Book Report on
Synthesis for Human-In-The-Loop Control Systems

I G Prasad 18111025

Prafulla Saxena 18111268

Vasu Bansal 160776

May 27, 2019

Contents

1 Introduction

Human in the loop systems are the models in which the autonomous system interacts with one or more human operators. In such models human operator is considered to be the part of the system and action taken by human operator and by the autonomous system together constitute the entire system. In fully autonomous system we may not mine all possible environment assumption always, which can be cause catastrophic failure in safety critical applications, and may cost even human lives. Involvement of Human operator in such systems is necessary. For example a human driver interacts with automated driver assistant automobile. But the longer interaction time with system may again increase the chances of failure. These challenges focuses the community to work on synthesizing a semi-autonomous controller with limited human intervention. Because of the risk associated with semi-autonomous systems, formally verifying the safety of such systems is critical. In this report we describe one such synthesis method [1] which guarantees the safety of semi-autonomous controller.

2 Problem Description

Many safety critical systems interact with human operators and hence action taken by the human operator is very important in such systems. Several studies show that inappropriate human actions are often the reasons for failure of such systems. One option to avoid such failures is completely removal of human interaction and make the system fully autonomous. However fully autonomous systems are not always feasible as making too many assumptions about the environment might abstract away from the physical limitations of the system such as sensor capabilities and unexpected environmental change. And without any proper assumptions, the environment behaviours as captured by high level mathematical specifications will make the synthesis algorithm to conclude that no controller exists which satisfies all the assumptions of the environment and fairness of the system. Systems that interact closely with the physical world are called Cyber Physical systems. Systems which interact with physical world and also have human interaction are called as human cyber physical systems [hCPS] [2]. Correctness of hCPS depend on both behaviour/actions of human operator and actions of synthesized controller. Therefore we need a methodology to synthesize a controller considering the interaction with human operator. Specifically we need to device a semi-autonomous controller with occassional human interaction.

3 Characterization of Human In the Loop Control System (h-CPS)

Semi-autonomous cars are one of the interesting domain of HuIL. Because of the interaction of human driver and an autonomous controller, challenge is to synthesize shared human and autonomous controller. The main challenges associated with designing h-CPS [2] systems like semi-autonomous cars are given below.

1. How h-CPS systems differ from typical CPS. How to model h-CPS?
2. How to capture the behaviour of human operator ?
3. What is the synthesis procedure which can model the interaction between both semi-autonomous system and human operator?

Modelling h-CPS

The model of human-Cyber Physical Systems will typically have 1) a plant which has to be controlled 2) A controller and human operator who interacts with the controller 3) An advisory system which mediates between autonomous system and human operator 4) Environment.
In the case of semi autonomous driving , car is the plant. A controller is a subsystem which performs different maneuvers of the car in an autonomous fashion until human presence is needed. Advisory control is a sub system which acts as a mediator and decides whether car has to be driven by human driver, or autonomous controller or combination of both. While autonomous controller and advisory controller can be modelled as Finite state Transducers which can capture all possible environment behaviours(assumptions) and corresponding actions of the controllers, modelling human behaviour is not a trivial task. We will look into modelling human drivers in later section.

Specifications/Requirements of HuIL

Considering safety issue and potential benefits of autonomous vehicles, the National Highway Traffic Safety Administration (NHTSA) has provided design for different levels of automation. Level-3 mode of automation is as below

"Limited Self-Driving Automation: Vehicles at this level of automation enable the driver to cede full control of all safety-critical functions under certain traffic or environmental conditions and in those conditions to rely heavily on the vehicle to monitor for changes in those conditions requiring transition back to driver control. The driver is expected to be available for occasional control, but with

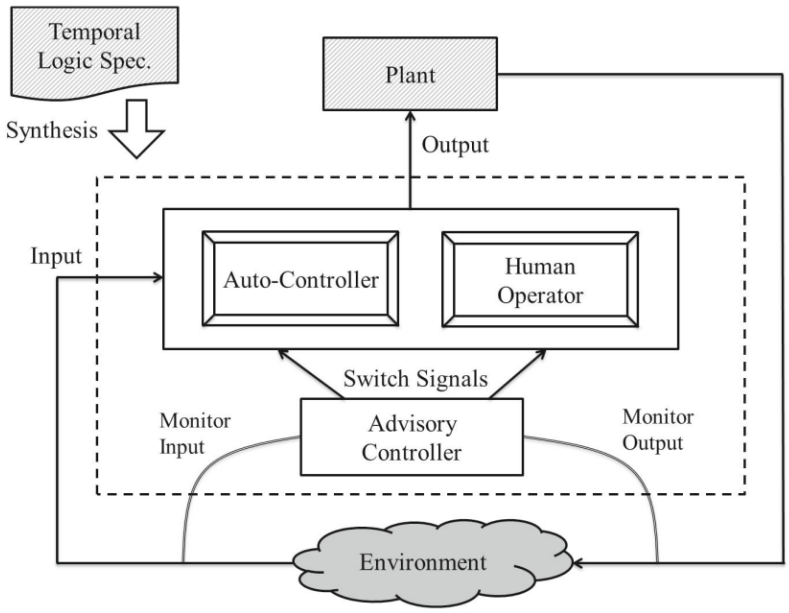


Figure 1: Component Overview

sufficiently comfortable transition time. The vehicle is designed to ensure safe operation during the automated driving mode" Based on the above instruction we can identify four important criteria that every semi-autonomous cars have to satisfy. However these criteria can be generalized to all h-CPS systems.

- **Monitoring** The controller should determine when the human intervention is needed based on some past and current information.
- **Minimally Intervening** Controller has to invoke the human operator only when it is necessary and in a way such that the interaction of human operator with the system is minimal.
- **Prescient** The controller has to find out if there is a possibility that specifications can be violated in future time.
- **Conditionally Correct** As the human operator is involved in minimal fashion and also the reaction time of human operator needs to be considered, the controller must behave correctly until the time the human operator takes over the control.

