Course Work Implementation – Sensor Fusion for autonomous navigation car-like robot

Filipenko Maksim - MS-AIR

I. Code description

The solution contains two main part. The first part addressed to autonomous robot navigation. The second part is dataset based. It has analyzed information from different sensors with applying different methods for processing. It also has methods for sensor fusion with Extended Kalman Filter and Particle Filter.

1. Autonomous navigation car-like robot

This repository was designed to make research platform RACECAR autonomous move to special goal.

Link to the repository: https://github.com/mfilipen/self-driving-car

The main packages:

- car_joystick - it is used for teleoperation mode.
- cmd_vel_to_ackermann_drive - it is used for solving forward kinematics problem.
- emergency_traxxas - check system consistency.
- racecar_description - URDF model of the robot.
- racecar_navigation - it is used for ROS navigation stack.
- razor_imu_9dof - it is used for working with IMU sensor.
- traxxas_driver - it is low-level actuators driver.
- zed-ros-wrapper-1.0.0 - it is used for working with zed camera sensor.

2. Sensor fusion

This repository was designed for analyzing data from LIDAR, ZED camera, IMU and also has modules for sensor fusion with EKF and PF

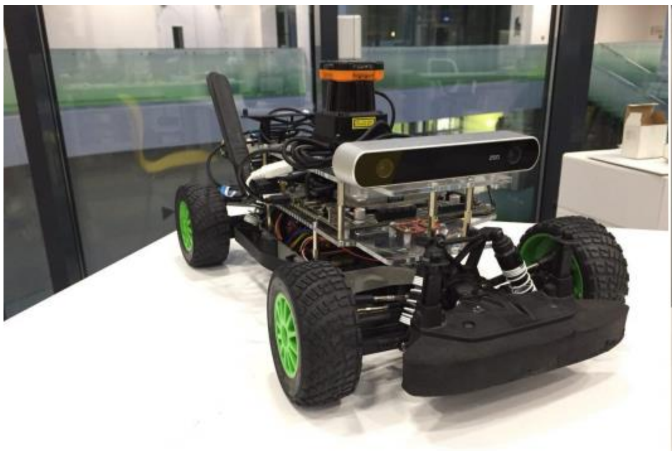
This research is dataset based. It requires dataset but suddenly it is too large for sharing via the internet (15 GB). If you need dataset contact to me.

Link to the repository: https://github.com/mfilipen/lidar_IMU_zed_fusion

The main packages:

- racecar_description - it is for rendering the dimensional model of the car in RViz
- sensor_fusion - it is for fusing data from different sensors.
- /sensor_fusion/scripts/ExtendedKalmanFilter/ - sensor fusion with Extended Kalman Filter
- /sensor_fusion/scripts/ParticalFilter/ - sensor fusion with Particle Filter
- other modules - data processing, error calculation, plotting data

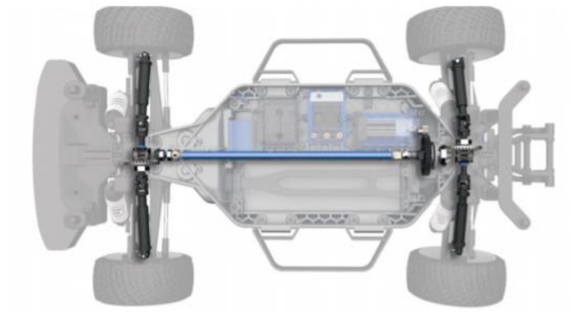
II. Experimental setup



The robot for experiment has Ackerman steering model, powerful NVidia Jetson TX1 for onboard computation, LIDAR (Hokuyo UTM-30LX) as the main sensor, driver, motor, servo for robot motion.

1. Hardware

Chassis - 4WD Traxxas #74076



It has Efficient Shaft-Driven 4WD, Low-CG Chassis. For more information: https://traxxas.com/products/models/electric/74076-1rally.

The computing unit - NVidia Jetson TX1

- NVIDIA Maxwell ™
- 256 CUDA cores
- Quad ARM® A57
- Memory: 4 GB 64 bit LPDDR4 25.6 GB/s
- Other: UART, SPI, I2C, I2S, GPIOs
- For more information: http://www.nvidia.com/object/embedded-systems-dev-kits-modules.html

2. Sensors

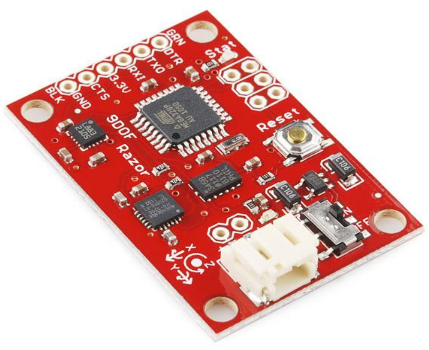
LIDAR - Hokuyo UTM-30LX

For more information: https://www.hokuyo-aut.jp/search/single.php?serial=169

Stereo camera - ZED

https://www.stereolabs.com/

Inertial measurement unit - Razor 9DoF SEN-10736



For more information: https://www.sparkfun.com/products/retired/10736

Driver & Motor SK-300058-04

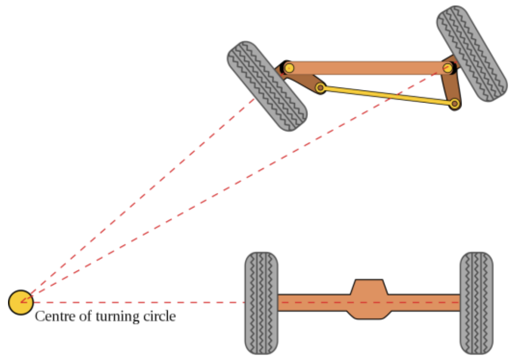
For more information: http://www.skyrc.com/Cheetah?search=%20SK-3000

3. Software

ROS indigo and Ubuntu 14.04 was chosen as foundation for reduce development cost. It are open source projects and have large community.

III. Robot model

1. Kinematic model - Ackerman steering model

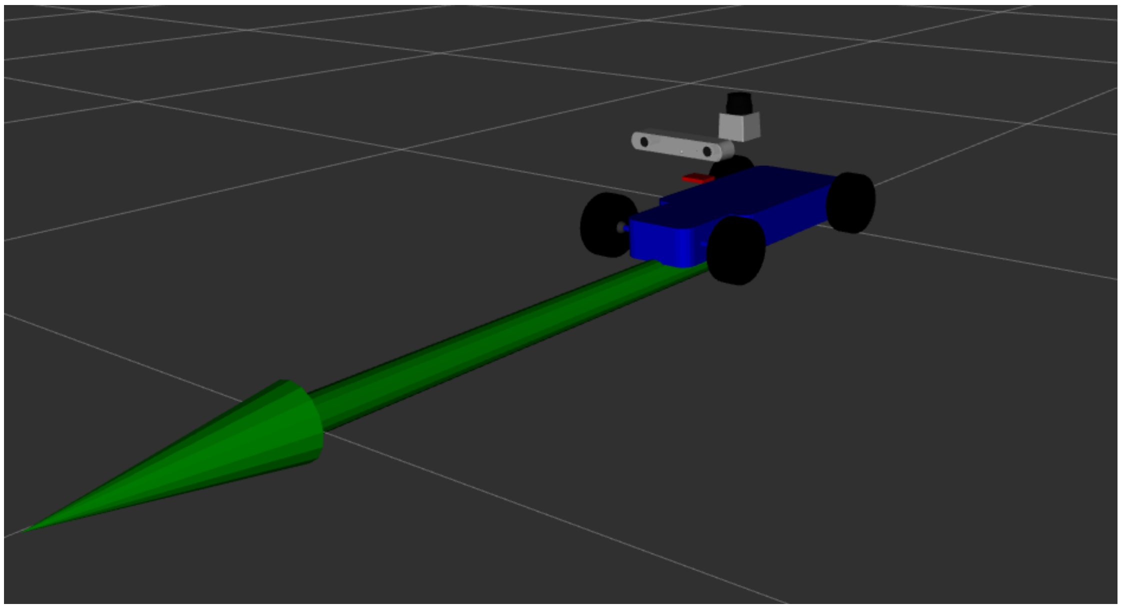


The robot platform satisfies to Ackerman steering model. It is used as motion model of experimental setup.

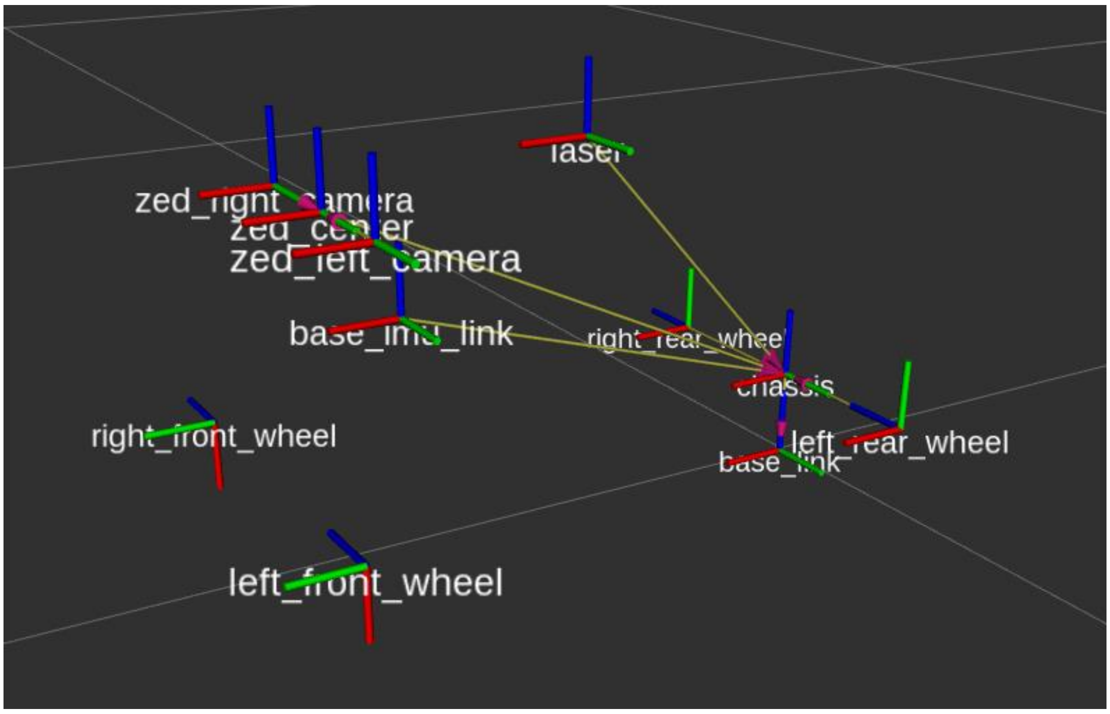
For more information: https://en.wikipedia.org/wiki/Ackermann_steering_geometry

