Grand Prix Grand Prix (White Paper)

Evan Lohn

December 2021

Abstract

We seek to bring the powerful modeling and verification capabilities of hybrid programs expressed in KeymaeraX to the domain of automated racecar driving. In particular, we model a single modified Dubins car racing on an arbitrarily complex racetrack, and seek to prove that it never crashes for an appropriate choice of controller. To reduce proof complexity and allow for proof re-use over multiple controllers and tracks, we prove safety properties for individual controllers and sections and use ideas from Component-Driven Proof Automation [1] to combine these proofs and show that the controller is safe for the entire racetrack. While we lack significant experimental results, we also produced several mathematical results involving the safety of a variety of controllers, and introduced a flexible system for modeling many different tracks and car controllers.

1 Introduction

Using Cyber-Physical Systems (CPS) to model systems that rely on the interplay of continuous dynamics and discrete decision making allows for formal verification of the safety of said systems using a theorem prover such as KeymaeraX. While any formal guarantees are only as strong as the modeling fidelity, proofs about well-justified models are a useful tool for increasing confidence about safety properties. In this work, we apply CPS safety analysis to the domain of racecar driving. In particular, we focus on proving that a certain class of racecar controllers will never cause a single racecar to crash into the boundaries of a racetrack.

Racecar drivers are agents acting in a highly complicated and dynamic space containing challenges such as avoiding walls, avoiding other cars, picking optimal routes, and managing tire wear via pit stops. Indeed, provably safe racecar driving can be seen as a special case of the well-studied provably safe autonomous vehicle (AV) driving domain [3]. Racecar driving is simpler than the general (AV) case in that in a given race there is a small, fixed number of cars to consider and no need to make high-level navigation decisions. However, racecar driving is also more complicated than the most common self-driving car scenarios, since racecars may need to navigate many bends in a track while operating at high velocity.

The main challenges of modeling racetrack safety with a hybrid program (HP) are the complexity of describing entire racetracks and the complexity of possible controllers that a racecar can use to navigate a given track. It is possible (although tedious) to verify properties of any particular controller for any particular racetrack, but in this work we are interested in techniques that allow analysis of a class of tracks and controllers large expressive enough to be useful in proving properties in the real world.

To address these challenges, we compose proofs about modular racetrack sections and controllers in order to create safety proofs for entire racetracks. By creating a system for describing and piecing together racetrack sections, arbitrarily complex tracks can be created. However, the simplicity of each section makes proofs about a controller's safety while inside them far more feasible. The final requirement for proof composition is entry and exit contracts that are specified for each (controller, track segment) pair. Each entry contract specifies what conditions the controller requires to be safe in the track segment, whereas exit contracts specify what properties the controller guarantees about the racecar if it reaches the end of the segment.

2 Related Work

2.1 Modular CPS proofs

Unsurprisingly, the idea to reduce complicated hybrid systems to their component parts is not a new one. One example of work that uses a similar structure of components and contracts is [2], which proved properties about complex traffic networks modeled as a graph of components connected by varying capacities of "traffic flow". While this work used similar techniques (e.g. components, contracts) to the ones presented here and also involves autonomous vehicles, it proved aggregate properties about traffic rather than safety of individual cars. High-level models of traffic flows would not be appropriate for proving racecar safety, as even a single racecar on a track is at risk of crashing simply due to the high speeds of racecar driving.

Another example of modular CPS proofs can be seen in [1], where the authors produced a roller-coaster track generation system and proved properties about passenger safety and comfort for any generated track. The idea of track generation and reusing proofs of individual sections (termed Component-Driven Proof Automation) is present in our work as well, but our work also includes producing controllers for non-trivial models of car physics and guaranteeing their safety. CoasterX proofs involved calculating and proving velocity and g-force constraints on roller coaster tracks without the need to specify any explicit control of coasters beyond that of physics.