We need a method which can synthesize a HuIL controller which is by construction monitoring, minimally intervening, prescient and conditionally correct. We will look into method of synthesizing HuIL controller which satisfies all aforementioned requirements which is specified by LTL formulae in the reactive system setting.

4 Preliminaries

Finite State Transducer

A Finite State Transducer (Mealy Transducer) is defined by a tuple

$$M = (\mathcal{Q}, q_0, \mathcal{X}, \mathcal{Y}, \rho, \delta)$$

where $\mathcal{Q}$ - Set of states
q_0 - Set of initial states
$\mathcal{X}$ - Input alphabet
$\mathcal{Y}$ - Output alphabet
$\rho : \mathcal{Q} \times \mathcal{X} \to \mathcal{Q}$ - transition function
$\delta : \mathcal{Q} \times \mathcal{X} \to \mathcal{Y}$ - Output function

Linear Temporal Logic

Linear temporal logic, defined by LTL formula is made of Atomic prepositions (AP), Boolean connectives such as negation($\neg$), conjunction($\wedge$), disjunction ($\vee$) and temporal operators X(next), $\mathcal{U}$(until), F(eventually), G(always) etc.LTL formulae are usually defined on infinite traces of input variables.

An LTL formula has the form :

$$\psi := \mathbf{true} \mid p \mid \neg\psi \mid \psi \vee \psi \mid \mathbf{X}\psi \mid \psi\mathcal{U}\psi$$

where p is an atomic preposition.

Generalized Reactive(1) specification

GR(1) specification is the subclass of LTL formula.

GR(1) has the form -

$$\psi = \psi^{env} \implies \psi^{sys}$$

where ψ^{env} represents Environment assumption and ψ^{sys} represent system guarantees. GR(1) specification comprises conjunction of sub-formulas of both environment and system which is represented as ψ^l where $l \in \{env, sys\}$.

- ψ_i^l : LTL formula characterizing initial states.
- ψ_t^l : LTL formula characterizing the transition.
- ψ_f^l : LTL formula characterizing fairness.

Synthesis of controller or deciding realizability from LTL specification requires construction of Buchi automata/Rabin automata and then construction of Buchi/ Rabin game because of which has complexity of order 2EXPTIME-complete which

is prohibitively very high. With GR(1) specification realizability can be decided in polynomial time complexity using μ-calculus and fixed point operators [3].

Generalized Reactive(1) Games and Counter strategies

In general, synthesis problem can be seen as a two player zero-sum game between system and the environment. If the system wins, env looses and if system looses env will win and the winning strategy of the environment is the counter-strategy. Finite state two-player game is a graph represented by the tuple

$$\mathcal{G} = (Q^g, \theta^g, \rho^{env}, \rho^{sys}, Win)$$

where $\mathcal{Q}^g \subseteq 2^{X \cup Y}$ - state space over input variable (X) and output variable (Y)
θ^g - initial condition over X $\cup$ Y
$\rho^{env} \subseteq \mathcal{Q}^g \times 2^X$ - Env transition determining possible next input
$\rho^{sys} \subseteq \mathcal{Q}^g \times 2^X \times 2^Y$ - System transition determining possible next output
Win - Winning condition of the form $\psi_f^{sys} \implies \psi_f^{env}$

A play π of a game $\mathcal{G}$ is maximal sequence of states $\pi = q_0 q_1 \ldots$ of states such that $q_0 \models \theta^g$ and each intermediate states satisfy transition relation of both environment and system. A play π is winning for system iff it is infinite and $\pi \models Win$. Otherwise π is winning for the environment. Whenever there is a winning strategy for environment, such strategies are called counter-strategies. Existence of counter-strategy implies that the system is un-realizable. Each counter-strategy itself can be treated as finite state machine. Algorithm that we are going to see in the next section is based on the counter strategy generated by the GR(1) synthesizer.

Convex Markov Chain

A labelled finite Convex Markov chain is a tuple, $\mathcal{M}_c = (S, S_0, \Omega, \mathcal{F}, X, L)$

where S - Finite set of states
S_0 - Set of initial states
Ω - Atomic propositions
$\mathcal{F}$ - Set of convex sets of transition probability distributions(TPD)
$\mathcal{F}_s \in F := \{f | 0 \leq \underline{f} \leq \overline{f} \leq 1 \text{ and } 1^T f = 1\}$ where $\underline{f}(\overline{f})$ is element wise minimum(maximum)
$X : S \to \mathcal{F}$ - Function that associates each state with convex set of TPD.

$L : S \to 2^{\Omega}$ - Function that associates each state with set of atomic propositions

In CMC, we need not have unique transition probability distribution at each state. Instead we can have convex set of TPD.

Probabilistic Computation Tree Logic

PCTL is extension of Computational Tree Logic(CTL) with a probability operator (P).

The syntax is := $\phi : \mathbf{true}|\omega|\neg\phi|\phi_1 \wedge \phi_2|P_{\sim p}[\psi]$

where $\sim \in \{ <, >, \geq, \leq \}$ is a comparision operator and p is threshold. We use CTLs for the purpose of quantitative verification which is not possible using standard LTL specifications. However to capture uncertainty we need probabilistic models. Therefore PCTLs are used to model CMC.