2. URDF Robot Model

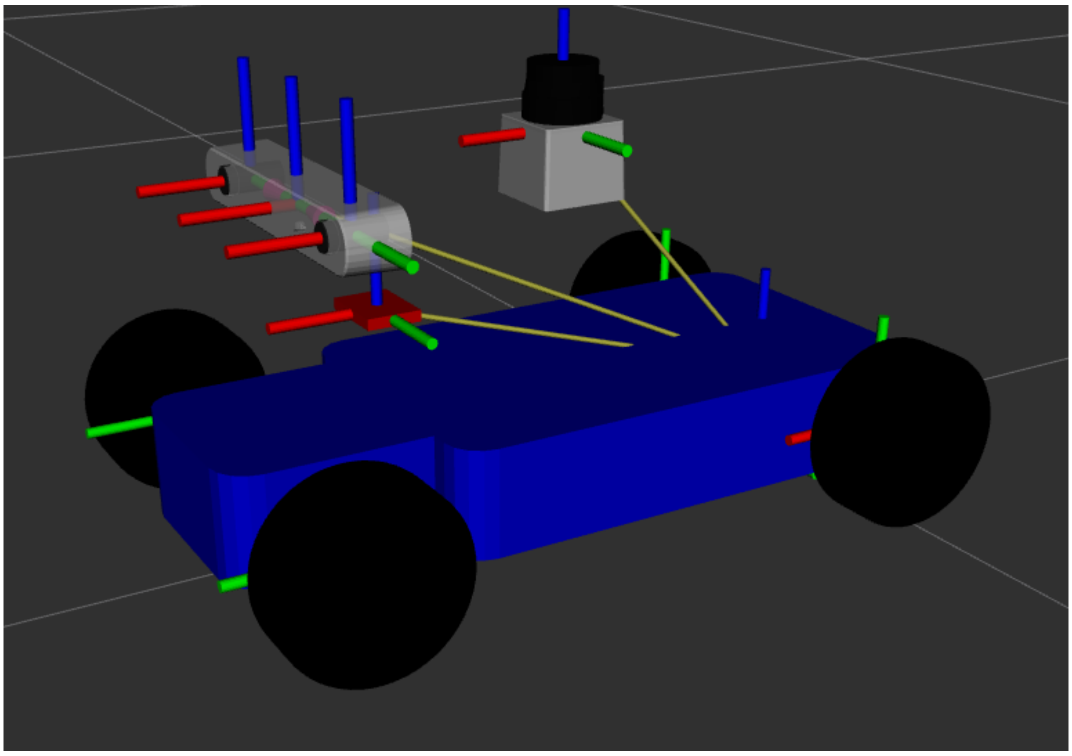
It was designed URDF model for simplify coordinate transformation and have easy understandable visualization of the robot.



It has coordinate frame for all-important part of the robot. These parts have hard connection to the base link, which was chosen according to kinematic model of the robot.

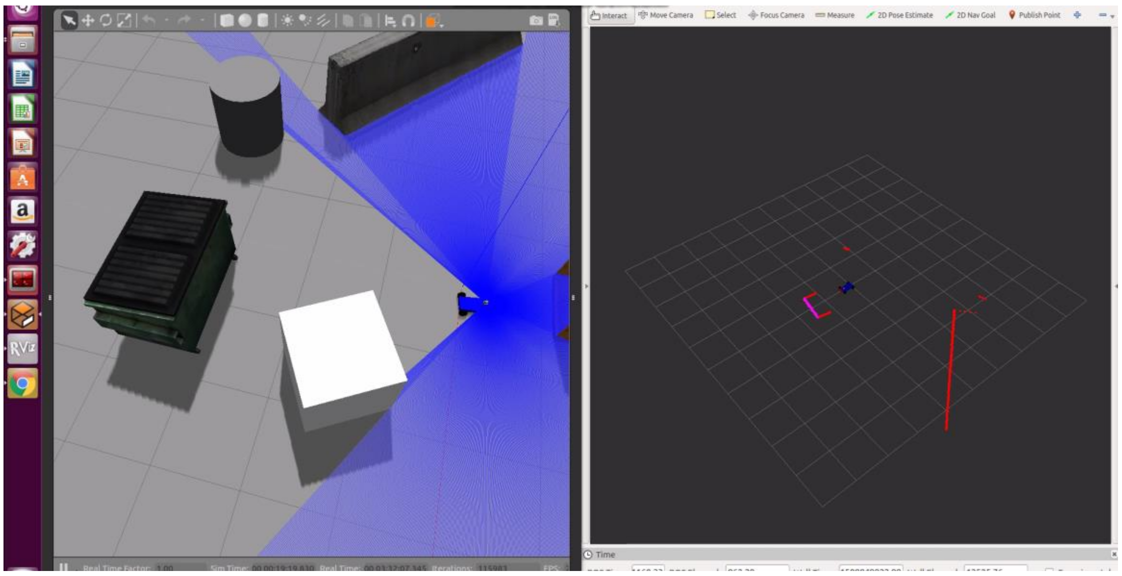


Distance between different components of the model corresponds with real robot model.



3. Robot modeling in gazebo

The robot model has mass and can be used in simulator gazebo. However, at this point it has not working kinematic plugin for gazebo.

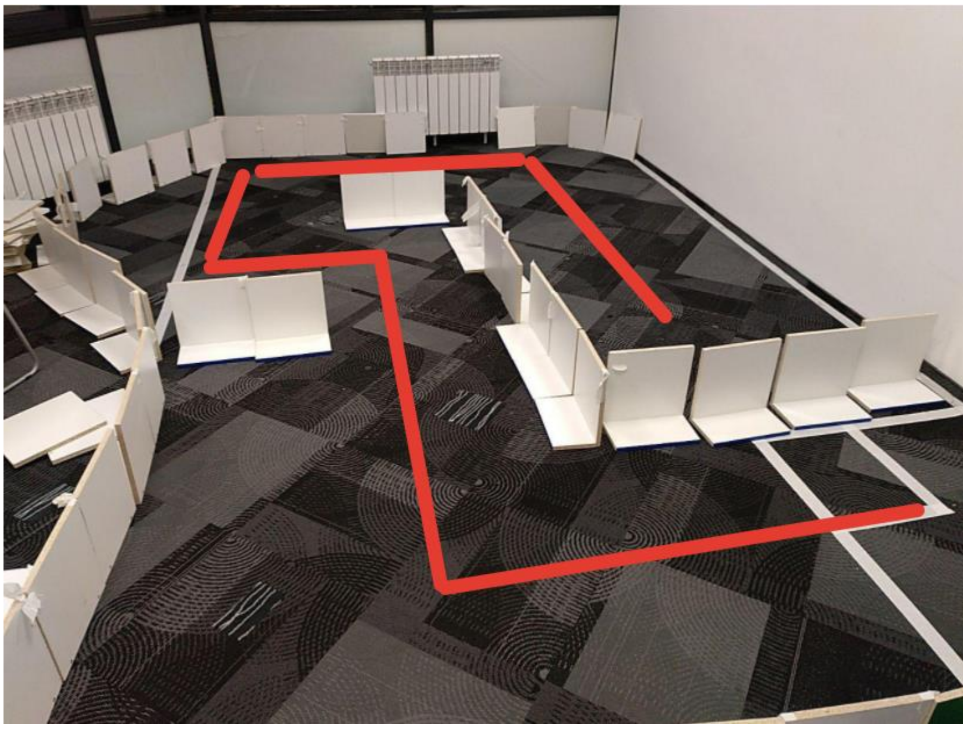


For more information: https://www.youtube.com/watch?v=yt9Jz-HH_zg

IV. Autonomous navigation

1. Test environment

The labyrinth with at least 5 turns was used as test environment for the experiment.

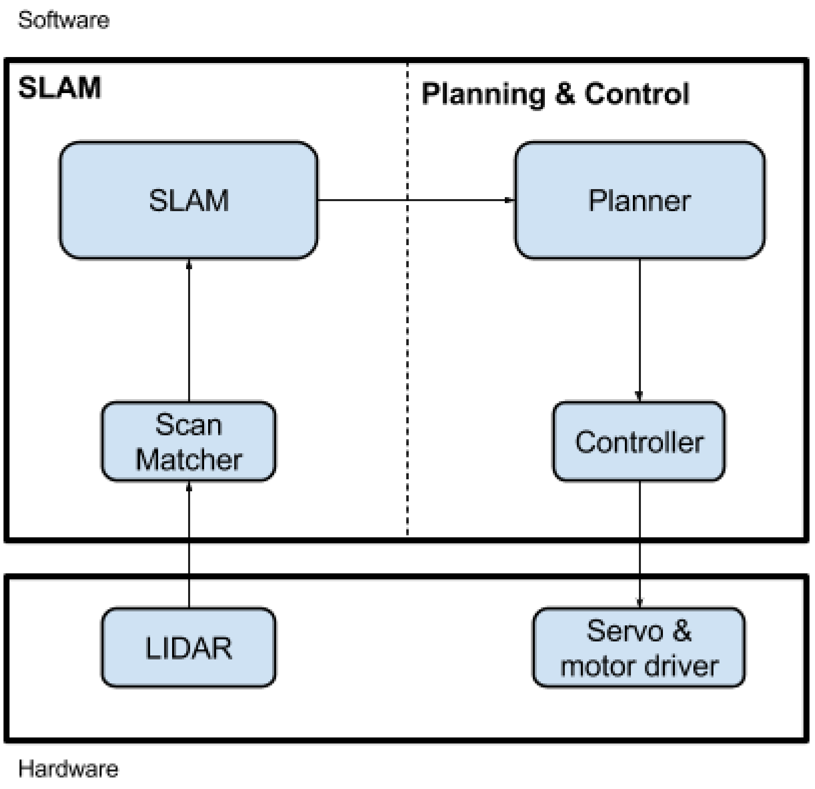


2. Experimental conditions

The robot was placed in test environment. All computation was on board. The host computer was connected via ssh connection and was used to send navigation goal command. The goal of the prototype is achieve to suggested goals.

3. Architecture

The main goal of this test setup is evaluate robot planning algorithm because of it was decided to do not do this system complicated with large amount of sensors. The LIDAR was used as main sensor. It provides information about environment. Software build map of environment and solve localization problem based on this information. Finally, planner build appropriate way and code this way to exact motions of actuators base on map and location.

