Technion – Israel Institute of Technology

The Andrew & Erna Viterbi Faculty of Electrical Engineering

Vision and Image Sciences Laboratory

Final Report for Project B

SLAM for Formula Driverless

Students:

Lital Guy

Maxim Aslyansky

Supervisor:

Eli Appelboim

Semester: Spring 2018/2019

Submission Date: April, 2019

Contents

List of figures

Abstract

This paper introduces our contribution for Technion Formula Student Team regards the Formula Student Driverless competition. In this competition, among other challenges, an autonomous racecar is tasked to autonomously complete 10 laps of a previously unknown racetrack as fast as possible and using only onboard sensing and computing. The two main challenges faced in such scenarios are (i) the lack of prior knowledge of the race track, and (ii) the possibility of a sensor failure hindering the operation of the car.

Our main goal in this project is to reconstruct the race track and estimate the vehicle position on the map while driving (SLAM). In order to achieve full speed while racing autonomously, the track must be known ahead. The car must thus drive carefully to discover and map the track. Once the map is known, the car can drive in Localization Mode which can exploit the advantage of planning on the previously mapped race-track. For Racing, it is essential that the mapping components are able to handle failures in the detection component. The camera or the laser could have short periods of blindness due to sunlight or uneven road conditions. In these cases, a cone might be missed.

We perform an extensive experimental evaluation on real world data, collected and labeled visual cone detection dataset, collected LIDAR and camera sequences as well as open source sequences used for evaluation. Our github repository containing all the code, roadmap and also some tutorials that we found useful.

This paper presents Introduction and prior work, our setup and system considerations, LIDAR Perception and LIDAR SLAM, Visual cone detection, Visual SLAM, building cone map, conclusions and future work.

Introduction and Prior work

"Formula Technion" is a cross-faculty endeavor that has been practiced and run (mainly by BSc. students) since 2012, with the goal being the construction of a fully functioning race car from scratch within only one academic year, every year. The "Formula student association" consists of over 600 universities from around the world. Every year the association organizes competitions in various countries, in which teams from universities from around the world compete. Through recent years, the Technion project has seen successes in competitions in Italy, Czech Republic, and Germany.
The rise of self-driving cars (led by the deep-learning revolution in computer vision and inference for decision problems) has had its effect on the competitions: 2017 was the first year in which FSG - the German competition - had opened its doors to student-made self-driving cars.

At the beginning of the project we focused on reading articles from the various groups that participated in the competition and particularly focused on the work of AMZ Driveless which is a group of students from ETH Zurich which won the competition two years in a row. Let us briefly review their solution
To ensure compliance with measurement errors and faults, they developed two parallel SLAM algorithms: one uses images from camera and IMU data as input-Visual SLAM and a similar algorithm that works with data from LIDAR- LIDAR SLAM. These two algorithms together with IMU data, GPS, wheel odometry etc. allows to achieve a high level of accuracy and fault tolerance.

After the first round algorithm detects the loop and performs its closure. Then it goes into Localization Mode where it only provides information on the relative position of the car on the map.

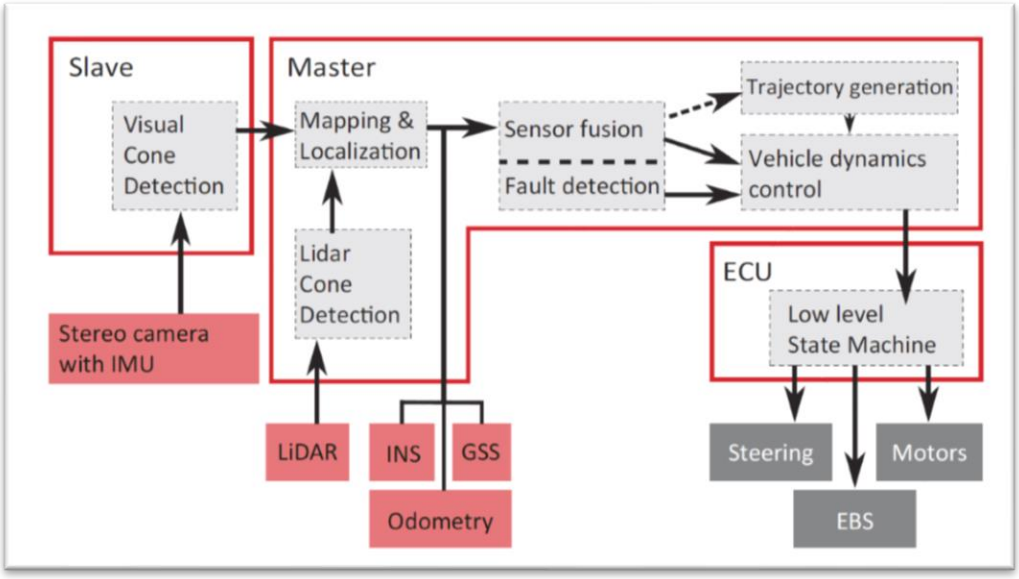

Figure 1. AMZ Driverless pipeline

Our approach

Similarly to AMZ we decided two keep LIDAR and Visual pipelines separate. The alternative to our approach would be fusing LIDAR and camera detections at the early stages of pipeline and then using a single SLAM algorithm two solve the problem, but in this case, we would have to develop it from scratch. At the beginning of the project, we also were limited by the lack of appropriate data set, and such separation would allow us to start the work faster.