2.2 Autonomous Vehicle Safety

In comparison to component-based CPS proofs, autonomous vehicle safety is a much more well-studied field that intersects with this work. In [5], the authors conclude that real-world testing will not be sufficient evidence of car safety, and identify "mathematical modeling and analysis" as one potential avenue for increasing our confidence in self-driving cars. [5] therefore provides some justification for why our work and similar analysis is important for the future. In [4], the authors also formally verify safety of autonomous vehicles. The authors use the logic-programming language Gwendolen to model Beliefs, Desires, and Intents of multiple agents, and verify that the high-level navigation decisions these agents make do not conflict with each other. Unlike our work, [4] abstracts away the low-level controller and does not consider each agent's safety relative to the road they are on, because it primarily focuses on analyzing large platoons of vehicles on highways.

When searching for literature handling low-level control of autonomous vehicles, the vast majority of work one will find uses machine learning (ML) to generate control inputs. This is largely because the enormous space of possible driving states is essentially impossible to account for with an explicit algorithm. In terms of work verifying controllers that use ML, [6] and [7] use an approach that only allows a reinforcement learning to select safe actions whenever the environment is modeled

correctly, ideally allowing the learned controller to be safe with minimal sacrifice of flexibility. One interpretation of our work and future work in this direction is that we prove the safety of a class of actions (controllers) under a set of scenarios (track segments) that could be given as input to an algorithm like the one developed in [6].

3 Approach

3.1 Car modeling

One of the most important decisions to be made in modeling racecar safety is the physical model of the racecar. Although models such as [8] are important for real-world control of racecars, the goal of this work is to analyze for safety any approximately dynamically feasible paths that a car controller could follow. Adding details about tire grip and suspension would restrict the possible trajectories we analyze down to a more realistic set, but this restriction is not necessary as long as the model we choose has a separate justification that any path a controller generates is dynamically feasible.

The model we use is an adapted version of the Dubins car [9], meaning a car represented by a point with orientation. The position (x, y) and orientation θ of a dubins car are updated over time via the model $\dot{x} = \cos(\theta), \dot{y} = \sin(\theta), \dot{\theta} = k$ where k is the controller input. The key to restricting dubins models to only allow dynamically feasible paths is restricting the range of k, as lower values of $|k|$ gives the dubins car a wider turning radius ($k = 0$ corresponds to straight-line movement). We leave the maximum value of k as a variable in our model, allowing the user of this proof approach to specify higher or lower max steering angles as desired.

The dubins car and its counterpart Reed-Shepps [11] are well-studied in control literature, with optimal (i.e. shortest dynamically feasible) paths being computable between any starting and ending position and orientation [10,12]. However, we note that this work is not preoccupied with the optimality of paths we seek to prove safe, since any real world controller is unlikely to be perfectly optimal in the sense of minimizing distance, as heuristic safety considerations (i.e. not veering too close to a boundary) make such "optimal" paths often undesirable.

Our adaptations to the classic dubins car to make it more interesting include adding the ability for the racecar to accelerate and break, as well as modifying the state space of the car to have an explicit $[dx, dy]$ unit vector describing the car's orientation for the purpose of modeling the car in KeymaeraX, which does not allow the use of trigonometric functions. In particular, the following is the single model of car physics we use for all race track segments ($'$ denotes the derivative operator with respect to time):

$$v' = a, \quad dx' = s * dy, \quad dy' = -s * dx, \quad x' = v * dx, \quad y' = v * dy, \tag{1}$$

where a is the acceleration (restricted to $-B < a < A$ for $A, B > 0$) decided by the car controller for the current segment and s is the "turning speed" of the car for the segment. We note that explicitly solving these equations for dx and dy yields $dx(t) = c_1 \cos(st) + c_2 \sin(st), dx(t) = c_2 \cos(st) - c_1 \sin(st)$ where c_1, c_2 are constants depending on initial conditions. For $s > 0$, the period of both dx and dy is $2\pi/s$, which implies that $s/(2\pi)$ complete rotations are completed per time unit, and therefore that the car changes orientation at a rate of s radians per time unit. The "turning speed" s is the equivalent to the k parameter mentioned earlier, and restricting its absolute value allows for proofs that show a racecar is safe on a racetrack even when it has a high turning radius.

