Senior Design Spring Research Activities with Trajectory Optimization based on Signal Temporal Logic Specifications

Yann Gilpin
Department of Electrical Engineering
University of Notre Dame
Notre Dame, IN, USA
ygilpin@nd.edu

***Abstract*—This is a report specifying continued efforts to improve optimal trajectory synthesis based on signal temporal logic constraints. Simply put, Signal Temporal Logic offers a convenient and rigorous way of describing high level tasks, such as motion planning, for automatic solution synthesis. While there are known methods to solve this problem, the computations scale exponentially with respect to the number of time steps, and the number of constraints. A wide variety of practical systems have many constraints and run for extended periods of time making this otherwise promising approach unusable. In the Fall semester, I tried a several of attempts, but was most successful with defining a pointwise robustness measure. This enabled a novel genetic evolution based optimization technique whose success along with a few other promising approaches is measured and compared against the latest work in this field.**

I. Introduction

As autonomous systems become increasingly independent, there comes a time when the system must plan in order to achieve a high level specification. Historically this was done by hand, though it is tedious. Imagine having a robotic butler that must water a plant while the owner is on vacation. To program this manually someone must calculate the position of the robot at every point in time to avoid collisions. Unfortunately the class of systems with invertible dynamics for which this calculation is significantly simpler, is small. Once the general form of this problem is solved, it will save a significant amount of tedious effort for programmers and increase the range of tasks that can be automated. Notable beneficiaries include the areas of search and rescue, domestic aids and national defense.

II. Previous Approaches and Related Work

One promising approach is to define the problem in terms of Signal Temporal Logic (STL) specifications. The core idea is to gather the variables of interest into a signal space. Then constraints upon the signal space can be written in terms of Boolean operators, *AND,* $\wedge$, *OR,* $\vee$, *NOT,* $\neg$ and temporal operators *ALWAYS,* $\mathbf{G}$, *EVENTUALLY,* $\mathbf{F}$, *UNTIL,* $\mathbf{U}$. This allows many practical tasks to be encoded rigorously in addition to control or dynamics limitations. For example, if a robot was tasked to water the plants once while the owner was on a short vacation, one could write a simple specification: *by time one week, water the plant beside the window using a feasible motion while ALWAYS avoiding the furniture.* A Gazebo simulation of a simplified version of this problem is available in Fig 1.

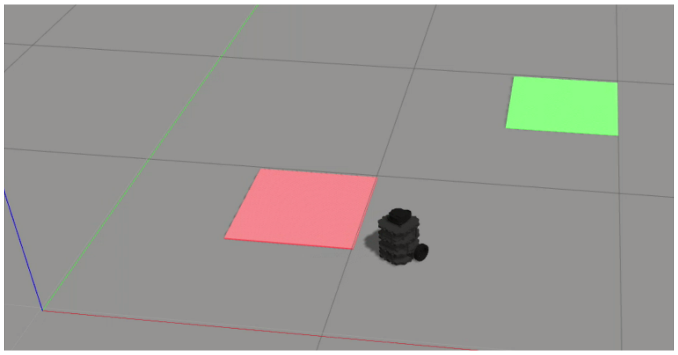


Fig. 1: A TurtleBot is on its way to water the plant (green) while avoiding the furniture (red). The solution being executed was automatically generated, see Section X for further details.

While it is easy to evaluate if a potential solution satisfies all STL predicates, a binary satisfied or unsatisfied result does not yield much information on how well a solution satisfies or how badly it violates the constraints. Hence the robust semantics of STL were developed. For a candidate trajectory $\boldsymbol{y}(t)$, predicates φ_1 and φ_2, the robustness, ρ, can be recursively defined as written formally below: 1

Definition 1 (Traditional STL Robust Semantics):

- $\boldsymbol{y} \models \varphi \iff \rho^{\varphi}((\boldsymbol{y},0)) > 0$
- $\rho^{\pi}((\boldsymbol{y},t)) = \mu^{\pi}(\boldsymbol{y}_t) - c$
- $\rho^{\neg\varphi}((\boldsymbol{y},t)) = -\rho^{\varphi}((\boldsymbol{y},t))$
- $\rho^{\varphi_1\wedge\varphi_2}((\boldsymbol{y},t)) = \min\left(\rho^{\varphi_1}((\boldsymbol{y},t)), \rho^{\varphi_2}((\boldsymbol{y},t))\right)$
- $\rho^{\mathbf{F}_{[t_1,t_2]}\varphi}((\boldsymbol{y},t)) = \max_{t'\in[t+t_1,t+t_2]}\left(\rho^{\varphi}((\boldsymbol{y},t'))\right)$
- $\rho^{\mathbf{G}_{[t_1,t_2]}\varphi}((\boldsymbol{y},t)) = \min_{t'\in[t+t_1,t+t_2]}\left(\rho^{\varphi}((\boldsymbol{y},t'))\right)$
- $\rho^{\varphi_1\mathbf{U}_{[t_1,t_2]}\varphi_2}((\boldsymbol{y},t)) = \max_{t'\in[t+t_1,t+t_2]}\Bigg($

$$\min\left(\left[\rho^{\varphi_1}((\boldsymbol{y},t')), \min_{t''\in[t+t1,t']}\left(\rho^{\varphi_2}((\boldsymbol{y},t''))\right)\right]^T\right)\Bigg).$$

For more information about the semantics and formal notation used with traditional robustness see [3]. This definition has successfully formed the basis for trajectory optimization (i.e. finding the most robust solution) using an approach called Mixed Integer Linear Programming [1]. Unfortunately, MILP struggles due to exponential complexity with respect to the length of the horizon and the number of predicates.

