Tell Me What You Can See: Sensor Fusion

Supervisor: Prof. Dr.-Ing. Olaf Hellwich; Company: Hella Aglaia

Anjaly Ealias
anjaly.ealias@campus.tu-berlin.de

Shreyas Gokhale
s.gokhale@campus.tu-berlin.de

Tuo Kang
tuo.kang@campus.tu-berlin.de

Christopher O'Hara
ohara@campus.tu-berlin.de

Hariprasanna Prabakaran
hariprassana.prabakarans@campus.tu-berlin.de

March 31, 2019

1 Introduction

The global report on road safety by WHO has highlighted the fact that the number of casualties due to road accidents has touched 1.35 million a year. That's an astounding 3700 people dying due to road fatalities every day. Millions suffer from life long injuries or are severely disabled. Such losses take a huge toll on the victim's families and communities. WHO reports that rapid urbanization, poor safety standards, lack of traffic rule enforcement, driver fatigue and distraction, driving under the influence of drugs and alcohol as the main cause of road accidents [11].

With the advent of self-driving vehicles, the automotive industry has responded and tried to allay these fears. However, a plethora of challenges lie ahead to execute the seemingly simplest of tasks. In reality, a task as simple as detecting and parking in an unoccupied parking space is not a trivial task for autonomous vehicles. This project, in cooperation with Hella Aglaia GmbH, seeks to do just that: detect unoccupied parking spaces. The focus of our group (the sensor fusion team) is related to the implementation of sensor fusion from the various sensor inputs. The secondary step is to use this information to validate the efforts of the other teams (Camera, LiDAR, and RADAR). The project is developed using Hella Aglaia's Cassandra development framework. A visual representation of the output of sensor fusion is provided in Figure 1.

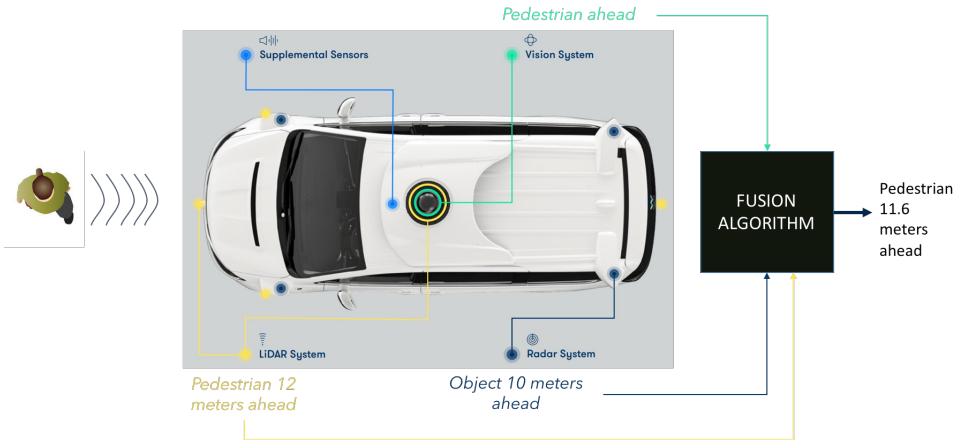

Figure 1: A visualization of the sensor fusion process for obstacle detection [15].

2 Motivation

Autonomous vehicles must perceive the environment around without human intervention (SAE Level 4 and 5) [4]. In order to enable this perception, the vehicles are equipped with different sensors such as Camera, LiDAR, and RADAR. However, each of the sensors exhibits certain strengths or characteristics that the other sensor is incapable of exhibiting. For example, LiDAR gives better accuracy in determining the object position compared to RADAR. However, RADAR gives good accuracy in determining a distance to an object and its velocity. The sensors exhibit complementary characteristics. This complementary behavior can be exploited to develop environment perception algorithms. A very commonly used technique called sensor fusion is used to fuse data from different sensors thereby enabling a better understanding of the surrounding environment of the autonomous vehicles.

The perception of the environment around the vehicle is required because the vehicle needs to know what objects are in its surroundings. These objects include dynamic objects, such as other vehicles, pedestrians, and bicyclists and they include static objects such as traffic signs, or parking spots. Since the vehicle itself is a dynamic and therefore a moving object, the surrounding objects continuously change. This fact requires the vehicle to perform an object detection algorithm regularly to detect any changes in its environment that would require an alteration in the behavior of the autonomous vehicle. To complicate this task, the viability of each sensor to detect an object varies depending on weather conditions and the nature of the perceived object.

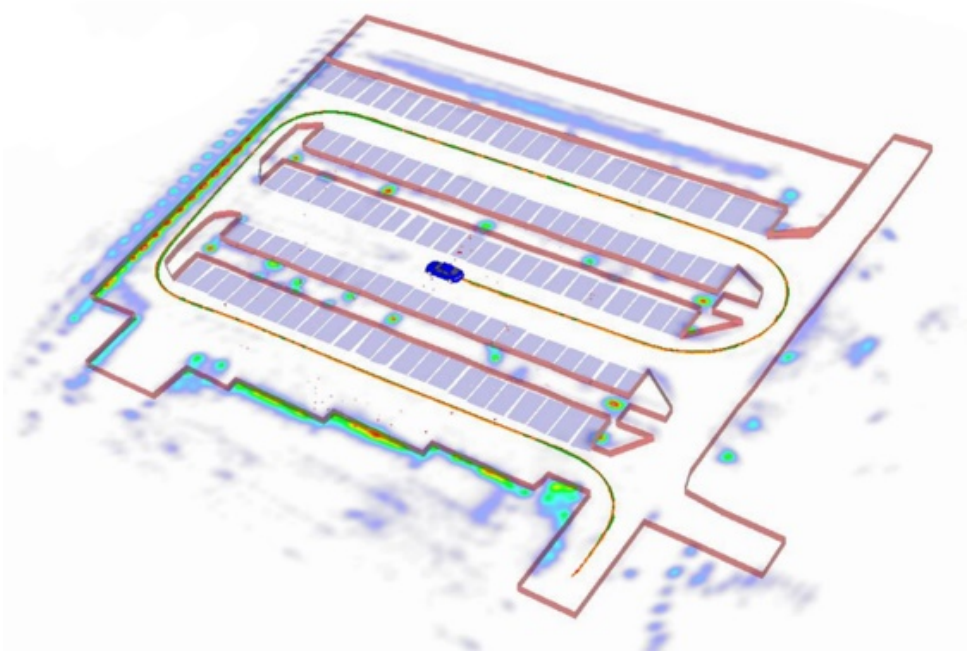

Figure 2: An example of the environment containing parking spaces [1].

2.1 Description of the Case study

The project will focus on implementing the parking pilot feature in autonomous vehicles. Different sub-functions for parking lot recognition and parking lot status assessment shall be implemented. Figure 2 above shows an example of the environment that Hella Aglaia is working around.

The starting point of the project was the overall goal of using various sensors to assist in the automatic detection of unoccupied parking spots as indicated in Figure 2.1. In detecting these unoccupied parking spots, an autonomous vehicle (SAE Level 3 [4]) is able to self-park. Within

our project, data from the camera, LiDAR and RADAR shall be used to identify and extract significant features around the vehicle's environment. Finally, through fusion of data from different sensors, a parking lot will be identified and validated. The entire implementation is done within Hella Aglaia's in-house rapid prototyping framework called Cassandra.

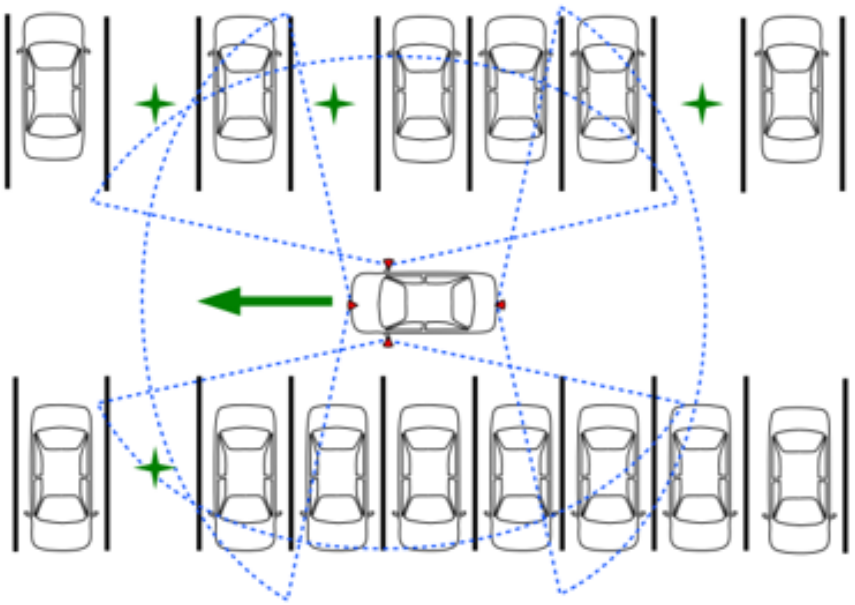

Figure 2.1: A clear, visual representation of the goal and purpose for the project [10].

2.2 Challenges

As a part of the sensor fusion team, our process is to implement a filtering method for the noise and combine the results from each sensor. This allows for certain 'weaknesses' from each sensor type to be minimized and focuses on an ideal prediction. Furthermore, our efforts will allow us to validate the results from the Camera and LiDAR teams as well as possibly validate the localization results. However, an external map was not provided to the team. As such, certain assumptions needed to be made and we decided to give the LiDAR results the most weight and assume a 'comparable' map based on their visualizations.

Other than the data sets from the camera (video feed), LiDAR, and RADAR, the Ego-Motion was provided. Ego-Motion (also called visual odometry) is utilized since, while the other objects are not moving, the vehicle is. This movement changes the model over time, including the orientation of the vehicle. So, it is imperative that a good model that describes the vehicle taking into account the velocity, acceleration and yaw rate needs to be formulated. As can be seen, the model becomes nonlinear and complex when such factors are incorporated. A simplified model such as the constant velocity model or the constant turn rate velocity model cannot be used as the vehicle's velocity and the yaw rate does not remain constant. A comprehensive yet moderately simple enough vehicle model needs to be developed.

In the normal object tracking algorithm, the frame of reference is stationary with the object undergoing movement. However, in our case, the objects remain stationary with the frame of reference subjected to motion. Hence, the normal object tracking algorithm cannot be applied directly. This is further complicated by the fact that as the vehicle moves, the object under consideration can be occluded and shall undergo a change in perspectives. This adds to existing complexity due to the vehicle motion.

Another obvious challenge for the sensor fusion team is to test the algorithms early enough in the project development phase, so that valuable feedback to other sensor teams could be

given. Since all the Sensor teams have to work in tandem with the Sensor Fusion team, each team had to agree what they would provide the Sensor Fusion team with. Also, since all the teams would provide their results quite late with respect to the project timeline, it became necessary for the fusion team to define a standard interface for receiving data from the sensor teams. It was also the duty of the sensor fusion team to drive the rest of the teams as a single coherent unit contributing to a bigger project thereby bringing in a sense of accountability.

3 Project Management

In a highly multi-disciplinary project such as this, it is quite probable that the project would deviate from the actually planned timelines if they are not managed properly. Especially with multiple teams as part of one big project, it becomes all the more challenging for the fusion team as coordination and communication with multiple teams become imperative. An effective project management strategy is required to tackle the different dynamics of the project. Following a presentation from Hella Aglaia, we have broken down the project management into four phases: Initiation, Planning, Implementation, and Finalization [14].

3.1 Project Initialization Phase

The initialization phase has three main sections: the start of the project, a stakeholder analysis, and a project goal description. As the overall goal description is listed throughout the report, we will not add redundant or superfluous information here. Therefore, we will primarily discuss the stakeholder analysis.