Our Visual SLAM pipeline will receive streams from stereo camera and IMU as an input and will consist of a Visual Cone Detection Algorithm as well as SLAM itself. We assumed that we should be able to find an off-the-shelve VISUAL SLAM solution, so our task would be to test it and later integrate it with the detector.

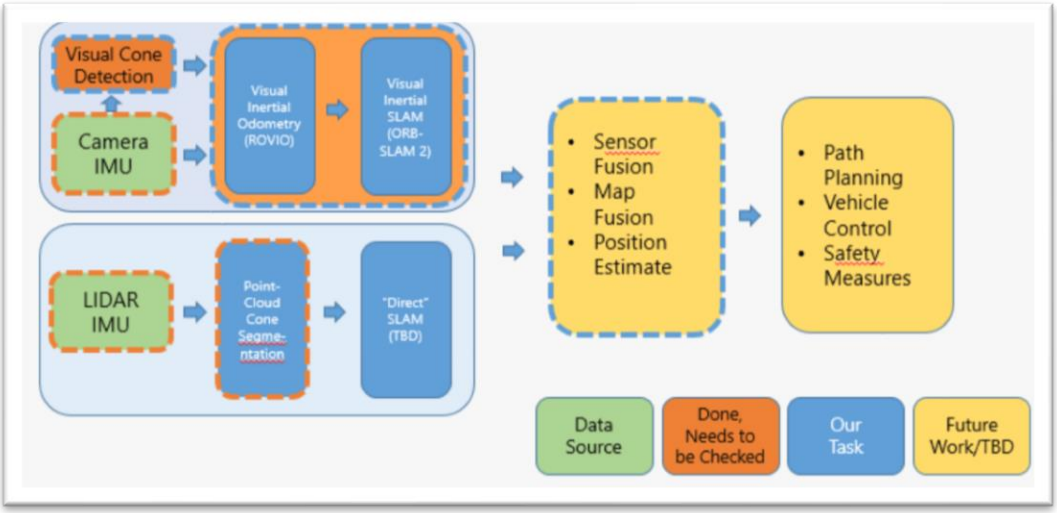

Figure 2. System overview

Although there are some algorithms that use LIDAR scans as an input (and we've evaluated some of them) we've decided to focus directly on building the cone map. This will obviously require a separate LIDAR Cone Detection Algorithm that we'll need to develop.

After these two parts are ready a separate algorithm for outlier rejection and map fusion should combine the maps and provide a accurate pose estimate in real-time.

We worked under Ubuntu 16.06 an ROS kinetic. We decided to use ROS library since it provides a wide variety of libraries, tools, packages and drivers which helped the library to gain a huge popularity in robotics community. So we decided to organize the algorithms as separate ROS nodes to allow easy integration to the main system in the future.

Our setup consisted of Velodyne VLP-16 LIDAR, ZED stereo camera, Nvidia Jetson embedded PC, Wi-Fi router and the car itself. During the project we managed to collect some data including several rosbags with video and lidar streams. Additionally we used some openly available datasets provided by AMZ driverless and EUFS Autonomous teams. These datasets included data from calibrated stereo rig, LIDAR and also wheel odometry, IMU and GPS.

LIDAR cone detection

LIDAR technology recently gained significant popularity in the field of autonomous driving due to their great accuracy, robustness and range. We found the sensor especially useful for cone detection, when it's placed correctly (which wasn't the case for our car).

The algorithmic steps are summarized below:

Implemented using PCL library and ROS (added a node receiving **velodyne_points**)

1. Remove distant points (higher than 1 metr and farther than 25)
2. Use RANSAC with planar model with following parameters: 3 cm threshold, deviation of at most 10* from z-axis
3. Filter all points below some threshold as points corresponding to the ground, and all points higher than 30cm as not relevant
4. Apply Euclidean clustering on the points that are left with some low threshold
5. Filter point clusters by size and extract centroid

We found the algorithm to perform reasonably well for all the sequences that we tested while being exceptionally fast. Some false positives are visible since we tried to leave as much real cones unfiltered. The algorithm can be farther improved by compenstaing motion distortion in LiDAR scans using the velocity estimates and applying temporal filter after aligning the scans.