As novel robustness measures are proposed there are some important formal properties to keep in mind. The first is *soundness*, which means that if a candidate trajectory has a positive robustness then it satisfies the specification. For some measures the converse is also true i.e. if the robustness is negative the trajectory does not satisfy the specification. The second property is *completeness*. In this context it means that if there is a solution the robustness measure will not obfuscate it. That is to say, when using a thorough optimization technique, in conjunction with a complete robustness measure, the globally optimal solution will be found. Furthermore, some measures have adjustable parameters, taking the limit of which, often yields the traditional $\max$ and $\min$. These are termed *asymptotically complete*, as there is no approximation error to conceal a solution in the limit.

Recent robustness measures makes progress against the non-smoothness of the traditional definition by replacing the $\min$ and $\max$ operators with smooth approximations. An example of such an approximation appeared in [8] and is the log exponential sum (LSE):

$$\widetilde{max}([\alpha, \beta, \gamma \dots]^T) \approx \frac{1}{k_1} \log(e^{k_1 \alpha} + e^{k_1 \beta} + e^{k_1 \gamma} + \dots) \quad (1)$$

Where α, β, γ would be robustness of a particular predicate at different points in time and $k_1 > 0$ is a free parameter that affects the accuracy of the approximation. Specifically the error of the approximation is limited by:

$$\left| \widetilde{max}([\alpha, \beta, \dots]^T) - \frac{log(e^{k_1 \alpha} + e^{k_1 \beta} + \dots)}{k_1} \right| \leq \frac{logN}{k_1} \quad (2)$$

Where N is the number of arguments. While this approximation opens the door to the use of much faster gradient based optimization techniques and nonlinear predicates, it presents numerical stability problems, due to the use of exponentials, and may not preserve the sign of the arguments. Hence this robustness measure is not sound. Lastly, since the approximation is bounded it is asymptotically complete. That is to say as k_1 grows large $\widetilde{\max} = \max$.

Similarly, the authors of [7] use a combination of arithmetic and geometric means to approximate the $\min$ and $\max$ operators. In doing so the authors avoid the numerical stability concerns associated with (1). This measure also preserves the sign of the arguments so it is sound. Additionally the approximation error is bounded so the method is complete. Unfortunately due to the switching between the arithmetic and geometric means, there are a finite number of discontinuities. That being said there are far fewer discontinuities than with $\max$ and $\min$ such that often, in practice, gradient based can be used. Lastly for the arithmetic and geometric means to work, the signals and predicates must be normalized, which is an additional step not required by other approaches.

III. Summary of Fall semester Activities

In addition to the developments in the literature, I made some progress in the Fall semester as well. My focus was initially to familiarize myself with STL and the aforementioned approaches so I could develop a novel approach. My main idea is to try to break the problem down in to manageable chunks, points in this case, to accelerate optimization. A necessary element to this line of thinking is a pointwise robustness measure, ρ_{pw}, the semantics of which are reproduced here for convenience.

Definition 2 (Pointwise STL Robust Semantics):
For a particular point in time t_k:

- $\boldsymbol{y} \vDash \varphi \iff \rho^{\varphi}_{pw}((\boldsymbol{y}, 0)) > 0$
- $\rho^{\pi}_{pw}((\boldsymbol{y}, t_k)) = \mu^{\pi}(\boldsymbol{y}_{t_k}) - c$
- $\rho^{\neg\varphi}_{pw}((\boldsymbol{y}, t_k)) = -\rho^{\varphi}_{pw}((\boldsymbol{y}, t_k))$
- $\rho^{\varphi_1 \wedge \varphi_2}_{pw}((\boldsymbol{y}, t_k)) = \min\left(\rho^{\varphi_1}_{pw}((\boldsymbol{y}, t_k)), \rho^{\varphi_2}_{pw}((\boldsymbol{y}, t_k))\right)$
- If $\exists t' \in [t + t_1, t + t_2] | t' \leq t_k, \rho^{\varphi}_{pw}((\boldsymbol{y}, t')) > 0$
  then $\rho^{\mathbf{F}_{[t_1, t_2]}\varphi}_{pw}((\boldsymbol{y}, t_k)) = \frac{|1 + t_k - t_2|}{|\rho^{\varphi}_{pw}((\boldsymbol{y}, t_k))| + \epsilon}$
  else $\rho^{\mathbf{F}_{[t_1, t_2]}\varphi}_{pw}((\boldsymbol{y}, t_k)) = \frac{-|\rho^{\varphi}_{pw}((\boldsymbol{y} t_k))|}{|1 + t_k - t_2| + \epsilon}$
- $\rho^{\mathbf{G}_{[t_1, t_2]}\varphi}_{pw}((\boldsymbol{y}, t_k)) = \min_{t_k \in [t + t_1, t + t_2]}\left(\rho^{\varphi}_{pw}((\boldsymbol{y}, t_k))\right)$

