2020 IEEE International Conference on Robotics and Automation (ICRA)
31 May - 31 August, 2020. Paris, France

Robust Lane Detection with Binary Integer Optimization

Kathleen Brandes, Allen Wang, and Rushina Shah

***Abstract*— Formula Student Driverless (FSD) is a competition where student teams compete to build an autonomous racecar. The main dynamic event in FSD is trackdrive, where the racecar traverses an unknown track with lanes demarcated by cones. One major challenge of the event is to determine the boundaries of the lane from cones perceived online despite false positive cone detections and sharp turns. We present a binary integer optimization to address this problem by leveraging *a priori* knowledge from competition rules on parameters such as average cone spacings and minimum track width. In this paper, we describe our approach, and analyze its latency, accuracy, and robustness to false positive cone detections. This approach is used on-board to solve the lane detection problem during the competition in real-time.**

I. Introduction

With the recent acceleration in developmental efforts for driverless vehicles, Formula Student Driverless (FSD) provides an avenue for student teams to innovate in this space. The FSD racecar competition started in 2017 to challenge students to develop a fully autonomous racecar. The most difficult dynamic event of this competition is trackdrive, where the racecar must traverse an unknown track for ten laps as quickly as possible. The track boundaries are demarcated by cones (yellow to the right, and blue to the left) as shown in Fig. 1.

Fig. 1. Example of the track layout at Formula Student Competitions with the rules-specified colored sides and cone spacings.

The software stack of the MIT/DUT Driverless team follows the structure shown in Fig. 2. The perception subsystem, which has both camera and lidar pipelines, detects the cone locations and colors. The state estimation subsystem estimates the full state of the vehicle and performs localization and mapping. The path planning subsystem generates a path for the car to follow, and the controls subsystem ensures the execution of this path. A crucial piece in this pipeline is the accurate detection of the track boundaries based on the cone locations, which define the feasible driving region and the possible paths for the vehicle. This paper describes the MIT/DUT Driverless team's approach to solve this problem.

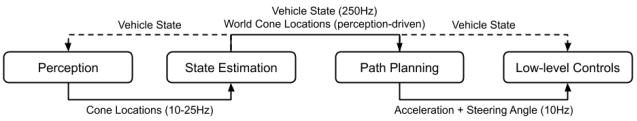

Fig. 2. The autonomous system software layout for MIT/DUT Driverless.

Several factors complicate the problem of lane detection. First, the competition track can contain very sharp turns, making the underlying geometry hard to capture. Further, the accuracy of the perception subsystem's detections can be affected by weather, lighting conditions, and sensor quality, resulting in a large number of false positives (15-25%) in the cone detections. The competition track does have color-coded left and right boundaries, but the color information provided by the perception subsystem can be inaccurate and incomplete. Further, the MIT/DUT team aimed to have complete perception redundancy using both lidar and camera pipelines, therefore the boundary detection system had to operate in scenarios (for the lidar-only system) where the problem of false-positives was exacerbated by having a small look-ahead and wide field of view, often including detections from other parts of the track or other nearby objects. In many cases, like the one shown in Fig. 3, given no other information, the number of false positives and inaccurate detections make the track geometry hard to decipher even for a human in some locations.

Additional information to aid in solving this problem is provided by the competition rules [1]. The cones on a single side are stated to be approximately 5m apart. In reality, this distance varied from 1m on tight turns to 5.5m on straight sections. The minimum track width is guaranteed to be 3m. A boundary detection system that can leverage this information, as well as information of the vehicle state, can potentially overcome the issue of uncertainty in cone detections.

A considerable amount of work has been done in detecting and tracking lane markers on the roads [2]. Most of these efforts are focused on detecting the lane markers, which are often then fit using polynomial or spline curves [3], [4], [5], [6]. Such curve-fitting approaches do not naturally

Kathleen Brandes, Allen Wang and Rushina Shah are students at the Massachusetts Institute of Technology, Cambridge, MA 02139, USA. Email addresses: kbrandes@mit.edu (K. Brandes), allenw@mit.edu (A. Wang), rushina@mit.edu (R. Shah)