Results of our algorithm on data captured by EUFSA team:

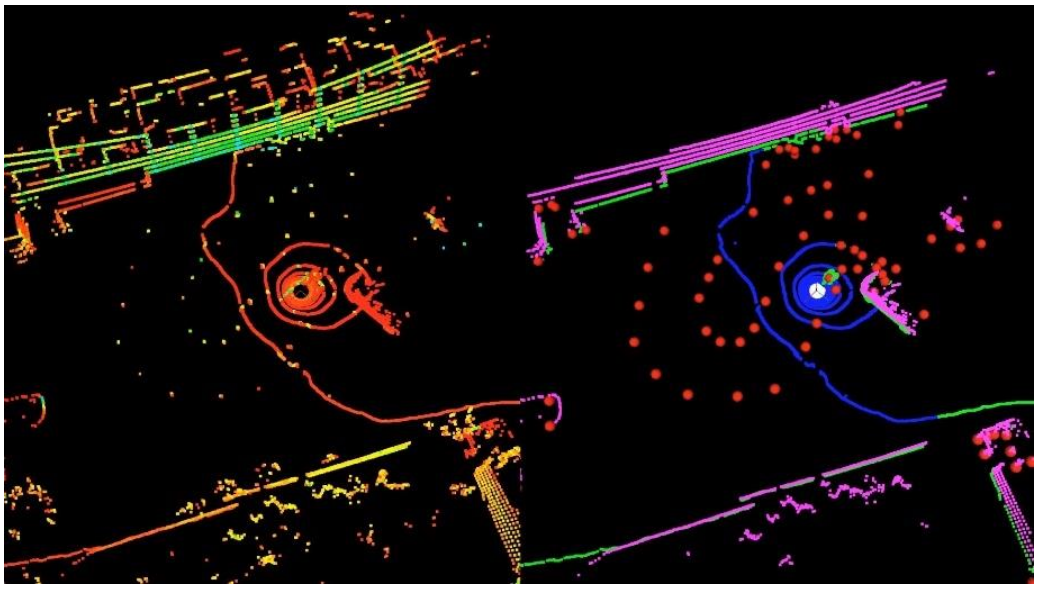

Figure 3. Input point cloud vs algorithm segmentation

Left: input point cloud (color – LIDAR intensity)
Right: segmentation produced by the algorithm (red- cone clusters, magenta- points that are too high, blue – ground removal, green – everything else)

LIDAR SLAM

Since developing a custom landmark-based SLAM algorithm from scratch is challenging and time-consuming task we decided to test some open-source algorithms for LIDAR SLAM. Usually they work by estimating rigid transformation between the map and current frame (which caused by the object motion and thus can be used to estimate it) and the fusing the aligned frame with the map.

We tested some off-the-shelve LIDAR SLAM algorithms including: loam_velodyne, hdl_graph_slam, LeGO-LOAM and cartographer. Loam_Velodyne performed the best but still wasn't good enough especially in more challenging scenes. LeGO-LOAM is based on the previous algorithm and promised better performance but actually was worse. hdl_graph_slam haven't compiled and cartographer needed too much effort for integration so we couldn't test it.

In our opinion the main reasons for such a poor performance were fast speed of vehicle, ego-motion artifacts and lack of distinctive features (large objects or planes rather than sparse features such as cones and rings of ground). Our conclusion is that a custom landmark-based SLAM algorithm should be developed from scratch.

