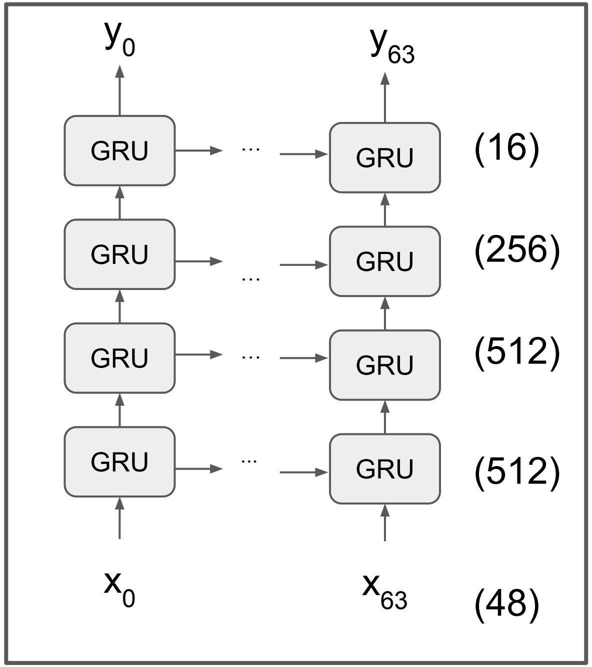
**Lidar points**
Shape = (n,  5)