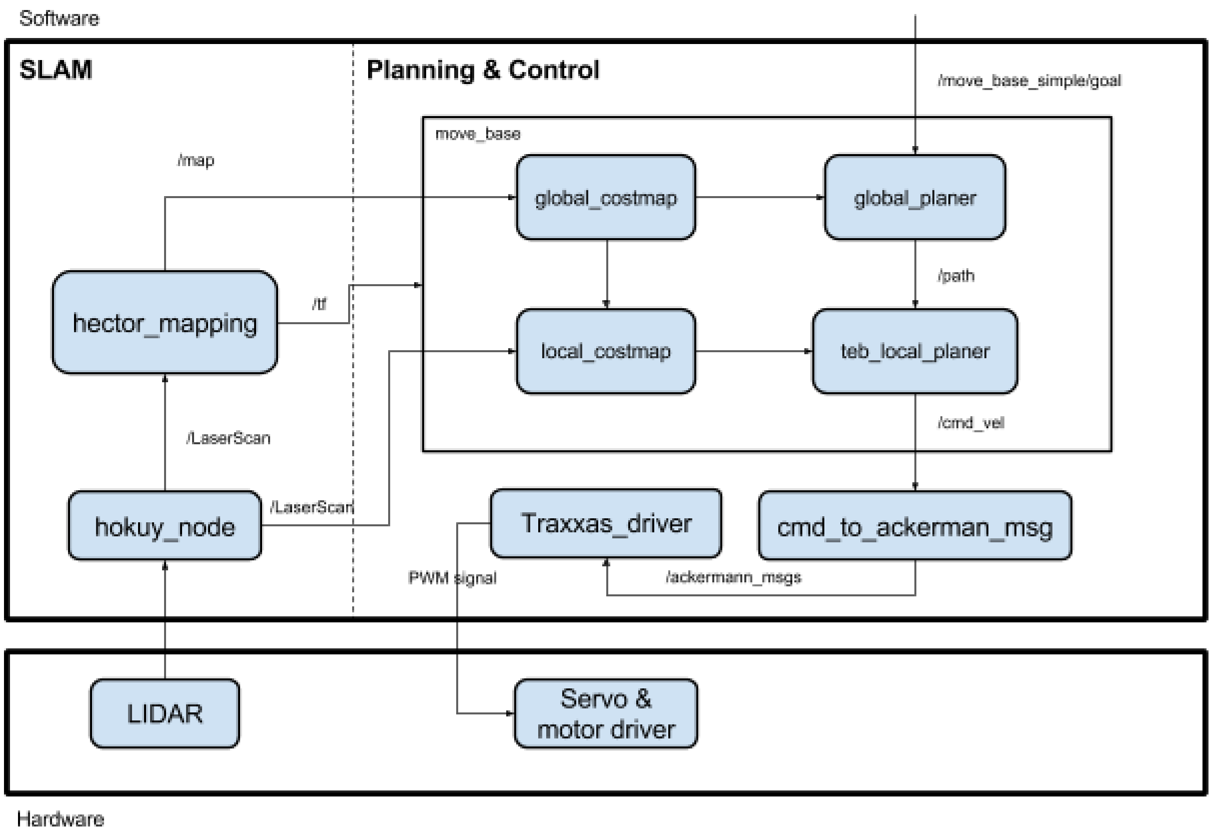


4. ROS Architecture

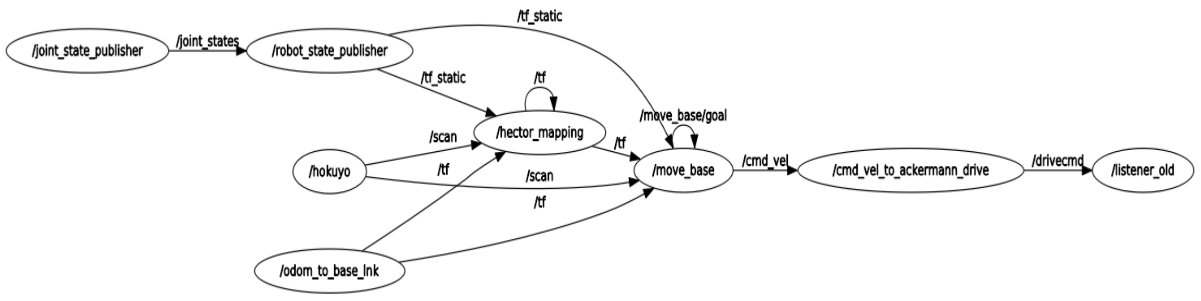
The following nodes were used:

hokuyo_node (http://wiki.ros.org/hokuyo_node) for collecting raw data from sensors. hector_mapping (http://wiki.ros.org/hector_mapping) was used for generating a map from LIDAR data and solving localization problem. move_base (http://wiki.ros.org/move_base) is part of ROS navigation stack (http://wiki.ros.org/navigation) and is used for solving global and local path planning problem. But default local planner is not applicable for Ackerman steering model. It was replaced with teb_local_planner (http://wiki.ros.org/teb_local_planner) for build a local plan for car-like robot. cmd_to_ackerman_msg is a node for solving kinematics for Ackermann steering model. Traxxas_driver is used for sending signals to the actuators.

The following schema describes how nodes connected and works.

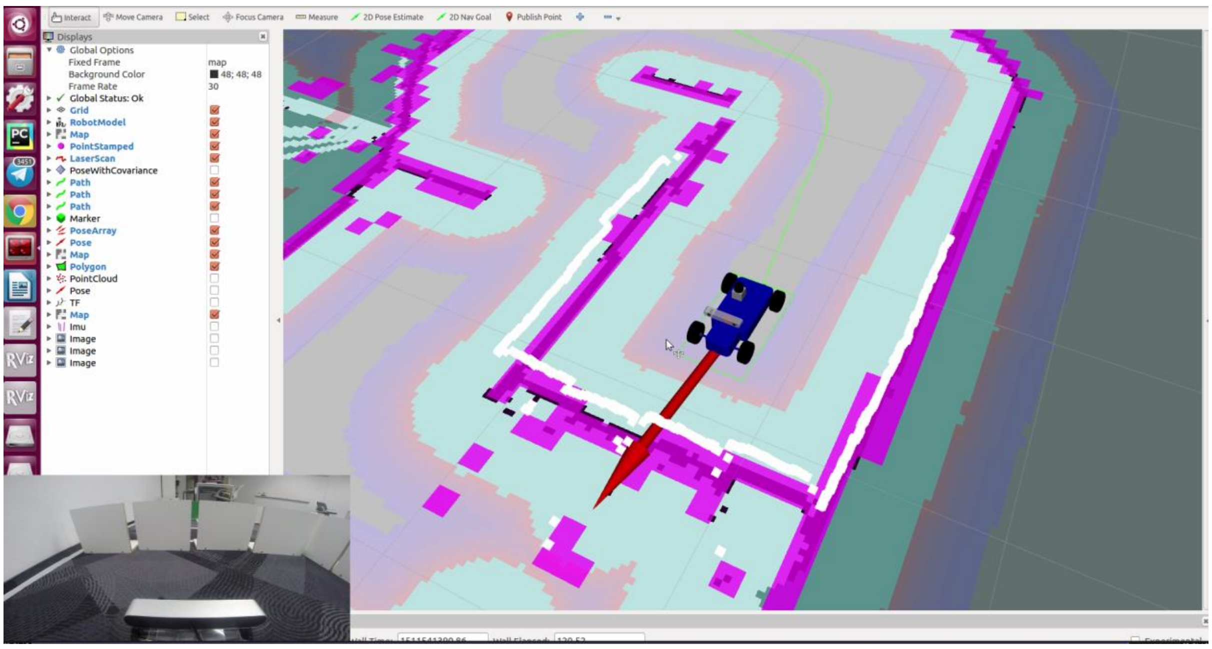


Next schema shows topic exchanges from different nodes.



After fitting all parameters of the system we have good enough autonomous mobile robot.

5. Example of working autonomous robot navigation



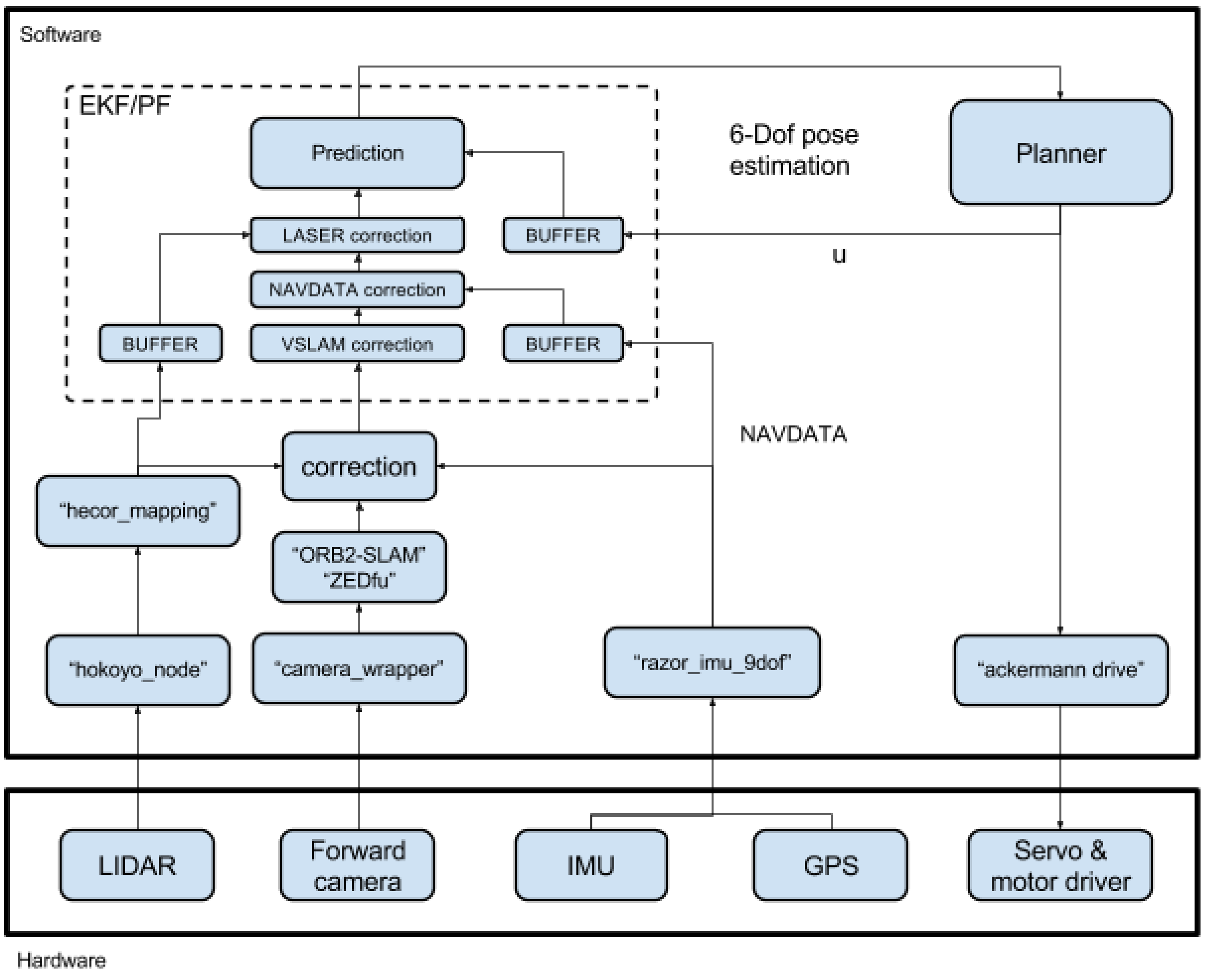
Global Options
Fixed Frame map
Background Color 48; 48; 48
Frame Rate 30

Video available via link: https://www.youtube.com/watch?v=mu7d6ygTFf0

V. Sensor Fusion

1. Architecture

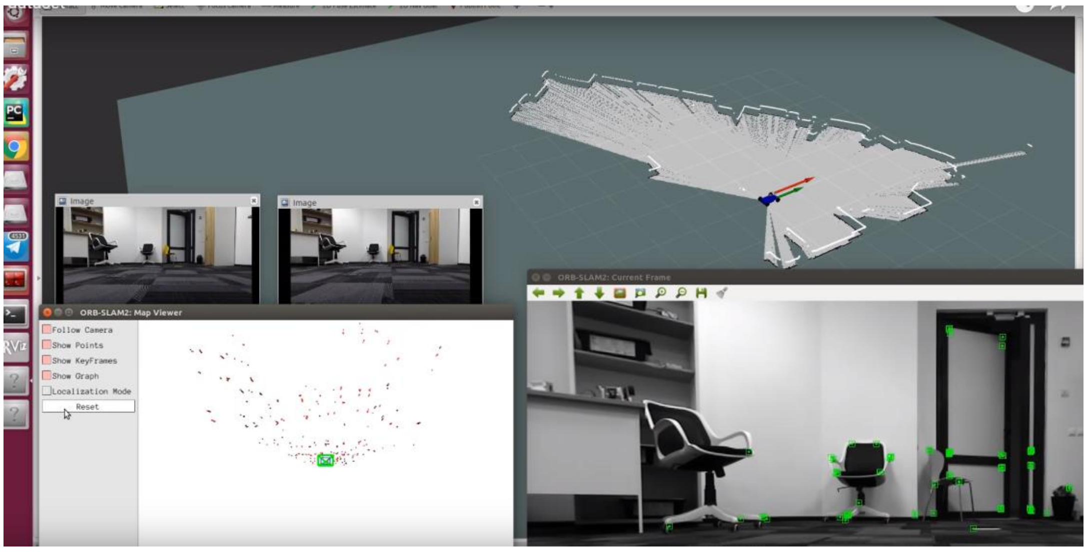
The goal is combine data from different sensors and control commands for more precise pose estimation.