State Discretization in Receding Horizon Fashion

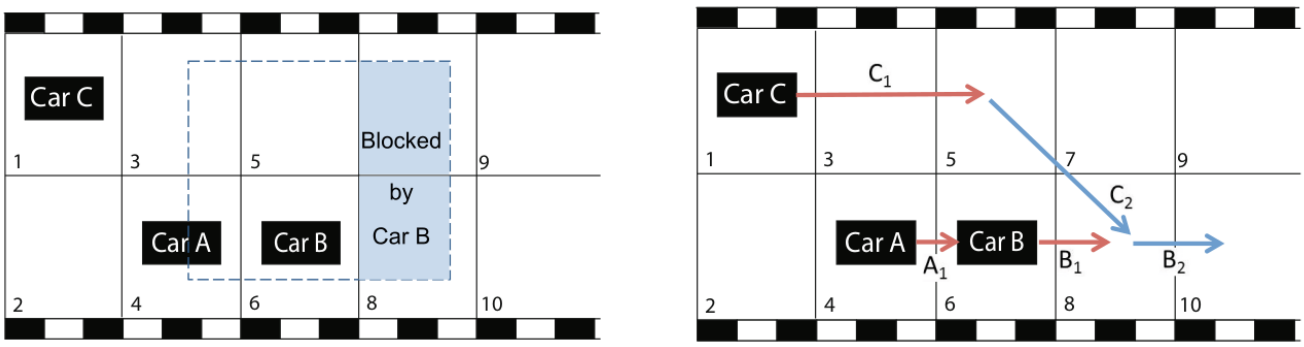
To discretized the continuous state space, the common approach is to construct a finite state transition system which represents an abstract model for physical system. The problem with this approach is that this may have infinitely many states and suffers from state explosion problem. However it is not required to synthesize entire execution with all possible environment behaviour at once. As the state which is far away from present state does not affect the near future plans. We can consider an example as a robot need to travel 100 km. It need to know about only near future plan as it can move with a limited speed. So it is sufficient to synthesize an execution plan for only 500 meters and implement it in receding horizon fashion. Receding horizon scheme allows us to perform synthesis on a smaller domain and substantially reduces the number of states of the automaton while ensuring the system correctness.

Receding horizon strategy [7]: Initially execution starts from state $v_0 \in T^{-1}(\mathcal{W}_i)$ and execute the Automaton $\mathcal{A}_i$ until it reaches the state $v \in T^{-1}(\mathcal{W}_j)$ where $\mathcal{W}_i \preceq_{\psi_g} \mathcal{W}_j$ and at that point it switches to automata $\mathcal{A}_j$. This procedure will be repeated until $\mathcal{A}_0$ is not executed. Here $\mathcal{W}_i$ is set of discrete states from discrete state space representation $\mathcal{V}'$ and $T^{-1}(\mathcal{W}_i)$ is inverse relation which maps discrete states of $\mathcal{W}_i$ to continuous states of dom(S).

5 Motivation

. We are considering semi-autonomous cars(Driver assistance systems) as an example. Driver assistance systems are of two types. One in which human driver's actions are considered to be fail prone and the driver assistance system acts as fail safe system and corrects the human drivers actions whenever needed. In the second case of driver assistance system, car is in the control of semi-autonomous controller for most of the time and human driver acts as fail safe system and when the semi autonomous system cannot handle the failure situation the system alerts the human driver to take over the control. Human in the loop control systems correspond to driver assistance systems in which human driver acts as fail safe system.

Example



(a) A's Sensing Range. (b) Failed to Follow.

Figure 2: Controller Synthesis - Car A following Car B

Let us consider a car following example where car A always try to follow car B. Car A is considered as part of the system where car B and car C is considered as part of environment which can not be controlled. The road is assumed to be divided into discretized regions as shown in the figure and the planning is done using the receding horizon approach as explained in previous section. Car A is equipped with the sensors which allows car A to see up to two block ahead of itself. Car A and C are allowed to move two block in one time step while Car B can move only one block in one time step. Whenever Car C comes in between A and B, Car A is unable to follow Car B, we consider this as a failure scenario. Here Goal is to automatically synthesize a controller where Car A always follows Car B and whenever it is unable to follow, control will be transferred to human

driver for correct actions with given enough time to driver. In Figure2, red arrow shows first step of movement of all the cars and blue arrow shows the second step of movement. It is visible that there exist a case where car C can come between A and B as shown with blue arrows. So here aim is that in such failure scenario the controller should transfer the car control to the human driver. The challenge in this HuIL synthesis is to decide when should human driver be notified about failure, giving him/her sufficient time to react and also assuring minimal intervention. We need to use the receding horizon based solution for this problem as the environment is completely dynamic and fixed horizon may not be a right approach to solve this.

6 Controller Synthesis for Semi-Autonomous Driving

Model of the Controller

In HuIL control synthesis, 3 Agents have been modeled as Finite state Transducer named as Automatic controller($\mathcal{AC}$), Advisory controller($\mathcal{VC}$) and Human controller($\mathcal{HC}$). Even though $\mathcal{HC}$ is not explicitly modeled it can be viewed as Finite state Transducer and uses the same input and output set as $\mathcal{AC}$. Both $\mathcal{AC}$ and $\mathcal{HC}$ uses an additional boolean variable named auto in their input set which is the only output of $\mathcal{VC}$. This auto variable is used by Advisory controller to indicate the Human operator, to take over control of the plant whenever necessary. $\mathcal{HIL}$ controller uses additional *active* variable. Whenever active is set to true $\mathcal{HC}$ overwrites the output of $\mathcal{AC}$.

Criterion for HuIL controller

As per the given criterion for semi-autonomous cars by the NHTSA (discussed earlier), here we can redefine the four important criterion for our semi-autonomous car example.

- **Monitoring** HuIL controller always monitors the situation of system and if the vehicle can reach a failure state $\mathcal{VC}$ signals the Human operator to take over control by setting *auto* variable to false, which was true when $\mathcal{AC}$ was controlling the vehicle. This condition is determined based on history information but not on the predictions.
- **Minimally Intervening** Whenever a failure occurs human intervention is needed for correct action and $\mathcal{HC}$ makes transition from non active state to active state. However the time duration in which the human driver controls

the vehicle should be minimal. This is done by optimizing a joint function of which combines (1) probability that environment forces $\mathcal{AC}$ in to a failure scenario where auto can be set to false and (2) cost of Human involvement which increases with the time the Human operator take control. Basically this is like a trade-off between cost and failure probability.

- **prescient** To counter the failure condition Human operator should take the control but with given enough time to respond. Whenever there may be a situation where environment can force $\mathcal{AC}$ to failure prone state, Advisory controller will set the auto to false before some T time of failure and signal the Human operator to take over control.
- **Conditionally correct** Whenever a failure prone state is reachable, Advisory controller will signal the Human operator to take over control before T time units from failure node. Until the Human operator takes the control Auto controller has to operate correctly. Whenever Human driver takes the control it overwrite the the output of $\mathcal{AC}$

Along with the four requirements mention above, following properties also have to be satisfied by the synthesized controller. Let p and q be two prepositional formulae.