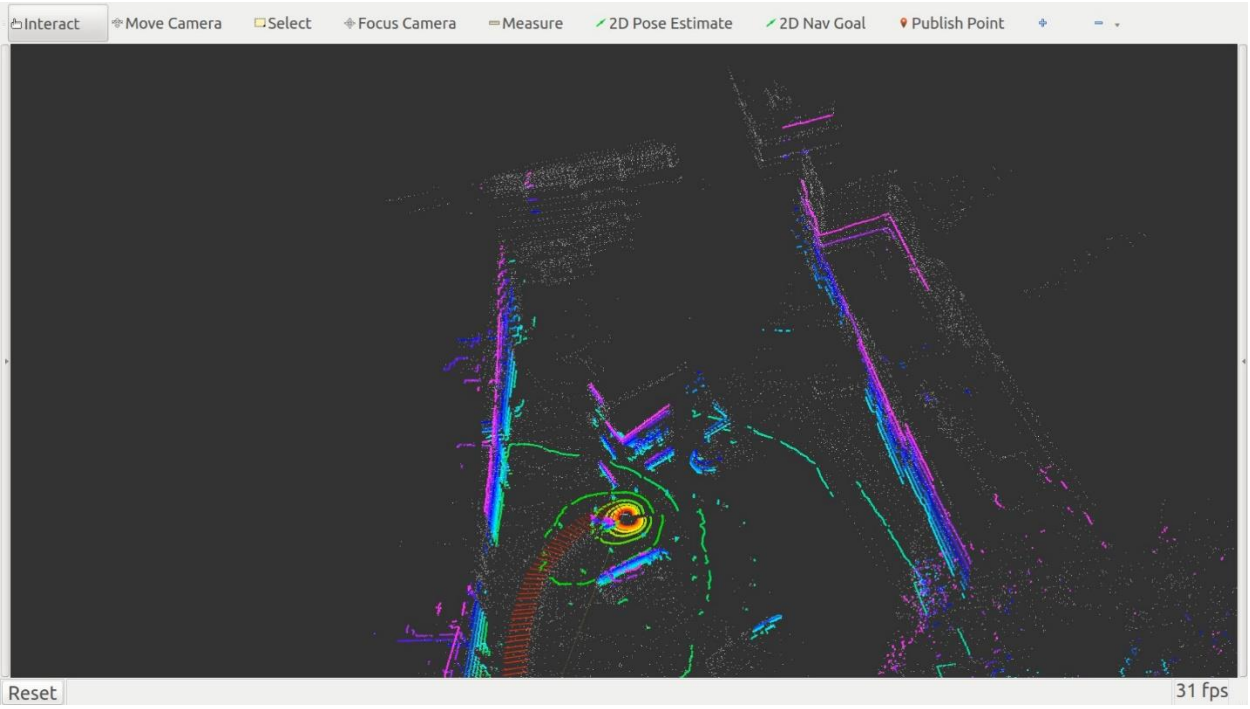

Figure 4. A frame from LOAM map
Gray: canonical model of the algorithm (map reconstruction by accumulating point clouds)
In color: current LIDAR frame
Large misalignment artifacts can be seen, in more challenging scenes algorithm won't converge at all

Visual Cone detection

Next, we decided to develop a Visual Cone Detection algorithm. Inspired by recent success of CNN-based object detectors we started preparing our own dataset contained over 400 frames from various videos with approximately 8000 hand-annotated cones in them (although later we used pre-trained network to initialize the dataset and then only corrected its mistakes).

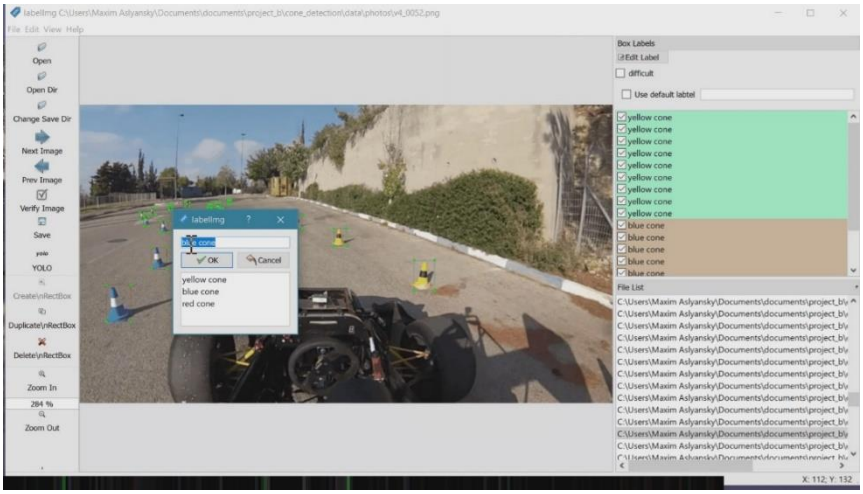

Figure 5. LabelImg – a GUI tool for object annotation

We also tried to use AirSim simulator which could provide us synthetic dataset with varying road and weather conditions. The only algorithmic change to be made is adding automatic bounding box calculation but, unfortunately, we hadn't have enough time to finish the work.
Additionally the simulator can provide ground truth depth data which can be used to train the network to regress it as well. This way Visual SLAM could become redundant or at least significantly simpler.

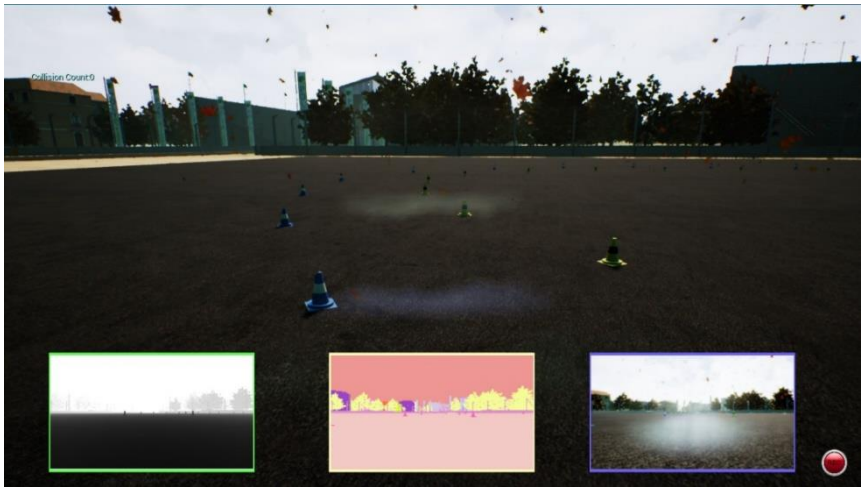

Figure 6. AirSim – racing simulator

After some research on the topic we decide to use YOLOv3 implemented in Darknet framework as our object detector, since it is the fastest algorithm while providing enough accuracy for our application. The network is trained on three classes: blue, yellow, and red cones. Network parameters like the anchor box size, non-maximum suppression and confidence thresholds are tuned on a self-acquired dataset to reduce both false positives and inaccurate bounding boxes. The network is trained on images with varying illumination and augmentation for better generalization and robustness in real world applications.