3.1.1 Stakeholder Analysis

This project was offered by via the Computer Vision and Remote Sensing group at Technische Universität Berlin in collaboration with Hella Aglaia GmbH. While we are not concerned with the end-user (customer), we do have stakeholders for our project:

- **Technische Universität Berlin; Prof. Dr.-Ing. Olaf Hellwich:** The Computer Vision and Remote Sensing Group is a member of the Institute of Computer Engineering & Microelectronics. This collaboration is given as a master-level project course. As such, our final evaluation is via the professor meaning that our contributions should satisfy certain university criteria.
- **Hella Aglaia:** Sven-Garrit Czarnian, Thomas Jahn, Micha Bruns, and Tino Kutschbach. Hella Aglaia wanted to evaluate our particular approaches to the project, weigh the cost and benefits from hardware (sensors) and software (algorithms, processing). We have considered that it is unlikely we, as a class, would come to an absolutely groundbreaking implementation. Therefore, our goal was to maintain proper communication with their team, provide meaningful results with little resources ("know-how," time, and documentation), and work diligently towards our deadlines.
- **Teams/Classmates:** As this project is a group collective it is dissimilar from traditional courses. We owe it to ourselves and our peers to collaborate effectively and efficiently while maintaining proper communication. This project has many co-dependencies and ultimately we will benefit (or suffer) if we do not recognize the value held with each other.

3.2 Project Planning Phase

Typically, the Sensor Fusion team works as a back-end team since it is dependent on the outputs of other teams. However, it is not sufficient for the Sensor Fusion team to simply wait until the other teams have outputs prior to making any preparation for receiving these outputs (and completing the project in less than two weeks is technologically unfeasible for our team). As such, our initial tasks were to fully understand the algorithms, attempt to model the process in simulation (in which Udacity projects were completed, see Section 7), and work directly with the other teams (throughout the course). Furthermore, it is the responsibility of the Sensor Fusion team to ensure as much interoperability and compatibility between the outputs of the other teams as possible to mitigate errors, computational overheads from additional processing, and consistency between other memory accesses/size. Several aspects of the general project management overhead are irrelevant for our project (i.e., financial planning) and instead, we will focus on the SMART criteria that are appropriate within our planning phase.

3.2.1 Planning of Project Phases (Work Packages)

1. **Literature Review and Research:**
    - Clearly define the project goals
    - Find appropriate and well documented literature for algorithm selection
    - Weight trade-offs between algorithm usages (i.e., use a single filter to fuse all inputs versus filter each individual input and then fuse)
    - We needed to find algorithms that innately use the available variable with as little transformations as possible (*Extended Kalman Filter*; *Global Nearest Neighbors*, see Section 6)
2. **Planning of Project Phases and Milestones:**
    - Estimate the time required to complete (implement) each phase
    - Assign tasks to individuals within the team
    - Negotiate deadlines for other teams (unique to the Sensor Fusion team)
    - Create Milestones (See Section 3.2.1 below)
3. **Management of Related Processes:**
    - Negotiate (when appropriate) with Hella Aglaia for additional resources
    - Re-distribute tasks that are following behind or if a member lacks the ability to complete the task individually
    - Communicate with other teams for specific data types, structures, implementation methods, and architectures (stations)

3.2.2 Milestone Planning

We decided on a timeline so that each team member could work independently of each other. This would enable us to meet the final timeline.

- 22.11 - Workshop at Hella Aglaia GmbH, Technische Universität Berlin
- 29.11 - Sharing of raw RADAR, LiDAR and Camera data by Hella Aglaia GmbH.
- 29.11 - 13.12 - Literature research for object tracking and sensor fusion algorithm.
- 13.12 - Milestone 1 - Architecture presentation and first steps.

- 14.12 - 17.01 - Milestone 2 - Prepare implementation framework and present the first version of requirements to sensor teams.
- 17.01 - 07.02 - Alpha version of the object tracking algorithm within Cassandra. Validation with simulated data,
- 07.02 - 10.03 - Beta version of the object tracking algorithm (also fusion algorithm) implemented as Cassandra stations. Reiteration of the requirements to sensor teams.
- 10.03 - 20.03 - Weekly meeting with sensor teams to understand their data format and other details of the implementation in order to integrate them with the fusion team's architecture.
- 20.04 - 25.03 - Validating the implementation and giving feedback to the sensor teams.
- 25.03 - Final presentation at Hella Aglaia GmbH.
- 26.03 - 31.03 - Final report submission to Hella Aglaia GmbH and Technische Universität Berlin

3.3 Project Implementation Phase

Next, the implementation of the work packages needs to proceed along with allowing for adaptions ('pivoting') whenever deviations occur. This section describes the inter-team collaborations and the Scrum/Agile method of continually evaluating our progress. While our negotiations were not with customers, they were instead with other teams and with Hella Aglaia (requesting a deadline extension; additional resources). The goal is to make a plan and follow it as closely as possible.

3.3.1 External Simulation and Validation

In Section 7, we describe our pre-Cassandra implementations to ensure validation and understanding of common sensor fusion algorithms for autonomous vehicles. Furthermore, simulation data was generated using Matlab and converting the required data into JSON format to work with Cassandra. The goal was to create working stations in parallel with the other teams and simply swap out their data whenever they had completed their processing.

3.3.2 Inter-Team Collaboration

To communicate with other teams, we created a Slack channel, WhatsApp group, email thread, and GitLab account. We also met weekly to ask for updates (progress checks). Initially, the Sensor Fusion team did not know the appropriate inputs to fuse. As such, we needed to communicate clearly if there were changes to the architecture or data structures (i.e., what type of vector/objects were needed). Naturally, each deviation would add additional time so it was critical to communicate as effectively as possible (which often leads to text-based communication being insufficient and was an incentive for personal communication).

3.3.3 Scrum/Agile Development

Fortunately, GitLab has the ability to directly incorporate Scrum-style development within the version control environment. As such, we would evaluate our 'hard' milestones and make progress towards implementation. If a particular member was falling behind (due to bugs or implementation issues), other members would switch tasks (if possible) to help ensure all work packages were completed by the milestone. However, we found in implementation that Agile development can be challenging. Any arbitrary error might take a few hours or a few days

to resolve without any indication of the amount of time required to resolve it. Furthermore, some work packages (issues that had been closed) would need to be reopened if there was an 'edge case' that had not been accounted for (i.e., data ranges outside of a bound) or if the RADAR/LiDAR team had made changes to their implementations.

3.4 Project Finalization

At a certain point, it is time to end the project, analyze the results, and ship the product. The Camera team had internal issues and were unable to provide their results to us in a meaningful manner (i.e., they did not have any Cassandra stations). As a result, we needed to make the decision to move forward with only the RADAR and LiDAR outputs. Even this proved to be a challenge, as the LiDAR and RADAR teams delivered their finalized outputs less than two weeks prior to the presentation (which was after the agreed upon milestone deadlines set by the Sensor Fusion team). Given the limited amount of time, we focused on filtering the LiDAR and RADAR inputs as well as conduct object tracking for each stream.

4 Sensor Characteristics

Since the other teams will be providing detailed descriptions of the individual sensors, their parameters, and characteristics, only the fused characteristics will be discussed in this section. In general, sensor fusion aims to alleviate the weaknesses within some sensors (e.g., cameras do not work in the dark but LiDAR is unaffected) to maximize the benefits from each sensor. Furthermore, fusing sensor readings allows for improved predictions/estimations that would be impacted severely in a stand-alone sensor given noise, missed updates, or other conditions [5].

4.1 Fused Sensor Characteristics

Below, the individual sensor characteristics are displayed. As can be seen in Figure 4.1, each sensor has a range of strengths and weaknesses.

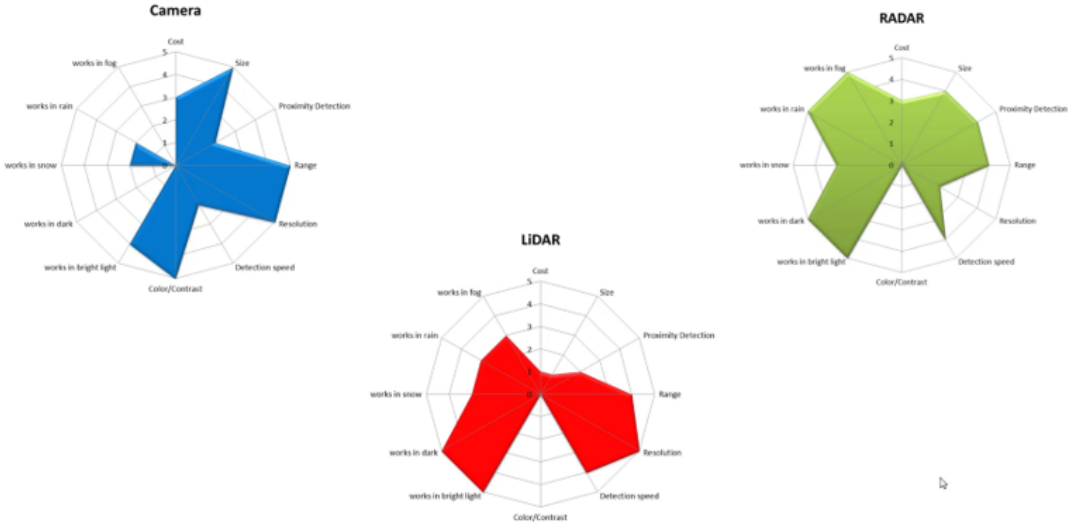

Figure 4.1: Individual Sensor Characteristics.

As defined, combining multiple sensors simultaneously allows for all of the strengths to be combined and many of the weaknesses mitigated (except for cost and size, there is little that can be done to avoid these aspects). Figure 4.1 demonstrates the combined efforts of fusing the camera, LiDAR, and RADAR together [5].

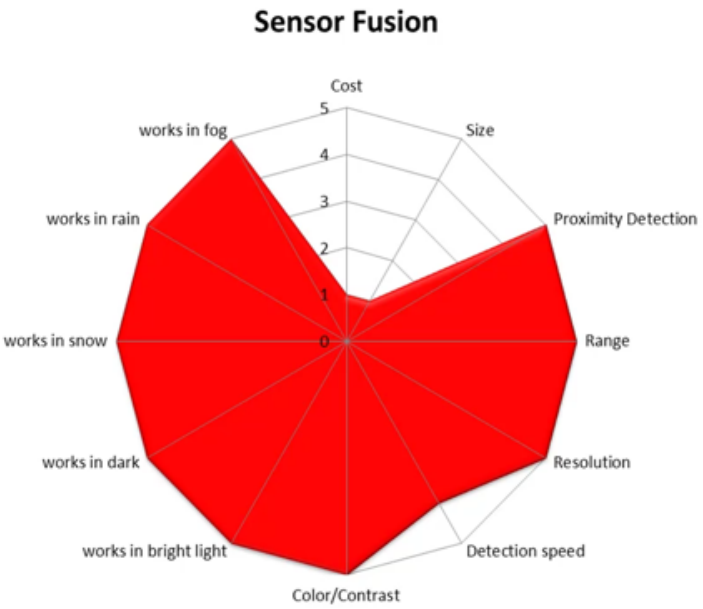

Figure 4.1: Fused Benefits (and Costs) of combining the camera, LiDAR, and RADAR sensors.

However, as the final implementation did not contain contributions from the camera, an updated radar chart was developed demonstrating the contributions from only the LiDAR and RADAR in Figure 4.1.