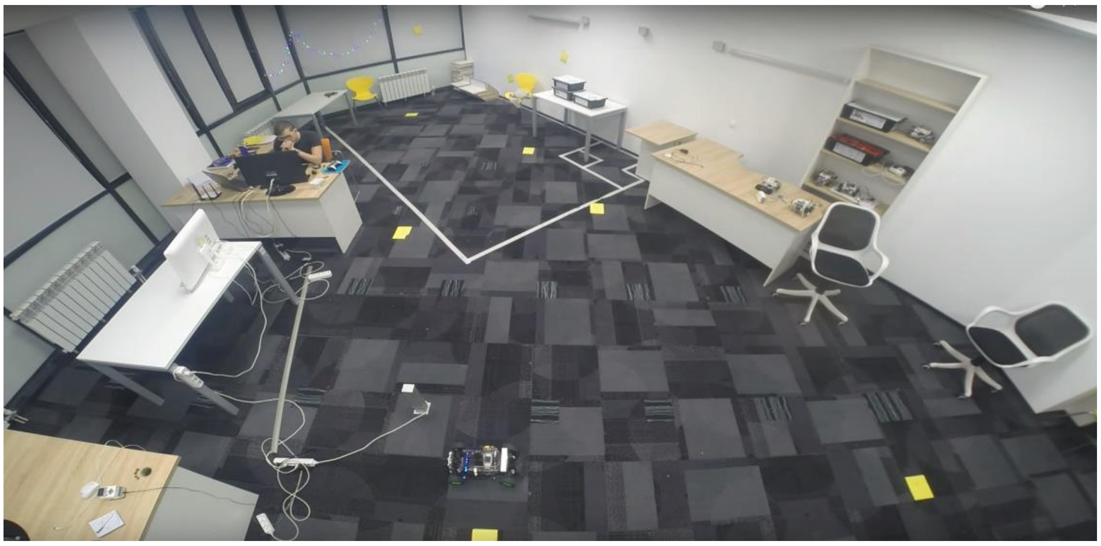
2. Dataset

It was collected a dataset to examine data from different sensors. Experiment was organized in this way. The robot was placed in test environment, which was usual office room. The robot was tele operated by kinematic mode from host computer via ROS. The robot has finished three full circles and record data from sensors.

The following videos show progress of experiment



Video available via link: https://www.youtube.com/watch?v=TLYLvVxFIdE



Video available via link: https://www.youtube.com/watch?v=RBj-wnN51l4

Only ORB SLAM was processed on dataset. All other algorithms computing in run time on board.

3. Sensor calibration

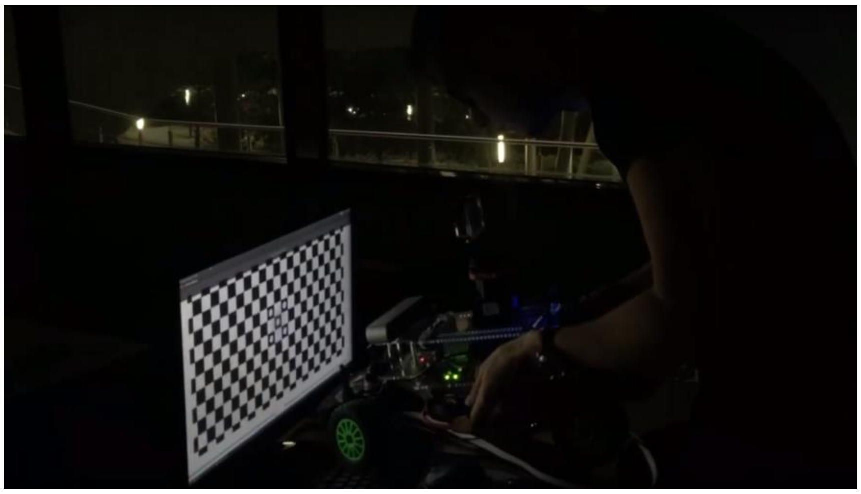
9DOF Razor IMU calibration

I found that default calibration does not provide enough precise information. It was necessary to do calibration before fusing data. The procedure of calibration was done how it was described here. I had many problems with the installation of programs for calibration on Jetson TX1 (which is used as the basis for the robot). It is found that there is a problem with running Oracle Java (which is required for run calibration software) on Jetson TX1 (last version, which I tried - Linux ARM 64 Hard Float ABI). Finally, calibration was done using another computer.

Video available via link: https://www.youtube.com/watch?v=6SqMOvdJ9xU

4. ZED camera calibration

ZED camera was calibrated. A precise calibration allows as to getting metric information about environment.

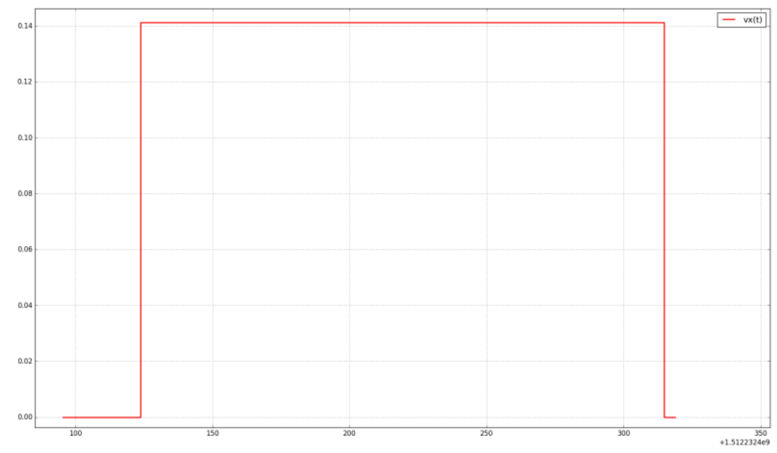


Video available via link: https://www.youtube.com/watch?v=oGXNo2LMekM

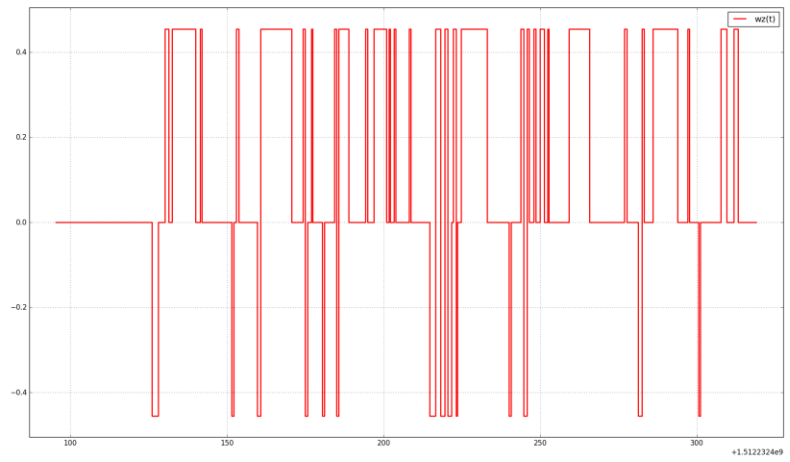
VI. Data

1. Control command Data

The robot has two main parameters. It are steering angle and speed of rotation of the motor. The robot already had a first principle model. Because of it we have as input control commands $vx(t) = f(rotation\ of\ the\ motor)$ and$wz(t) = g(sterling\ angel, parameters\ of\ the\ robot)$.



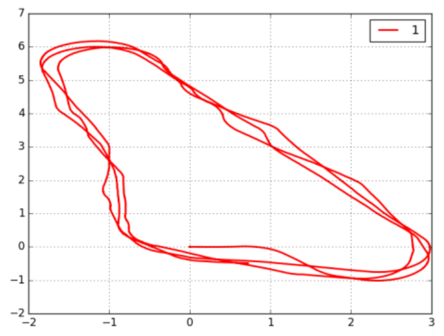
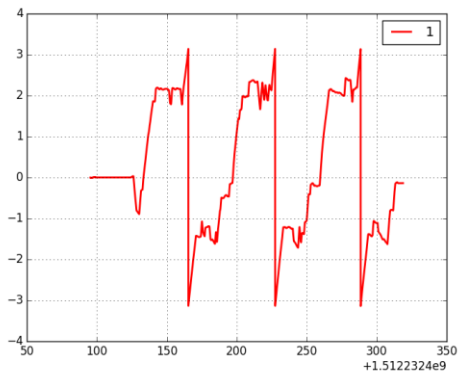
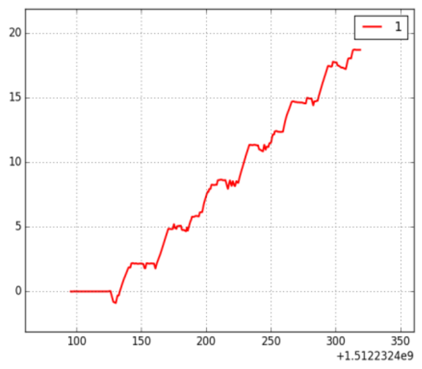
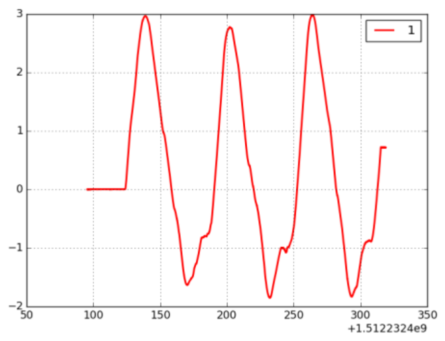
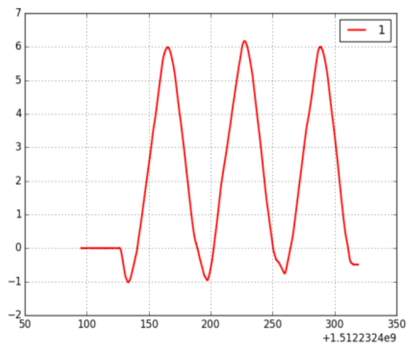
$vx(t)$ - Linear speed in coordinate frame attached to the robot.



$wz(t)$ - Speed of rotation.