One notable failure of the dubins model in the racecar context is that it does not account for wheel slipping that can occur at high speeds. Wheel slipping while turning has the effect of increasing turning radius, so this failure can be mitigated by using conservative estimates for the max turning speed of the car being controlled. For the issue of wheel slipping while accelerating or braking, the maximum allowed magnitude of possible braking B and accelerating A can be reduced to improve the model's fidelity to real-world conditions.

3.2 Track Sections

In order to make this approach generalizable to a large class of tracks, we pick a small set of track segments and allow for arbitrary composition of said segments satisfying the following rules:

1. each segment has an exit and entrance defined as a specific line segment. $Goal(i)$ is a predicate that is true when the current car position is within the exit of the ith segment, and false otherwise.
   (a) For example, we might have $Goal(1) := x >= 0 \quad \&\& \quad x <= 1 \quad \&\& \quad y = 1$, meaning the goal of segment 1 is the line segment from $(0, 1)$ to $(1, 1)$
2. The exit of segment i must be the entrance of segment $i + 1$, except for the first and last segments which can have an arbitrary entrance and exit respectively.
3. no two segments in a track overlap, except at their entrances and exits

3.2.1 Straight segments

The simplest type of segment we allow for use in composition is a straight segment. We allow for arbitrary width and height straight segments. Our current work only proved safety (see Controller Proofs section below) for vertical straight segments (i.e. the boundaries are vertical line segments, the entrance and exit are horizontal), but it is trivial to do the same proofs for horizontal and even skew segments via a change of coordinates approach. The compositionality of multiple straight segments in a row actually adds significant expressive power to the racetrack safety model, as it allows the racecar to switch to another provably safe policy partway through a straight section. For a given section of straight track that a user of this methodology would like to model, increasing the number of conjoined straight segments used for the racetrack model allows for arbitrarily fine-grained control of the car.

3.2.2 Curve segments

Without a curve segment, it would be very difficult to model racetrack turns. Each curve section is a subsection of some circle C_{out} with radius r_{out}. In particular, the curve segment is a sector of C_{out} swept out by the angle range $[\theta_0, \theta_1]$ and excluding all point that are part of a circle C_{in} concentric with C_{out} but with smaller radius r_{in} (see Figure 1). Each segment is restricted to sweeping out at most 180 degrees of the circle, but this does not restrict the space of turns that can be modeled, as it is perfectly valid to compose a 180 degree curve segment with another curve segment that simply extends the former to sweep out a greater angle. The fact that this segment is defined for arbitrary $r_{in}, r_{out}, \theta_0, \theta_1$ allows for arbitrarily tight and loose curves (for example, large r and small $|\theta_1 - \theta_0|$ corresponds to a slight bend). The 180 degree restriction for the curve is used because it allows

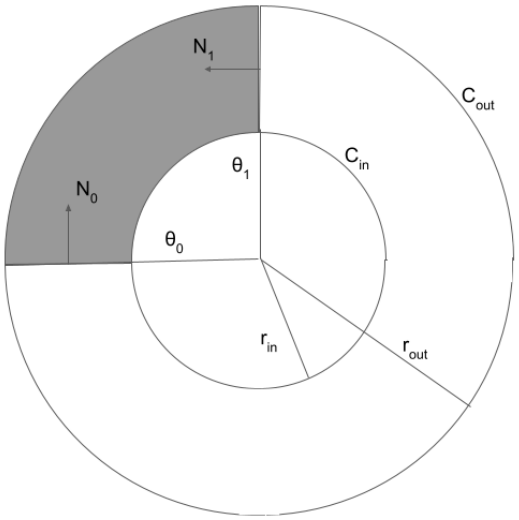

Figure 1: An example curve segment (colored gray) with the circles and angles used to form it.