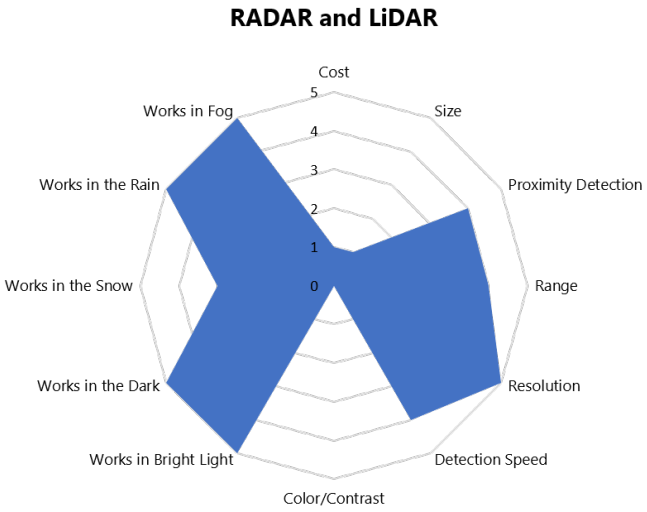

Figure 4.1: Results of only combining the LiDAR and RADAR sensors. As can be seen, a lot of functionality is still provided.

Based on the resulting values of the combined sensors, we posit that it is still entirely possible to detect (un-)occupied parking spaces with an increased difficulty in evaluation and validation (considerably more so without an external map to extrapolate these precise locations).

5 Design

In this section, design options and software architectures (flow-charts and pipelines) are provided.

5.1 Approach

Prior to any implementation, it is often important to approach the project from a top-down (systems engineer) perspective to consider what the final goal and view of the project should look like. In Figure 5.1.1 below, we look at the generally accepted project planning for autonomous vehicles [15]. This allows engineers to consider all of the possibilities and potential inputs and outputs of a system. For this project, we are mainly concerned with the *Sensor* and *Perception* phases of the diagram.

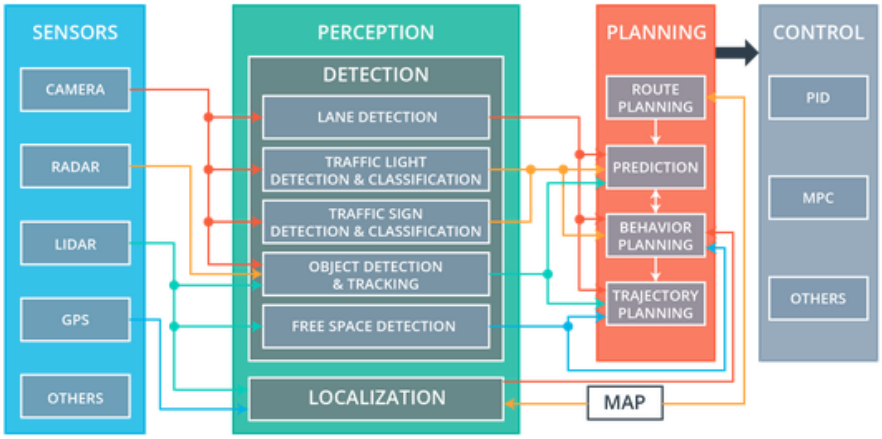

Figure 5.1.1: Planning Pipeline for Self Driving Cars. We are primarily interested in the *Sensor* and *Perception* phases [15].

In general, sensor fusion algorithms follow a typical pattern. Since most filters are based on Bayes Filter, only minor modifications need to be made for various implementations. Below, Figure 5.1.2 shows the general process for conducting sensor fusion based on feeding in sensor data [15].

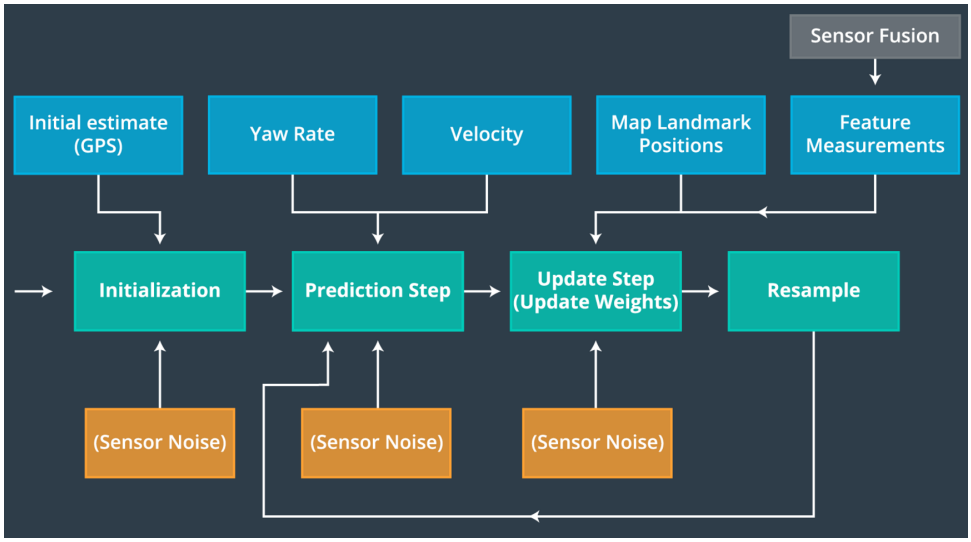

Figure 5.1.2: General Sensor Fusion Algorithm [15].

5.2 Architecture

In the initial development phases of the project, it was crucial to create an overall architecture and plan for implementation. Initially, all three sensor teams were planned to provide their processed data to the Sensor Fusion team. After sensor fusion had been completed, the results (probabilities, bounding boxes) would be validated against an external map as shown below in Figure 5.2.

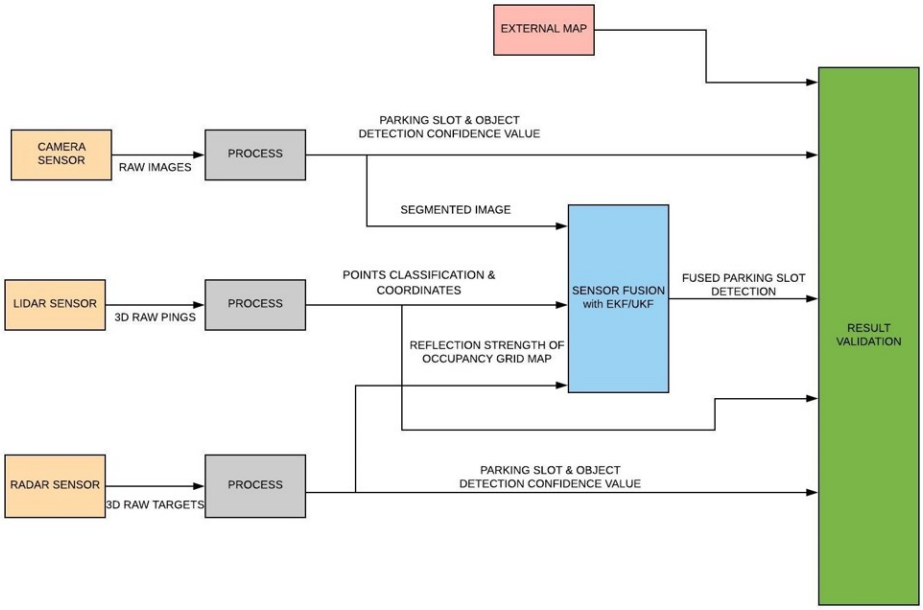

Figure 5.2: Original Architecture Flow-Chart to describe the planning of the project. The sensor fusion team's result would be compared (and validated) against an external map.

6 Related Work

In this section, we will introduce the theory and related work that was used throughout the project. We use the EKF and notation as defined in Thrun et al. [16]. The pseudocode is provided for the *Bayes Filter*, the EKF, and the UKF [17]. The primary challenge in estimation is due to unreliable sensors (noise) and updates (sampling) that do not occur simultaneously. For self-driving vehicles, it is critical to implement estimation and prediction during real-time. Therefore, real-time behavior is tightly coupled with safety as any delay could lead to an accident. In current times, the Extended Kalman Filter (EKF) is utilized for filtering out the noise and for properly implemented sensor fusion. The EKF is, as the name implies, an extension of the Kalman Filter that allows for the linearization surrounding nonlinear transitions and measurement models based on the current state. In practice, the Unscented Kalman filter (UKF) is often considered a better alternative as it is more accurate in estimation. While it might have a comparable runtime, the computation overhead is much higher given the number of data points we are receiving from the various teams and their data sets. Furthermore, the UKF has issues if an update (sampling) is missed when calculating sigma points. As such, it was decided to use the EKF since the behavior (in documentation) is more robust and reliable. The Unscented Kalman Filter (UKF) was not used in the final implementation but is considered for future (and related) works. The goal of each of these filters is the same in implementation, minimizing the *Root-Mean-Square Error* (RMSE) [3].

6.1 Bayes Filter

The *Bayes Filter* is a common approach to utilizing the information from the motion prediction into the state estimation. *Bayes Filter* is the basis for various forms of *Kalman Filters*. Initially, the next state is predicted based on a current state and a controlled input. The belief state is kept and applied to the estimate after which the prediction is updated again (based on an observation). Next, the update feeds the previous estimate together with the observed sensor readings. A new belief is returned and is compared to the model. The notation and structure were initially based on Thrun et al. [16]. The BAYESFILTER function, as well as the other algorithms are given in their default form (for replication). In practice, a number of callbacks might be required due to missed updates, asynchronicity, or other design choices.

Algorithm 1 General Bayes Filter algorithm. For specific filters such as the Kalman Filter or the particle filter, the representation for *bel* and $\bar{bel}$ changes, and the corresponding mathematical updates take specific computational forms.

function PREDICT($bel(x_{t-1}), u_t, \Delta t$)
  $\bar{bel}(x_t) = \int p(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1}$
  return $\bar{bel}(x_t)$
function UPDATE($\bar{bel}(x_t), z_t$)
  $bel(x_t) = \eta p(z_t|x_t) bel(x_t)$
  return $bel(x_t)$
function BAYESFILTER
  $u_t = \text{COMPUTECONTROL}(bel(x_{t-1}))$
  $\bar{bel}(x_t) = \text{PREDICT}(bel(x_{t-1}), u_t, \Delta t)$
  $z_t = \text{READSENSOR}()$
  $bel(x_t) = \text{UPDATE}(\bar{bel}(x_t), z_t)$

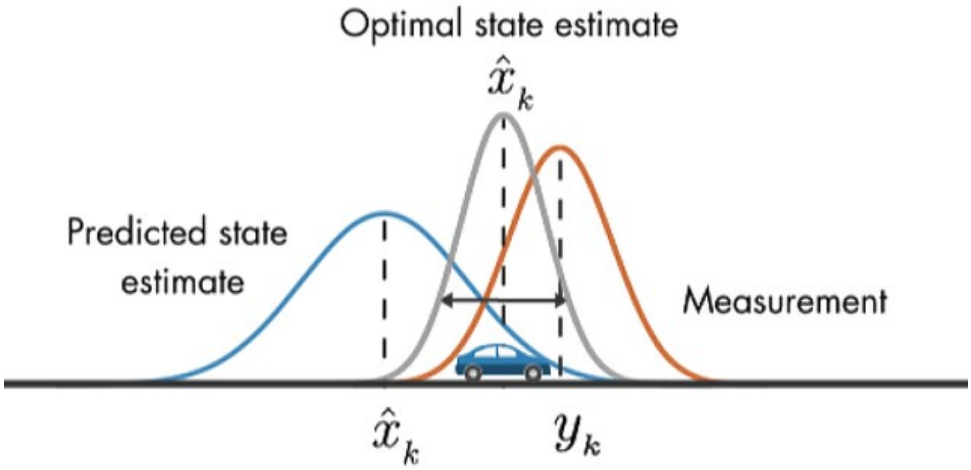

Figure 6.1: A visual example of the benefits from filtering (Kalman, Bayes, etc.) [15] [16].