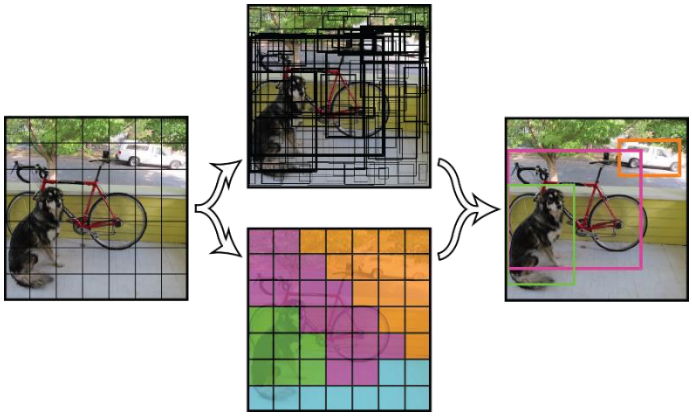

Figure 7. YOLO detector architcture

Additionally, we performed network architecture search in order to achieve the best speed/accuracy tradeoff. We found that increasing grid resolution, network resolution, adding two more YOLO layers and adding skip connection increased the network accuracy and ability to recognize smaller objects while staying inside our computational budget (60 fps on gtx 1050, 100 fps on gtx 1080 ti).
We also trained the network on grayscale images (in order to estimate impact of using a monochromatic camera) and observed some accuracy degradation especially for small and far cones.

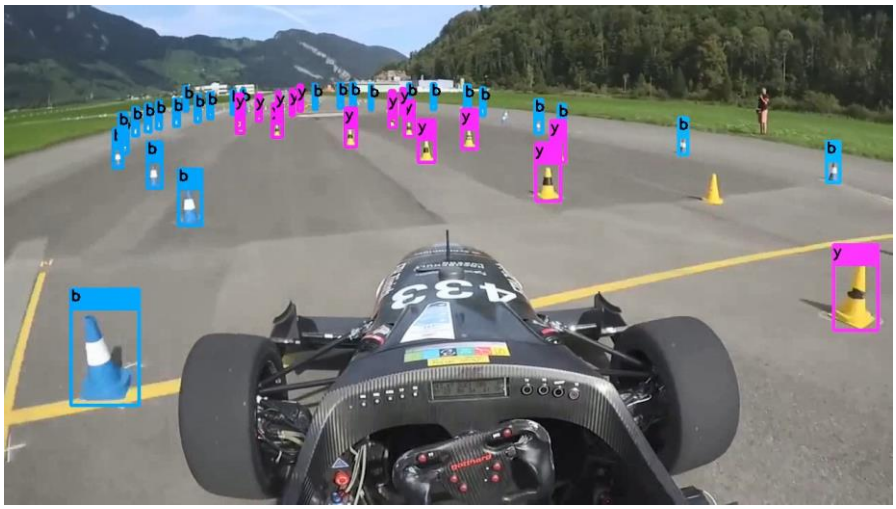

Figure 8. Visual Cone detection – results on test set

After evaluating our network on our test set, we concluded that network generalized well on new data, significantly outperforming competing solutions (probably due to dense annotation).

Visual SLAM

After some literature overview we decide to use ORBSLAM 2 since the system has many distinct advantages over other SLAM algorithms:

1. Efficiency and robustness achieved by reusing the same features for tracking, mapping and place recognition as well as careful key-frame selection which helps to reduce computations compared to filtering approaches
2. Loop closure after which the map is aligned and global bundle adjustment step is performed. This one is especially useful to detect and close the loop after the first round.
3. Localization mode which works by freezing the map and only providing pose localization data, thus greatly reduces computational cost of the algorithm. This feature will be useful for us during the next rounds after loop closure.

Due to its popularity ORBSLAM 2 has many modifications and improvements some of which we also tested. Since at the time we haven't had appropriate hardware we sticked to the dataset provided by AMZ Driverless which featured a calibrated stereo rig as well as synchronized imu. The car in the sequence drove very fast which later proved to be a big challenge for the algorithms that we've tested.