n~30k    {x,y,z,i,r}




**Sequences**
Shape = (32, 64, 48)

sectors/ batch_size    angles/ timesteps    features*/ input_dim

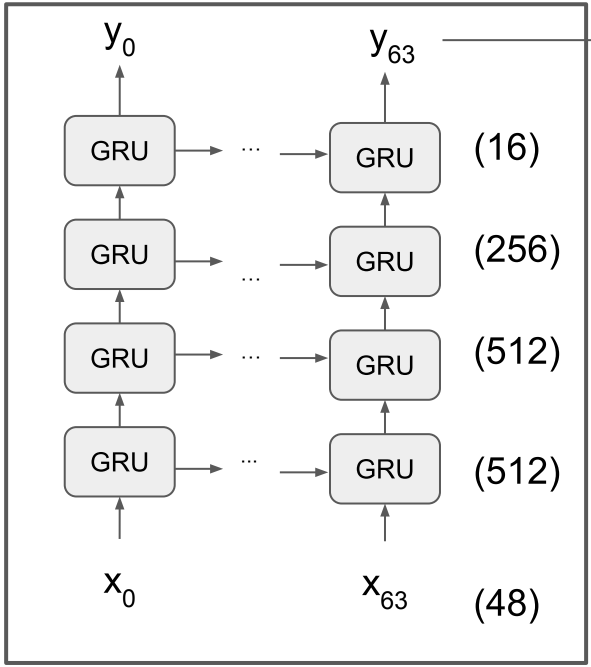



**RNN**
Output shape = (32, 64, 16)

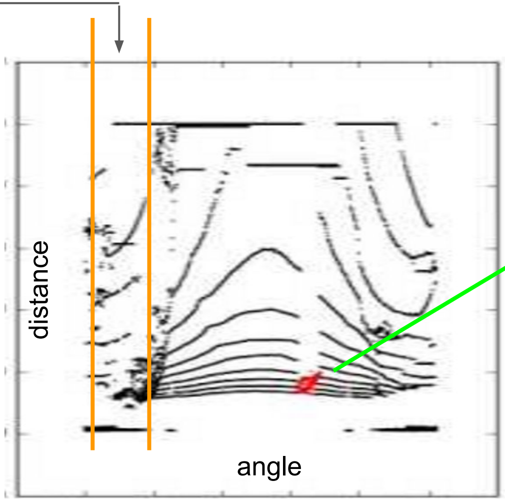
rings

* $[d_0,\ldots,d_{16}, h_0,\ldots,h_{16}, i_0,\ldots,i_{16}]$

# Obstacle segmenter: output

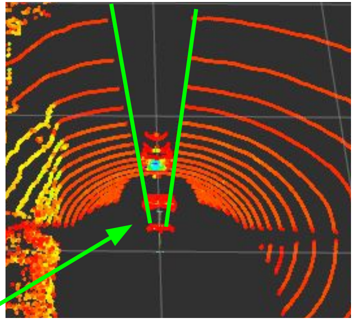



**RNN**
Output shape = (32, 64, 16)




**Obstacle segmentation**
Shape = (32, 64, 16)

Probability that *r*-th ring contains obstacle

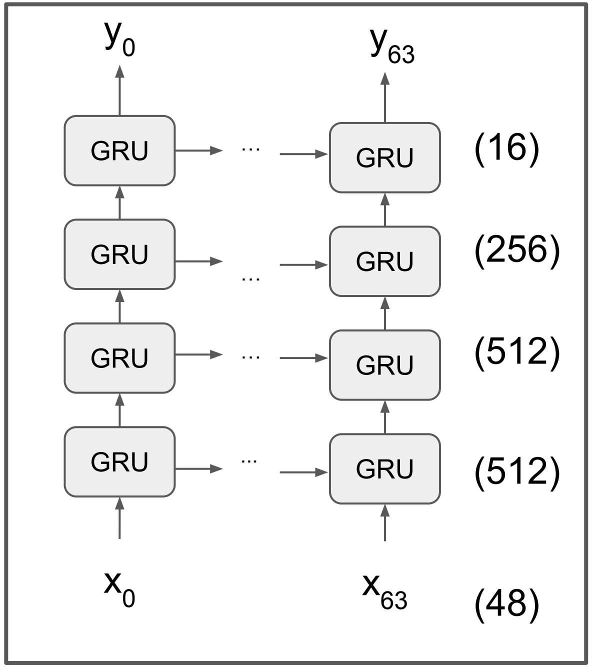


Extract points
Shape = (m, 4)

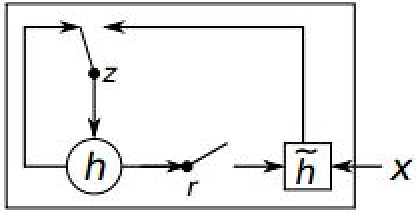
thresholded points

{x,y,z,i}

# Obstacle segmenter: RNN




GRU = Gated Recurrent Unit (Cho et al. 2014)