6.2 Extended Kalman Filter

The *Kalman Filter* (KF) and *Extended Kalman Filter* (EKF) make the assumption that the distributions over belief state are Gaussian, represented as a mean and covariance matrix. Compared to the *Bayes Filter*, the distributions *bel* and $\bar{bel}$ are represented as a mean and covariance matrix. The KF assumes that the transition and observation models are linear, and can be defined by a matrix. The EKF is the extension to the nonlinear case, where we make use of the Jacobian matrix of the transition and observation functions to compute a point-wise linear estimate and then do the same updates as the Kalman Filter. We define the EKF algorithm following Thrun et al. [16]. We altered it to include separate PREDICT and UPDATE methods and to use our notation. The transition or prediction covariance is Q_t; the measurement covariance is R_t. These matrices are often taken to be constant, but also sometimes people change them over time depending on the sensor model [9].

For the KF, the matrix G_t is constant every iteration of the function and does not need to be recomputed each time (except for Δt). Essentially, the implementation of g' ignores its state input. Similarly, for the KF, the matrix H_t is constant every iteration of the function and does not need to be recomputed. Another way to say it is that the function h' ignores its state input. For the EKF, these matrices change each iteration, because it linearizes around the current state [8].

Algorithm 2 EKF algorithm.

function PREDICT($\mu_{t-1}, \Sigma_{t-1}, u_t, \Delta t$)
  $\bar{\mu}_t = g(u_t, \mu_{t-1})$
  $G_t = g'(u_t, x_t, \Delta t)$
  $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + Q_t$
  return $\bar{\mu}_t, \bar{\Sigma}_t$
function UPDATE($\bar{\mu}_t, \bar{\Sigma}_t, z_t$)
  $H_t = h'(\bar{\mu}_t)$
  $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + R_t)^{-1}$
  $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
  $\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$
  return μ_t, Σ_t
function EXTENDEDKALMANFILTER
  $u_t =$ COMPUTECONTROL(μ_{t-1}, Σ_{t-1})
  $\bar{\mu}_t, \bar{\Sigma}_t =$ PREDICT($\mu_{t-1}, \Sigma_{t-1}, u_t, \Delta t$)
  $z_t =$ READSENSOR()
  $\mu_t, \Sigma_t =$ UPDATE($\bar{\mu}_t, \bar{\Sigma}_t, z_t$)

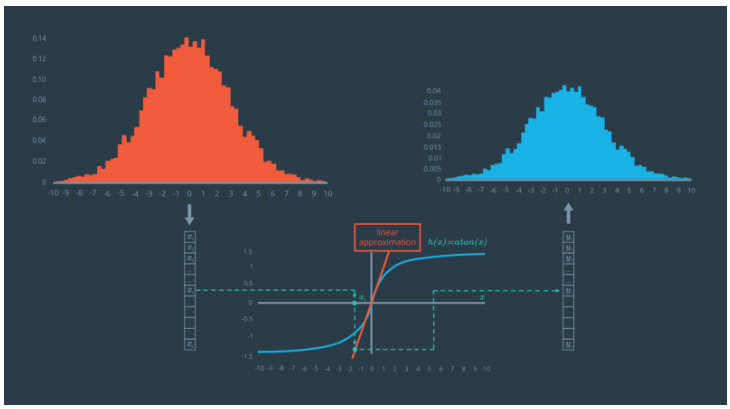

Figure 6.2: Using linear approximation (via Taylor series approximation and Jacobian partial derivative matrix), a Gaussian distribution can be returned with relatively high accuracy [15] [16].

6.3 Unscented Kalman Filter

For comparison, the *Unscented Kalman Filter* (UKF) is also defined. For stochastic, non-linear systems, the UKF is able to provide a good estimate of the state without the usage of Jacobians. This makes the implementation somewhat easier to realize. The UKF [17] is similar to the EKF in that it handles nonlinear transition models g and measurement models h. However, instead of linearizing around the current estimate, the UKF selects sigma points (sample points) depending on the current state estimate and covariance. Sigma points are sent through the nonlinear transition (or observation function). The resulting sample means and covariances from the sigma points are used to construct a new Gaussian μ and σ. Sigma points are computed using the matrix S which is defined from the covariance matrix, Σ_t. S_i denotes the ith column of the matrix S. Initially, it was planned to use the UKF for the RADAR sensors but it was later chosen to only rely on the EKF for consistency. However, in future work, it would be ideal to either incorporate the UKF for a 'final fusion' or at least for benchmarking (as autonomous vehicles are essentially large embedded systems).

$$S = \sqrt{\Sigma_t} \tag{1}$$

The defined sigma points $X_{i,t} \in X_t$ are as follows:

$$X_{i,t} = \begin{cases} = \mu_t, & i = 0 \\ = \mu_t + \gamma S_i, & i = 1, \ldots, N \\ = \mu_t - \gamma S_{i-N}, & i = N+1, \ldots, 2N \end{cases} \tag{2}$$

The sigma points are defined using the mean and covariance matrix of the distributions. Selecting several representative points S_i away from the mean, it is possible to use a relatively small set of points to represent the entire distribution via approximation and weights. The weights, w_i^m are given as:

$$w_i^m = \begin{cases} = \frac{\lambda}{N+\lambda} & i = 0 \\ = \frac{1}{2(N+\lambda)}, & i = 1, \ldots, 2N \end{cases} \tag{3}$$

The weights when computing the sample covariance, w_i^c are:

$$w_i^c = \begin{cases} = \frac{\lambda}{N+\lambda} + (1 - \alpha^2 + \beta) & i = 0 \\ = \frac{1}{2(N+\lambda)}, & i = 1, \ldots, 2N \end{cases} \tag{4}$$

The parameters are defined as:

$$\gamma = \sqrt{N + \lambda} \tag{5}$$

$$\lambda = \alpha^2(N + \kappa) - N \tag{6}$$

See Kandepu et al. [12] for tuning suggestions when implementing the filter.

Algorithm 3 Unscented Kalman Filter.

function COMPUTESIGMAS(μ_t, Σ_t)
  return $X_{0,t}, \ldots, X_{2N,t}$
function PREDICT($\mu_{t-1}, \Sigma_{t-1}, u_t, \Delta t$)
  $X_{t-1} = \text{COMPUTESIGMAS}(\mu_{t-1}, \Sigma_{t-1})$
  $\forall_{i=0}^{2N} \bar{X}_{i,t} = g(X_{i,t-1}, u_t, \Delta t)$
  return $\bar{X}_t$
function UPDATE($\bar{X}_t, z_t$)
  $\bar{\mu}_t = \sum_{i=0}^{2N}(w_i^m \bar{X}_{i,t})$
  $\bar{\Sigma}_t = \sum_{i=0}^{2N} w_i^c (X_{i,t} - \bar{\mu}_t)(X_{i,t} - \bar{\mu}_t)^T + Q_t$
  $\forall_{i=1}^{2N} Z_{i,t} = h(\bar{X}_{i,t})$
  $\mu^z = \Sigma_{i=0}^{2N} w_i^m Z_{i,t}$
  $\Sigma_t^z = \Sigma_{i=0}^{2N} w_i^c (Z_{i,t} - \mu^z)(Z_{i,t} - \mu^z)^T + R_t$
  $\Sigma_t^{xz} = \Sigma_0^{2N} w_i^c (\bar{X}_{i,t} - \bar{\mu}_t)(Z_{i,t} - \mu^z)^T$
  $K_t = \Sigma_t^{xz}(\Sigma_t^z)^{-1}$
  $\mu_t = \bar{\mu}_t + K_t(z_t - \mu^z)$
  $\Sigma_t = \bar{\Sigma}_t - K_t \Sigma_t^z K_t^T$
  return μ_t, Σ_t
function UNSCENTEDKALMANFILTER
  $u_t = \text{COMPUTECONTROL}(\mu_{t-1}, \Sigma_{t-1})$
  $\bar{X}_t = \text{PREDICT}(\mu_{t-1}, \Sigma_{t-1}, u_t, \Delta t)$
  $z_t = \text{READSENSOR}()$
  $\mu_t, \Sigma_t = \text{UPDATE}(\bar{X}_t, z_t)$

6.4 (Global) Nearest Neighbors

Nearest Neighbors (NN) is an algorithm that is useful for data association. An additional flavor, *Global Nearest Neighbors* (GNN) is useful for *Multi-Object Tracking* (MOT). Since we have the case of multiple objects (clusters) being provided from the other teams that change over time, it is crucial to keep track of objects in recent memory. The task is to minimize the total distance function for the sum of all distances related to individual assignments (objects). GNN works by taking in the current data, clustering the results, measuring the association, and followed by filtering the track using an EKF [6] [13]. Figure 6.4 shows an example.

In our case, we choose to minimize the Euclidean Distance based on the work by Collins [7]. The goal is to match the features along the track (where the clusters are defined as gates) and the optimal observation value is selected to be incorporated into the track. We also added a "freshness" value that allows objects that have not been detected after a predefined number of cycles (4 cycles in our case) to drop out of memory (i.e., be forgotten). This was needed to prevent memory overflow which causes issues within the Cassandra Framework.

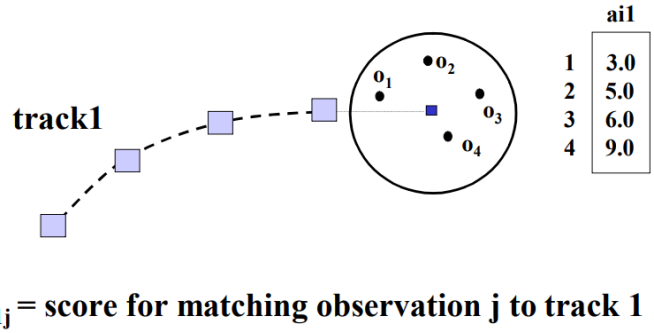

Figure 6.4: Example of Global Nearest Neighbors tracking with observation scores (higher is better). [7]

7 Testing - Pre-Cassandra / Other Team Inputs

In this section, the details of the preparatory work are explained. This is meant to give practical experience with common sensor fusion algorithms and practice coding them in C++ (the core language of Cassandra).

7.1 Udacity Background

As the Sensor Fusion team is dependent on the outputs of the other teams, it was critical that we were prepared to receive their contributions. This requires a full understanding of the implementation of various sensor fusion algorithms, understanding how the data works, and anticipating further work to obtain a reasonable result. To gain a deeper understanding and practice of sensor fusion, we cloned projects from Udacity's Self-Driving Car Nanodegree program. The goal of these projects is to filter out the noise of a vehicle in motion as it moves around a track in a simulator. As a result, we were able to gain implementation experience with the EKF and UKF [15].

This was highly beneficial for our understanding and practice. However, the case of Hella Aglaia is very different as we are not focusing on the model of the car, but rather how the model of the external environment is changing. Furthermore, it is was not possible to directly integrate the algorithms as Cassandra requires particular formats and data structures. Regardless, this was an excellent opportunity to proof out the algorithms and ensure they work in some environment (as future errors would then, hopefully, only be related to implementations in Cassandra and not related to the formulae). The project also allows for GPS/odometry sensor data. Kalman filters do not change their behavior based on the type of input. However, since we would not have access to GPS in our project, we only implemented RADAR and LiDAR data. Lastly, a member from the Hella Aglaia team said these efforts would benefit our overall understanding would be beneficial to the sensor fusion team.