Where $\epsilon > 0$ is small constant to prevent division by zero. The only major departure from Definition 1 is in the eventually operator. The thinking behind it is that of constant speed. If the eventually statement is not reached within the time limit, each point is scored based on how far it is from the goal and how much time is remaining to satisfy the predicate. The further away and the less time left (before t_2) the poorer its robustness. Similarly when the operator is satisfied the points are scored to reward arriving earlier. A more detailed explanation is available in [4]. This definition enabled the development of a variety of custom optimization techniques. The main disadvantage of this robustness measure is that it is non-smooth due to the definition of the *EVENTUALLY* operator in addition to those caused by the use of $\max$ and $\min$. On the bright side it is sound and complete.

One more note about pointwise robustness. Sometimes it is convenient to have a single number to represent the robustness of entire trajectory. This is easy to obtain from the robustness trajectory. Since a satisfactory trajectory requires satisfactory robustness at all points I applied the $\min$ operator to the set of each point's robustness to obtain a robustness the entire trajectory. This worst case thinking comes directly from the traditional robustness definition.

The first somewhat successful optimizer based on pointwise robustness is called, Worst Point First (WPF). It starts by simply generating a population of potential solutions. Next picking the best one based on the overall robustness, it identifies the weakest point. By stocastically proposing additive adjustments to this point, it is able to improve the robustness of the entire trajectory. Once the point is replaced with a better one, the process repeats by once again finding the worst point (within the same candidate) and trying to improve it until a the iteration limit or robustness level is reached. While in some cases this approach was able to find a satisfactory solution, this was not always the case, due to the non-convexity of the cost function, i.e. it would get stuck with a sub-optimal result.

The second promising idea was a genetic evolution based algorithm. I chose each point as "gene" with an infinite number of "alleles" (signal values). The mutation rate is a function of the robustness, with less robust points more likely to receive a stronger change. Conversely good points are less likely to be adjusted significantly. Similarly, the cross-over function favors transferring more robust points to children. Parents are picked based on their overall robustness such that only the best trajectories in a population reproduce. In this way the ideas behind the algorithm originally designed for black-box (single valued) cost function optimization can take advantage of the information provided by the pointwise robustness.

The other development of note, was a connection with traditional optimal path planning from the optimal control framework. These approaches in discrete-time generally minimize an additive cost function J of the form:

$$J = \sum_{i=1}^{N} L(x_i, u_i) \tag{3}$$

Where N is the number of time steps in the finite horizon and x_i and u_i are the current position and control inputs respectively and L is a real valued scalar function [2]. This has not been applied to STL based trajectory optimization, because optimizing over the traditional robustness is not additive.

With pointwise robustness however, these approaches appear viable. Thinking back to the smooth approximation of the $\max$ function (1), and noticing that $log(x)$ is convex, I realized that:

$$argmax(log(e^{k_1\alpha} + e^{k_1\beta}...)) = argmax(e^{k_1\alpha} + e^{k_1\beta}...) \tag{4}$$

Which led me to try the following additive, cost function:

$$J = \sum_{i=1}^{N} e^{-k_1 p_n} \tag{5}$$

Where k_1 is still an adjustable parameter as in (1) and p_n is the robustness of a particular point in time n. The negative in the exponent is so that this can be used for minimization. There is no control dependence for simplicity. To find the optimal solution in this framework, I tried single shooting [2], which is to use a generic optimizer on the cost function, (5) in my case. I used a custom black-box differential evolution optimizer. Though slow, it converged nonlinearly.

Based on this progress, I proposed the plan for the spring semester in Table I.

IV. Summary of Spring Semester Activities

The spring semester (and the rest of this paper) largely followed the plan laid out in Table I with one major exception. The first portion of the semester was dedicated to writing a proposal to IEEE Control System Society Letters.

TABLE I: Spring Semester Proposed Schedule

Month	Activity
January	Implement and verify convergence of genetic evolution algorithm.
February	Implement and verify direct collocation approach
March	Implement and Compare MILP, Log/Exp, and Arithmetic-Geometric approaches
April-May	Run most promising algorithm on a Turtlebot and demonstrate results

V. Control Systems Society Letters Proposal

Working with Dr. Lin and Vince Kurtz we put together a proposal on a very similar research topic [5]. The main idea behind the paper was to combine $\min$ and $\max$ approximations to obtain a result that was sound and smooth. Recalling, that the main draw back of using the LSE (1) approximation is that it does not preserve the sign of the arguments. That is to say it is an over approximation. Thankfully there are some smooth $\max$ approximations that are under approximations. The one we used is the exponential fraction (EF):

$$\widehat{max}([x_1, x_2 \ldots, x_N]^T) = \frac{\sum_{i=1}^{N} x_i e^{k_2 x_i}}{\sum_{i=1}^{N} e^{k_2 x_i}} \tag{6}$$

Where $k_2 > 0$ is an adjustable parameter. Notice as $\lim_{k_2\to\infty} \widehat{max} = \max$. By combining LSE and this EF for $\min$ and $\max$ carefully, we were able to define a robustness measure that was smooth everywhere, sound, and asymptotically complete. The formalized definition is reproduced next for convenience, though more details including proofs are available in [5].