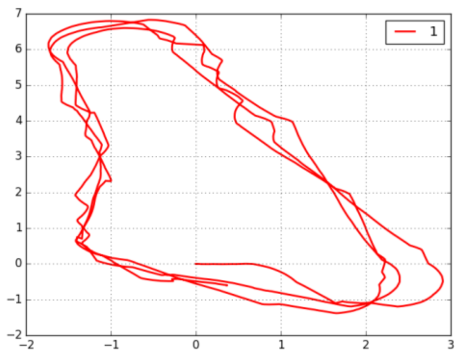
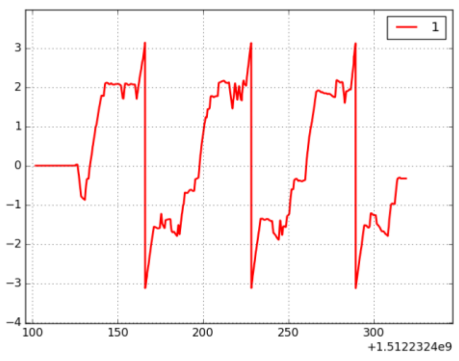
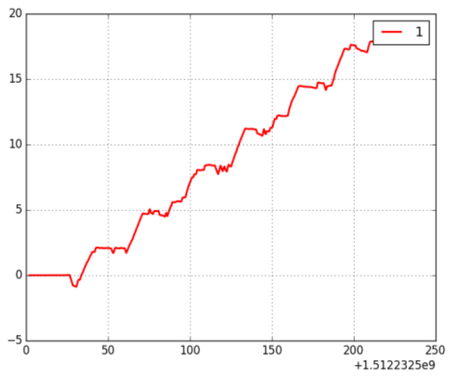
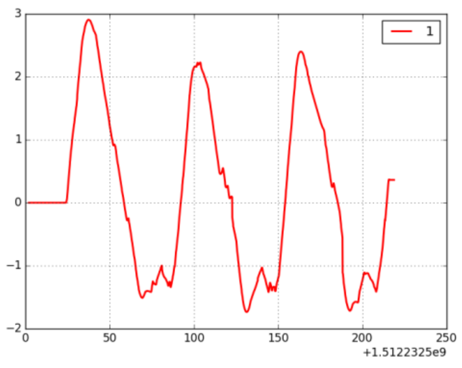
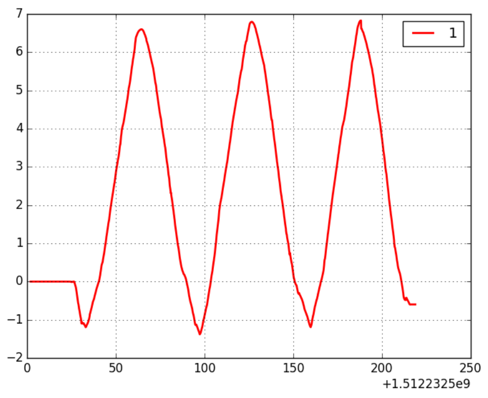
2. Pose Lidar Data

hector_slam (http://wiki.ros.org/hector_slam) was used for solving localization problem from LIDAR data.

	
The trajectory of the robot (x,y).	
	
yaw(t) normalized	yaw(t)
	
x(t)	y(t)

3. Pose ZED camera Data

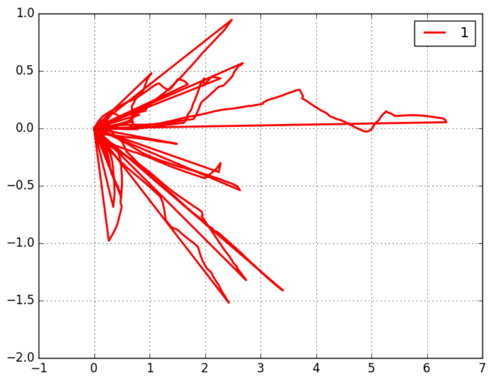
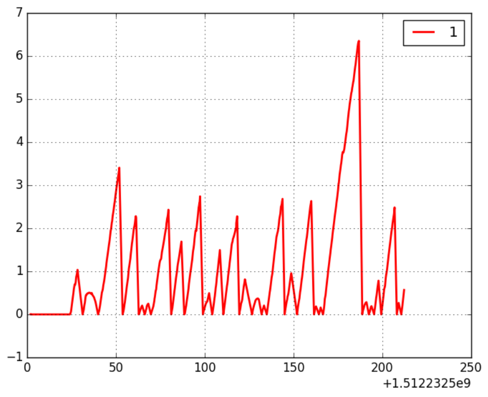
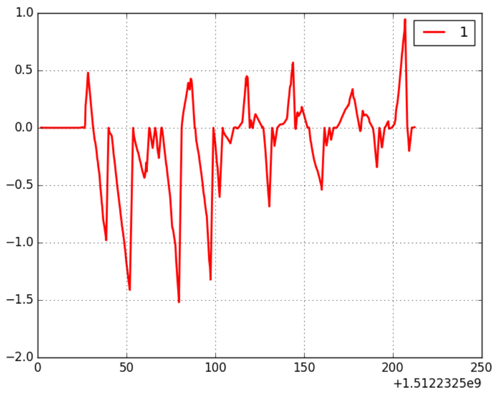
ZEDfu provides metric information about pose.

	
The trajectory of the robot (x,y).	
	
yaw(t) normalized	yaw(t)
	
x(t)	y(t)

4. ORB SLAM Pose Data

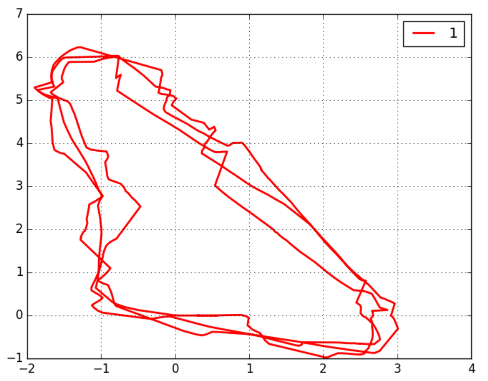
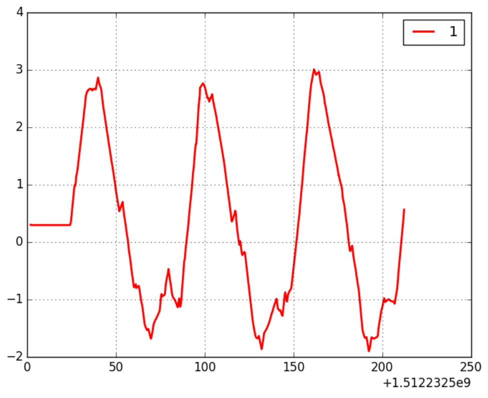
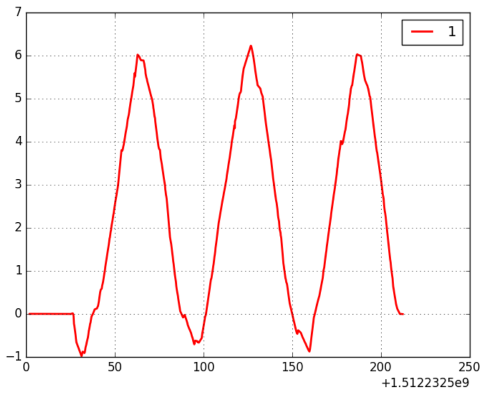
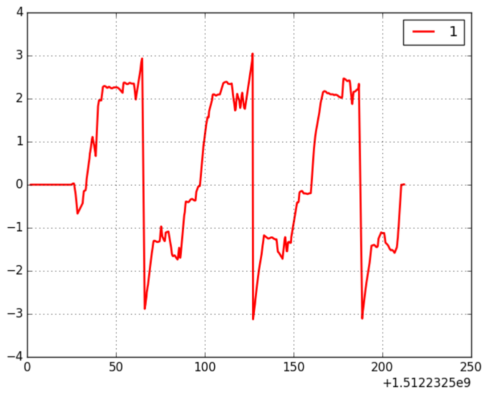
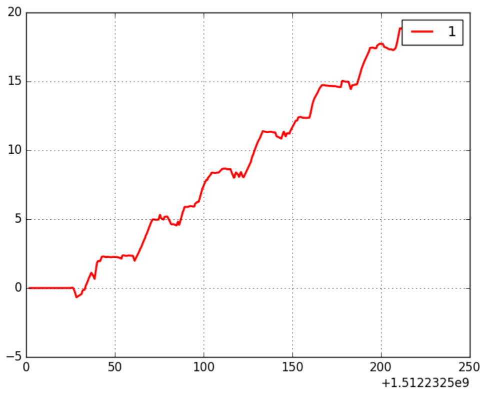
As i have calibrated stereo camera it is possible with ORB_SLAM to get metrical pose information. But the main problem that this approach is not resistant to the fast rotation, but it is provide additional information on straight line moving.

Raw data ORB SLAM odometry

	
(x,y)	
	
x(t)	y(t)

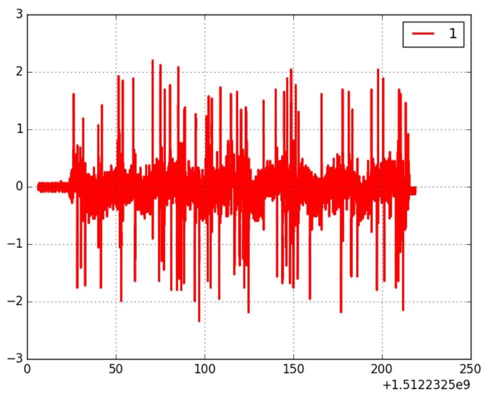
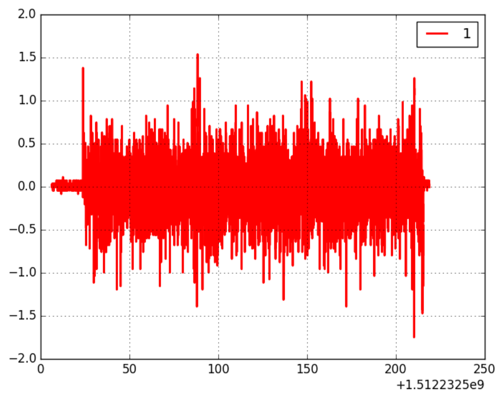
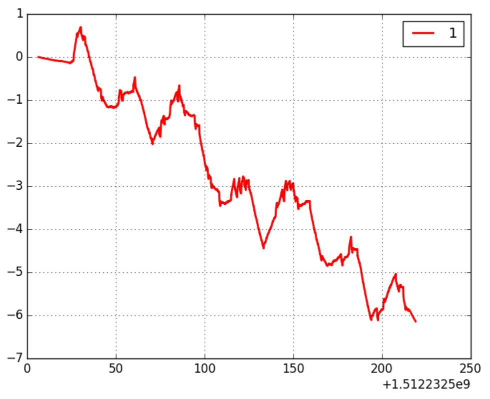
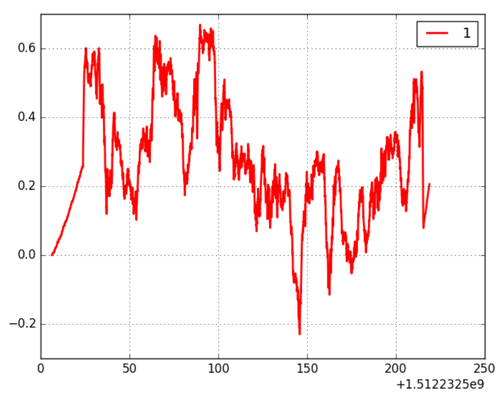
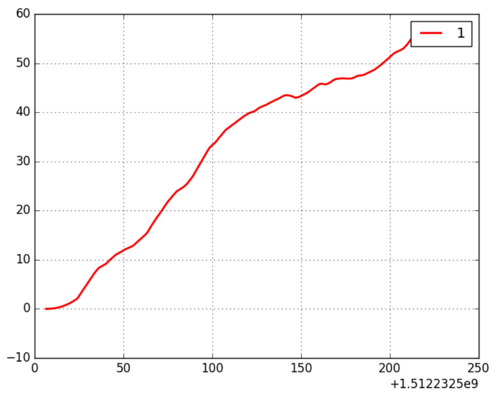
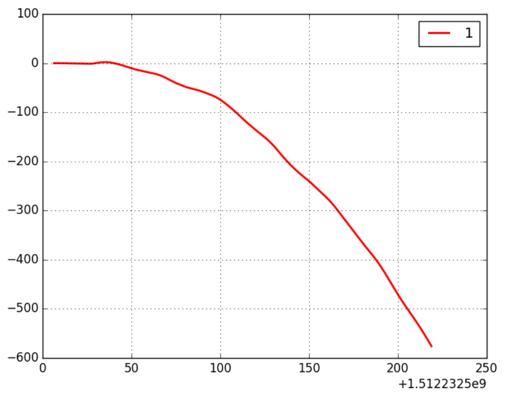
The result complete in line with expectation. As it lose track we have map reset and pose to zero update. The idea is initialize the each fragment with data from lidar.