7.2 Mercedes Data Set

For the same Udacity projects, a Matlab script was given to automatically generate sample points (randomly) in order to test and tune the EKF. This script and framework are provided by Mercedes. This allows for visualizations to be generated with the project to analyze the behavior of added noise (random or fixed) [2]. Originally, it was planned to take the text files and convert them into JSON format to work with the stations in Cassandra. This would have allowed for creating simulated data entirely independent from the other teams (since we are dependent on their throughput). However, the input station (or player station as per Cassandra station phrasing) was not able to innately take in the data we had created, even though it was in the proper format. Upon consulting a Hella Aglaia employee, we found that the station suitable for our needs was an in house station and would not be provided to us (probably due to intellectual property or other logistics).

7.3 Results

As these results are not critical to the case study for Hella Aglaia, they will only be briefly shown below. Figure 7.3.1 shows the initial fusion of the LiDAR and RADAR as compared to a ground truth with the state predictions. As can be seen, the state predictions deviate quite drastically from the ground truth. Figure 7.3.2 shows an improved implementation with more accurate covariances (adjustable).

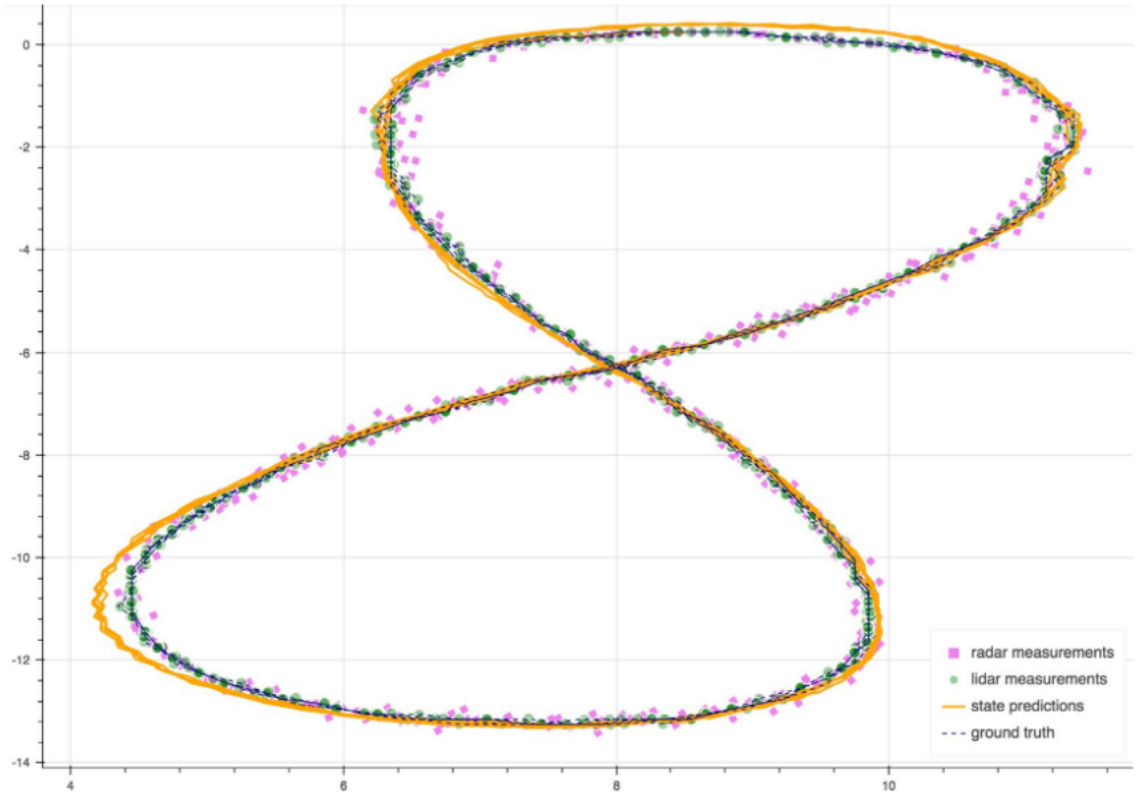

Figure 7.3.1: Visualization of predictions using covariances. The deviation of the state predictions is quite noticeable form the ground truth.

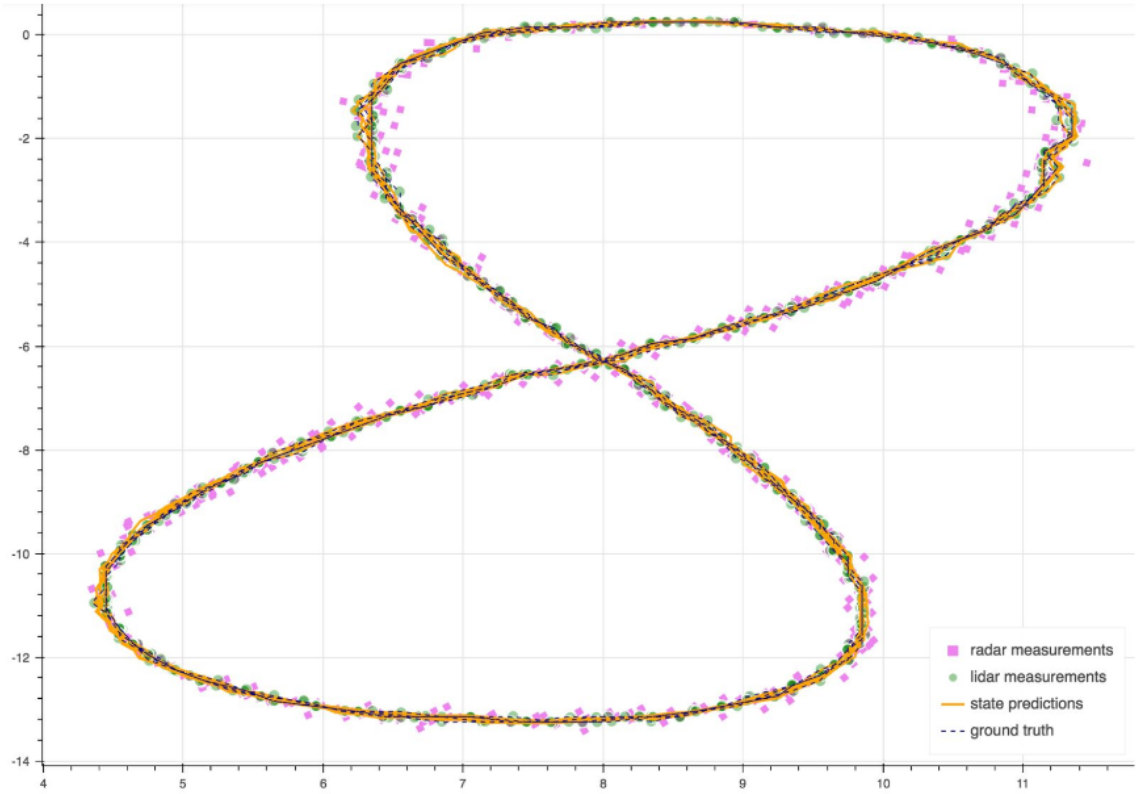

Figure 7.3.2: Visualization of predictions using improved covariances. The tracking of the state predictions has drastically improved.

8 Vehicle Model

As was already alluded to previously, the project is not only concerned with the model of the changing vehicle. Here, the goal is to observe the changing environment and identify unoccupied parking slots. As such, a different model is needed in order to handle the discrepancies between only filtering the noise from the movement of the vehicle and analyzing the input data from the other teams. Recall that in the final implementation, only the LiDAR and RADAR information was available for fusion and analysis.

We define a model for the autonomous vehicle based on the Hella Aglaia case. The vehicle traverses forward and backward in the x direction and left and right in y. The state transitions (x is for the state; z is for the measurement; a and v are the acceleration and velocity) function as follows:

$$x_k = [x, y] \tag{7}$$

$$z_k = [x, y] \tag{8}$$

We define the control input u_k as:

$$u_k = [a_x \; a_y \; v_x \; v_y \; v_{yaw}] \tag{9}$$

Next, the derivations of the formulae are provided (note that w and ν are the process and observation noise respectively):

$$x_k = f(x_{k-1}, u) + \omega_n \tag{10}$$

$$z_k = h(x_k) + \nu_z \tag{11}$$

$$f(x) = x' = x \cos v_{yaw} t - y \sin v_{yaw} t - v_x t - \frac{1}{2} a_x t^2 \tag{12}$$

$$f(y) = y' = -x \sin v_{yaw} t + y \cos \nu_{yaw} t - v_y t - \frac{1}{2} a_y t^2 \tag{13}$$

$$h(x, y) : x' = x, \; y' = y \tag{14}$$

8.1 Transition Model

Then we calculate the Jacobian matrix for the EKF:

$$F = \left. \frac{\partial f}{\partial x} \right|_{x_k} = \begin{bmatrix} \cos \nu_{yaw} t & \sin \nu_{yaw} t \\ -\sin \nu_{yaw} t & \cos \nu_{yaw} t \end{bmatrix} \tag{15}$$

8.2 Prediction

$$x_k^* = f(x_{k+1}, u_{k+1}) \tag{16}$$

P is the posteriori error covariance matrix; F is the state-transition model; Q is the covariance of the process noise; process noise is assumed static:

$$P_k^* = FP_{k+1}F^T + W_kQ_{k-1}W^T \tag{17}$$

$$Q = q_0 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \; ; \; W = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{18}$$

8.3 Update

H is the observation model; R is the covariance of the observation noise:

$$K_k^* = P_k^* H^T (HP_kH^T + V_kR_kV_k^T)^{-1} \tag{19}$$

$$H = \frac{\partial h}{\partial x}\bigg|_{x_k} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{20}$$

8.4 Measurement Noise Covariance

$$R_k = R_0 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} ; R_0 >> 1 \tag{21}$$

Assume Measurement Noise is static:

$$V = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{22}$$

8.5 Data Specifications

The sensor fusion team received various data from the LiDAR and RADAR teams. During the planning of the Cassandra stations, it was critical to work directly with these teams to ensure the compatibility and interoperability of the data for proper fusion. Based on the setup for the vehicle model, we specifically asked for the coordinates of the bounding boxes (x_{min}, x_{max}, y_{min}, y_{max} which gives the lower left and upper right coordinates of the bounding box), its covariances (Cov_{xx}, Cov_{xy}, Cov_{yy}, Cov_{yx}), and the time-steps (Δ_t, for proper synchronization). The means (μ_x, μ_y) were also requested (and provided). Initially, we planned to use the centroid of the bounding box (for data reduction) but the results were not ideal. Therefore, we focused on implementing with the bounding box coordinates.

```
class SensorObject {
public:
    float32 meanX;
    float32 meanY;
    float32 Cov_xx;
    float32 Cov_xy;
    float32 Cov_yy;
    float32 Cov_yx;
    float32 maxX;
    float32 maxY;
    float32 minX;
    float32 minY;
    int label;
    sensortype sensor;
```

Figure 8.5: Screenshot of the Class/Objects utilized by the Sensor Fusion team (based on the outputs of the LiDAR and RADAR teams).

8.6 Object Tracking

Below, two images (Figures 8.6.1 and 8.6.2) are shown as provided from the LiDAR and RADAR teams. As can be seen, even visually, it is difficult to make much sense of the data. LiDAR provides a much higher level of clarity and differentiation for discrete objects. Due to the nature of the RADAR (and noisy data), RADAR is unable to provide the same level of detail. As such, the clustering work from the other teams is initially incompatible. For example, a single cluster from the RADAR team (given as a single object) might be composed of several objects (including different combinations of vehicles, infrastructures, and other obstacles). Therefore, it became the responsibility of the Sensor Fusion team to try to find a region of interest that was similar between both inputs.