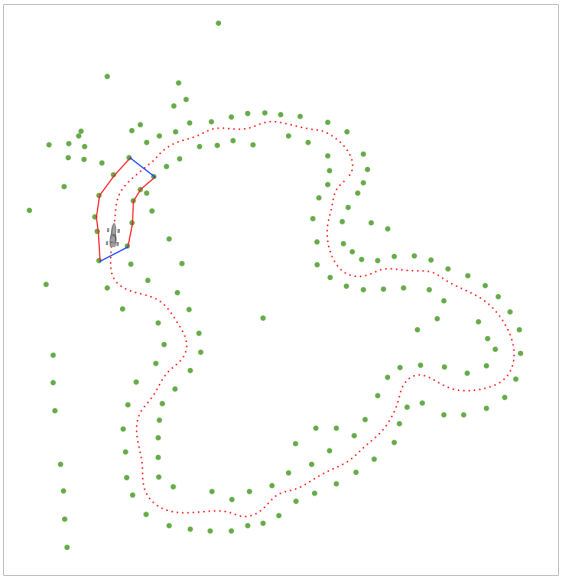

Fig. 3. Completed track with false positive accumulation. The computed boundaries are visualized surrounding the car. The green markers represent the perception system's cone detections, which include false positive detections. The red dotted line represents the path traversed.

lend themselves to including rule-based heuristics and can be skewed by outlier false positive detections. In [7], a dynamic programming approach is used to connect nodes on the lane by minimizing the cost to the goal node, after some heuristics-based edge extraction. In [8], a filter-based approach is used to generate road boundaries for scenarios with multiple different markers (such as painted lanes or "Botts Dots"). The above mentioned approaches work well for the case of road-driving, where lanes are typically marked using a solid line segment. However, these methods, along with outlier rejection methods such as random sample consensus as used by [9] and [10], are not easily adaptable to the competition scenario where cones mark the track boundaries, and are not always capable of handling larger amounts of false positives or detection inaccuracies.

To address the problem of generating track boundaries that are marked by cones, we present a novel binary integer program that is robust to false positives and false negatives, and flexible enough to incorporate rule-based information. The Robust Lane Detection optimization (RLD) is implemented in both simulation and on an autonomous electric racecar, and used as the basis for the path planning subsystem to define a driveable region in real-time, allowing the car to decipher and traverse a previously unseen competition-compliant track. Our method can be modified to take into account known lane parameters (such as the spacing between markers) or rules based on road construction standards [11]. Ideas from RLD may be useful for real-world lane detection applications in the presence of high numbers of false positives, as is likely to happen in inclement weather or at construction sites.

The main contribution of this paper is an optimization-based method for robustly determining the boundaries of the lane of a track in real-time. Section II defines the optimization problem and the constraints required to accurately produce the boundaries of the track. The implementation, including details for runtime improvements, are discussed in Section III. Section IV describes the analysis and validation of the optimization's robustness and efficiency.

II. Robust Lane Detection Optimization

We represent the boundaries of the lane with an undirected graph $G = (V, E)$ where the set of vertices V is the set of observed cones, including false positives, and E is the set of edges connecting cones that are adjacent to each other on the same boundary. The overall strategy is to solve a binary integer program to find the adjacency matrix, A, of the lane graph that is optimal with respect to a specified cost function.

A. Definitions and Notation

Definition 1. A graph $G = (V, E)$ is said to be a *lane graph* if every edge $e \in E$ is an element of one of two simple paths P_1, P_2 and $P_1 \cap P_2 = \varnothing$.

Definition 2. A cone c_i in a lane graph is said to be an *endpoint cone* if it has degree one.

Definition 3. A cone is said to be an *inlier cone* if it is a member of one of the two simple paths of a lane graph.

The optimization operates on $|V| = n$ cones, denoted c_i for $i \in [n]$, and computes $|E|$ total edges, denoted a_{ij}, connecting cones c_i and c_j. Let n_i denote the number of inlier cones. Let $\mathbf{a}$ be the vectorized form of the lower triangular portion of A arranged in column-major order and let n_a denote the number of elements of $\mathbf{a}$. Let D denote the distance matrix of size $n \times n$ where entry D_{ij} is the Euclidean distance between c_i and c_j. Throughout the paper, bold letters denote vectors, and capital letters denote matrices. The $m \times m$ matrix of ones is denoted J^m, and similarly the $m \times 1$ vector of ones is $\mathbf{j}_m$. The matrix operations of the Kronecker product and element-wise product are denoted as $\otimes$ and $\circ$, respectively. A subscript denotes the index into the vector or matrix.