ORBSLAM 2. The main issue was tracking loss caused by large and fast camera rotation leading to no overlap between consecutive frames aggravated by the slow speed of ORBSLAM in stereo mode (which doubles the load on detection and tracking Front-end) which led to slow insertion of new keyframes. We also tested the algorithm in sub-real-time modes of operation where video stream was fed at half and quarter frame rates which slightly improved performance but the algorithm still was not able to complete the loop.

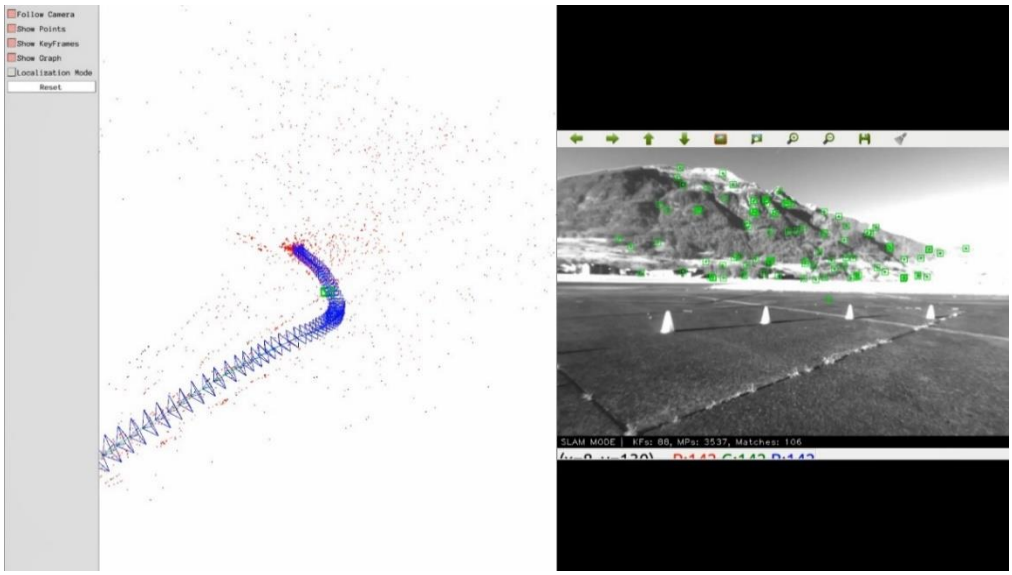

Figure 9. Results of ORBSLAM 2 – tracking loss can be observed

We also tested orbslam_DWO which features double window optimization in order to account for IMU data. The algorithm ended up being significantly slower than ORBSLAM 2 while not providing any benefit in terms of robustness, so we discarded the algorithm as well.

SERVO is algorithm developed by AMZ Driverless team. The main contribution of the algorithm is using ROVIO to initialize SLAM algorithm's pose estimate which greatly improves overall robustness to fast motions and high speed. Although initially we run into some issues caused by incompatibility of ROS versions we managed to solve them and test the algorithm.
Although the algorithm crashed when loop was detected and also showed some inconsistency in results during different launches in general it performed much better mainly due to the fact that now ROVIO (robust visual intertial odometry) provided very good initialization (pose estimate) to ORBSLAM which helped to keep tracking and finish the lap successfully.

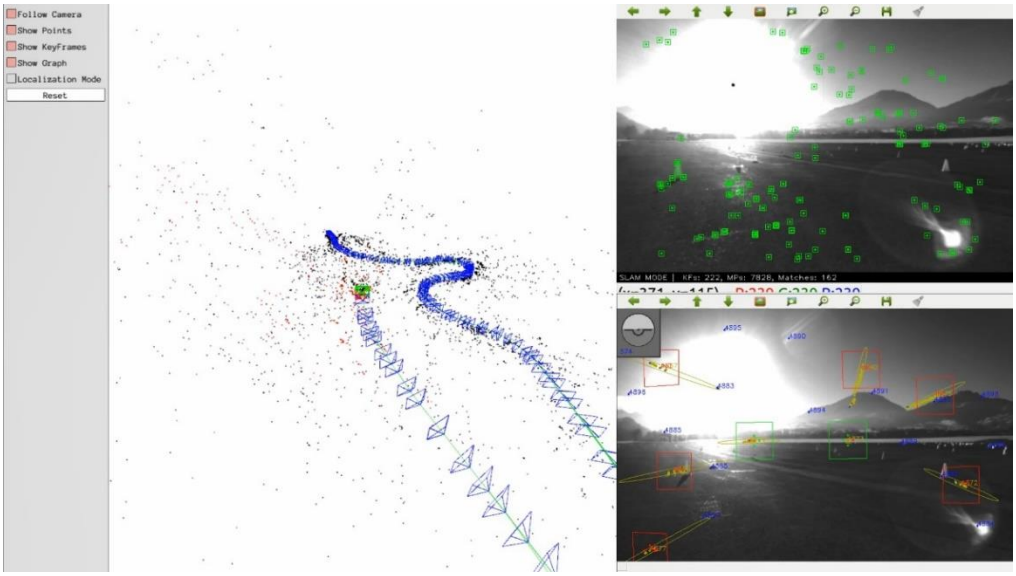

Figure 10. Results of SERVO

Although SERVO performed very well on the provided dataset it should be integrated to our system and evaluated on captured data to test it in a wider range of use-cases and also check Loop Closure and Localization Mode features more extensively.