- **Safety:** This states that nothing bad should happen throughout the execution of the controller. This can be stated with the formula **G** p ("Always p").
- **Guarantee:** There should always be a guarantee that vehicle will reach its destination. This can be stated with formula **F**p.
- **Obligation:**This states the disjunction of Safety and Guarantee with formula **G**p $\vee$ **F**q.
- **Progress:** System should always make progress and can be capture by formula GFp.
- **Stability:** At some point of time system should reach stable(desired) condition. It can captured by the formula **FG**p
- **Response:** It depicts how system should react when there is a change in the environment. Captured by formula $\mathbf{G}(p \implies \mathbf{F}q)$

All the requirements that need to be satisfied by the controller are specified using GR(1) specifications. GR(1) synthesizer is fed with these specifications. If the GR(1) formulae are satisfiable controller is returned by the GR(1) synthesizer otherwise counter-strategies are returned. We will use the counter strategy returned by the GR(1) synthesizer to synthesize a controller.

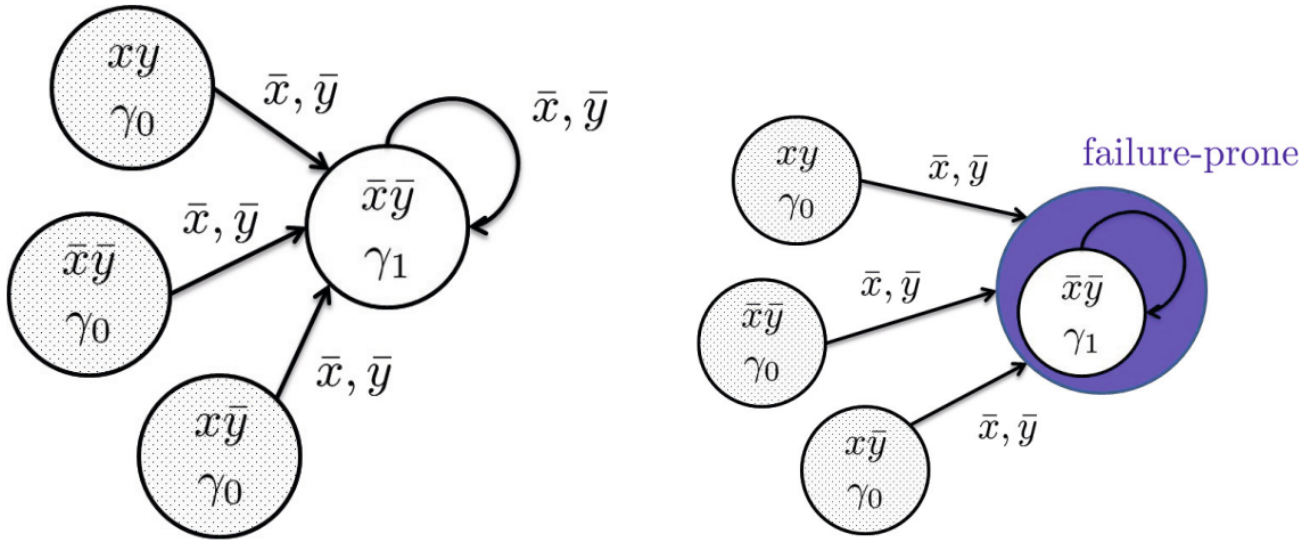


Figure 3: Counterstrategy graph G^c for unrealizable specification

Example: Consider $X = \{x\}, Y = \{y\}$ and the following GR(1) sub-formulae which together form $\psi = \psi^{env} \implies \psi^{sys}$

$1.\psi_i^{env} = \mathbf{true}$
$2.\psi_t^{env} = \mathbf{true}$
$3.\psi_f^{env} = G(F\neg x)$
$4.\psi_i^{sys} = \mathbf{true}$
$5.\psi_t^{sys} = G(\neg x \implies \neg y)$
$6.\psi_f^{sys} = G(Fy)$

A counter-strategy for this example is shown in the figure 3. A state is defined as combination of input picked by environment, output produced by system and content of the finite memory. Here γ_0 denotes the initial memory content γ_i denotes the memory at time i. First environment chooses input $\bar{x}$ and then system picks the output $\bar{y}$ as per the transition specifications. As per the specification, System fairness is described by the formula $\psi_f^{sys} = \mathbf{G}(\mathbf{F}y)$. But the transition constraint of the system $\psi_t^{sys} = \mathbf{G}(\neg x \rightarrow \neg y)$ forces the system to produce output $\bar{y}$ whenever environment chooses input as $\bar{x}$ which violates the fairness of system. As depicted in figure 3, system may reach a failure doomed situation where it will be stuck forever in an unsafe state. So the goal of the synthesis algorithms is to identify such failure scenario and notify the human to take over the control with giving him/her enough time to respond. In the next section we will describe the counter-strategy guided synthesis of HuIL controller in detail.

Types of Failures

In the Above example (Fig.4) we can see that system can reach a failure prone node where it may no longer satisfies the desired specifications and violate some system guarantees. We can consider two type of failure in this condition.

- **Safety violation:** From the counter-strategy when environment picks an input and system is unable to pick any output and doesn't satisfy all the guarantees. Such nodes are called failure imminent nodes.
- **Fairness violation:** When environment forces system may reach a node which is part of Strongly Connected Component(SCC) in counter-strategy graph Such nodes are called as failure doomed nodes. Once the system reaches a failure doomed node, it may enter an infinite loop. In figure 3 node$(\bar{(}x), \bar{y}, \gamma_1)$ is a failure doomed node.

In the counterstrategy graph each SCC is condensed into single node and a Directed Acyclic Graph (DAG) $\hat{G}^c = (\hat{Q}^c, \hat{Q}^c, \hat{\rho}^c)$ is generated.