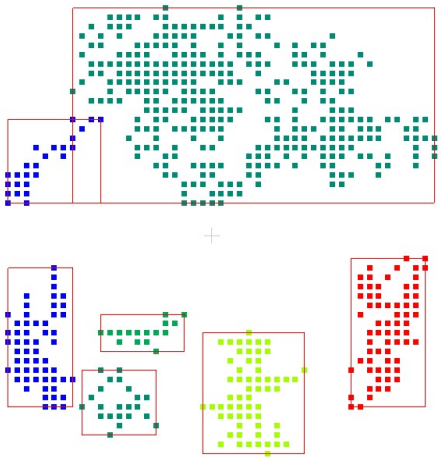

Figure 8.6.1: Example of the bounding boxes and clusters provided from the RADAR team.

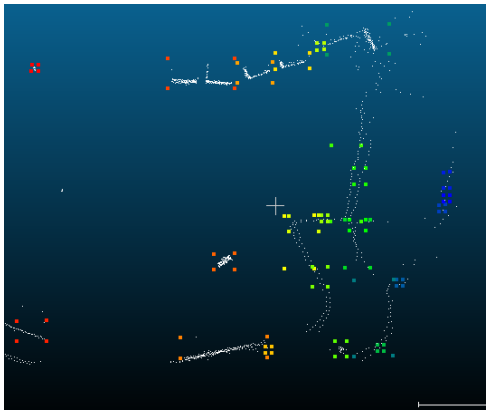

Figure 8.6.2: Example of the bounding boxes and clusters provided from the LiDAR team.

To resolve these discrepancies, it was required to break each cluster into meaningful objects. Then, the objects are given unique object identification numbers (Object ID). In this manner, objects would be able to be maintained in memory and compared across the initial input data. This was completed by using a *Multiple-Object Tracking* data association algorithm based on *Global Nearest Neighbors*.

8.7 Final Architecture

Below, the final architecture is updated based on the data that the sensor fusion team received. After each team performs its coordination transformation, the Fusion Team applies the EKF and GNN algorithms (on each input, independently), conducts tracking (Object ID assignment), and proceeds to fuse the results (Figure 8.7.1).

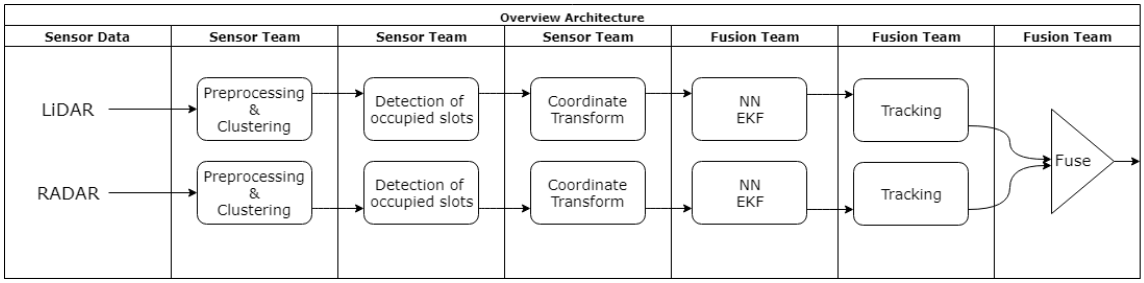

Figure 8.7.1: Final Pipeline for Filtering, Tracking, and Sensor Fusion.

Below, Figure 8.7.2 shows the state diagram of how tracking was achieved using measurements from individual sensors. NN is used for data association between the sensor measurements and the tracked objects.

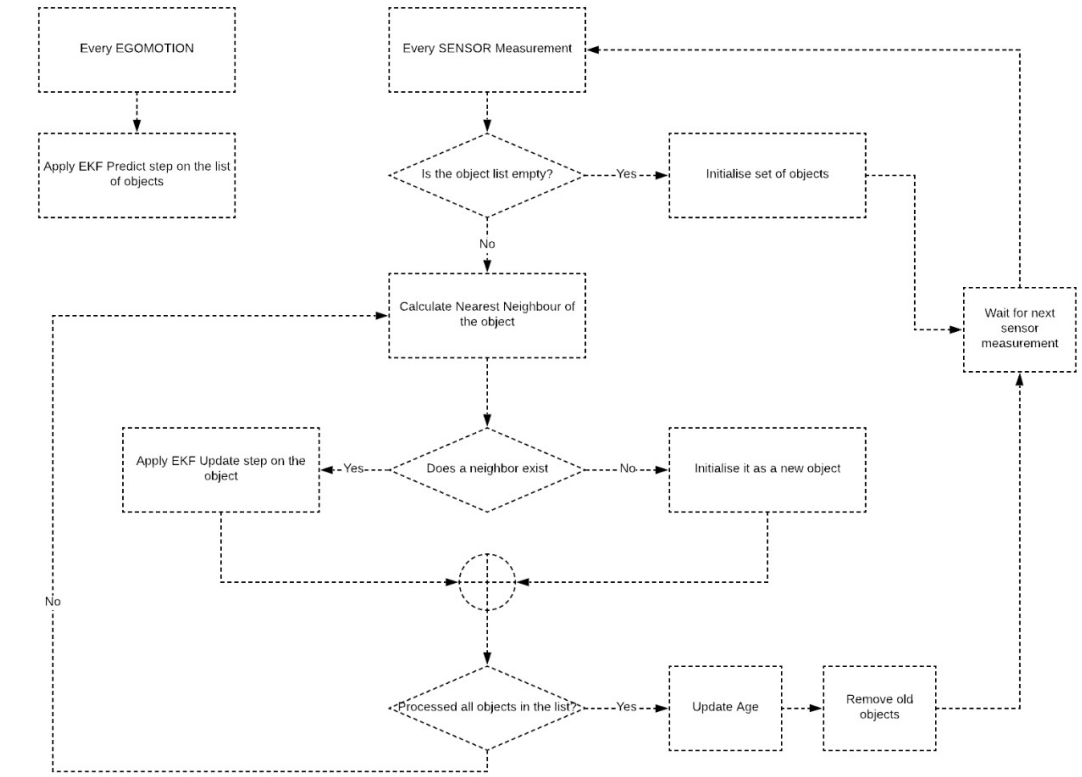

Figure 8.7.2: State Diagram Object Tracking for individual sensors.

In Figure 8.7.3 below, the flow diagram of how sensor fusion between LiDAR and RADAR data is done is depicted. NN is applied across the tracked objects from the two sensors to evaluate whether an object can be fused or not.

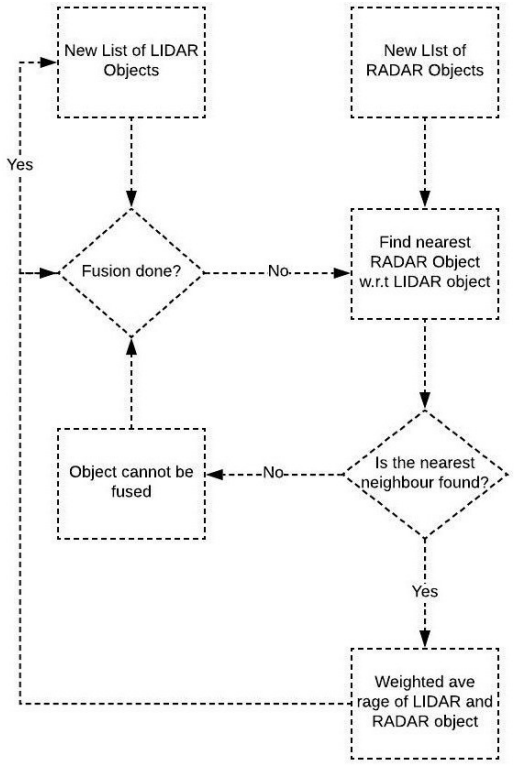

Figure 8.7.3: Fusion of LiDAR and RADAR tracks

8.8 EKF Station

The EKF station is the main station in the fusion pipeline. When we receive any input from the sensor, it has to be converted into a correct format. This is done by the converter station. This data is then given to the EKF station. The sensor input is used as a measurement to the update function of the *Kalman Filter*, and *Ego-Motion* values are used to provide a prediction. The process occurs in the following way:

1. A global vector for detected objects; *ekfObjectMap*, is initialized in the beginning. This vector holds all the objects detected. If it is zero, a new set of objects is created.

2. The data received from the sensors can be inaccurate and invalid, hence it needs to be filtered out. All the bounding boxes which are less than specific width (for example 0.5 meters) are assumed to be invalid and thus filtered out.

3. The bounding box is checked with the existing object list in order to find if it has a matching nearest neighbor. For this purpose, distances are computed for each bounding box. In order to ensure that an object is not matched with its neighboring object, the maximum distance threshold is determined. This is calibrated by observing the complete data from the LiDAR and RADAR and deriving a general pattern out of it to come up with the average distance threshold. We have calculated the average distance threshold as around half the value of the object size detected by each sensor (with the observation that a valid object will be present in the sensor range for more than one cycle). The values that we have used in the code is 1.8m for LiDAR and 4m for RADAR.

4. If any matching object is detected, it is updated via the EKF update function using the value received. Otherwise, a new object is created. The EKF saves each object as (x_{min}, y_{min}, x_{max}, y_{max}); the bounding box coordinates and tracks them separately.

5. When all the newly detected objects are processed, their ages are updated. In our code, the age of an object is 5 when it is recently detected, and it decreases by 1 per cycle. Thus if the object is detected again, its age is set to 5.

6. After this, the ages of all the objects is reduced by 1. Finally, if any object has age ≤ 0, it is removed from the system.

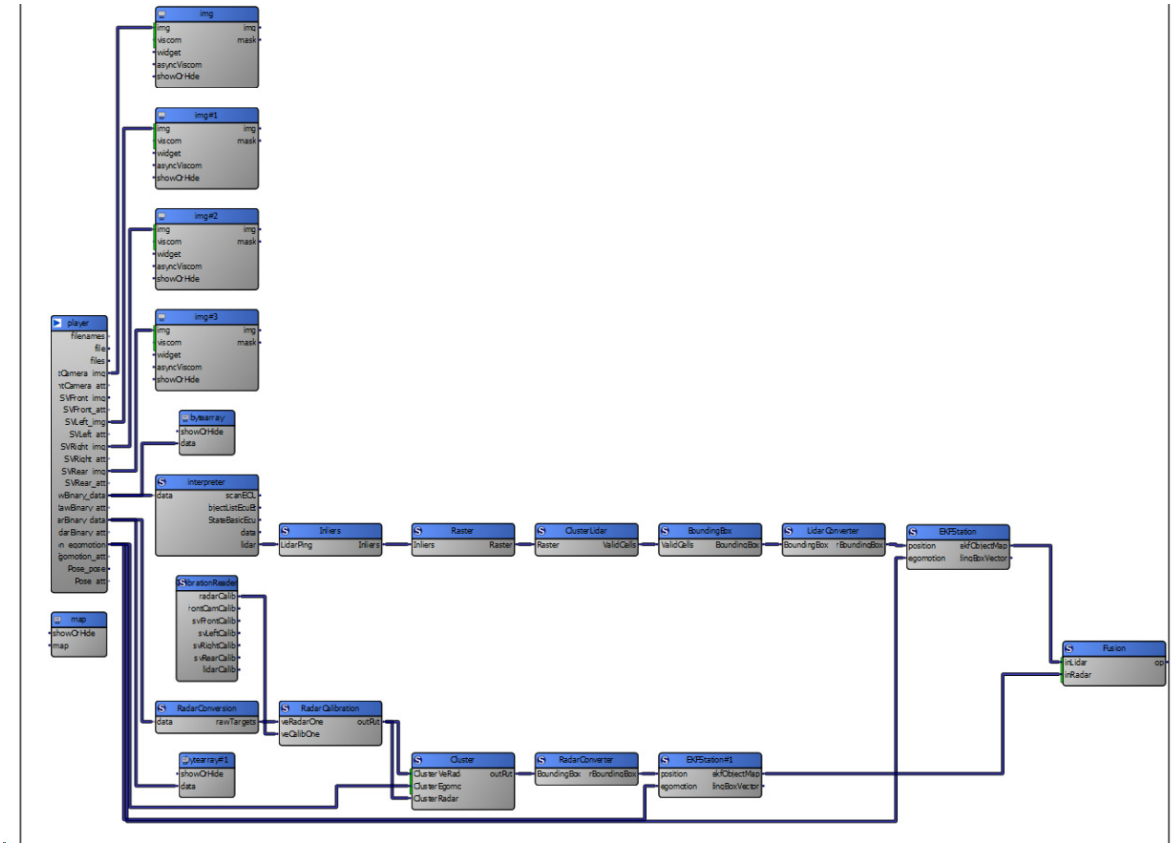

Figure 8.8: Full Station Design.