Adding Cones to the Map

Unlike AMZ driverless we decided to extend the usage of the visual slam beyond pose estimation by also building the cone map as part of main ORB feature map.

We found two approaches to the problem:

The first is to assign a cone label to each of the detected features which overlap with bounding boxes of detected cones. Such approach was already used in the field and commonly known under the name of semantic slam. At the end we would get a 3d point cloud with semantic labels for potential cones which could be clustered to obtain the 3d cone map.

The second option being explicitly adding new features for each detected cone.
In this case one or several features would be extracted from the region of detected bounding boxes (or even a slightly bigger region containing more additional "context") and then adding them to the list of regular features detected by FAST algorithm. Optionally, to improve matching performance and reduce interference between the two types of features, FAST features that overlap with cones could be removed and also constraint to match "cone features" only with features of the same type and label could be introduced.

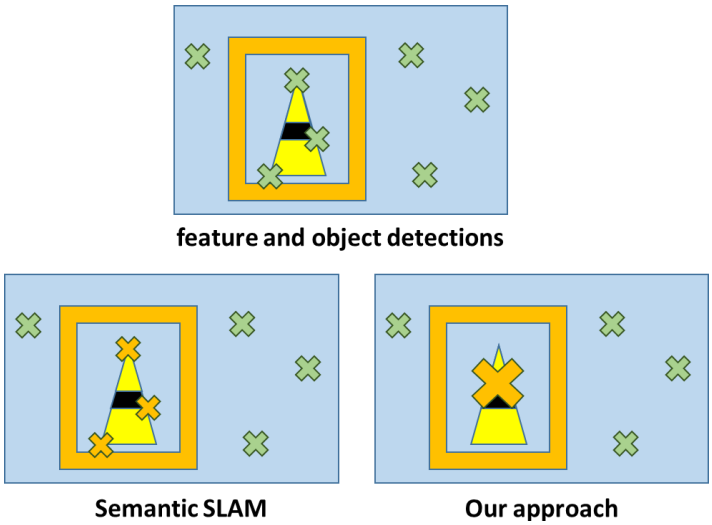

Figure 3. Possible approaches for adding detected objects to the ORBSLAM map

So our algorithm would crop the bounding box region extended by some margin **m** and then resize it to the size used to extract ORB features (21x21 px) and perform descriptor calculations. Usually feature orientation need to be estimated first in order to assure rotation invariance but we haven't applied it since cones' orientation assumed to stay constant.

We've implemented both approaches but couldn't test them due to lack of data - we needed a sequence containing the cones from RGB or monochromatic stereo rig calibrated and synchronized with an IMU. Unfortunately, we had only a sequence from AMZ driverless containing the wrong cones and calibrated camera rig with IMU wasn't available to us at the time.

Cone Tracking

Inspired by the success of ROVIO which uses direct visual tracking approach to achieve higher tracking robustness (instead of feature matching used by ORBSLAM) we decided to implement a custom cone tracking algorithm based on our detector and also KLT tracker implemented in OpenCV library.

The goal of our tracking algorithm is to provide the ability to distinguish between different objects by assigning each one of them an unique ID which should stay the same across different frames. To achieve this at each frame **n** we calculate the motion of a small patch surrounding each detected object in order to propagate the boxes to frame **n+1**. Then we "connect" each propagated object to the closest detected object in frame **n+1** which lie in certain radius. In case such object wasn't found we continue to propagate that object up to three times and then remove t from our active list.