B. Lane Graph Constraints

In this section, we show how linear constraints can ensure the results of an optimization problem over A form a lane graph and explicitly reject outliers. We begin by showing how boolean decision variables can explicitly mark inlier cones. Let $\mathbf{g}$ be a vector of n boolean decision variables that mark inlier cones. Let $A_i = \sum_{j=1}^{n} a_{ij}$, and note that A_i is equivalent to the degree of the i_{th} cone, c_i. Then, we can ensure that $g_i = 1$ if and only if c_i is an inlier cone by enforcing the following linear constraints $\forall i \in [n]$:

$$g_i - A_i \leq 0 \tag{1}$$

$$\frac{1}{2} A_i - g_i \leq 0 \tag{2}$$

Note that (2) also constrains the degree of cones to be no more than two. In addition, since $g_i = 1$ if and only if $A_i > 0$, we have that $A_i - 2g_i = -1$ if and only if $A_i = 1$.

Thus, we can constrain the number of endpoint cones to four by enforcing the following affine constraint:

$$\sum_{i=1}^{n}(A_i - 2g_i) = -4 \tag{3}$$

The number of cones with connecting edges is $n_i = \sum_{i=1}^{n} g_i$, which is also the number of inlier cones. The number of edges is $|E| = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}$. To ensure there are at least two disjoint subgraphs, introduce the affine constraint:

$$\frac{1}{2}\sum_{i=1}^{n} A_i \leq \sum_{i=1}^{n} g_i - 2 \tag{4}$$

This constraint holds because, for any graph, the sum of the degrees of all vertices equals twice the number of edges. Therefore, $\sum_{i=1}^{n_i} \text{degree}(c_i) = 2n_i - 4 = 2|E|$, and all other non-inlier cones do not contribute in the sum. However, the smallest connected graph will have at least $|V| - 1$ edges, which contradicts (4). Furthermore, a connected graph has no cycles when $|E| \leq |V| - 1$. When there are exactly two disjoint, connected subgraphs, constraint (4) ensures that neither will have cycles.

The constraints thus far ensure the result consists of exactly two disjoint paths, but allows for additional disjoint cycles. Additional disjoint cycles can be prevented with an additional constraint. Let $\mathcal{P}(V)$ denote the power set of the set of cones. For each set $S \in \mathcal{P}(V)$ and each cone $c_i \in S$, the following constraints can prevent cycles:

$$\sum_{c_i, c_j \in S} a_{ij} \leq |S| - 1, \quad \forall S \in \mathcal{P}(V) \tag{5}$$

While it is possible to prevent cycles by imposing additional constraints of the form (5) described in [12], we found that, in practice, cycles would never be a part of the optimal solution due to the high angle costs introduced by cycles and the relatively small number of cones optimized over. To guarantee the solution is a lane graph, a post-optimization check eliminates any potential cyclic subgraphs at a lower computational cost than enforcing (5).

C. Pairwise Edge Constraints

The competition rules specify the track will have a minimum width of at least 3m, and similarly, for more general applications, all road lanes should at least be the width of the car. Therefore, we want to ensure the lane graph boundaries will be at least the minimum distance apart. To constrain the minimum width of the lane to be greater than a distance, d_{min}, we introduce pairwise edge constraints:

$$a_{ij} + a_{kl} \leq 1 \tag{6}$$

This restricts certain edges a_{ij} and a_{kl} to not both be members of the optimal lane graph. We only use pairwise edge constraints to enforce the minimum width of the track, but they can be used to enforce any condition that specifies two edges should not simultaneously be members of the optimal lane graph.

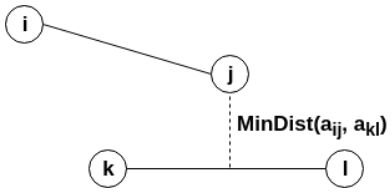

Fig. 4. Example of two line segments satisfying $\text{End}(a_{ij}, a_{kl}) = 0$.