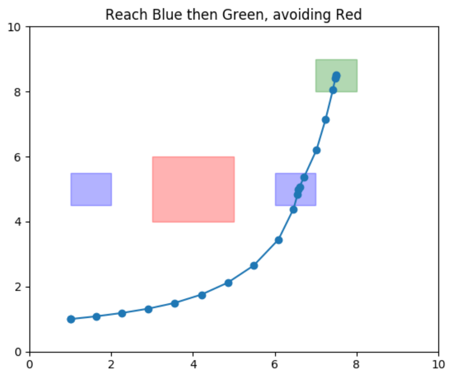
Definition 3 (Smooth STL Robust Semantics):

- $\tilde{\rho}^{\varphi}((\boldsymbol{y}, 0)) > 0 \implies \boldsymbol{y} \vDash \varphi$
- $\tilde{\rho}^{\pi}((\boldsymbol{y}, t)) = \mu^{\pi}(\boldsymbol{y}(t)) - c$
- $\tilde{\rho}^{\neg\pi}((\boldsymbol{y}, t)) = -\tilde{\rho}^{\pi}((\boldsymbol{y}, t))$
- $\tilde{\rho}^{\varphi_1 \vee \varphi_2}((\boldsymbol{y}, t)) = \widehat{\max}\big(\tilde{\rho}^{\varphi_1}((\boldsymbol{y}, t)), \tilde{\rho}^{\varphi_2}((\boldsymbol{y}, t))\big)$
- $\tilde{\rho}^{\varphi_1 \wedge \varphi_2}((\boldsymbol{y}, t)) = \widetilde{\min}\big(\tilde{\rho}^{\varphi_1}((\boldsymbol{y}, t)), \tilde{\rho}^{\varphi_2}((\boldsymbol{y}, t))\big)$
- $\tilde{\rho}^{\mathbf{F}_{[t_1,t_2]}\varphi}((\boldsymbol{y}, t)) = \widehat{\max}_{t' \in [t+t_1, t+t_2]}\big(\tilde{\rho}^{\varphi}((\boldsymbol{y}, t'))\big)$
- $\tilde{\rho}^{\mathbf{G}_{[t_1,t_2]}\varphi}((\boldsymbol{y}, t)) = \widetilde{\min}_{t' \in [t+t_1, t+t_2]}\big(\tilde{\rho}^{\varphi}((\boldsymbol{y}, t'))\big)$
- $\tilde{\rho}^{\varphi_1 \mathbf{U}_{[t_1,t_2]}\varphi_2}((\boldsymbol{y}, t)) = \widehat{\max}_{t' \in [t+t_1, t+t_2]}\Bigg(\widetilde{\min}\Big(\Big[\tilde{\rho}^{\varphi_1}((\boldsymbol{y}, t')), \widetilde{\min}_{t'' \in [t+t1, t']}\big(\tilde{\rho}^{\varphi_2}((\boldsymbol{y}, t''))\big)\Big]^T\Big)\Bigg).$

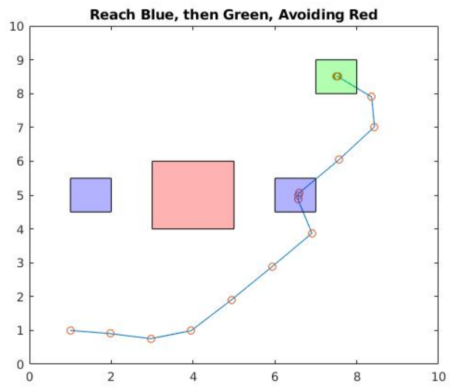
Where $\widetilde{min}$ is the corollary of the LSE $\widehat{max}$ (1). Additionally we found that by setting k_1 and k_2 small an averaging effect was produced similar to the desired results of the AGM robustness measure. For example, see Fig.2

The corollary to this property is by setting k_1, k_2 large, the approximation error reduces (to zero in the limit) proving that the measure is asymptotically complete. This feature is important in practice as the approximation error can mask solutions. See Fig. 3 for a visual representation.

With the success of the smooth approximations replacing $\max$ and $\min$ (henceforth the EF method), there has been

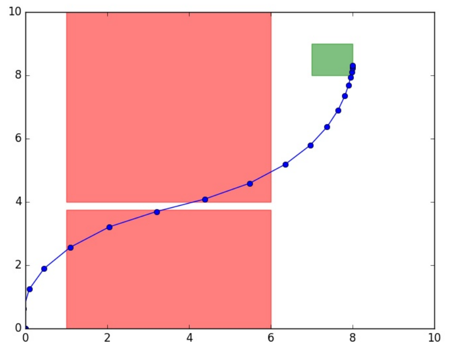


(a) Locally optimal trajectory with respect to the smooth robustness (Def. 3) for relatively low parameter values ($k_1 = k_2 = 2$). The robot takes several extra steps in the blue target and green goal regions, suggesting greater robustness to external disturbances.