for cleaner math to compute whether the car is inside the curve section. In particular, the formula for $InBounds(i)$ (a predicate used in the Hybrid program, see "Hybrid Program Structure") for curved segment i is

$$Inbounds(i) := car \cdot N_0 >= 0 \quad \&\& \quad car \cdot N_1 >= 0 \quad \&\& \quad r_{in} \leq \|car\|_2 \leq r_{out} \tag{2}$$

Where $car := [x - C_x, y - C_y]$ is the car's position relative to the center $[C_x, C_y]$ of C_{in}, N_0 and N_1 are the normal vectors of the entrance and exit respectively of the curved segment pointing into the segment. Finally, we note that our proofs only handled the case where the racecar turns right, but it is trivial to redo the proof to obtain another section where the entrance and exit are flipped.

3.3 Hybrid Program Structure

As our goal is to prove the hybrid system of a racecar driving around a track is safe, we model each racetrack safety proposition of interest with differential Dynamic Logic (dDL) [13]. The structure of the dDL formula we seek to prove in all cases is as follows:

$$P \rightarrow [\alpha^*]G. \tag{3}$$

For some premise P (often expressed as a long conjunction of facts), we prove that all runs of the hybrid program α repeated some number of times result in G holding true, where G is also commonly a conjunction.

We now explain the structure of each full-track safety proof. The premises P in this case restrict the starting position and orientation of the car to be at the "starting line" of the first track segment facing forwards. The goal G has two conjuncts; the first is an assertion that the racecar is somewhere within the bounds of the racetrack. Once we define the dDL formula $InBounds(i)$ to

be true when the current position of the car is inside or on the boundary of track segment i, this conjunct can be expressed as $\bigvee_{i=1}^{n} InBounds(i)$. The second is a guarantee that dx and dy form a unit vector, as this fact is important for establishing confidence that the car is never moving faster than whatever its current velocity variable v is set to.

The hybrid program α describes the evolution of a racecar's position over time as it traverses the racetrack. It is structured as a loop containing a "control phase" where the controller decides its setting for a and s based on its current position and orientation, and a "dynamics phase" where it follows its dynamics as outlined in the previous section.

3.3.1 Control Phase

The control phase selects its control parameters based on the car's current position and information about the track segments. Additionally, we define $Goal(i)$ to be true when the car's position has reached the exit of segment i. Finally, we define $controller(i)$ to be the hybrid program that chooses how to assign a and s based on x, y, dx, dy that the car intends to use to traverse segment i. The control segment is structured as

$$c_1 \cup c_2 \cup ... \cup c_n \cup c_{err} \tag{4}$$

where n is the number of track segments. Each c_i is defined as a HP as follows:

$$c_i := \quad ?InBounds(i) \quad \&\& \quad \neg Goal(i); \quad controller(i); \tag{5}$$

c_{err} is a simple test that is the negation of the disjunction of all the tests used in c_i. This extra choice is provably never taken by the model, as taking it implies that the car has already crashed. For completeness, it is important that this fact is proven with c_{err} in the model, since without c_{err} the global safety condition would be trivially satisfied by the tests of the control phase if the car started outside the track. For simplicity, we do not add this test into our model and simply ensure that the car's starting position is inside the racetrack, because any controller that allows the racecar to leave the track will fail the test of G immediately after the dynamics phase.

3.3.2 Dynamics Phase

The dynamics phase is slightly more complicated than simply following the car dynamics for some max amount of time, as the car must be able to change its control strategy when it reaches a segment boundary to allow the composition of individual segment safety proofs. To allow this switch, the dynamics phase consists of the nondeterministic choice $\cup$ between several nearly identical differential equation HPs:

$$d_1 \cup d_2 \cup ... \cup d_n \cup O \tag{6}$$

where n is the number of track segments. Each d_i is a HP containing the dynamics from the above section (because the car should have the same physics and control available regardless of what segment it is in), but contains the extra evolution domain constraint $InBounds(i)$.