Let $\text{MinDist}(a_{ij}, a_{kl})$ be the minimum Euclidean distance between the edge segments a_{ij} and a_{kl}. If we enforce constraint (6) whenever $\text{MinDist}(a_{ij}, a_{kl}) > d_{min}$, we would limit the maximum length of the lane the optimization routine can identify. Thus, the minimum distance between edges exceeding d_{min} is not a sufficient condition for introducing constraint (6). Let $\text{End}(a_{ij}, a_{kl})$ be a function that returns 1 if the minimum Euclidean distance path between a_{ij} and a_{kl} connects an endpoint of a_{ij} with an endpoint of a_{kl} and 0 otherwise, as shown in Fig. 4. Then, for $i \neq j \neq k \neq l$, introduce the constraint (6) if the following condition is met:

$$\{\text{MinDist}(a_{ij}, a_{kl}) > d_{min}\} \wedge \{\text{End}(a_{ij}, a_{kl}) = 0\} \tag{7}$$

If the distance condition is met, then, in all scenarios encountered in practice, $\text{End}(a_{ij}, a_{kl}) = 0$ only if a_{ij} and a_{kl} do not belong to the same side. The pairwise constraints are implemented in matrix form:

$$P\mathbf{a} \leq \mathbf{j}_{n_p} \tag{8}$$

The matrix P is computed during pre-processing such that each row consists of two entries that are one and will meet the constraint (6). P has n_a columns, but the number of rows of P, n_p, varies depending on the number of pairwise constraints that need to be introduced based on condition (7) for any given instance of the optimization problem.

D. Heading Constraint

The car is assumed to be traveling in approximately the correct direction along the track, so we want to also ensure the nearest boundaries are pointing in a direction similar to the car's heading. Additional constraints are added such that the angle between the car's heading vector and at least two edges within d_h meters of the car have a difference of at most $\theta_{heading}$ radians. In pre-processing, we compute a vector $\mathbf{h}$ of size n_a such that $h_i = 1$ if the closest point on the i_{th} edge is within d_h meters of the car and the angle between the edge and the car's heading vector is within $\theta_{heading}$ radians of each other. Otherwise, $h_i = 0$. We then enforce the linear constraint:

$$\mathbf{h}^T\mathbf{a} \geq 2 \tag{9}$$

E. Endpoint Distance Constraints

We also constrain the endpoint cones of the lanes such that at least two of the degree one cones are within d_{near} meters of the car and the other two are at least d_{near} meters away from the car. This constrains one end of the lane graph to be near the car, ensuring the vehicle is within or near the feasible

driving region defined by the boundaries and reducing any needed path extrapolation from the car to the driving region. Let $\mathbf{m}$ denote a vector of n variables such that:

$$m_i = -(A_i - 2g_i) \tag{10}$$

Then, $m_i = 1$ if and only if the i_{th} cone has degree one. In pre-processing, we compute the vector $\mathbf{r}$ such that $r_i = 1$ if the i_{th} cone is within d_{near} meters of the car and $r_i = 0$ otherwise. We can ensure that exactly two endpoints are within d_{near} meters of the car by enforcing the linear constraint:

$$\mathbf{r}^T\mathbf{m} = 2 \tag{11}$$

F. Maximum Endpoint Distance Constraint

In practice, we found it helpful to also constrain the maximum distance, d_{max}, between the two endpoints that are further than d_{near} meters away from the car. This allows us to leverage the fact that there is a maximum reasonable lane width and prevents one side of the lane from being significantly longer than the other. Note that $m_i - r_i = 0$ if and only if the i_{th} cone is an endpoint cone and is within d_{near} meters of the car. To mark the endpoint cones that are further than d_{near} meters of the car, we introduce a new vector of boolean decision variables $\mathbf{u}$ and enforce the constraint:

$$m_i(1 - r_i) + u_i = 1, \forall i \in [n] \tag{12}$$

Define the following matrices:

$$\tilde{M} = \mathbf{j}_n^T \otimes \mathbf{m} = [\mathbf{m}, ..., \mathbf{m}] \tag{13}$$

$$\tilde{R} = \mathbf{j}_n^T \otimes (\mathbf{r} \circ \mathbf{m}) = [\mathbf{r} \circ \mathbf{m}, ..., \mathbf{r} \circ \mathbf{m}] \tag{14}$$

$$\tilde{U} = \mathbf{j}_n^T \otimes \mathbf{u} = [\mathbf{u}, ..., \mathbf{u}] \tag{15}$$