Weighted counter-strategy Graph

To satisfy the notion of minimal intervention there is need for minimizing the cost of human involvement. We need to optimize the joint objective cost function explained before. We can assign weights to each edge in counter-strategy graph for this purpose. Longer the human driver is involved with driving, the cost will be more. However safety is the primary concern and hence the driver has to be notified of the failure that may happen well ahead of time. The weight function can be written as

$$\varpi\left(\hat{q}_i, \hat{q}_j\right) = \begin{cases} 1 & \text{if } \hat{q}_j \text{ is failure-prone} \\ \frac{pen(\hat{q}_i) \times \text{len}(\hat{q}_i)}{c(\hat{q}_i)} & \text{Otherwise} \end{cases}$$

where $pen : \hat{Q}^c \rightarrow Q^+$ is user defined penalty, $len : \hat{Q}^c \rightarrow Z^+$ is length of shortest path from node q_j to any failure prone node and $c(q)$ represents total number of legal action which environment can take. We can clearly observe that the weight function assigns more weight to those edges which are near to the failure node as well as very far from the failure nodes. This makes intuitive sense as the edges near to the failure node will have more probability of failures and number of legal actions the environment can take becomes very small. Similarly for the edges that are far away from the failure node more weight should be given to minimize the human intervention. The used cost function also behaves in the same way. When an edge is far away from the failure node the length of the shortest path to the failure node increases and the assigned weight also increases.

Counter-strategy guided synthesis

As we have counterstrategy graph and condense it with removing all SCC by single node. Whenever this node reachable to any non failure node system may reach to an unsafe state. So if these moves of environment, which may lead system to failure prone sate, are disallowed then non of the failure non will reachable from any non-failure node and we can remove counterstrategy. So we need to mine some extra assumptions which additionally need to be satisfied with environmental assumptions. We need to assert negation of the condition which is making system to move failure prone node. Formally we can say these additional assumption as

$$\phi = \Lambda_i \left(\hat{\mathbf{G}} \left(a_i \rightarrow \neg \mathbf{X} b_i \right) \right)$$

where a_i is Boolean formula which describing a set of assignment over variable X, b_i represent a Boolean formula describing set of assignment over X. We can see this in this context that from a non failure state which satisfy condition a_i then system cannot go to a state which satisfy b_i which make system to go unsafe state by forcing transition of system to a failure prone node. As in above example we can see that there are 3 outgoing edges from non-failure nodes $(x, y, \gamma_0), (\bar{x}, \bar{y}), (\gamma_0, x, \bar{y}, \gamma_0)$ which are making transition to failure-prone node $(\bar{x}, \bar{y}, \gamma_1)$. So we need to restrict these transition by adding extra assumptions.

$$\phi = (\mathbf{G}((x \wedge y) \rightarrow \neg \mathbf{X} \overline{x})) \wedge (\mathbf{G}((\overline{x} \wedge \overline{y}) \rightarrow \neg \mathbf{X} \overline{x})) \wedge (\mathbf{G}((x \wedge \overline{y}) \rightarrow \neg \mathbf{X} \overline{x}))$$

Under this assumption ϕ if $\phi \wedge \psi^{env} \rightarrow \psi^{sys}$ then we can say that we have successfully synthesized an auto-controller that satisfy specification ψ. clearly this mining is equivalent to finding set of edges so that no failure prone node can be reached from any non-failure node. We can denote such set of edges as E^{ϕ}. The notion of Prescient says that human driver should get enough time (say T) to respond to any failure where he/she must need to intervene. But the longer interaction of human further increase the cost which we introduce by adding weights to edges in counterstrategy graph. So we need to minimize a joint objective function $\mathcal{C} = \sum_{e \in E^{\phi}} \varpi(e)$. where E_{ϕ} can be computed as discussed earlier, and $\varpi(e)$ is weight assign to edge e.

To give enough time T to human, first we need to remove $\hat{Q}_T$ nodes from $\hat{G}^c$ which are backwardly reachable within T-1 steps from any of the failure-prone nodes. After which we will get a graph $\hat{G}^c_T$ and we can formulate this as an s-t min cut

problem (shown in figure 4) to obtain E^{ϕ} by adding a source and terminal node with sufficiently large weight on edges.

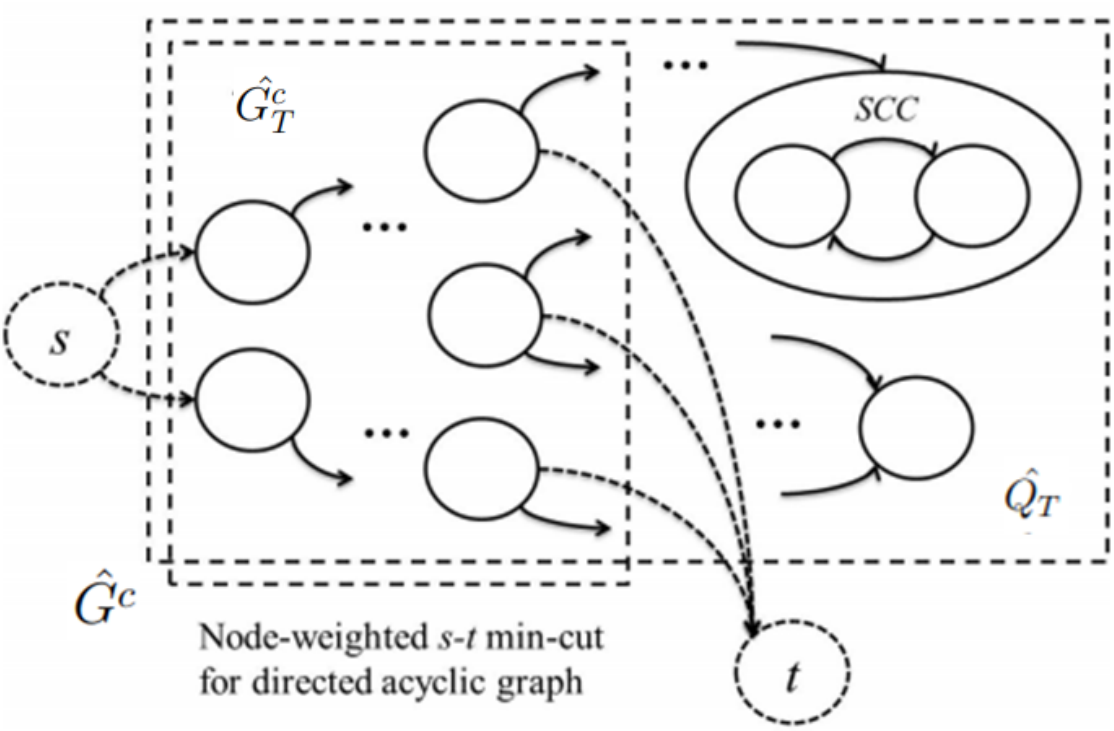


Figure 4: Formulation of s-t min cut problem in

Basically this E^{ϕ} represents the set of edges from which a system can reach failure-prone node (in the example T=1) or at alarming situation (where T-1 nodes before any failure-node are considered to produce alarming signal, where Human need to take over the control).

