Capstone Progress Report

GFR Group 61 & 62

CS461 Senior Software Engineering Project

Fall 2018

December 3, 2018

Abstract

Research and development of camera, lidar, odometry, and SLAM subsystems for the GFR driverless formula style racing car has begun. This document will cover the progress of the GFR driverless formula student project over the fall 2018 term.

CONTENTS

1 INTRODUCTION

1.1 Terminology

- **Lidar** A visual sensor that fundamentally works like a radar but uses lasers instead of radio waves
- **GPS** Global Positioning System
- **IMU** Inertial Measurement Unit
- **ROS** Robot Operating System
- **Faster RCNN** Faster Region-based Convolutional Neural Network
- **YOLO** You Only Look Once
- **SSD** Single Shot Multibox Detector
- **SLAM** Simutaneously Localization And Mapping
- **GFR** Global Formula Racing

1.2 Purpose

Autonomous vehicles require robust internal representations of the world in order to make correct driving decisions. The driverless vehicle must utilize sensor input from lidar, camera, accelerometer and GPS in order to construct a virtual representation of the world around it as well as localize itself within that world. Due to the inherent error and low sampling rates of the sensors, high speed motion can cause data about the environment to be sparse and noisy. Robust statistical analysis is needed to evaluate the confidence of the vehicle's internal representation of the world around it. Our team is tasked with constructing the vision, mapping, and localization systems for a driverless formula style racecar for the Global Formula Racing (GFR) team. The driverless vehicle is expected to compete in the Formula SAE (FSAE) competition in August 2019.

1.3 Scope

The purpose of our system is to create a robust virtual representation of the vehicle's environment. The virtual representation will be generated using sensor input from the camera, lidar, GPS, and IMU. Algorithms will be developed for detecting the location and color of the cones using lidar and camera. The cone locations and colors will be used to construct a dynamic map of the world, updating the map as the vehicle moves and new cones are seen. As the vehicle moves, our system will need to update the position of the vehicle within the map. The location of the vehicle and cones will be used to compute the optimal path and velocity to successfully maneuver around the track.

2 PROGRESS

2.1 Camera

2.1.1 Research

Three object detection models were considered for solving the problem of cone detection: YOLO, SSD, and Faster RCNN. After comparing results of the three models, Faster RCNN has been determined to be the best solution for our problem. The bounding boxes produced by YOLO had significantly lower accuracy on smaller

objects when compared to Faster RCNN while SSD had trouble even detecting some of those small objects. Detecting small objects is very useful for our vision system because it allows us to correctly localize cones much farther from the vehicle. The main drawback of Faster RCNN is the speed. Faster RCNN performs slower than the others by almost 50% in some benchmarks. However, our team has access to current top-of-the-line GPUs, so the speed of Faster RCNN is currently not much of a concern.

2.1.2 Development

Four datasets of images have been generated from videos found of driverless FSAE vehicles maneuvering around the same cones used for the competition. 31 images have been labeled using the web service LabelBox and exported to a JSON format. The json labels were then converted to the COCO data format using a custom python script. 1000+ more images from three additional videos are available to label. In the coming weeks, these images will be uploaded to LabelBox and help will be requested from all members of the driverless project to help label the images.

The open source project Detectron (developed by Facebook's AI Research Division) along with machine learning frameworks Pytorch and Caffe2 are being tested to evaluate the efficacy of Faster RCNN for our cone detection algorithm. A prototype Detectron model has been trained with promising initial results. While evaluating the prototype on two videos the model wasn't trained on, 0 false positive cones were observed and at least 50% of all cones within 5 meters of the car were accurately detected. However, these results were acquired with a model that was trained on a laptop GPU with 3GB of video memory when Detectron expects at least 4GB of video memory. To get around the memory limitations, the training and testing images needed to be scaled down. Once our team has access to a better GPU, we will be able to realize the full potential of this algorithm, and have many more ways to tune the configuration of the model.

2.1.3 Problems

The installation of Caffe2 was met with frustration and compilation errors. Multiple combinations of various CUDA and cuDNN versions (both required for Detectron) were installed in an attempt to get Caffe2 to compile correctly. Prebuilt binaries were successfully installed using conda. However, this is a temporary solution since the prebuilt binaries do not support the AVX512 CPU instruction set, making CPU bound instructions significantly less efficient. By the end of the year, we hope to find the right set of dependencies that allow us to compile from source code. Additionally, g++ 5.5.0 and Python 2.7.9 have incompatibility issues with Caffe2 and Detectron. However, g++ 6.5.0 and Python 2.7.8 worked just fine.