The i_{th} row of the matrix $\tilde{M} - \tilde{R}$ is all ones if the i_{th} cone is an endpoint cone greater than d_{near} meters from the car; otherwise, the row consists entirely of zeros or negative ones. Due to the endpoint distance constraints, exactly two rows will consist of ones; let these rows be at indices i^* and j^*. In matrix $\tilde{U}$, there will be all zeros in rows i^* and j^*. Otherwise, the rows consist of ones or twos. Define a new matrix:

$$Q = \tilde{M} - \tilde{R} - \tilde{U}^T \tag{16}$$

The matrix Q has ones only at the entries: $Q_{i^*i^*}, Q_{j^*j^*}, Q_{i^*j^*}, Q_{j^*i^*}$. All other elements are zero or negative. Thus, we have that $\max D \circ Q$ is equivalent to the distance between the two endpoint cones that are further than d_{near} meters away from the car. So, we enforce the linear constraint (by introducing a slack variable):

$$\max D \circ Q \leq d_{max} \tag{17}$$

G. Cost

The RLD optimization is a minimization of a cost function derived from expected spacings between cones, the angles between adjacent edges, and the parallelism of edges and the car's heading. This cost function serves as a heuristic for rejecting false positives and finding the correct lanes from the set of perceived cones.

Let d_e be the expected spacing between two cones and d_t be a tunable threshold parameter. The spacing cost for edge a_{ij} is:

$$s_{i,j} = \left(\frac{D_{ij} - d_e}{d_t}\right)^4 \tag{18}$$

Let $\mathbf{s}$ be the vector of spacing costs $s_{i,j}$, and w_s be the spacing weight. The spacing cost term is thus:

$$w_s\mathbf{s}^T\mathbf{a} \tag{19}$$

Let angle(a_{ij}, a_{jk}) be the acute angle between the two adjacent edges a_{ij}, a_{jk}, let θ_e be the expected angle, and let θ_t be the threshold angle. In principle, it is possible to change θ_e based off prior knowledge of the track or the current state of the car, but, in practice, we found that setting θ_e to π proved to be effective. This is because connecting false positives would often introduce very "sharp" angles into the resulting lane graph. The cost for the angle between adjacent edges is:

$$t_{i,j,k} = \left(\frac{\text{angle}(a_{ij}, a_{jk}) - \theta_e}{\theta_t}\right)^4 \tag{20}$$

To introduce these costs linearly into the cost function, we define a new vector, $\mathbf{f}$, of $n_f = \frac{n^2(n-1)}{2} - n(n-1)$ binary variables with elements:

$$f_{ijk} = \begin{cases} 1 & a_{ji} = a_{jk} = 1 \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

To achieve this behavior, we enforce the constraints:

$$2f_{ijk} - a_{ji} - a_{jk} \leq 0 \tag{22}$$

$$f_{ijk} - a_{ji} - a_{jk} \geq -1 \tag{23}$$

These constraints can be understood by logically considering all combinations of the binary edge variables a_{ij} and a_{jk}. When $a_{ij} = a_{jk} = 0$, then (22) implies that f_{ijk} must be 0 while (23) is automatically satisfied. When $a_{ij} = a_{jk} = 1$, both (22) and (23) imply that $f_{ijk} = 1$. In the case that either a_{ji} or a_{jk} equals one and the other equals zero, (23) is automatically satisfied while (22) implies that $f_{ijk} = 0$. So, let $\mathbf{t}$ be the vector containing all of the $t_{i,j,k}$ costs according to an order consistent with $\mathbf{f}$, and let w_t be the angle cost weight. The angle cost term is thus:

$$w_t\mathbf{t}^T\mathbf{f} \tag{24}$$

The vehicle state is utilized to determine whether an edge is parallel to the car's heading. During nominal operation, the edges near the car will be approximately parallel to the car's heading, so the angle between the edges near the car and the car's heading can be used in the cost function. Let the vector pointing from cone i to cone j be $\mathbf{v}_{ij}$ and let $\mathbf{h}_c$ be the car's heading vector. Note that $|\mathbf{v}_{ij} \times \mathbf{h}_c|$ will be larger when the acute angle between the two vectors is larger; additionally, the absolute value is needed so that $\mathbf{v}_{ij}$ and $\mathbf{v}_{ji}$ have the same cost. Edges further away from the car are not expected

to be parallel to the car's heading, so the cost falls off based on distance of the midpoint of the edge to the car, $d_{\text{car}}(a_{ij})$. The heading cost for a_{ij} is:

$$p_{ij} = \left(\frac{|\mathbf{v}_{ij} \times \mathbf{h}_c|}{d_{\text{car}}(a_{ij})} \right)^4 \tag{25}$$

Let $\mathbf{p}$ be the vector of p_{ij} with an ordering consistent with $\mathbf{a}$, and let w_p the the heading cost weight. The cost term is thus:

$$w_p \mathbf{p}^T \mathbf{a} \tag{26}$$

Finally, let w_b denote a uniform "benefit" of adding an edge. Without this term, the optimization will never find edges since a pair of cones without an edge does not contribute cost and all other costs are positive and additive.