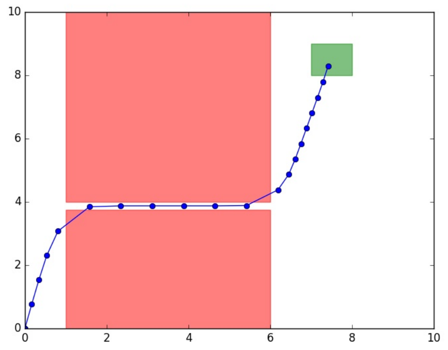


(b) Locally optimal trajectory with respect to AGM robustness [7]. Note that the robot spends several extra timesteps in the blue target and green goal regions, suggesting greater robustness to external disturbances.

Fig. 2: Comparison of our approach (2a) with AGM robustness (2b). Starting at $(1, 1)$, a robot must avoid the red obstacle, visit one of two blue target regions, and eventually arrive at the green goal region. By choosing low values for the parameters k_1, k_2, our proposed smooth robustness similarly demonstrates the desired conservativeness of AGM robustness.



(a) Locally optimal trajectory with respect to our smooth robustness (Def. 3) with low parameter values ($k_1 = k_2 = 1$).



(b) Locally optimal trajectory with respect to our smooth robustness (Def. 3) for relatively high parameter values ($k_1 = k_2 = 10$).

Fig. 3: Comparison of low parameter values 3a to high parameter values 3b. Starting at $(0, 0)$, a robot must avoid the red obstacles, passing through a narrow tunnel before reaching the green goal region. By choosing high values for the parameters k_1, k_2, our proposed smooth robustness can demonstrate asymptotic completeness.

.

significant progress in mitigating the effects of the non-smoothness of the traditional robustness. That being said there is still some remaining challenges. The most inhibiting is the non-convexity. Simply put, in most practical scenarios there are multiple ways to satisfy a particular specification. Unfortunately gradient based approaches can only guarantee finding a local optimum. Hence the rest of my work for the semester was built around dealing with the non-convexity.

A. Smooth Approximations and Pointwise Robustness

Recalling that the definition of pointwise robustness relies upon $\min$ and $\max$ operators (see 2) I tried the EF method (from the proposal). I found that this largely helped so unless otherwise stated, I used it henceforth.

Though these smooth approximations greatly reduce the number of discontinuities present in the definition of pointwise robustness they do not eliminate all of them. The remaining source is in the definition of the pointwise robustness of the *EVENTUALLY* operator. Specifically when a potential solution crosses over from unsatisfactory to satisfactory, the sign and the fraction flip in a non-smooth way. Unsurprisingly the effects of this appeared in a few optimization attempts as is discussed in section VII.

Lastly, smooth approximations make no constraint on the type of signal a predicate can constrain (as long as it is also smooth). For example obstacles, targets, dynamics and control limits are all easily specified using the STL framework, independent of robustness measure used.

VI. Worst Point First with Automatic Random Restart

Based on the success of the worst point first algorithm last semester, I decided to try to improve it by including a random restart. I hoped that this would help get around the local optima on the way to the global one. Despite using large populations to help find a choice candidate to improve this approach still got stuck too often to be practical. I tried to further improve it by following its best result with a gradient based optimizer. I used a generic Sequential Least Squares Quadratic Programming function (SLSQP) for the gradient based optimizer yet it still returned a non-satisfactory solution too often to be useful.

From this experiment, I determined that the WPF approach is able to tweak a trajectory to improve it, however it is not able to perform global optimization on its own. In light of this, I used this method in my hybridized genetic algorithm as explained in section VIII.

VII. Direct Collocation

In the previous semester I was able to optimize a trajectory in a very similar way to the signal-shooting method from optimal control literature. To see if other successful approaches from this area could similarly be adapted, I was planning to attempt direct collocation. Direct Collocation works by discretization of the trajectory and enforcing the system and state dynamics constraints at each collocation point [6]. However for this approach to be successful one, I realized that would need an optimization algorithm that can handle the non-smooth nature of pointwise robustness (due to the eventually definition) in addition to algebraic constraints. As this is non-trivial to find or implement, given the time constraints, I decided to forgo further investigation in this area.

VIII. Hybrid Genetic

Unlike the previous two attempts, I was able to achieve some positive results with a custom hybrid genetic algorithm. Before delving into the details it is useful to review the classical genetic algorithm.

A. Classical Genetic Optimization

The algorithm starts with an initial population of binary strings that are usually randomly generated. The quality of each candidate is then measured. Next the intermediate population is generated by randomly copying a parent, in such a way, that favors fitter candidates. Then each candidate in the initial population is paired, and crossed over, before being mutated. After mutation, the modified intermediate population is merged with the initial population (replacing the less fit candidates). The cross over function randomly selects a point along two candidates which splits each trajectory into two pieces. Two full trajectories are made by swapping and re-linking the partial trajectories. The mutation operation randomly flips a bit in a string and usually has very low probability. For more information on classical genetic optimization see [9].