The idea is to initialize each fragment with data from LIDAR odometry. Using this approach we can additional information from sensor about pose on straight enough segments of way. Finally, we have following graphics.

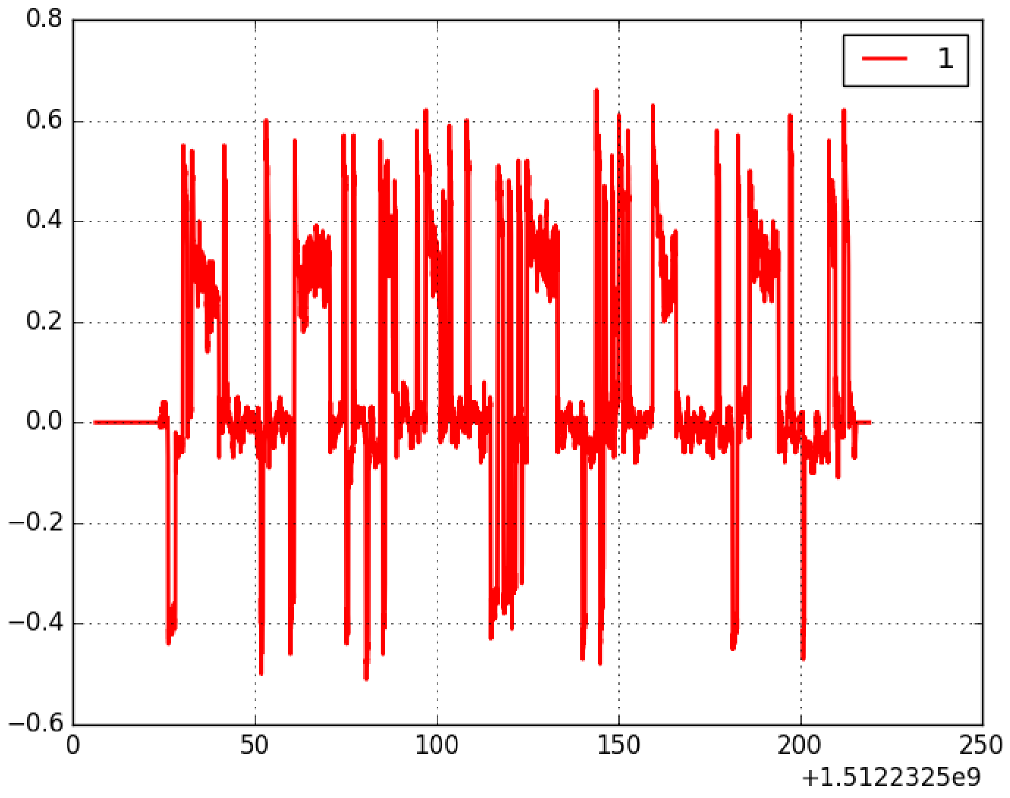
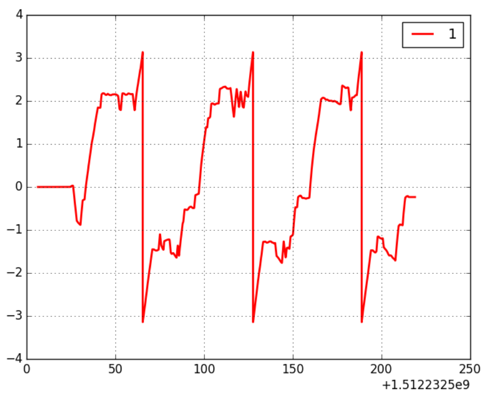
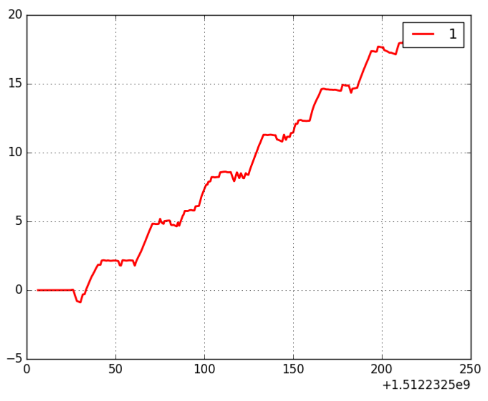
	
(x,y)	
	
x(t)	y(t)
	
yaw(t) normalized	yaw(t)

5. IMU data

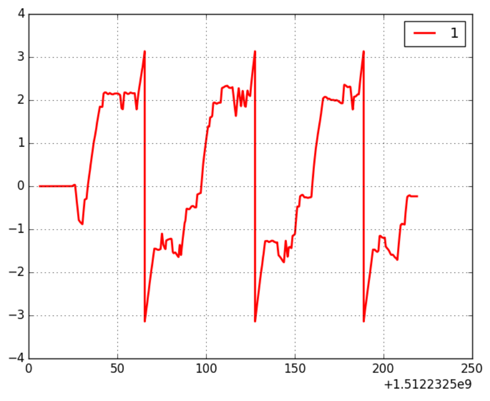
Accelerometer data

	
ax(t)	ay(t)
	
vx(t) - numerical first integral ax(t)	vy(t) - numerical first integral ay(t)
	
x(t) - numerical second integral ax(t)	y(t) - numerical second integral ay(t)

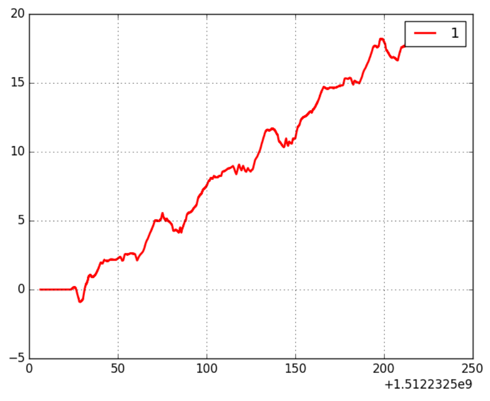
Gyroscope data

	
$wz(t)$ - speed of rotation.	
	
yaw(t) - numerical first integral with normalization	yaw(t) - numerical first integral

Magnetometer data



yaw(t) normalized

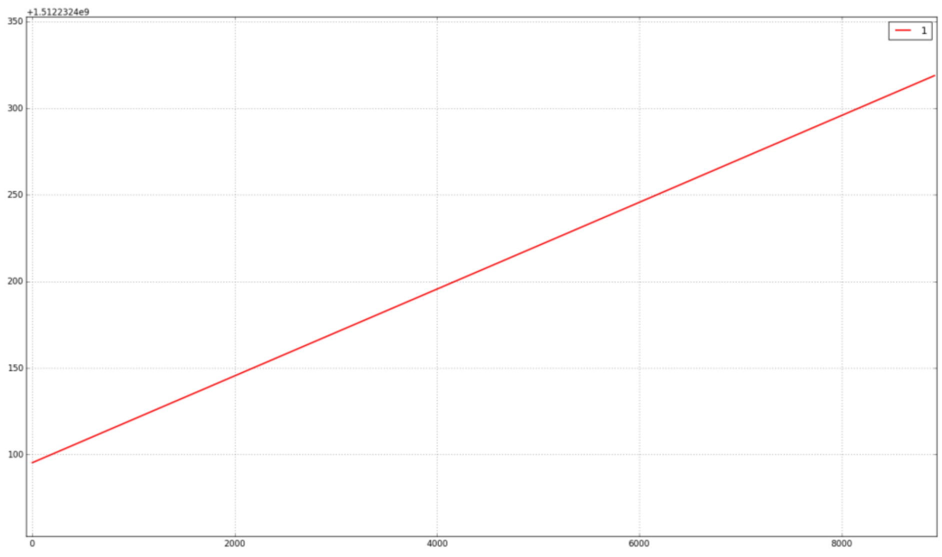


yaw(t)

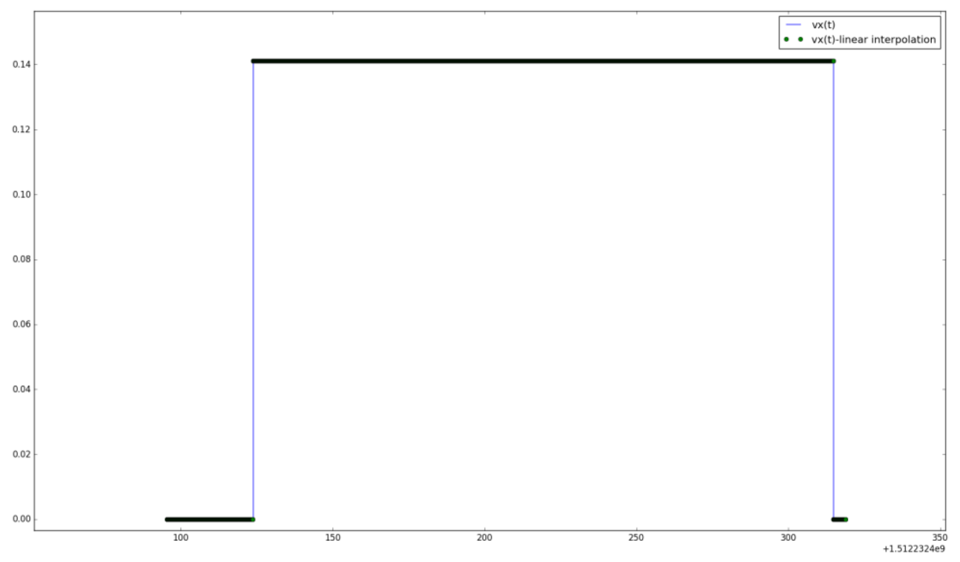
VII. Data Set Preparation

6. Control command interpolation

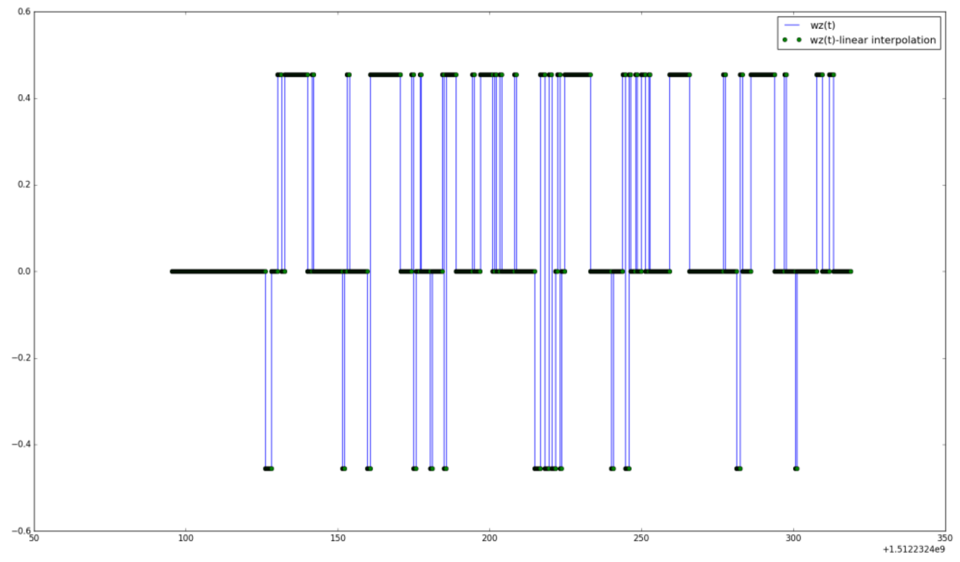
As we have different frequency of the data from control command and lidar. Control commands data were interpolated in time grid of hector_SLAM.