Algorithm 1: Counterstrategy-Guided HuIL Controller Synthesis

input : GR(1) specification $\psi := \psi^{env} \implies \psi^{sys}$
input : $\mathbf{T}$: parameter for minimum human response time.
output: $\mathcal{AC}$ and $\mathcal{VC}$. $\mathcal{HUIL}$ is then a composition of $\mathcal{AC}$,$\mathcal{VC}$ and $\mathcal{HC}$
if *ψ is realizable* **then**
  Synthesize transducer M $\models \psi$ (using standard GR(1) synthesis);
  $\mathcal{HUIL}$:= M (fully autonomous).
else
  Generate G^c from ψ (assume a single G^c; otherwise the algorithm is performed iteratively);
  Generate the DAG embedded $\hat{G}^c$ from G^c.
  Reduce $\hat{G}^c$ to $\hat{G}^c_T$;
  Assign weights to $\hat{G}^c$ using ϖ by removing $\hat{Q}^c_T$ - nodes that are within $T-1$ steps of any failure-prone node;
  Formulate a s-t min-cut problem with $\hat{G}^c_T$;
  Solve the s-t min-cut problem to obtain E^ϕ;
  Add assumptions ϕ to ψ to obtain the new specification $\psi_{new} := (\psi^{env} \wedge \phi) \implies \psi^{sys}$;
  Synthesize $\mathcal{AC}$ so that $M \models \psi_{new}$; Synthesize $\mathcal{VC}$ as a (stateless) monitor that outputs **auto** = **false** if ϕ is violated.
end

Correctness

In the above algorithm we will construct the advisory controller which will monitor the system continuously. The design of algorithm is such that the human intervention is minimum. This fact is taken care by the cost function used in the algorithm. Also to make the system prescient we are removing all the transitions that can take the system to failure state in next T-1 time steps. And by construction the controller is also conditionally correct.

Experimental Results on Algorithm 1

We can consider the example shown in figure 2 where Car A try to follow car B. Let p_A, p_B, p_C LTL formulae which denotes the position of car A,B,C respectively. There are some LTL specification(Assumptions and Guarantees) need to

be satisfied. Few of them are shown bellow.

- Any position(discrete grid on road) can be occupied by at most one car

  $\mathbf{G}\,(p_A = x \rightarrow (p_B \neq x \wedge p_C \neq x))$

  $\mathbf{G}\,(p_B = x \rightarrow (p_A \neq x \wedge p_C \neq x))$

  $\mathbf{G}\,(p_C = x \rightarrow (p_B \neq x \wedge p_A \neq x))$

- Car A should always follow Car B

  $\mathbf{G}\,((v_{AB} = \text{ true } \wedge p_A = x) \rightarrow \mathbf{X}\,(v_{AB} = \text{ true }))$
  where (v_{AB} states that Car A can see Car B.

- Two cars can not simultaneously cross each other

  $\mathbf{G}\,(((p_C = 5) \wedge (p_A = 6) \wedge (\mathbf{X}p_C = 8)) \rightarrow (\mathbf{X}\,(p_A \neq 7)))$

As we can see in figure 2, if position of car C is 5 and Car C is 6, both cars are side by side. If Car C makes a transition to grid 8 then Car A is not allowed to go at position 7 as they can crash into each other.
Same as these specification other specification also need to be satisfied which are not shown.

It is visible that in step two (In figure 2(b) with blue arrows) Car C can violate the specification and can block Car A to see Car B if it comes to position 8.
By applying the algorithm 1 with this unrealizable specification with time T=1 we can get the ϕ as given bellow

$\phi = \mathrm{G}\,(((p_A = 4) \wedge (p_B = 6) \wedge (p_C = 1)) \rightarrow \neg\mathrm{X}\,((p_B = 8) \wedge (p_C = 5))) \wedge$

$\mathrm{G}\,(((p_A = 4) \wedge (p_B = 6) \wedge (p_C = 1)) \rightarrow \neg\mathrm{X}\,((p_B = 6) \wedge (p_C = 3))) \wedge$

$\mathrm{G}\,(((p_A = 4) \wedge (p_B = 6) \wedge (p_C = 1)) \rightarrow \neg\mathrm{X}\,((p_B = 6) \wedge (p_C = 5)))$

This additional assumption ϕ need to be satisfied by the car A. If car A failed to satisfy these conditions then advisory controller will signal the human operator that car A may not be able to follow Car B.

7 Data Driven Probabilistic Modelling of Human Driver Behavior

As mentioned above, in most of the cases human driver is the reason for most of the accidents. Studies show that 22%-50% of the times drivers distraction is the reason for accidents. Therefore, modelling the driver and predicting his actions is an important factor. Attentiveness of the driver can help the model to guarantee safety of the system.

Modelling the driver however is not a trivial task. Also each individuals behaviour varies from the other. Therefore deterministic models are not practical. We need to consider the uncertainties and the variability into consideration. Therefore stochastic models can be used [4]. Also we need to quantify the attentiveness which improves the intervention predictions confidence. Therefore we can use models such as Convex Markov Chains along with Probabilistic Computation Tree Logic to model complex driver behaviours. We can make use of computer vision techniques to identify different poses of driver to identify the attentiveness and also can make use of other Machine Learning techniques to find out the mode of the driver combined with his attention level using some past observed data. We can collect driving data from different set of users in different environment conditions and with different attention levels. We can use simulation or real-time driving for this purpose.