H. High Cost Constraints

Certain possible edges and angles have costs that are too high to be members of any reasonable solutions and can therefore be ruled out with constraints. Let s_{crit} be the maximum allowed spacing cost for an edge. Now let $\mathbf{s}^c$ be a pre-computed vector of Boolean variables such that $s^c_{ij} = 1$ if $s_{ij} > s_{crit}$ and $s^c_{ij} = 0$ otherwise. Similarly, we introduce t_{crit} and vector $\mathbf{t}^c$ to constrain large angle costs between edge pairs. We then impose the constraints:

$$\mathbf{s}^c \circ \mathbf{a} \leq \mathbf{j}_{n_a} \tag{27}$$

$$\mathbf{t}^c \circ \mathbf{f} \leq \mathbf{j}_{n_f} \tag{28}$$

I. Optimization Problem Statement

The complete Robust Lane Detection optimization for determining a lane graph from a set of cone detections is as follows:

$$\min_{\mathbf{a},\mathbf{f},\mathbf{g},\mathbf{u}} (w_s \mathbf{s} + w_p \mathbf{p} - w_b \mathbf{j}_{n_a})^T \mathbf{a} + w_t \mathbf{t}^T \mathbf{f} \tag{29}$$

$$s.t. \quad g_i - A_i \leq 0, \quad \frac{1}{2} A_i - g_i \leq 0 \tag{30}$$

$$\sum_{i=1}^{n} (A_i - 2g_i) = -4 \tag{31}$$

$$\frac{1}{2} \sum_{i=1}^{n} A_i \leq \sum_{i=1}^{n} g_i - 2 \tag{32}$$

$$P\mathbf{a} \leq \mathbf{j}_{n_p}, \quad \mathbf{h}^T \mathbf{a} \geq 2, \quad \mathbf{r}^T \mathbf{m} = 2 \tag{33}$$

$$\mathbf{s}^c \circ \mathbf{a} \leq \mathbf{j}_{n_a} \quad \mathbf{t}^c \circ \mathbf{f} \leq \mathbf{j}_{n_f} \tag{34}$$

$$\max D \circ Q \leq d_{max} \quad A = A^T \tag{35}$$

III. Implementation

The RLD optimization, described in Section II-I, is a binary integer program. It is modelled in CVX and solved with Mosek [13], [14]. Intel's Math Kernel Library, [15], is used to accelerate the matrix optimizations within Mosek. Extensive work has been done to decrease the latency of RLD such that the algorithm can be run on vehicle during mapping laps without inhibiting speed. Warm starting RLD with previous successful results decreases the optimization solve time and improves the result quality. Additionally, the RLD instance can be built at the program's start, and parameters (e.g. cone positions) can be updated before computing each optimization, reducing latency by removing repeated computations independent of situational information like cone positions.

The worst case scaling of the number of RLD decision variables is $O(n^3)$, but, in practice, the large number of introduced constraints makes the effective search space much smaller. The optimization directly takes the cone positions as input, requiring minimal preconditioning and reducing solve times, as opposed to Delaunay triangulation or other methods which reduce the problem space by inputting favorable edges, yet require more computation time.

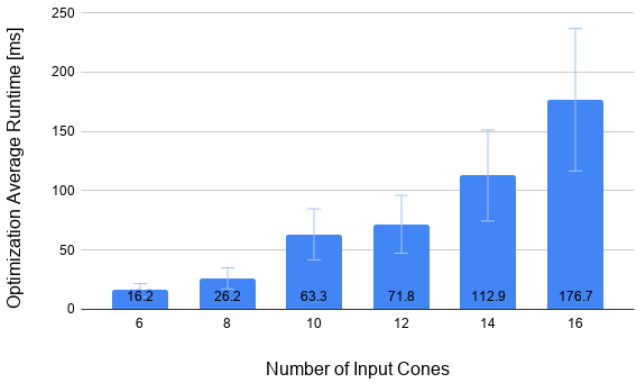

Fig. 5. Average solve time analysis for multiple different cone set sizes. The error bars represent one standard deviation from the average solve time.