After the differential equation, each d_i includes a rather counterintuitive test that ensures that the car has reached an edge of segment i (i.e. either a racetrack boundary, the exit, or the entrance of the segment). This condition is disjuncted with $dx^2 + dy^2 \neq 1$ because any run of the HP where $dx^2 + dy^2 \neq 1$ must pass this test to allow the ovrall HP to exit and evidence the case where the

unit vector condition does not hold. However, the unit vector condition always should hold simply based on the dynamics, so we focus on the more interesting condition.

The "reached an edge" part of the test disallows any "partial" runs of the car through segment i, which is actually fine for the purposes of proving safety! if a run of d_i failed due to the test, that run ended in a safe state and is therefore irrelevant for further analysis. The boundary test essentially declares that we only care about what happens when (or if) the racecar reaches an edge of the segment, as at that point if it reaches a boundary it is considered to have crashed, whereas if it reaches the exit the race continues (and if the controller is well designed, it satisfies its exit contract).

If only $d_1...d_n$ were included, the safety property of the car staying in the track would be trivially provable via differential weakening. Therefore, our model also includes O, a version of the dynamics with evolution domain constraint $Evo(O) = (Evo(d_1) \cup Evo(d_2) \cup ... \cup Evo(d_n))^C \cup T$, i.e. a restriction that the car is outside the track or on its boundary T. T also does not include the exit of the last segment in order to disallow a car from violating the safety property after finishing a race.

3.4 Proof Strategy

With the structure of the model defined, one of the main remaining questions is how to break this proof down into manageable, modular pieces. It is here that the idea of contracts and modular segment proofs is most useful. The loop invariant is structured as a conjunction, including implications of the form

$$contract := \bigwedge_{i=1}^{n} Goal(i) \rightarrow ExitContract(i). \tag{7}$$

Additionally, it contains a conjunct of the following form:

$$bookmark := \bigvee_{i=1}^{n} Goal(i) \vee P, \tag{8}$$

Where P in this case refers to the premise set of the overall proof. The loop invariant also holds $dx^2 + dy^2 = 1$, but that is something proved by each modular proof (discussed below) and will be ignored in favor of the main safety discussion.

both *contract* and *bookmark* are true initially, as all $Goal(i)$ are false and P is true. Additionally, *bookmark* easily proves the safety condition, because every disjunct indicates that the car is within the racetrack boundaries. The *step* section of the loop invariant proof starts by applying $\vee L$ to *bookmark*. In each proof branch where a $Goal(i)$ is true, *contract* also produces the hypothesis $ExitContract(i)$. Here it is required that the prover produce a proof of $ExitContract(i) \rightarrow EntryContract(i+1)$; if this is not possible, then the racetrack is not provably safe! Similarly, P must imply the entry contract of segment 1 (if it does not, the racecar's initial position or direction is wrong in the problem setup). Finally, the new EntryContract hypothesis causes only a single branch of the control phase to pass, and a similarly only a single branch from the dynamics phase.

At this point, the proof state in each branch is (roughly) of the form

$$EntryContract(i) \rightarrow [d_i](loop\_invariant). \tag{9}$$

At this point, we invoke the modular proofs to be describe in the next section, which are of the form

$$EntryContract(i) \rightarrow [d_i](Goal(i) \quad \&\& \quad ExitContract(i)). \tag{10}$$

Using the monotonicity of $[\cdot]$, we can change the goal of proving the loop invariant to instead prove the result of the modular proof, at which point the modular proof is exactly what remains to be proven. This completes the proof structure section; as we have seen, the only new things to be proven for a new racetrack are the exit-entry contract implications, which are much simpler by design than proving a controller safe for a track segment of unknown shape and size.