Methods for Modelling Driver Behaviour

We need to build a model which can predict the model behaviour before we can use it formally. Once we design the model we combine the model with formal verification techniques. Previous studies suggest that driver behaviour depends on some particular modes which depend on drivers state and environment state. We can collect driver state from past 2 seconds and also collect future 4 seconds of environment state and predict the trajectories. Using collected data we can use K-means clustering algorithm to predict the mode, the driver's behavior depends on. Once the clustering is done we can use the modes to predict the future steering angles and further predict the trajectories. This driver model can be used to find the driver behaviour and future trajectories continuously.

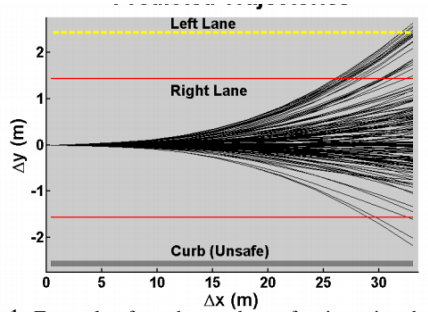
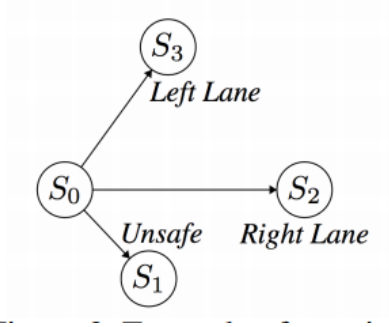


Figure 5: Predicted Trajectories



Transition	Transition Probability Interval
$S_0 \rightarrow S_1$	[0.019,0.021]
$S_0 \rightarrow S_2$	[0.890,0.980]
$S_0 \rightarrow S_1$	[0.048,0.053]

Figure 6: Converted Empirical Probabilities

Stochastic Modelling of Driver

Because the driver behaviour is not deterministic we need to consider the uncertainties and also the error in predicting the driver attentive levels or the modes on which driver behaviour depends on. Therefore we need to convert the trajectories generated after K-Means clustering into a transition model with associated probabilities. We can consider each trajectory as safe,unsafe and left lane (Fig.6). We can consider more complex transition systems as well. We can then find the transition probabilities using the empirical data. The frequency of the trajectories predicted can be used to calculate the empirical transition probabilities. Now we can build a CMC model using these states and the empirical transition probability.

Formal verification of Driver Models

After the model for the driver is created and the environment and system space is discretized CMC model can be created. Now we can use PCTL to verify the quantitative requirements the model has to satisfy. This method is based on the data collected from past driving experience either using simulation environment or real driving with different drivers. Such methods are called data driven methods.

8 Safe semi Autonomous Control with Enhanced Driver modelling

In this section we describe 2nd h-CPS system in which the human operator is fail prone and semi autonomous system is considered as fail-safe system. Considering again the semi-autonomous driving example, we can see that drivers distraction might cause accidents. However if we can predict the drivers intention and future trajectories we can construct a safe semi autonomous system which can correct the

drivers behaviour. Instead of considering driver as disturbance, we can model the driver as specified above and include the driver model along with synthesis procedure. Now we can combine both driver model and controller design procedure [5] to design a safe semi-autonomous system.

Methodology

Consider a vehicle whose evolution is given by $\mathbf{x(k+1) = f(x(k),u(k))}$, where x(k+1) $\in R^n$ is the state of vehicle and u(k) $\in R^m$ is the input at time k. Let $\overline{N} \in N$ be a fixed horizon.
Let us consider some functions that we use in this model.

Constraint Function: $\varphi_{\overline{N}}(x(k), k) \leq 0$ if vehicle state is safe at time k and unsafe otherwise.

Cost Function : J$(R^n, u, \overline{N}) \in R$ is a function which gives us the cost over the entire horizon $\overline{N}$, of using an input u at a given vehicle state.

Assume that an algorithm $\mathcal{A} : R^n \times \mathcal{U}_{\overline{N}} \times N \rightarrow \mathcal{U}_{\overline{N}}$ which computes $\underset{u \in \mathcal{U}_{\overline{N}}}{\text{argmin}}$J(X(0),u,$\overline{N}$) exists. (Ex : We can construct such algorithm by using Model Predictive Control).

Driver state Function : $\theta_{\underline{N}}$ is a driver state function which describes how the driver has evolved in previous $\underline{N} + 1$ time step.

Using above four functions we can design a semi-autonomous framework which consists of following two components.
Component1 : **Vehicle Prediction Multi Function (VPM** Fixing $\overline{N}, \underline{N} \in N$ and choosing $\alpha \in [0, 1]$ and $k \in \{0, \ldots, \overline{N}\}$, we can define a function $\Delta(\alpha, k, \mathcal{O}, \mathcal{I})$ as solution to,

$$\begin{aligned} &\underset{\Delta \subset R^n}{\text{argmin}} |\Delta| \\ &\text{s.t.} \quad P((X(k) - x(0)) \subset \Delta | \mathcal{O}, \mathcal{I}) \geq \alpha \end{aligned}$$

Intuitively, the above component will gives us the shortest possible alpha probable vehicle trajectories in k time steps.
Component2 : **Vehicle intervention Function :** Let us define a function whose value will be 1 when any unsafe state is predicted to be reached and 0 otherwise.

$$g(\alpha, \mathcal{O}, \mathcal{I}) = \begin{cases} 1 & \text{if } \cup_{k=0}^{\overline{N}} (\Delta(\alpha, k, \mathcal{O}, \mathcal{I}) \cap \mathcal{C}(k, \mathcal{I})) \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

Now we can use these two components and design an algorithm which helps to prevent accidents.