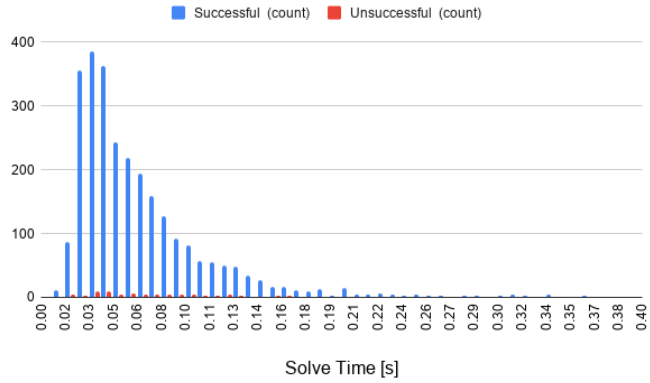

Fig. 6. Distribution of solve times for a cone set of n=12 run on vehicle for 10 consecutive mapping laps. The average solve time for successful solves is 71.79ms, and only 2.81% of solves were unsuccessful.

IV. Results

A. Runtime Analysis

Latency of the RLD optimization is measured from the input of cone positions and optimization parameters to the output of the boundary edge set. The resulting latencies were collected on a fully stressed, 7-core CPU machine running RLD for 20 iterations in mapping mode (no prior cone information) on data collected for a 340m track. The RLD

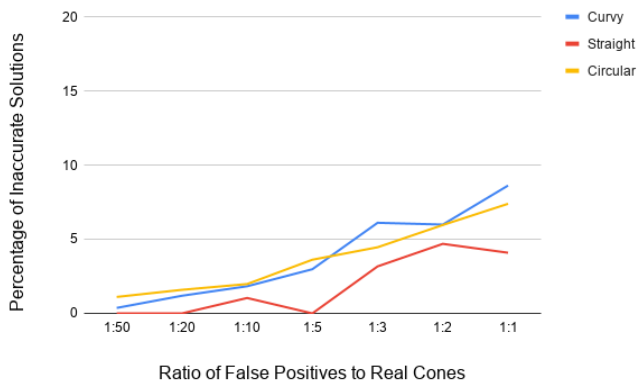

Fig. 7. Percent of inaccurate solves on different track topologies for increasing ratios of false positives. Less than 10% of all solves are incorrect for any track topology, even with a 1:1 ratio of false positives to cones.

optimization was run with variable cone set sizes to measure how the solve times will scale with increasing number of input cones, as shown in Fig. 5. RLD can reliably compute solutions in under 80ms, on average, for cone sets less than 14 cones.

The results of RLD are classified as successful when the optimizer result is found within a time limit of 0.4 seconds and achieves an absolute optimality gap less than some threshold, $\epsilon = 1 \times 10^{-5}$. The absolute optimality gap is defined as the difference between the minimum possible objective and the minimum computed objective; it is calculated by the optimizer through the duality gap as described in [16]. For a fixed size cone set of $n = 12$, Fig. 6 shows the distribution of RLD solve times. 97.19% of all iterations of the optimization are successful, with an increase in average latency of only 1.71ms for unsuccessful solves. RLD can solve 95% of all $n = 12$ optimizations within 155.88ms (standard deviation $\sigma = 51.96$ms). Additionally, Fig. 6 shows only a tail end (.1%) of the successful solve times are greater than .26 seconds; a trade-off can be made to sacrifice these successful results for guarantees on lower latency by capping the maximum solve time for the optimizer.

In practice, we found $n = 12$ to utilize the entire perception sensor look ahead and field of view with the lowest average latency. With the low latency of RLD and the other subsystem latencies, we have been able to complete mapping laps at speeds up to 5.65 m/s.

B. Robustness and Accuracy

The algorithm is robust to many different situations in which it is difficult to determine the correct lane graph from the cone set, such as many false positives or inaccurate positions. To evaluate the robustness and determine potential failure modes, the RLD optimization was tested on multiple concave and convex track topologies. Random, uniform noise is inserted as additional cone detections in simulation.

The optimization results are first classified as successful using the absolute optimality gap described in Section IV-A. The result is then classified as accurate if it passes a series of quality checks: less than or equal to one cone different from the expected boundary cone set, one boundary on each side of the vehicle, no edges crossing, and the distance between boundaries falling within the minimum and maximum width.