B. Custom Genetic Optimization

I started with the algorithm that I adapted last semester which was designed to specifically take advantage of the robustness trajectory provided by pointwise robustness and to work with continuously valued vectors instead of binary strings. This algorithm converged too quickly yielding sub-optimal results. I attributed this to favoring higher robustness at every step which limited the searching ability of this approach. In an effort to better search the space, I changed the way children are generated. Specifically, I found making the children a random linear combination of the two parents worked well [10]. By producing two children for each pair of parents, I was able to bias the system to produce one child like parent 1 and another like parent 2. Before entering the population each child undergoes some slight mutations. The parents are picked randomly from top 10 solutions as necessary to replace the bottom 1/3 of the total population with their children.

To further improve the speed and converge of this approach, I added a training step to each iteration. Specifically each member of the population is improved through worst point first tweaking except the best solution which is left untouched. After this phase finishes the process repeats, going back to the parent selection step. The cycle ends upon reaching a maximum number of iterations, or after being unable to improve the best candidate for three iterations. Thinking about the hybridization of WPF and genetic, I found that the genetic aspect helped get around the many local minimums that limited the WPF only approach. Secondly the WPF helped add the innovation that would otherwise have to come from mutation and cross over in the purely genetic approach. Together I believe these algorithms compliment each other well.

That being said the hybrid method has a hard time converging to the optimal solution. That is to say, it gets within the vicinity, but tends not to set all the vector elements just right. Consequently, I follow the best hybrid genetic result with further improvement by using SLSQP. The combined result of all of these methods and ideas is able to find consistently find satisfactory solutions to a variety of problems. While the parameters are problem dependent I found that $k_1 = k_2 = 25$ worked well for the genetic portion, while $k_1 = k_2 = 10$ was better the SLSQP portion. This discrepancy is due to numerical overflow in the implementation. These values should be adjusted to match the type of results and the nature of the problem of interest.

IX. Scalability Tests

Throughout this paper a range of methodologies have been described. To see how well the hybrid genetic (GEN), Log Sum Exponential (LSE) (1), and Exponential Fraction (EF) (6) methods compare I devised a two part scalability test with increasing amounts of non-convexity.

To ensure that all the tests had equal starting points, they all had access to the same initial guess method. Since the dynamics where those of a Turtlebot Burger, after the zero initial conditions, I guessed a constant speed $0.20\frac{m}{s}$

(below top speed) with uniformly distributed random turning, $\omega \in [-0.1, 0.1]\frac{rad}{s}$ (also within physical limitations). The sampling frequency is 1 sample/sec. Note that by choosing a random initial guessing procedure the results are random. To help reduce the impact of this randomness, each trial was conducted three times and the results averaged. Each test was performed on a Dell XPS 13 with a Intel i5 processor; no other programs were running. Similarly, all the algorithms used the same SLSQP function for the smooth optimization portion with $k_1 = k_2 = 10$. The only exception to these parameter values was the hybrid genetic during the genetic-WPF portion which used $k_1 = k_2 = 25$. I selected somewhat arbitrary high k's so that the algorithms would be able to find a solution even as the approximation error increased with the number of time steps and predicates. The genetic algorithm used even higher parameter values because it could do so without running into frequent numerical overflow. That being said overflows did not appear to cause any issues, as long as they were not too common. Perhaps this is because as the candidate solution improves they become much less likely, thereby not upsetting the final result.

A. *Horizon Length Test*

One challenge of applying the STL frameowrk in practice is the potentially long horizon's needed to accomplish a particular task. The more points, N, in the plan, the more difficult it is to find a satisfactory solution. To measure the performance in this area I used the specification depicted in Fig 4 while varying the number of points in the horizon. The robot must travel from the starting position (0,0) to the goal (green) while avoiding obstacles (red) within the finite horizon allowed.

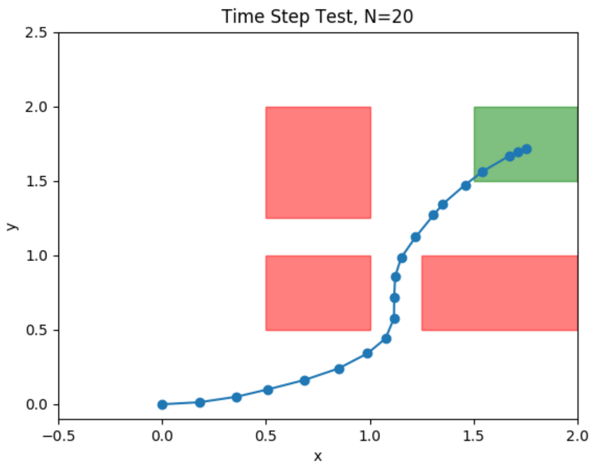


Fig. 4: Example of the Horizon length test with 20 timesteps. The robot must travel from the starting position (0,0) to the goal (green) while avoiding obstacles (red) within the 20 seconds allowed. The particular solution depicted was determined using the hybrid genetic algorithm and has a traditional robustness of 0.034