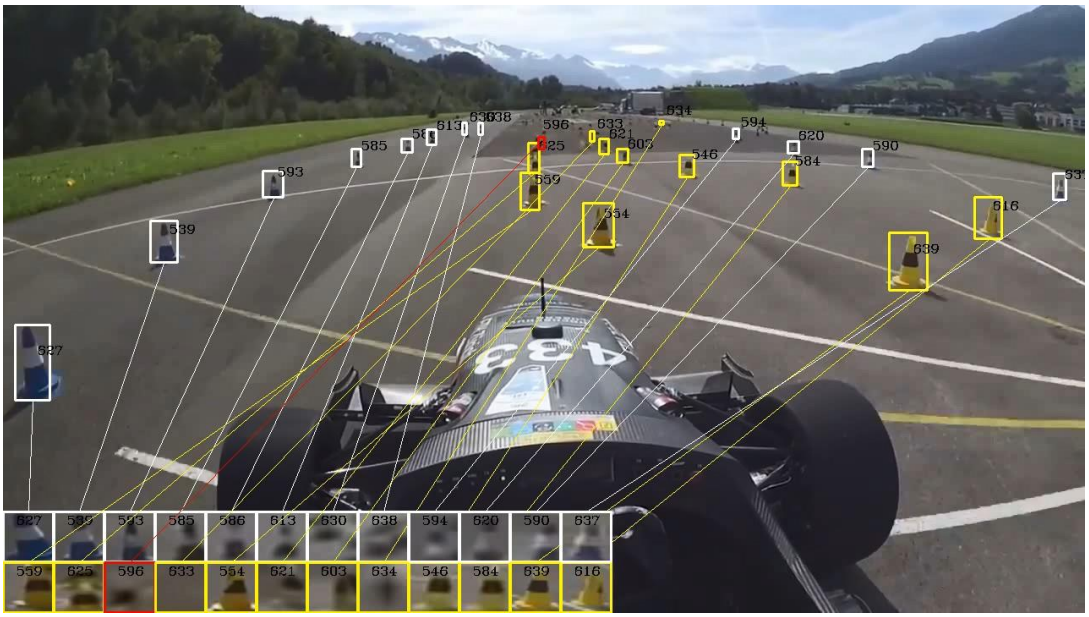

Figure 4. KLT object tracking

From our evaluation we conclude that algorithm performed well in most cases but sometimes tracking was lost due to fast motion, especially near the edges of the frames. The algorithm could be significantly improved by utilizing depth prediction and IMU data which could provide accurate estimate of future cone position reducing search radius and thus increasing speed, accuracy and robustness.

Future Work

LIDAR cone detection

- Need to compensate for ego-motion
- Discard outliers better by using neural network or by using color data from the camera
- Track or filter temporarily (after alignment of consecutive maps)

Data and Equipment Availability

- LIDAR should be placed better (higher) and calibrated with camera
- Need for a calibrated stereo rig with IMU with global shutter

Better Cone Detection Dataset and Network

- Use AirSim: diverse and accurate data, depth prediction and modify the network accordingly (currently bounding box generation is lacking)
- Add depth and IMU information to improve tracking (similarly to ROVIO)

Integrating the Pipelines

- Option A: Finish our work, Implement Kalman Filter and FastSLAM/g2o-based similarly to AMZ: the FastSLAM 2.0 algorithm needs to be adapted to map and localize in real-time using the output of either one or both perception systems.
- Option B: Develop robust fusion of LIDAR and camera data and use single custom SLAM algorithm

To summarize, in our opinion the best way to solve given problem is to modify our neural network to be able to regress depth and train it on synthetic data. In this case with help of epipolar geometry 3D cone point cloud can be received each frame. After calibration with LIDAR these two sensors could now be much easily fused together with a single SLAM algorithm removing the need for complicated multiple-stage pipelines.

Conclusions

This paper presented the approaches developed to ensure reliable operation of an autonomous race car by introducing redundancy into the perception and state estimation pipelines. It has been shown that accurate cone position estimates can be obtained from LiDAR and, and accurate positions can be estimated from cameras using prior knowledge of objects.

We developed several crucial algorithms with emphasis on high accuracy, robustness and speed, still there is a lot of work to be done to achieve final working solution.

LIDAR SLAM: proposed cone detection algorithm was fast and robust but it needs some refinement. Evaluated SLAM algorithms performed poorly, so a custom solution is needed.

Visual Cone Detection: our fine-tuned network achieved better accuracy compared to the competition (subjectively), we also believe the network can be further improved with synthetic data and extended

with depth prediction. We also developed an algorithm for cone tracking which would benefit from depth from the network and IMU.

Visual SLAM: SERVO demonstrated great robustness on provided data but needs to be tested with our HW. We proposed and implemented two ORBSLAM 2 extensions for adding a semantic label to SLAM map and thus building map of custom objects (cones in our case). This algorithm should be tested in the future.

References

1. Miguel I. Valls, Hubertus F.C. Hendrikx , Victor J.F. Reijgwart, Fabio V. Meier, Inkyu Sa, Renaud Dube, Abel Gawel, Mathias B ´ urki, and Roland Siegwart. “Design of an Autonomous Racecar: Perception, State Estimation and System Integration” International Conference on Robotics and Automation (2018)
2. Nikhil Bharadwaj Gosala, Andreas Bühler, Manish Prajapat, Claas Ehmke, Mehak Gupta, Ramya Sivanesan, Abel Gawel, Mark Pfeiffer, Mathias Bürki, Inkyu Sa, Renaud Dubé, Roland Siegwart. “Redundant Perception and State Estimation for Reliable Autonomous Racing”. International Conference on Robotics and Automation (2019)
3. Raul Mur-Artal and Juan D. Tardos. “ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras” IEEE Transactions on Robotics (2017)
4. Joseph Redmon, Ali Farhadi . “YOLOv3: An Incremental Improvement”. Computer Vision and Pattern Recognition (2018)