8.9 Results

In this section, we describe the results from the sensor fusion and tracking for the LiDAR and RADAR. As can be seen from the figures below, the tracking of the coordinates over time is relatively accurate (given the sensors are independent). In the images, a single object (given ID of 0 in the form 'Object[0]') is tracked and plotted against time. The values of x_{min}, x_{max}, y_{min}, and y_{max} are shown for each image. Each value is shown with the predicted, updated, and measured (sensor input) values. In cases in which the plotted data is not readily distinguishable, this is due to a perfect overlap of data (i.e., the measurement values and updated values are almost same until two decimal places due to the lower measurement covariance values obtained from the sensor teams). The images below show an example for Object[0] but these results are also valid for additional objects throughout the process.

8.9.1 LiDAR Results

Below are the LiDAR results:

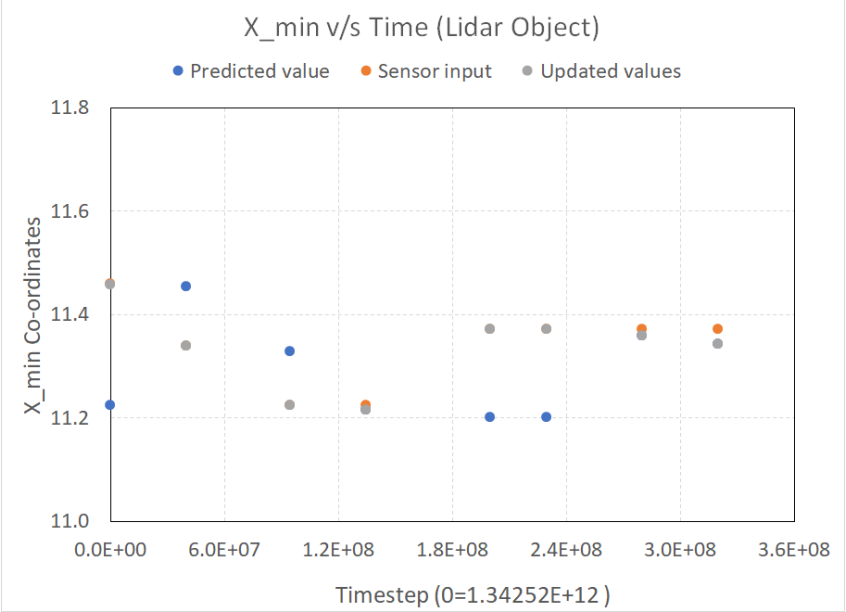

Figure 8.9.1.1: LiDAR - x_{min} vs time for Object[0].

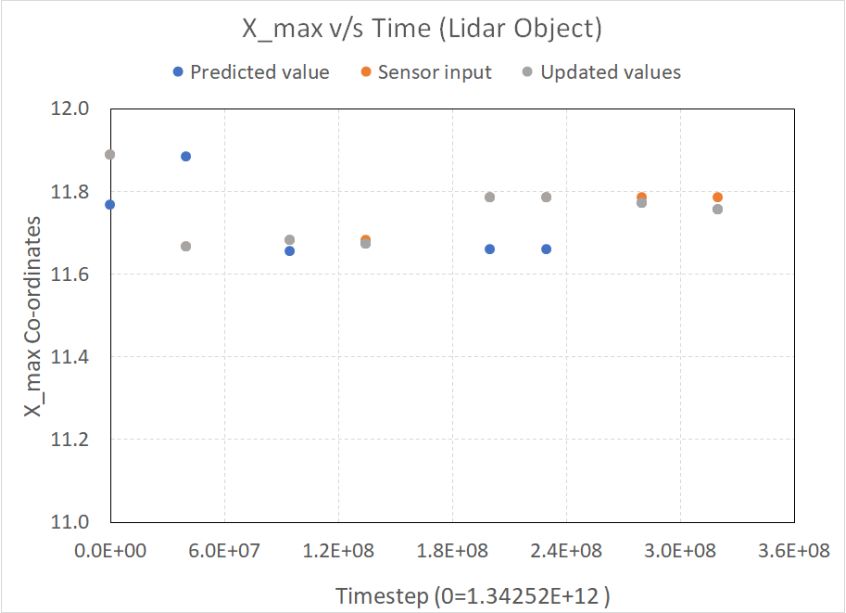

Figure 8.9.1.2: LiDAR - x_{max} vs time for Object[0].

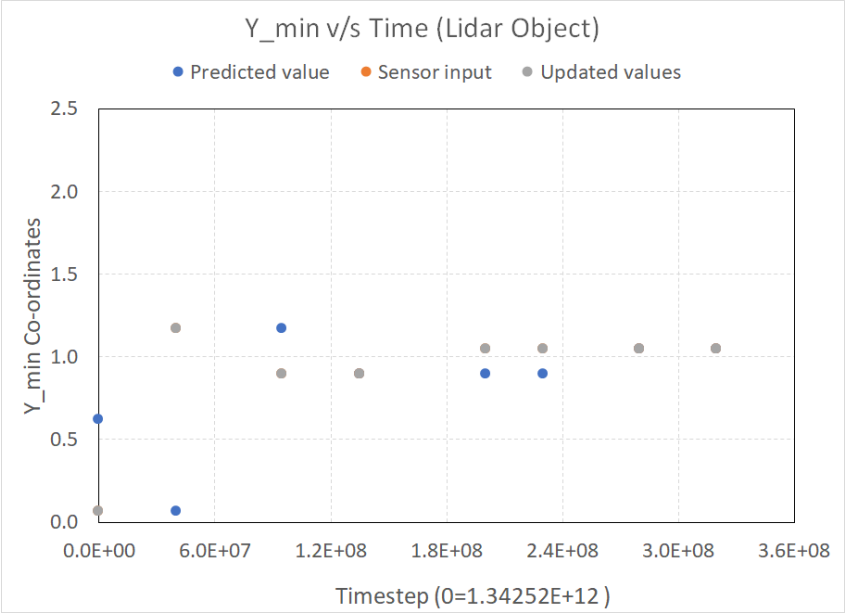

Figure 8.9.1.3: LiDAR - y_{min} vs time for Object[0].

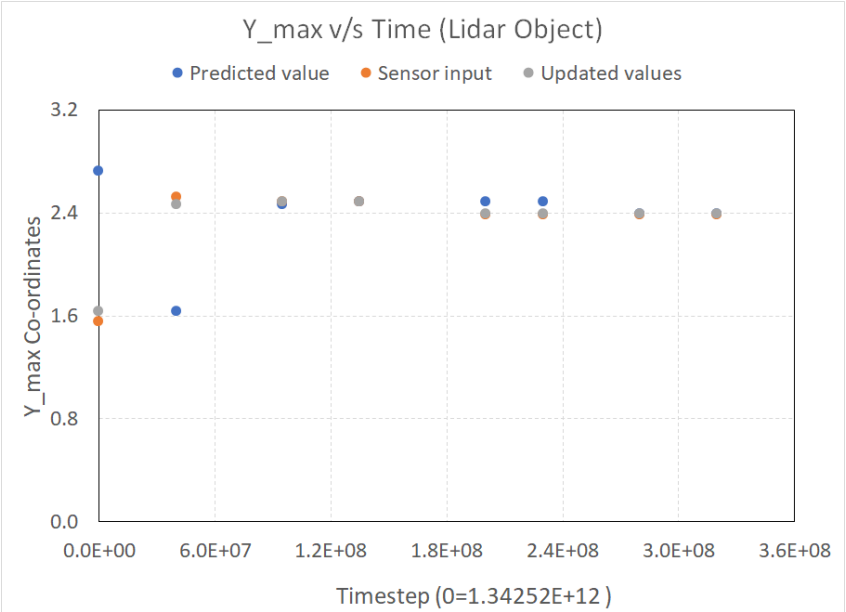

Figure 8.9.1.4: LiDAR - y_{max} vs time for Object[0].

8.9.2 RADAR Results

Below are the RADAR results:

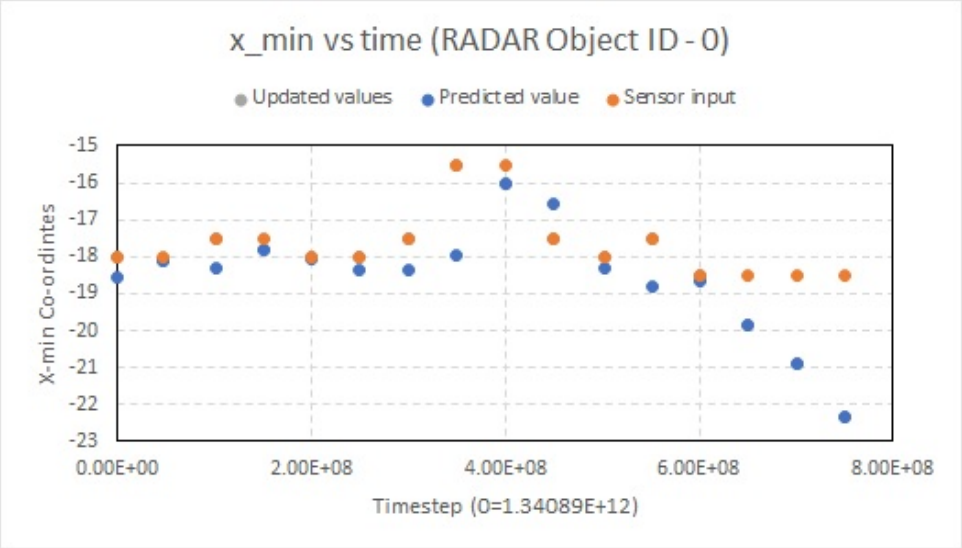

Figure 8.9.2.1: RADAR - x_{min} vs time for Object[0].