2.2 Lidar

In the lidar component, we spent the first 1-2 weeks get up to speed with GFR organization and the development and communication tools that are being used. Week 3-6 was spent to learn about lidar, how to process its scanning data, and most importantly, the state of real-time lidar object detection in self-driving car. In week 7 to week 10, we discussed, decided, and designed our solution to the problem.

2.2.1 Research

The lidar is responsible for scanning the surrounding physical environment of the car. In the current state of lidar processing in self-driving/self-racing car system. The ultimate goal and core of lidar processing pipeline is the cone detection task. There are 3 candidates for our cone detection algorithm: DBSCAN, SqueezSeq, and RANSAC (along with some 3D transformation, suggested by Dr. Grimm). DBSCAN is a very classic density-based clustering algorithm for spatial data. SqueezSeq is the state-of-the-art neural-based real-time object segmentation model for general self-driving car. RANSAC (random sample consensus) is an iterative fitting algorithm that is very resilience to datasets that contain many outliers and is effective with known fixed-size objects like cones. Based on our discussion with Dr. Grimm, we decided to implement RANSAC for our cones detection task because this method fit well with our problem and situation.

2.2.2 Development

The operations and development processes are quite sophisticated. The scanning data flows through many stages of processing before the output is produced. From the scanned data, the ROS node detects and calculates the location of cones relative to the car as well as estimates the color of cones based on intensity values of the laser scans corresponding to those cones. The lidar continuously scans the physical environment around the car as the car is moving at a speed of at most 60 miles per hour until being turned off. Point cloud data from the lidar sensor is then published to lidar processing node in ROS. In real time, the lidar processing node differentiates between points that are cones and points that are not cones. The lidar processing node then computes the 2D coordinate of the cones relative to the physical location of the lidar sensor mounted on the car. And finally, the lidar processing node predicts and differentiates between the different colors of detected cones based on their position relative to the lidar sensor and the intensity values of the points in the point cloud corresponding to those cones. Based on this nuance of semi-dependent, multi-stage data processing, the pipeline data processing architecture fit our problem the best and we, like other racing teams and self-driving car teams in the world, chose this data processing architecture for our lidar component. For the cone detection algorithm in specific, we use RANSAC along with 3D hough transform to find the ground plane and then remove the ground plane. RANSAC is also used to fit cone located in the point cloud after the ground plane is removed.

2.2.3 Problems

One of the problems is the performance of our implementation. The RANSAC and 3D hough transformation and all things related have to perform under 50 ms, which is the time between 2 point cloud scan streamed from the lidar. We are quite optimistic about solving this problem because the team is well-experienced in high-performance C++ implementations and the algorithms have been tested in other real-time tasks in various projects at Oregon State. Another problem is that there may be rain and/or snow that will make the lidar data and our processing output extremely unreliable. Our solution for this scenario is to develop a method to detect, at any moment in the middle of the race, if it's raining and/or snowing. Once one of these two bad condition is detected, the lidar will automatically turn off and the SLAM component will only use the output from the camera component from then on.

2.3 Odometry

2.3.1 Research

The odometry system uses sensor data from 3 accelerometers and a GPS to determine where the car is and how fast it's moving. This output data will then be used in localization and dynamic controls. The fusion of the sensor data requires a nonlinear multiple-input multiple-output filter model. For this, several options were considered: extended Kalman filters, unscented Kalman filters, and particle filters. We were heavily pushed to implement a particle filter solution by our client, which was also validated by independent research.
A motion model for the car was also necessary. Originally a simple translation/rotation model was considered, but after communicating with robotics department faculty a more robust three step model was chosen. This model uses an initial rotation to rotate to the proper heading, translates a fixed distance along that heading, and then applies a final rotation to correct for the end heading.

2.3.2 Development

Most of the work for this term has been heavy research and design, considering different sensor fusion methods or existing implementations that may work for our purposes. That being said, existing ROS odometry packages have been tested in the simulation environment we are expected to use, and a generalized particle filter has been implemented. Thus we now have a strong understanding of the technologies available and the requirements of the odometry component as they stand. The whole driverless team is expected to produce a working implementation that functions at a basic level by January 7th, which is a realistic time frame for odometry.

2.3.3 Problems

The definition, interfaces, and expectations for this component were continuously refined throughout the term, we had to respond with reevaluations and redesigns of our own. This meant implementation work wasn't worth the effort, because a changed design requirement could - and has - invalidate most of the prototyping that has been done.