The accuracy of the optimization was evaluated on a curvy track (concave and convex portions of the track), a straight track, and a circular track. These geometries can be combined to represent an expected track topology at the FSD competition. For all three track topologies, the RLD optimization has only 3.62% inaccurate solves for 1:5 false positives to real cones, which is a comparable amount of noise for tracks of 250-400m length experienced by the racecar. The type of track topology has minimal effect on the accuracy of results, with the percent incorrect deviating by at most 4.53% across all track topologies. However, RLD tends to be slightly less accurate for curvier tracks since this deviates from assumptions made by the cost terms in equations (20) and (25). All track topologies achieve less than 10% inaccurate solutions for extreme amounts of false positives (1:1 ratio of false positive to real detections).

The robustness and stability of the optimization can allow the perception subsystem to lower thresholds for detection; this can introduce more false positives, but also increase look ahead range. A video of the RLD optimization running successfully during the Formula Student Germany competition on an autonomous racecar completing a mapping lap at 4m/s is available at https://youtu.be/bEQi4eglH2Y.

V. Conclusion

To create an efficient and robust system to identify track boundaries during the Formula Student Driverless competition, we developed the Robust Lane Detection optimization to find two fully connected paths connecting cone detections. RLD utilizes the vehicle state and track information specified by the competition rules. The optimization is shown to be robust to false positive detections, failing to find the accurate solutions only 3.62% of the time for different track topologies with false positive detections matching those of the vehicle operating in real time. The RLD optimization ran in real-time on an autonomous electric racecar in mapping mode during competition and testing.

Additional constraints to prevent edge crossing and cost terms utilizing color information of the cones were incorporated into the optimization for improving the result specifically for Formula Student Driverless. The RLD optimization could be extended beyond the scope of Formula Student as a robust method for detecting lanes based on consistent markers, such as mile markers. For future work, we plan to incorporate detection confidences into the optimization cost function and relax binary constraints to decrease latency.

ACKNOWLEDGMENT

The authors thank MIT/DUT Driverless for the opportunity to develop software for an autonomous racecar and for the consistent support throughout the process.

REFERENCES

[1] "Formula student rules 2019," 2019. [Online]. Available: https://www.formulastudent.de/fsg/rules/

[2] A. Bar-Hillel, R. Lerner, D. Levi, and G. Raz, "Recent progress in road and lane detection: a survey," *Machine Vision and Applications*, vol. 25, pp. 727–745, 2011.

[3] Y. Wang, E. K. Teoh, and D. Shen, "Lane detection and tracking using b-snake," *Image and Vision Computing*, vol. 22, no. 4, pp. 269 – 280, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0262885603002105

[4] Y. Wang, D. Shen, and E. K. Teoh, "Lane detection using spline model," *Pattern Recognition Letters*, vol. 21, no. 8, pp. 677 – 689, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167865500000210

[5] A. S. Huang, D. Moore, M. Antone, E. Olson, and S. Teller, "Finding multiple lanes in urban road networks with vision and lidar," *Autonomous Robots*, vol. 26, no. 2-3, pp. 103–122, 2009.

[6] D. Neven, B. De Brabandere, S. Georgoulis, M. Proesmans, and L. Van Gool, "Towards end-to-end lane detection: an instance segmentation approach," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 286–291.

[7] S. P. Adhikari and H. Kim, "Dynamic programming and curve fitting based road boundary detection," in *Proceedings of the 9th WSEAS international conference on computational intelligence, man-machine systems and cybernetics*, 2010, pp. 236–240.

[8] J. C. McCall and M. M. Trivedi, "An integrated, robust approach to lane marking detection and lane tracking," in *IEEE Intelligent Vehicles Symposium, 2004*. IEEE, 2004, pp. 533–537.

[9] A. Borkar, M. Hayes, and M. T. Smith, "Robust lane detection and tracking with ransac and kalman filter," in *2009 16th IEEE International Conference on Image Processing (ICIP)*, Nov 2009, pp. 3261–3264.

[10] H. Du, Z. Xu, and Y. Ding, "The fast lane detection of road using ransac algorithm," 01 2018, pp. 1–7.

[11] "Standard specifications for construction of roads and bridges on federal highway projects."

[12] L. Klein, "Combinatorial optimization with one quadratic term," Ph.D. dissertation, 2014.

[13] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, version 2.1," http://cvxr.com/cvx, Mar. 2014.

[14] E. D. Andersen and K. D. Andersen, "The mosek interior point optimizer for linear programming: an implementation of the homogeneous algorithm," in *High performance optimization*. Springer, 2000, pp. 197–232.

[15] *Intel Math Kernel Library Reference Manual*. Intel Corporation, 2009.

[16] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge University Press, 2004.