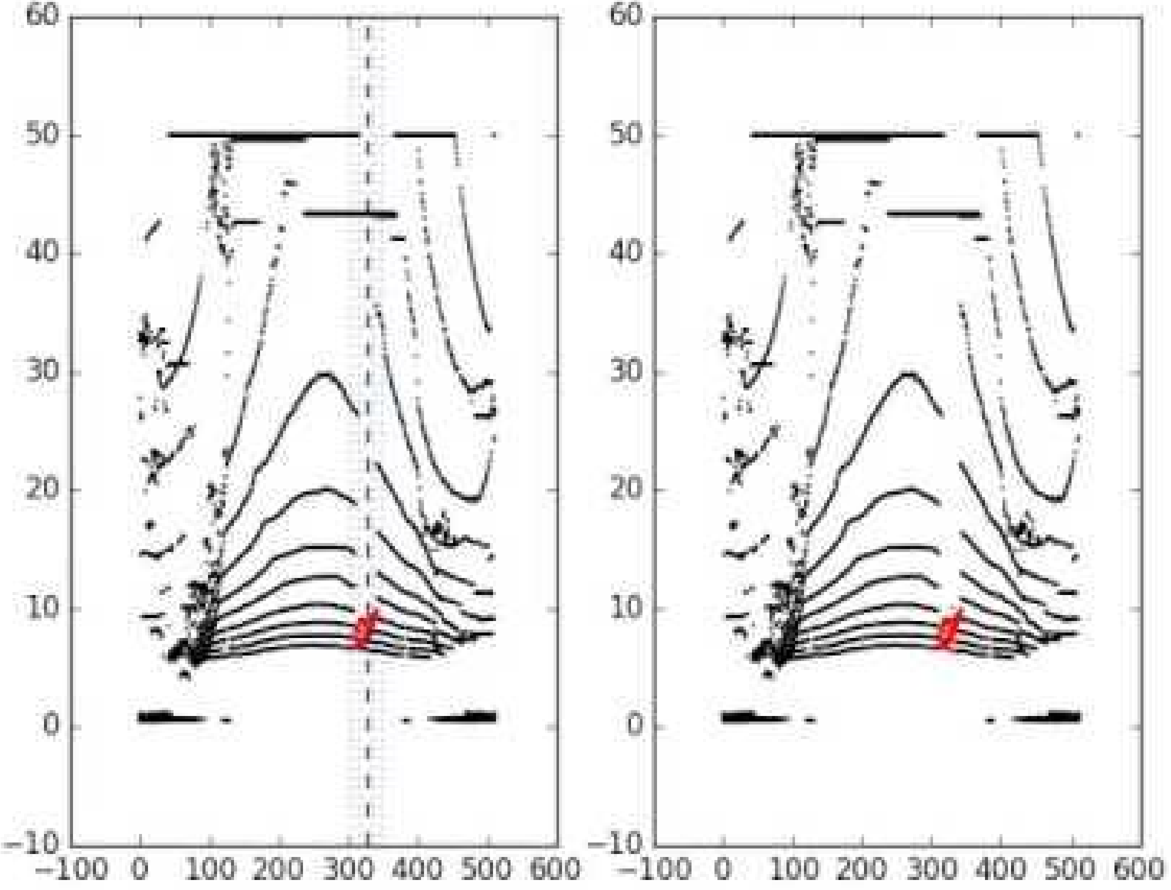

$$r_j = \sigma\left([\mathbf{W}_r\mathbf{x}]_j + [\mathbf{U}_r\mathbf{h}_{\langle t-1\rangle}]_j\right)$$

$$z_j = \sigma\left([\mathbf{W}_z\mathbf{x}]_j + [\mathbf{U}_z\mathbf{h}_{\langle t-1\rangle}]_j\right)$$

$$\tilde{h}_j^{\langle t\rangle} = \phi\left([\mathbf{W}\mathbf{x}]_j + [\mathbf{U}(\mathbf{r} \odot \mathbf{h}_{\langle t-1\rangle})]_j\right)$$

$$h_j^{\langle t\rangle} = z_j h_j^{\langle t-1\rangle} + (1 - z_j)\tilde{h}_j^{\langle t\rangle}$$

- Last GRU layer uses *sigmoid* and *dropout* 0.1, rest use *tanh* and *dropout* 0.2
- 2.6m parameters, trained w/ binary x-entropy



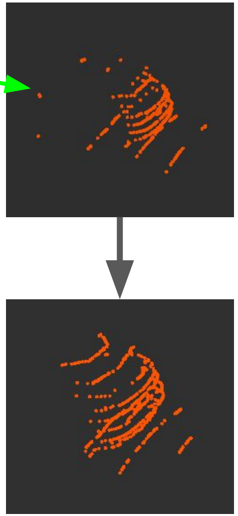
60
50
40
30
20
10
0
−10
−100
0
100
200
300
400
500
600
60
50
40
30
20
10
0
−10
−100
0
100
200
300
400
500
600

# Clutter removal

- Clusterize segmented points

- Pick cluster nearest to the last known position

- Remove outlier points (statistically)



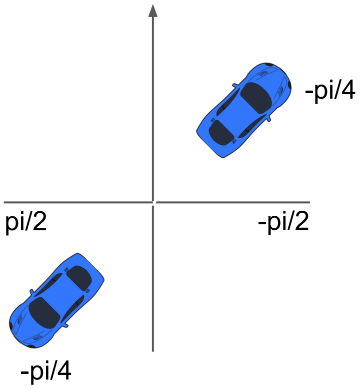
# Bbox regression: problem statement

Given an *unordered* set of segmented points, regress the bounding box properties:

- *Size* (width, length, height)
- *Centroid*
- *Orientation [-pi/2, pi/2]*

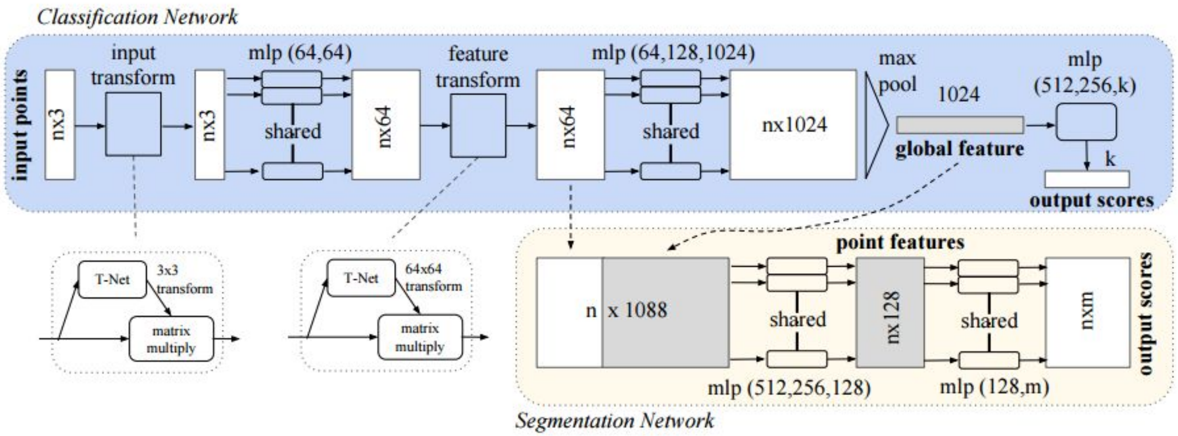
The solution must be robust to:

- Order of the points (n! permutations)
- Outliers, deleted points, perturbations
- Affine transformations



# Bbox regression: PointNet

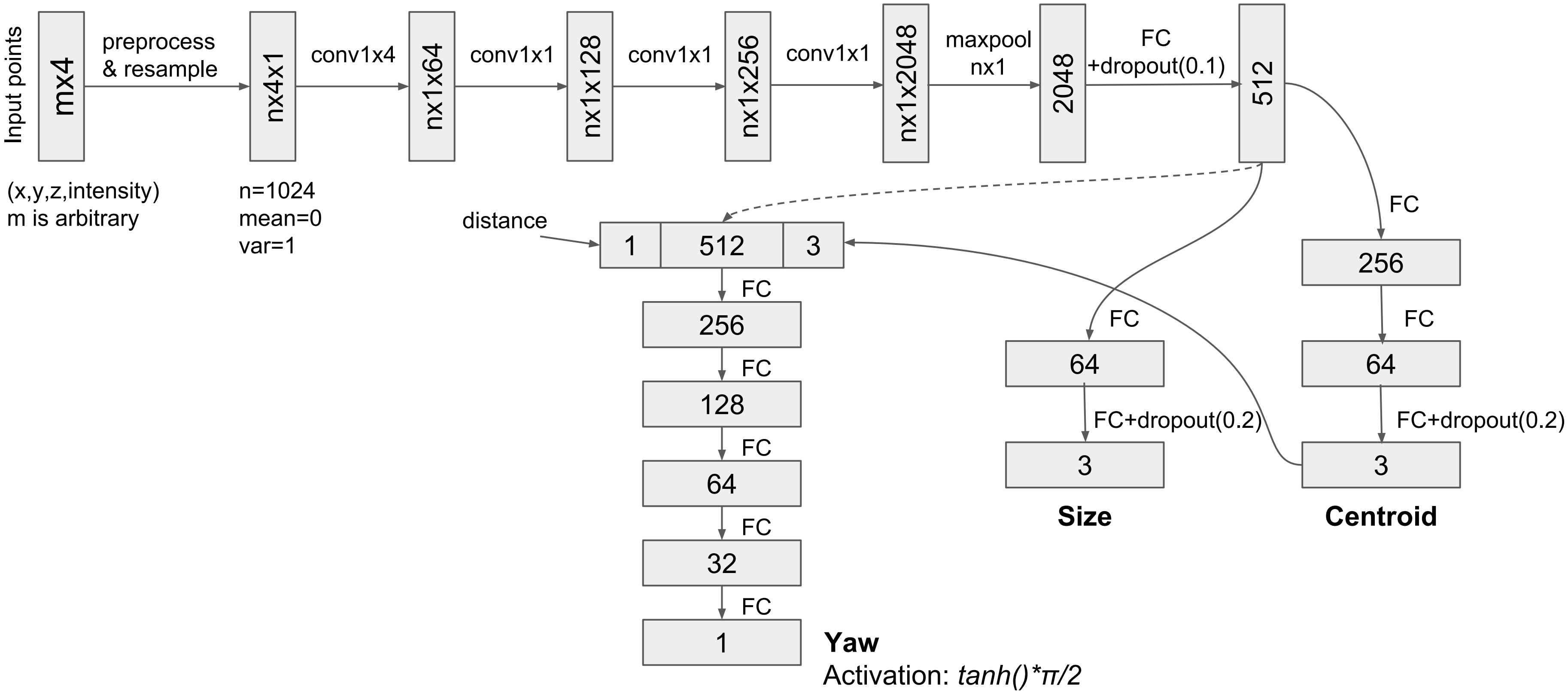
**PointNet*** is able to classify a point cloud *as is.*




We can adjust PointNet for bbox regression by adding custom layers.

*PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation, CVPR 2017, Qi et al

# Bbox regression: modified PointNet



# Bbox regression: training

$$L_i = (s - s^*)^2 + (c - c^*)^2 + (r - r^*)^2$$

size loss centroid loss angle loss

$s$ = {w, l, h}
$c$ = {x, y, z}
$r$ = {cos(yaw), sin(yaw)}
* - true values

Train/val ratio

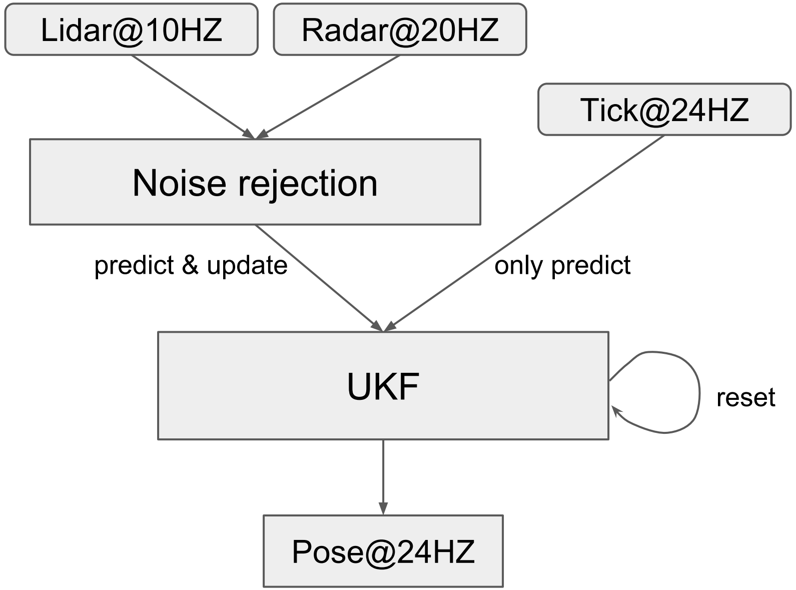
- Vehicle: 90/10
- Ped: 85/15

Size, centroid metric: mean absolute error
Angle metric: mean absolute angle error

# Fusion: model

- State vector:
  $\{x, x', x'' y, y', y'', z, z', z'', \varphi\}$ ← Lidar-fixed coordinate frame
- Transition model:
  $x_{i+1} = x_i + x'_i dt + 0.5x''_i dt^2$ ← Newton law / inertial frame assumption
  $x'_{i+1} = x'_i + x''_i dt$
  $x''_{i+1} = x''_i$
  $\varphi_{i+1} = \varphi_i$
- Input
  - Lidar $\{x, y, z, \varphi\}$
  - Radar $\{x, x', y, y'\}$
  - Camera tick
- Output
  - Pose $\{x, y, z, \varphi\}$

# Fusion: model



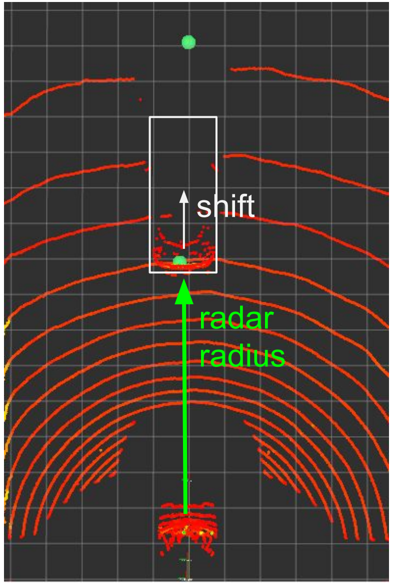

- We use *Unscented Kalman Filter*
- Reject noisy input statistically (3-sigma rule)
- Reset filter when covariance is too high

- Camera ticks are used to predict pose @24Hz
- Prefer lidar measurements over radar measurement at close distances

# Bbox filtering

- Vehicle
  - moving average for bbox length, width, height
- Pedestrian
  - constant cylinder radius and height (allowed by the rules)

*Trick*: shift radar distance by a constant value to better fit vehicle's bbox centroid



# Results

| | Team | Score |
|---|---|---|
| 1 | abccba | 0.433 |
| 2 | Robodreams | 0.409 |
| 3 | zbzc | 0.397 |
| 4 | Tea | 0.391 |
| 5 | ICTANS | 0.346 |

# Conclusion

- Implementation, performance & gotchas:
  - No resolution lost when using raw lidar points
  - Trained using single 1080 GTX Ti
  - Code primarily in Python, optimized lidar cloud interfacing in C++
  - Trained GRU (RNN) w/ *theano* (2x faster than *tensorflow*)
  - Used *tensorflow* for inference (*theano* segfaulted when using two models sequentially)

- Areas of improvement:
  - Train two networks end to end (need differentiable filtering and resampling)
  - Track ego and obstacle position in a fixed global frame, separately
  - Account for time lags in lidar frames
  - Fuse camera, odometry

**Code:**
https://github.com/antorsae/tea
https://github.com/antorsae/torbusnet

Thank you!