4 Controller Proofs

Having described the general structure that allows for safety proofs about arbitrary racetracks under well-designed controllers with safety contracts, we now move on to describing several relatively simplistic examples of combinations of controller, track segment, and safety contract.

4.1 Basic Straight Segment Controller

The simplest controller for a straight segment is to do no turning or acceleration, i.e. $a = 0, s = 0$. This controller has an extremely restrictive entrance contract that enforces that the car's direction vector upon entering the segment is parallel to the direction of the segment. For the vertical straight segments we use, this condition is simply $dx = 0$, along with the condition that the racecar's position is somewhere at the entrance of the segment.

The constraints upon the entrance contract directly dictate the exit contract constraint; it is valid to have the position constraint give no other information about the car's position besides the fact that it is on the entrance line segment. In this case, the segment's exit contract must be similarly vague and indicate that the car will end on the segment exit with dx = 0. However it is equally valid to have an entrance contract specifying the exact position of entrance of the car, which would allow an exit contract with specificity as well.

4.2 Accelerating/Decelerating on a straight section

As a racecar driver, it is useful to be able to speed up on straight sections to make up for lost speed on turns. However, sometimes it is important to be able to slow down in order to take a turn safely. This controller can be adjusted to set a to any value between A and $-B$, and again requires and ensures that $dx = 0$ while setting $s = 0$. However, it is different from the Basic Straight Segment Controller in that the exit contract will have a modified velocity range. In particular, if the entry contract required $v_a \leq v \leq v_b$ and the segment has length h, The segment's exit contract will have $\sqrt{max(0, v_a^2 + 2 * a * h)} \leq v \leq \sqrt{max(0, v_a^2 + 2 * a * h)}$ as dictated by kinematics and the fact that braking down to $v = 0$ leaves the car stuck with 0 velocity

4.3 Slight turns on a straight section

If a car driver wants to move closer to the inner wall of a track, for example, they may wish to turn slightly even on a straight section. Although we have not implemented this controller, we can calculate the maximum turning speed usable on a straight section of height h with speed v and

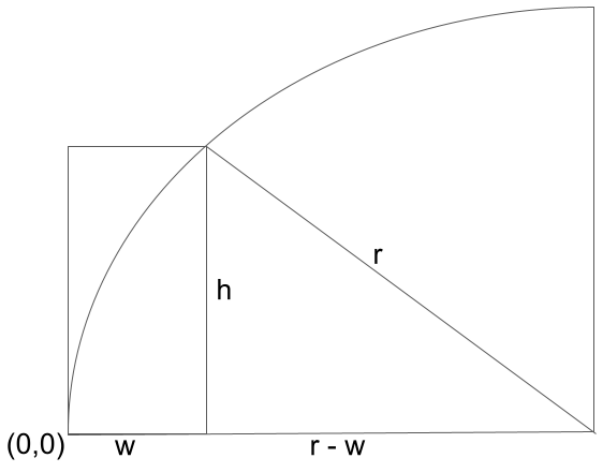

Figure 2: Slight turn Straight controller radius calculation

$a = 0$. For simplicity, we consider only slight right turns from (0,0) to reach the upper right corner of a rectangle of a rectangle at (w, h), with $0 < w < h$ for feasibility. We additionally $dx = 0, dy = 1$ initially, and note that as long as a solution exists (i.e. a reasonable entry contract is satisfied that does not allow a car to enter and immediately crash into the racetrack boundary), this simple case calculation is re-usable via a change of coordinates.

For any nonzero s, the radius r of the circle that the car follows obeys $r \cdot s = v$, a relationship that comes from the earlier derivation of s as a measure of "radians moved per unit of times" (see under Eqn 1), and the definition of r which can be interpreted as "distance units per radian". Therefore, the radius of the circle being followed obeys $r^2 = h^2 + (r - w)^2$ (see figure 2), and thus $r = \frac{h^2+w^2}{2w}$. This allows us to conclude that the maximum allowable s is

$$s_{max} = \frac{2vw}{h^2 + w^2} \tag{11}$$

.