B. *Horizon Length Test Results and Discussion*

The average convergence time and traditional robustness values for several different time lengths, N, are shown in Table II. Where the over-bar represents sample mean. First and

TABLE II: Horizon Length Results

	EF		LSE		GEN	
N	$\overline{Time(s)}$	$\overline{\rho}$	$\overline{Time(s)}$	$\overline{\rho}$	$\overline{Time(s)}$	$\overline{\rho}$
16	34	-0.029	42	-0.009	571	0.009
18	54	-0.014	49	-0.057	1227	0.016
20	85	-0.034	58	-0.068	982	0.025

foremost it is clear that the hybrid genetic algorithm yields solutions that are more robust than the other two. I believe this to be due to the genetic algorithm's ability to search the solution space, a strength of this heuristic. Additionally the robustness improves as the length of the horizon is extended. This is expected of the optimal solution as the robustness against the speed limit is increased.

In looking at the optimization times we can see that the EF and LSE approaches increase linearly with the length of the finite horizon. The hybrid genetic algorithm does not show a clear increase though this is likely to due to insufficient trials.

Perhaps most strikingly the genetic algorithm is significantly slower than the other two. There are a two main reasons for this, which are both consequences of the implementation. This method must manipulate large arrays of potential candidates. In the Python implementation used here, it is difficult to control whether or not new memory is allocated or if values are simply overwritten. The fastest option is to allocate all the needed memory at the beginning and simply read and write from it. An efficient C++ implementation could significantly improve this aspect. Moreover this Python implementation was written to be as flexible as possible and as such computes the robustness of each point one at a time. While this is fine for the traditional robustness, it increases the order of the computational complexity for pointwise eventually calculations. Specifically the pointwise eventually must evaluate the trajectory to see if at some point the predicate is satisfied. This computation only needs to be done once per predicate per trajectory, yet to maintain flexibility, the calculation of the robustness of each point is done separately thereby repeating the search for satisfaction unnecessarily. By addressing these inefficiencies, the speed of the genetic algorithm can be greatly improved. Unfortunately, I did not have time to implement these improvements. More importantly, however, the optimization time from N=16 to N=20 less than doubled for the hybrid genetic algorithm whereas the EF technique more than doubled. This suggests that with efforts focused on efficient implementation of this algorithm would be scalable.

Overall, this experiment showed that the hybrid genetic algorithm often yields more robust solutions than EF and LSE. Additionally it has comparable scalability.

C. *Predicate Disjunction Test*

Another challenge of applying the STL framework in practice is the many different ways a particular task may be accomplished. Hence, the idea behind this experiment is vary the number of available options. Specifically the TurtleBot Burger must travel to the goal (green) while avoiding the obstacles (red). Along the way the TurtleBot must visit a charging station (blue) before reaching time 10. See Fig. 5 for a visualization.

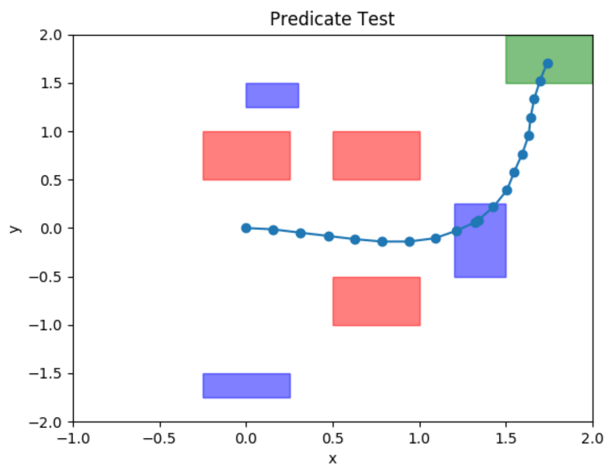


Fig. 5: Example of the Predicate test. The robot must travel from the initial condition (0,0) to the goal (green) while avoiding the obstacles (red). Along the way by time step 10 the robot must also visit a charging station (blue). In this case there are three charging stations available. The example solution was generated using the hybrid genetic approach and yielded a traditional robustness of 0.027

D. Predicate Disjunction Test Results and Discussion

The results of this experiment are tabulated in Table III where P is the number of available charging stations. Once

TABLE III: Predicate Disjunction Results

	EF		LSE		GEN	
P	$\overline{Time(s)}$	$\overline{\rho}$	$\overline{Time(s)}$	$\overline{\rho}$	$\overline{Time(s)}$	$\overline{\rho}$
1	110	-0.050	131	-0.063	1654	-0.002
2	75	-0.040	69	-0.025	1357	0.009
3	117	-0.070	108	-0.045	2287	-0.011

again the over-bar represents the sample mean. Similarly to the previous experiment the genetic algorithm yields results that are more robust than the other two, while taking significantly longer. The timing of the genetic algorithm still grows slowly with P suggesting that an efficient implementation may scale well. Looking at the computational time of EF and LSE methods, it is clear that they do not grow linearly. This implies that these approaches do not really search the space, but merely improve the initial guess to the nearest local optimum.