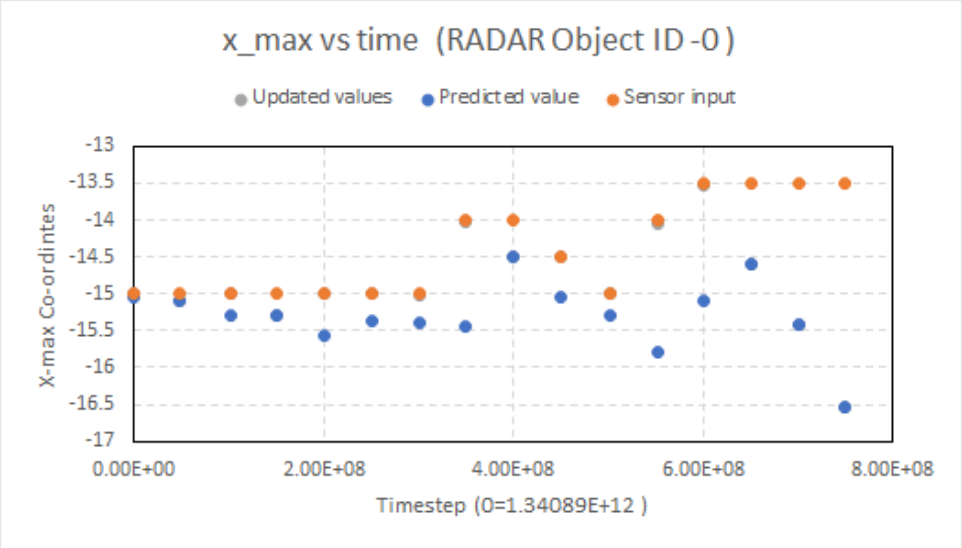

Figure 8.9.2.2: RADAR - x_{max} vs time for Object[0].

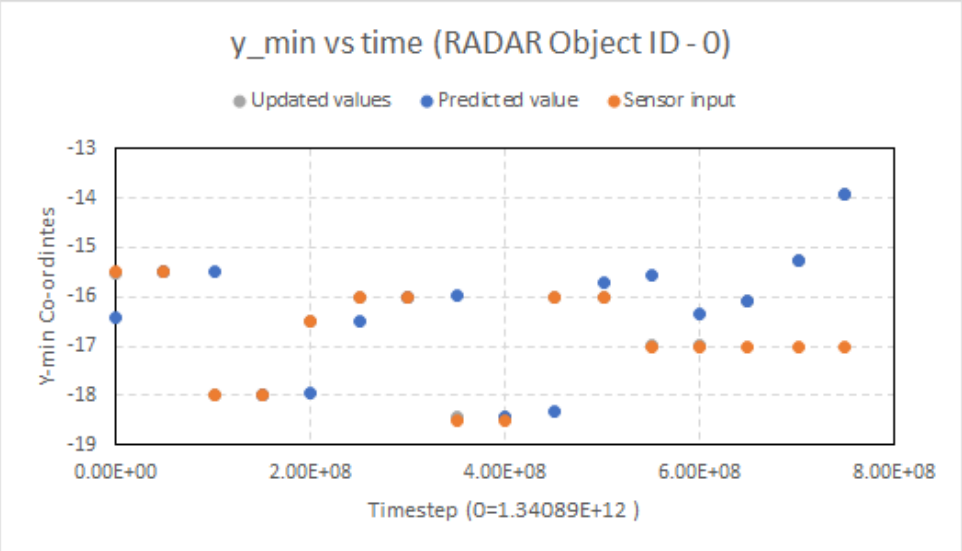

Figure 8.9.2.3: RADAR - y_{min} vs time for Object[0].

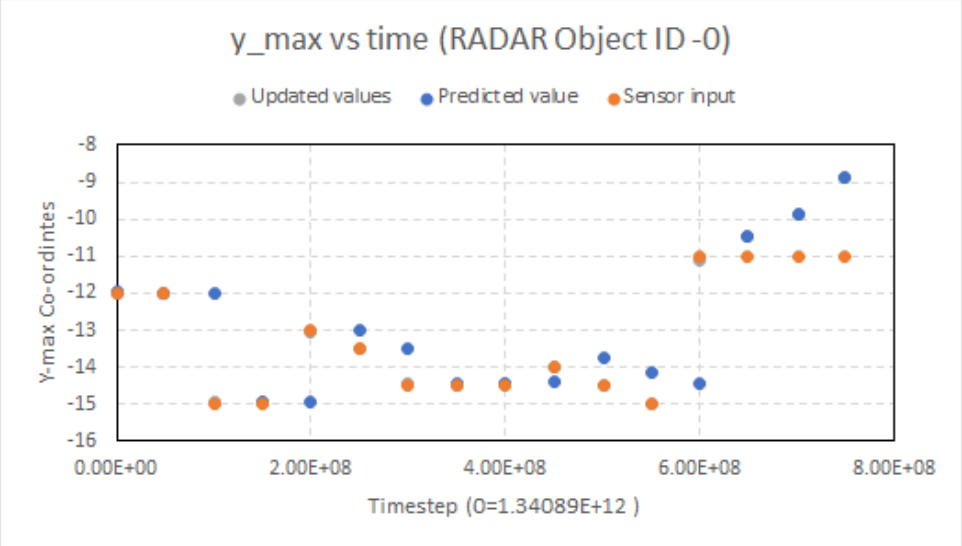

Figure 8.9.2.4: RADAR - y_{max} vs time for Object[0].

9 Conclusions

In conclusion, the project was an excellent learning opportunity. There were many challenges faced for each of the individual teams as well as the entirety of the class. As our duty to communicate with other teams, we created a Slack channel (for communication), a Google Drive, a GitLab Repository, a WhatsApp group, and a typical email thread. These tools allowed us to properly communicate with the other teams. The LiDAR and RADAR teams were able to meet with us in person nearly every Friday. This was very useful when the needed data types and data structures would change based on a lack of understanding or issue with implementation. We are very grateful for their patience as we changed some of the requirements for the final implementations during the project. Together, we ensured that the different teams know the data types, structures, and ordering (in an array) so that we would be able to have compatible data for our stations.

For a ”typical” academic project, the results seem incomplete as the final implementation did not contain the material from the camera team. The fact that we received the final data from the LiDAR and RADAR teams less than two weeks before the final presentation deadline was also a major factor in the final projects' result. However, building our communication skills and ability to adapt under pressure is arguably more important that have visual representations of parking spots. When working in industry, there are hard deadlines set for projects and minimal resources available (including support and documentation). This project gave us direct exposure to these types of situations and there were many times in which we needed to move forward with what we currently had available or find a different solution to our issues. In short, we are very satisfied with our end results and the opportunity to engage in an industry environment solving real challenges, especially for autonomous vehicles.

We have found that the EKF is a robust filtering algorithm that is highly appropriate for autonomous vehicles. Synchronization of data (time-steps) and missed measurements are real challenges that do not have trivial solutions. As such, we can see the real motivation for implementing these projects in Cassandra. Finally, we learned that the scientific community does not have complete consensus on the implementation of algorithms (i.e., EKF versus UKF, using GNN for data association), and as such, we learned that even scholarly articles and results need additional testing prior to a final implementation (i.e., it is not prudent to simply take a well-cited algorithm an attempt to implement it).

9.1 Future Work

Naturally, future work would include the contributions of the camera team. In some sense, we have an intuition that the parking spaces could be explicitly categorized via semantic segmentation (since the texture and color are different than the street in the case of the provided environment). At the very least, the camera should be able to identify obstacles and the intersection of the results from the other sensor teams would provide a higher probability and accuracy of correctly identifying objects. Furthermore, it would be ideal to have an external map to manually extrapolate free spaces and objects based on a particular time and orientation. This would improve the performance rate and quality of parking lot recognition.

While some of the technologically feasible aspects are out of our scope and ability, we feel we are capable of addressing implementations such as using the UKF. It would a good opportunity to measure the effectiveness (benchmarking) of additional filter types. Furthermore, the final fusion of the tracked objects has not been proven to be optimal. The results of the RADAR and LiDAR need to be properly synchronized and fused.

During the Q&A session of the final presentation, Prof. Dr.-Ing. Olaf Hellwich suggested that we use probabilistic approaches to our object tracking and data association. Currently, we have been using a deterministic implementation as it was easier to evaluate. The next

logical step would be to evaluate the probability distributions and use the results in our object tracking.

Also, the EKF can be altered to account for multiple sampling rates from different sensors based on the work of Friedland, by setting data flags for synchronization (event-driven or time-driven) [9]. However, compared to the method that Cassandra uses, it is not clear which method would have a lower amount of overhead or optimal real-time behavior. The augmented EKF was testing in simulation but not incorporated in Cassandra. This would help assess aspects based on the requirements of computational times, requirements on update rates, and potentially the stability.

Finally, we were left with a research question for curiosity. As we did not have access to the camera data we wonder if designing a project in which it was completely omitted from the beginning would have also been valid. We consider that the camera requires dedicated hardware resources, requires additional computation time, is susceptible to blockage, and may have instability in the output (missed frames, lag/jitter/drift, etc.). Considering these aspects, benchmarking with and without a camera would allow for quantitative results in the analysis of trade-offs (likely the performance rate and quality versus resource utilization and stability). This is something that we are interested in and would have liked to explore (based on what we know now). We could make the case that we "only needed more time" but this is a common statement from anyone working on an industrial project.

9.2 Acknowledgements

We would like to thank the following Hella Aglaia employees for their continued support and collaboration throughout the project: Tino Kutschbach, Thomas Jahn, Sven-Garrit Czarnian, and Micha Bruns. Furthermore, we would like to thank Prof. Dr.-Ing. Olaf Hellwich for his feedback during the presentation and for offering the course. Of course, we would also like to thank all of the other teams for their communication and collaboration (we literally depended on it).

References

[1] Hella aglaia technology, Accessed March 29, 2019. https://hella-aglaia.com/technologie.

[2] Carnd-mercedes-sf-utilities, Accessed March 29, 2019. https://github.com/udacity/CarND-Mercedes-SF-Utilities.

[3] Root-mean-square deviation, Accessed March 29, 2019. https://en.wikipedia.org/wiki/Root-mean-square_deviation.

[4] Sae levels of driving automation, Accessed March 29, 2019. https://www.sae.org/news/press-room/2018/12/sae-international-releases-updated-visual-chart-for-its-E2%80%9Clevels-of-driving-automation%E2%80%9D-standard-for-self-driving-vehicles.

[5] Texas instruments, adas overview with sensor fusion, Accessed March 29, 2019. https://training.ti.com/adas-advanced-driver-assistance-systems-overview?cu=1136060keyMatch=camera%20radar%20lidar%20sensor%20fusiontisearch=Search-EN-Everything.

[6] Y. Bar-Shalom, F. Daum, and J. Huang. The probabilistic data association filter. *IEEE Control Systems Magazine*, 2009.

[7] R. Collins. Data association, Accessed March 29, 2019. http://www.cse.psu.edu/ rtc12/CSE598C/datassocPart1.pdf.

[8] B. Friedland. Control system design: An introduction to state-space methods. 2005.

[9] B. Friedland. Estimation and control with all kinds of data. 2016.

[10] S. Houben, M. Komar, A. Hohm, S. Lke, M. Neuhausen, and M. Schlipsing. On-vehicle video-based parking lot recognition with fisheye optics. *16th International IEEE Conference on Intelligent Transportation Systems*, ITSC 2013.

[11] A. Khan I. Noreen and Z. Habib. A comparison of RRT, RRT* and RRT*-smart path planning algorithms. page 8.

[12] Rambabu Kandepu, Bjarne Foss, and Lars Imsland. Applying the unscented kalman filter for nonlinear state estimation. *Journal of process control*, 18(7-8):753–768, 2008.

[13] P. Konstantinova, A. Udvarev, and T. Semerdjiev. A study of a target tracking algorithm using global nearest neighbor approach. *International Conference on Computer Systems and Technologies*, 2003.

[14] S. Sab and T. Jahn. Hella aglaia project management presentation at tub, Accessed March 29, 2019. https://isis.tu-berlin.de/pluginfile.php/1114406/mod_resource/content/1/20181025__ProjectManagement_ProjectSetu

[15] Sebastian Thrun. Udacity's self driving car nanodegree, Accessed March 29, 2019. https://eu.udacity.com/course/self-driving-car-engineer-nanodegree–nd013.

[16] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[17] Eric A Wan and Rudolph Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, pages 153–158. Ieee, 2000.