Equation 11 allows us to write decent controllers without guesswork, because the equation for s holds for any distance w (not just for the width of the segment!). So, one simply must plug in the desired horizontal distance $w < h$ to travel to obtain a corresponding s to use in this straight section controller.

In order to complete this controller it is also important to calculate the resulting dx value for any given s in order to include this information in the segment's exit contract. Because $dx = 0$ at $t = 0$, the solution for $dx(t)$ is $dx(t) = \sin(s * t)$ The angle traversed from $(0, 0)$ to (w, h) is $\theta = \arcsin(\frac{h}{r})$ radians, and the time taken is $t = \theta/s$. Combining these facts leads to the interesting conclusion that the final value of dx is $\frac{h}{r}$.

4.4 re-orienting on a straight section

In addition to our slight-turn straight controller that allows cars to move laterally in straight sections, it is necessary to define a controller that takes a non-zero dx as part of its entry contract in order to allow the slight-turn controller to ever be useful. The idea of the re-orienting controller is to use some horizontal space to slowly reduce dx to 0 from some positive starting value. We begin by noting that the scenario is actually modeled by Figure 2, with the car starting at (w, h) and moving along the arc to $(0, 0)$.

Working backwards, this means the current value of dx is $\frac{h}{r}$, where h is fixed as the height of the segment. With $r = \frac{h}{dx}$, we can calculate w via the same identity of $h^2 + (r - w)^2 = r^2$, yielding $w^2 - 2rw + h^2 = 0$ and thus $w = r - \sqrt{r^2 - h^2}$. Finally, we can plug in the values of h, w, v into $s = -\frac{2vw}{h^2+w^2}$, at which point we have calculated the input s as well as the displacement w that informs this controller's exit contract. If the exit contract indicates an impossible new position due to w, the exit contract from the previous segment allowed too much turning to get the car back on track!

Note for graders: I also did not have time to implement this controller, but I effectively proved a version of this controller for my project proposal submission.

4.5 Curve controller

Finally, a controller is needed to successfully navigate arbitrary curve segments. This curve controller is as simple as possible; its entry contract is that dx, dy is parallel to the curve entrance with any position in the entrance being acceptable. The controller then exactly follows the curve contour by setting $s = \frac{v}{r}$ where r is the euclidean distance between the car and the center of the circle the curve segment is based on. The exit contract position can then be exactly calculated as the point on the circle being followed that intersects with the exit, and the exit contract dx, dy are simply the unit vector perpendicular to the exit in the direction of the circular motion.

It was mentioned in the accelerating straight segment controller that it might be necessary to slow down for turns, and mentioned in the introduction that a reasonable way to mitigate the downsides of this approach when modeling real racetracks is to cap the max turning speed s. Indeed, the best way to enforce turn safety within this framework is to simply require that each curve segment enforce a maximum velocity as part of its entry contract. Because any turn controller must necessarily turn more with higher velocity in order to follow the curve (this is seen very explicitly in this simple controller, with $s = \frac{v}{r}$), limiting the maximum velocity when entering a turn via entry contract is an effective way of capping maximum turning speed.

5 Future Work

In our proposal, we expected to complete far more work on this project, and highly underestimated the amount of effort many of the anticipated deliverables would require. In particular, we have listed the following items for potential future work, along with an explanation of how difficult they would be to complete and why.

5.1 Multi-Agent safety proofs

While it is interesting to build a framework for safety proofs on arbitrary tracks via modular set of track segments and associated controllers, it would be far more interesting to do so for races involving multiple cars at once. This endeavor seems extremely difficult with our current framework and may require a paradigm shift to complete. However, one way that I thought was plausible was to account for each of the n^k possible assignments of agent to track component with a set of dynamics enforcing that assignment as the evolution domain constraint. If each car was given a sub-track it was required to stick to when within 1 segment of another car, this approach would even allow safety proofs about control in the presence of multiple agents. However, its nearly prohibitively tedious generation and proofs may indicate that there is a superior method of modeling this situation that avoids the issue of event-space size explosion.