(№, t_i)- the time grid is linear. Time stamp is step 0.0250586 s.



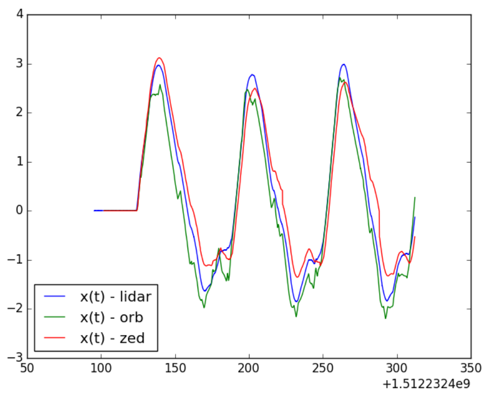
$vx(t)$- linear interpolation.

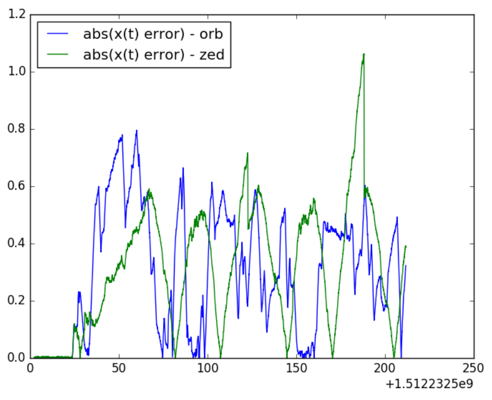


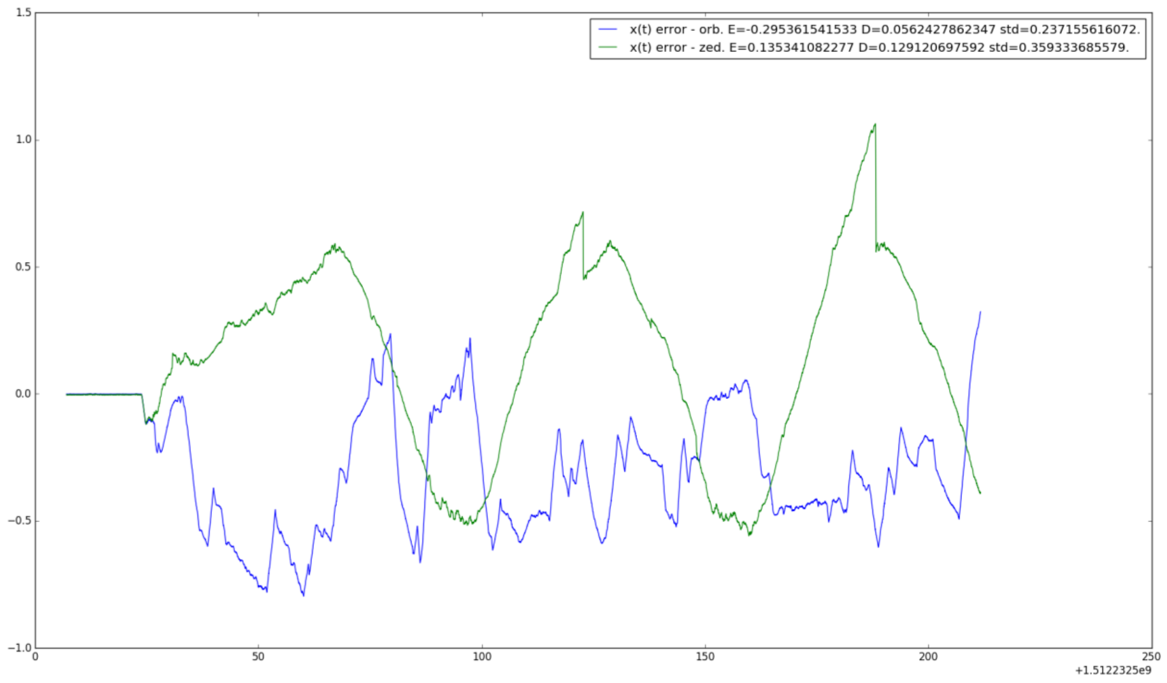
$wz(t)$- linear interpolation.

8. Data for sensor fusion

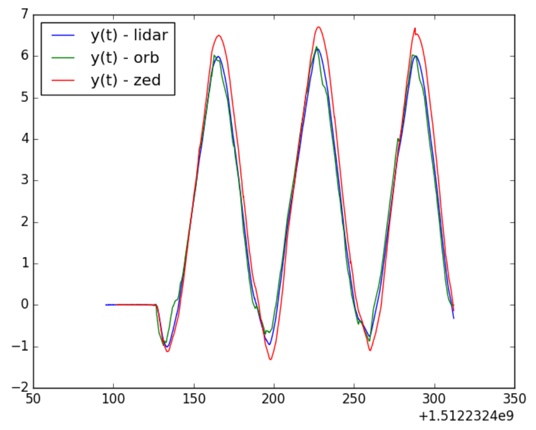
X_POSE data



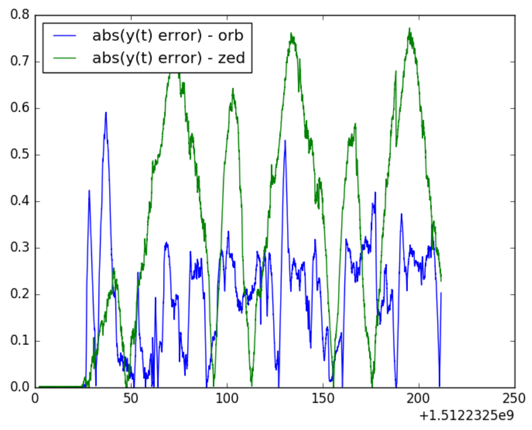




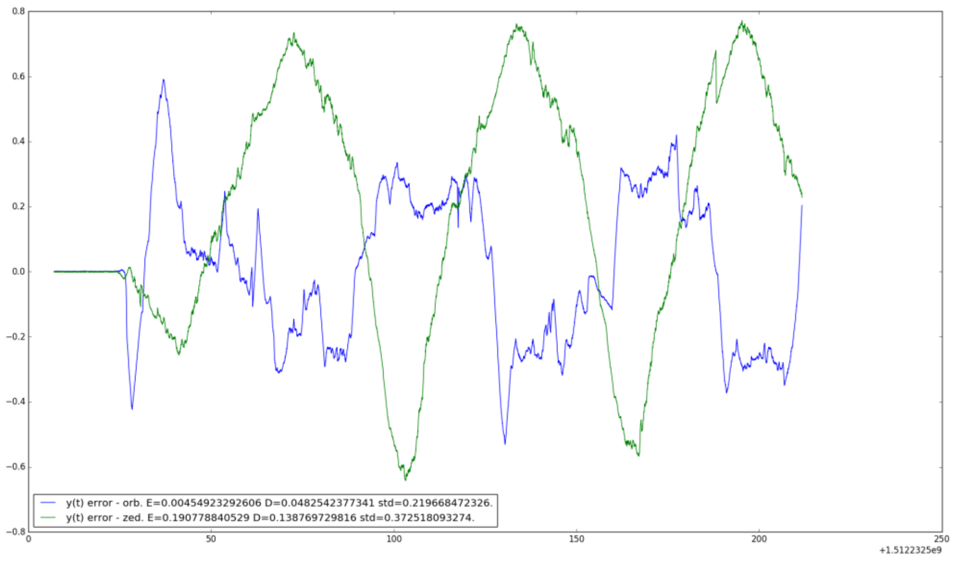
Y_POSE data



y(t) - lidar
y(t) - orb
y(t) - zed
7
6
5
4
3
2
1
0
−1
−2
50
100
150
200
250
300
350
+1.5122324e9

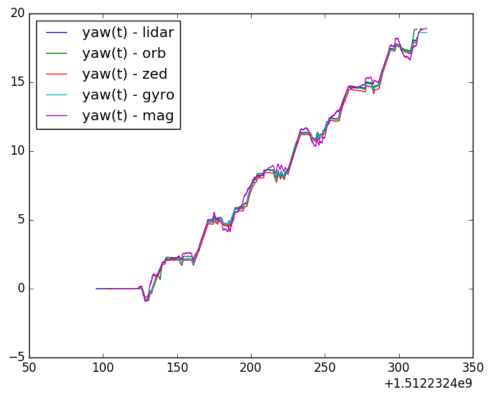


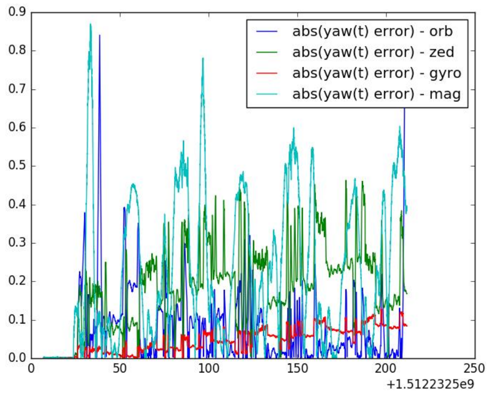
abs(y(t) error) - orb
abs(y(t) error) - zed
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
50
100
150
200
250
+1.5122325e9