Most strikingly none of the algorithms produced on average satisfactory solutions. In addition to reflecting the difficulty of this problem, it shows the importance of picking a good initial guessing scheme. If I were to repeat this experiment I would reduce the variance of the steering angular velocity and possibly reduce the fixed speed of the initial guess generating method.

X. Simulation

While the genetic hybrid algorithm was not implemented as efficiently as possible, it does often yield satisfactory solutions despite working with non-linear system dynamics. As such I used it to generate a solution to a simple situation depicted in Fig. 6. The robot must travel to a plant in order to water it while avoiding the obstacle. Using an open loop feed forward controller, a TurtleBot Burger in Gazebo Simulation followed this path quite closely, though a closed loop controller would have produced even better results. A screenshot from this is printed in Fig. 1 and the full video is available upon request to the author.

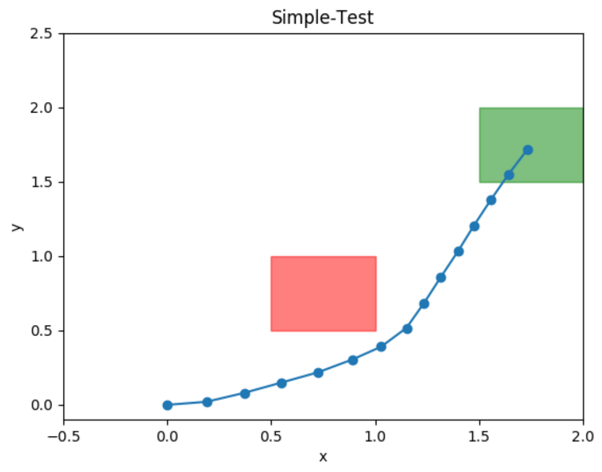


Fig. 6: Example of a simple situation in which a TurtleBot Burger must must travel from the initial condition (0,0) to the plant (green) while avoiding the obstacle (red). The solution was generated using the hybrid genetic approach and yielded a traditional robustness of 0.029

XI. Conclusion and Opportunities for Future Work

The accomplishments of this semester can be summarized into two categories of advancements in planning based on STL specifications. First with the help of Vince Kurtz and Dr. Lin, in writing the proposal for the Control Systems Society Letters, I helped reduce the non-smoothness of the problem through the judicious use of smooth approximations, forming the EF approach. These approximations yielded a robustness measure that is sound and differentiable everywhere. Additionally it combines some of the advantages of previous work namely the averaging effect of the AGM measure and the asymptotic completeness of the LSE approach.

In the second part of the semester I made some headway on the non-convexity aspect of the problem with the hybrid genetic optimization algorithm. Specifically, based on the preliminary results measured here, this algorithm on average produced better solutions and appeared to grow in computation time more slowly than the LSE and EF approaches. Recalling that the LSE and EF approaches are two of the best performing in recent work this suggests that a genetic style heuristic has potential to make measurable progress on this problem. Despite the promising metrics of my genetic hybrid algorithm more implementation work needs to be done to minimize the computational complexity before being put into practice.

Looking forward, it would be very interesting to see if a smooth definition of pointwise robustness could be developed. If so it would open the door to further application of optimal control trajectory synthesis within the STL framework. Additionally such a development has the possibly of further accelerating some of the ideas presented here.

Acknowledgment

Thank you Dr. Lin and Vince Kurtz for their guidance in selecting this project, help co-authoring the Control Systems Society Letters proposal, and support throughout the project.

References

[1] Calin Belta and Sadra Sadraddini. Formal methods for control synthesis: An optimization perspective. *Annual Review of Control, Robotics, and Autonomous Systems*, 2:115–140, 2019.

[2] Moritz Diehl, Hans Georg Bock, Holger Diedam, and P-B Wieber. Fast direct multiple shooting algorithms for optimal robot control. In *Fast motions in biomechanics and robotics*, pages 65–93. Springer, 2006.

[3] Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 92–106. Springer, 2010.

[4] Yann Gilpin. Senior design fall research activities with trajectory optimization based on signal temporal logic specification. University of Notre Dame, 2019.

[5] Yann Gilpin, Vincent Kurtz, and Hai Lin. A smooth robustness measure of signal temporal logic for symbolic control. IEEE, 2020.

[6] Matthew Kelly. An introduction to trajectory optimization: How to do your own direct collocation. *SIAM Review*, 59(4):849–904, 2017.

[7] Noushin Mehdipour, Cristian-Ioan Vasile, and Calin Belta. Arithmetic-geometric mean robustness for control from signal temporal logic specifications. In *2019 American Control Conference (ACC)*, pages 1690–1695. IEEE, 2019.

[8] Yash Vardhan Pant, Houssam Abbas, and Rahul Mangharam. Smooth operator: Control using the smooth robustness of temporal logic. In *2017 IEEE Conference on Control Technology and Applications (CCTA)*, pages 1235–1240. IEEE, 2017.

[9] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

[10] J. Wu. Real-coded genetic algorithm-based particle swarm optimization method for solving unconstrained optimization problems. In *2010 International Conference on Electronics and Information Engineering*, volume 1, pages V1–194–V1–198, 2010.