5.2 Calculate/Prove Controller Efficiency

In addition to safety, it would be interesting to prove bounds on the time different "cars" (i.e. different combinations of controllers used on the same track) took to complete a track. This idea is actually not difficult to implement in the current framework; bounds on the total time taken so far can be calculated and propagated via exit contracts, and the final property G could include the calculated result for the range of possible end times given that $Goal(n)$ was reached. The calculated result would then be easily provable by the exit contract of segment n.

5.3 Overlapping track segments

Some types of racetrack have bridges and tunnels. Allowing for this type of layering is also not extremely difficult (one might consider adding a simple counter of what segment a car is on, and not allow entering a new evolution domain if its associated segment number is not at most 1 from the current one). However, this idea is also less interesting than calculating controller efficiency, as most real-world race tracks are flat.

5.4 Track generator

Writing a script that allows the creation of entire racetracks (similar to [1]) and generates a KeymaeraX proof while automatically calculating reasonable exit and entrance contracts would be a key addition to this work. We wrote all our models and proofs by hand, and did not end up with time to automate the process. However, the formulaic nature of the models and proofs means this is likely very feasible.

6 References

1. Bohrer, B., Luo, A., An Chuang, X., and Platzer, A. (2018). CoasterX: A Case Study in Component-Driven Hybrid Systems Proof Automation. In International Federation of Automatic Control. 55–60.

2. Müller, A., Mitsch, S., and Platzer, A. (2015). Verified Traffic Networks: Component-based Verification of Cyber-Physical Flow Systems. In International Conference on Intelligent Transportation Systems.

3. N. Rajabli, F. Flammini, R. Nardone and V. Vittorini, "Software Verification and Validation of Safe Autonomous Cars: A Systematic Literature Review," in IEEE Access, vol. 9, pp. 4797-4819, 2021, doi: 10.1109/ACCESS.2020.3048047.

4. Kamali, M., Dennis, L., McAree, O., Fisher, M., and Veres, S., (2017). Formal verification of autonomous vehicle platooning. In Science of Computer Programming. Volume 148. 88–106.

5. Kalra, N., Paddock, S. (2016). Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?. Transportation Research Part A: Policy and Practice. volume 94. 182-193.

6. Platzer, A. (2019). The Logical Path to Autonomous Cyber-Physical Systems. QEST.

7. Fulton, N., Platzer, A. (2018). Safe Reinforcement Learning via Formal Methods: Toward Safe Control Through Proof and Learning. AAAI.

8. Campbell-Brennan, J. Racecar Vehicle Dynamics explained. URL https://www.racecar-engineering.com/tech-explained/racecar-vehicle-dynamics-explained/

9. Dubins, L.E. (1957). On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents. American Journal of Mathematics, 79, 497.

10. Xuan-Nam Bui, J. -. Boissonnat, P. Soueres and J. -. Laumond, "Shortest path synthesis for Dubins non-holonomic robot," Proceedings of the 1994 IEEE International Conference on Robotics and Automation, 1994, pp. 2-7 vol.1, doi: 10.1109/ROBOT.1994.351019.

11. Reeds, J.A., Shepp, L.A. (1990). OPTIMAL PATHS FOR A CAR THAT GOES BOTH FORWARDS AND BACKWARDS. Pacific Journal of Mathematics, 145, 367-393.

12. J. -. Boissonnat, A. Cerezo and J. Leblond, "Shortest paths of bounded curvature in the plane," Proceedings 1992 IEEE International Conference on Robotics and Automation, 1992, pp. 2315-2320 vol.3, doi: 10.1109/ROBOT.1992.220117.

13. Platzer, A. (2008). Differential Dynamic Logic for Hybrid Systems. Journal of Automated Reasoning. volume 41. 143–189.