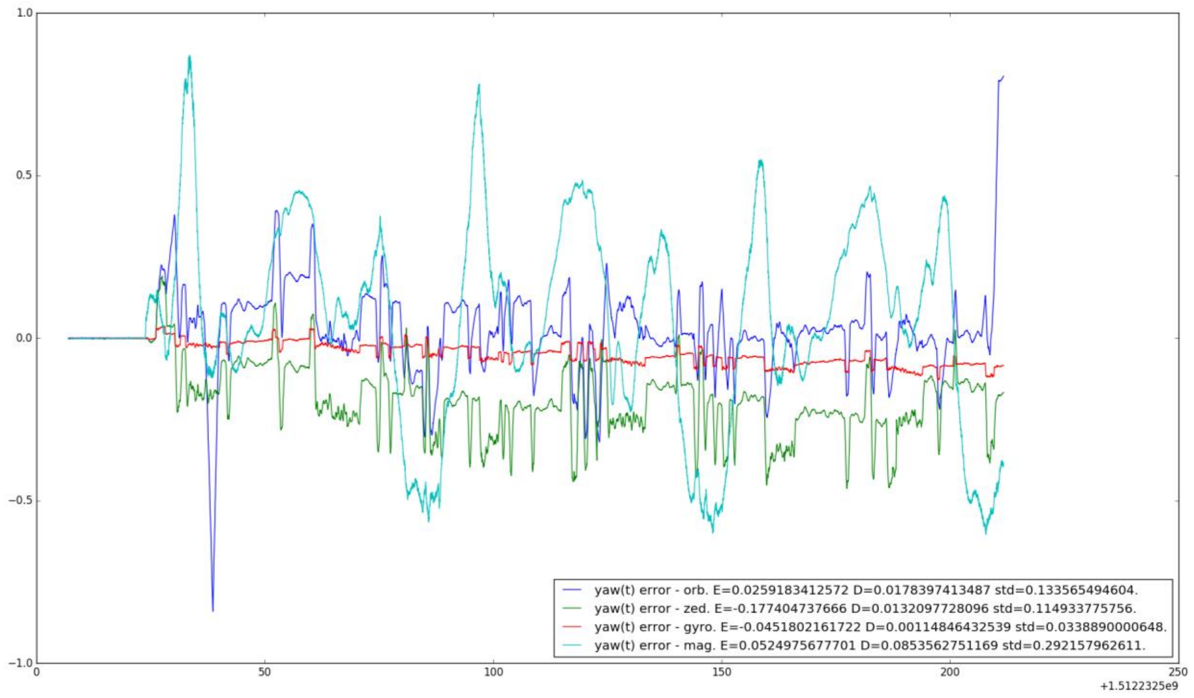


y(t) error - orb. E=0.00454923292606 D=0.0482542377341 std=0.219668472326.
y(t) error - zed. E=0.190778840529 D=0.138769729816 std=0.372518093274.
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
0
50
100
150
200
250
+1.5122325e9

YAW_pose data







9. Sensor Fusion

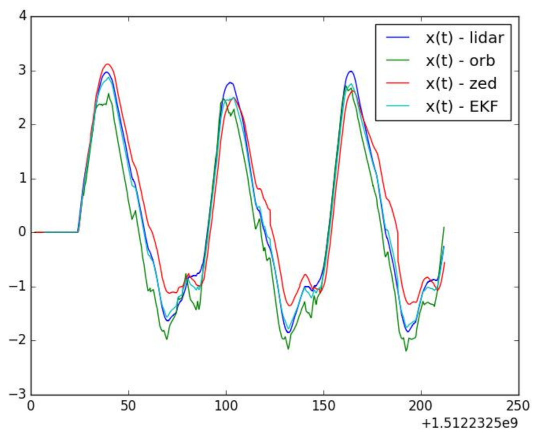
Motion model

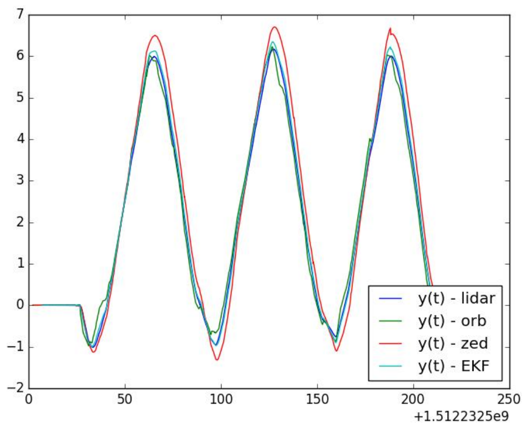
$$yaw_i = yaw_{i-1} + w_{z_i} \cdot dt_i$$

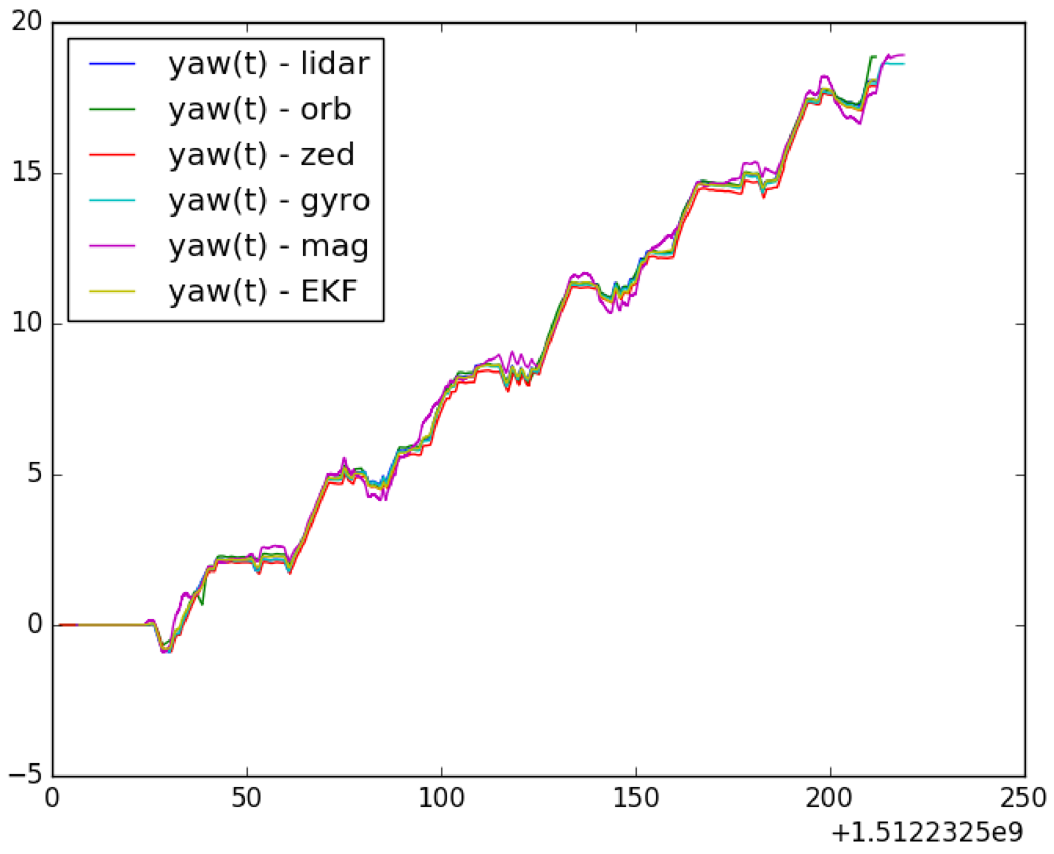
$$x_i = x_{i-1} + v_i \cdot cos(yaw_i) \cdot dt_i$$

$$y_i = y_{i-1} + v_i \cdot sin(yaw_i) \cdot dt_i$$

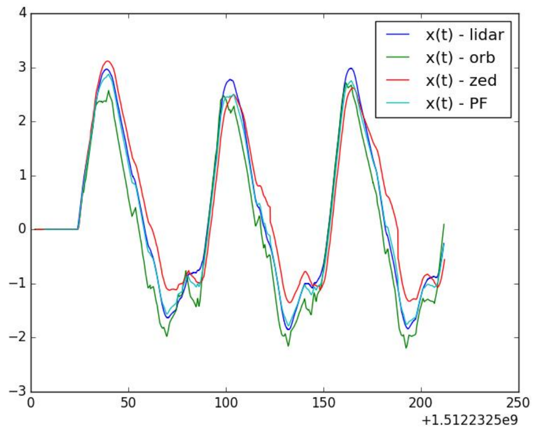
Extended Kalman Filter

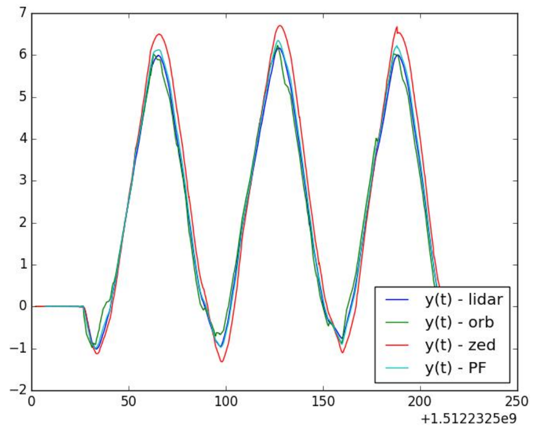


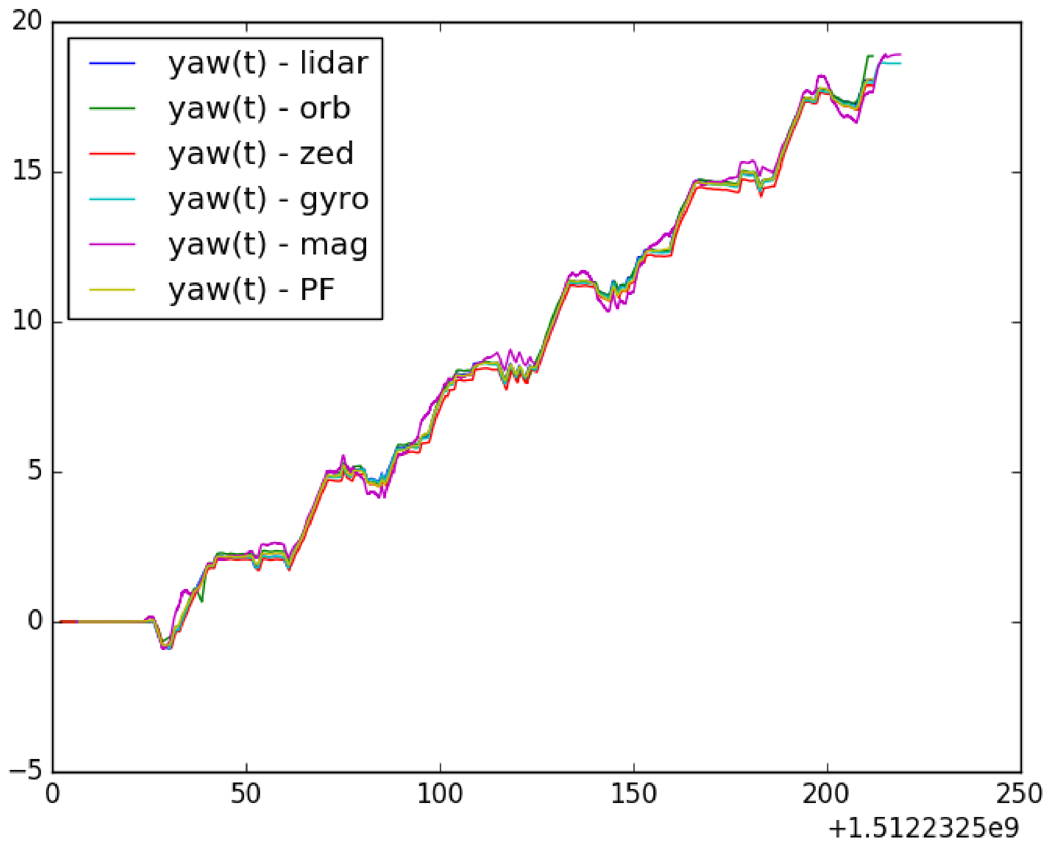




Particle Filter







VIII. Conclusion

During the work was designed real world autonomous car-like robot. Created robot model. Evaluated and comparing work of different sensors and methods for precise pose estimation of robot.