2.4 SLAM

2.4.1 Research

The SLAM system is responsible for mapping the surroundings of the car, as well as localizing the car within the map. There were three SLAM methods considered. Kalman filter, particle filter, and graph SLAM. The first few weeks of the term were spent researching SLAM methods. The decision was to use a particle filter. A particle filter would be implemented using the FastSLAM algorithm. Graph SLAM is not as common as the particle filter or Kalman filter. Graph SLAM also had less documentation than the other options so it was decided against. The Kalman filter is a common method for SLAM implementations, but it was also decided against due to its reliance on linear models and Gaussian noise. The extended Kalman filter (EKF) partially solves this problem by linearizing non linear models, but was ruled out in favor of the particle filter. The particle filter was chosen as the best option because of its ability to fit non linear models and increased accuracy. The particle filter is more robust in this respect in comparison to the EKF.

2.4.2 Development

The SLAM algorithm is divided into two subsections: open loop (i.e. the first lap) and closed loop (i.e. the following laps). Initially we decided to implement the particle filter for both subsections since that optimized accuracy. However, after much research and speaking to the head of OSU's robotics department, Dr. Grimm, we decided to use the particle filter only on the open loop calculations. This is when accuracy is the highest weighted criteria. For closed loop, speed becomes the primary area of concern. The Kalman filter has much less computational overhead and can increase the speed of our vehicle dramatically, but it has trouble working with nonlinear environments. This is why we decided to use an extended Kalman filter which linearizes the data produced by the open loop calculations. This allows us to use the computational speed of the Kalman filter but with a nonlinear system.

Therefore, our development will be based upon a particle filter for open loop and an extended Kalman filter for closed loop. The SLAM team considered using an open source FastSLAM algorithm for the open loop, particle filter implementation but chose instead to build one from scratch in order to keep the implementation as simplistic as possible.

2.4.3 Problems

Because the GFR is an international team, we have to remotely work with German side to develop our algorithm. The communication between us and Germany is one of the challenges we are facing. Our original plan was to have the Germans finish implementing open loop localization on FastSLAM1.0 by the end of November and then we will develop closed loop localization and upgrade the algorithm to FastSLAM2.0. However the German team did not finish the code by the aforementioned deadline so we are still trying to getting contact with them to come up with an alternate plan. In addition, we still have some design decisions that need to be made. Specifically, in order to get a better synchronization with sensors, we have an option to turn our SLAM algorithm into a set of two asynchronous nodes, one containing a localization thread that is always running and updates at the same rate of the cameras and lidar. This thread would use a EKF and will not update the map; another mapping thread that runs slower, uses a particle system from FastSLAM, and is less concerned with localization. This lack of communication between both teams is a primary concern that we wish to ameliorate as soon as possible.

2.5 Preliminary Design Review (PDR)

A presentation was developed for Dr. Cindy Grimm, Dr. Bill Smart, and Bob Paasch, three professors of mechanical engineering and robotics at Oregon State University. The presentation consisted of in depth overviews for all subsystems of the vehicle. Useful feedback was given to aide in the direction and development of our systems. The PDR was used to evaluate if each subsystem would be ready to enter into competitions in the summer of 2019..

3 RETROSPECTIVE

Component	Positives	Deltas	Actions
Camera	Created initial dataset of 31 images. Trained initial model on laptop with promising results. Created script which takes live video and draws bounding boxes around cones.	Create larger dataset of 150+ images to use for version 1 of camera vision node. Train on higher end GPU in order to generate more accurate model. Create prototype of ROS node to take camera data and output detections.	Client plans to bring in personal desktop PC with high end GPU to use for training. Templates for camera nodes in ROS can be found online and used for first pass.
Lidar	Finalized the data processing architecture for lidar node Got the lidar scanning visualized in simulated environment	Train cone detection machine learning model with RANSAC and 3D transformation on data from simulation and real life test drives	Implement the lidar data processing with pipline architecture Implement cone detection machine learning model and test it in simulated environment
Odometry	Plenty of faculty resources available for Odometry. ROS implementations of odometry already exist and may be usable for part of the project.	Definitions and interfaces changed often despite being presented as "decided upon".	Meet with client, decide upon one final definition and adhere to it.
SLAM	There are plenty of online sources that are able to point us in the right direction when it came to researching the different types of filters. The Germans have already begun implementing the first version of our system (i.e. FastSLAM1.0).	Changing the filters we are using for closed loop. Altering the interfaces interacting with SLAM by deciding to not use velocity data for our open loop calculations since the vehicle will be moving at such a slow pace.	Take over the implementation of FastSLAM1.0 from the Germans and begin working on the closed loop algorithm.