Algorithm 2: Semi Autonomous Vehicle Framework with Driver Modelling

input : $\overline{N}, N \in N$ and $x(0) \in R^n$
output: Next input to be applied to vehicle Dynamics
for *Each time step* **do**
  update $\varphi_{\overline{N}}$ and $\theta_{\overline{N}}$, and set $x(0)$ equal to the vehicle's current state.
  if $g(\alpha, \mathcal{O}, \mathcal{I}) = 1$ *and* $\mathcal{A}(x(0), u, \overline{N}) \neq \emptyset$ **then**
    apply intervention procedure.
  end
  if $\mathcal{A}(x(0), u, \overline{N}) \neq \emptyset$ **then**
    set $u = \mathcal{A}(x(0), u, \overline{N})$
  end
end

Algorithm

In each step we can update both the safety function and driver state function. To update driver state function we can use computer vision techniques along with previous history of drivers state. Set initial state of the vehicle at each step equal to its current state. And at each step we solve the problem for a fixed horizon of length $\overline{N}$. We apply vehicle intervention function and see if intervention of controller is needed. If needed then intervention procedure is applied. Else we can find the next possible input to vehicle dynamics and apply that input. Repeat the procedure for each time unit until the final position is reached. In the above algorithm , the vehicle intervention function actually depends on the Vehicle prediction function. However it may not always be true that prediction of VPM is correct. Hence the accidents can still happen. But we can see that if the intervention occurs then the safety is always guaranteed.

9 User Interface Design and Verification

In the previous sections we have seen methods to synthesize semi autonomous controller considering the interaction with human operator as well as methods on modelling the human driver in case of semi autonomous driving. But the major problem in h-CPS system is interacting with the Human operator. In modern

days, we expect a better interface between the system and human operator which satisfies the expectation of human operator. One solution for this is by incorporating the data from system, environment and the driver where user interface can act as a communication medium. These data can be collected using various methods [6] like
Vehicle to vehicle communication : vehicles will communicate with each other where they can share the status.
Sensory Information : By using sensors like Lidar, Radar etc surrounding data can be collected.
Driver monitoring : The state of driver can be tracked by eye tracker cameras head positions, steering wheel touch, audio etc.
After collecting data,UI should be deterministic and should display concise information in user friendly manner. Not all data is crucial, so only crucial information should be displayed to user as driver might not be aware of bad impact of some information. So a one to one mapping can be done from informative part to collected data which needs to be provide to user.

10 RRT Based Motion planning of Semi autonomous cars.

Motion planning algorithms can be used to generate path and we can display it to the driver. This would help the driver visualize the path and may help him take the decision of when to take control from the Autonomous System and also guide him when he has taken control from it.

These algorithms are called Sampling-based algorithms. These are named so because they represent the configuration space in which we are working as a road-map of sampled configurations. The algorithms sample some configurations in space and retains those to use them as milestones. A road-map is then constructed which connects two milestones if the line segment joining them is completely in it. We can also check collision detection by not including the line segment which cuts an obstacle.

Some of the notable algorithms are :-
- A* search algorithm
- D* algorithm
- Rapidly-exploring random tree
- Probabilistic road-map.

We will focus on the RRT (abbreviation for Rapidly-exploring random tree) approach, which we can try to use for Motion Planning of Urban Driving Vehicles.

The primary challenges generally faced in the designing of motion planning systems for an autonomous vehicle result from

1. Complex and unstable vehicle dynamics, with considerable drift
2. limited sensing capabilities in uncertain and changing environment
3. Temporal and logical constraints, arising from the rules of the road.

RRT algorithm is an incremental sampling based algorithm. Main reasons for choosing this sampling based algorithm for guiding the Human controller are :-

1. These algorithms are applicable to very general dynamical models
2. As the algorithm is incremental, it is easy implement it in real-time and for online systems.
3. These algorithms do not require the explicit enumeration of constraints but allow trajectory-wise checking of possible very complex constraints.

So it is assumed that we are given a target, and a low-level controller that can track a path and a speed command. Then the motion planner has to provide a path and a speed command to the controller in such a way that the vehicle does not collide with obstacles and stays in lane boundaries. The information which is obtained through the sensors will have noise in it, thus the RRT algorithm must be robust to account for uncertainties.

Rapidly Exploring Random Trees (RRT) :

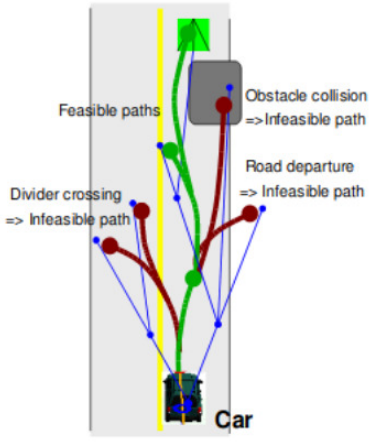


Figure 7

The blue points and lines are those of RRT. The algorithm randomly selects a point which is at step-distance from the current point. Various distance metrics like Minkowski, Euclidean etc. can be used for the same. Then joins the two points if there is no obstacle between the points. Continuing in this way until the algorithm reaches the destination point, we may get different paths to the object. However the RRT calculates the most optimal path, say the one with minimum distance or the user may define a cost function over the space.

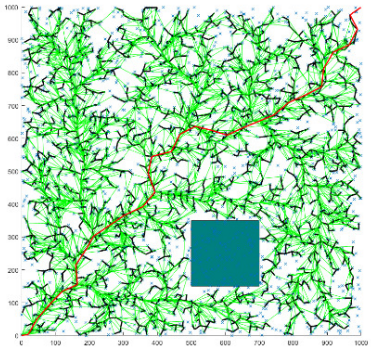


Figure 8

However, the path generated by RRT is not guaranteed to be dynamically feasible. This means that the vehicle may not be able to take sharp turns (an example), as given in the path by RRT.
So, a feasible trajectory is obtained by running forward a simulation of the closed-loop system which consists of the vehicle model and the controller. By using a stable closed-loop system, RRT algorithms can be efficiently employed on vehicles with unstable open-loop dynamics. In Fig.7, the green path is the dynamically feasible trajectory.
Following the standard RRT, the algorithm performs sampling, selection of a node, expansion and constraint check. The planner provides the command to the controller at a fixed rate, and the expansion of the tree continues until this time limit is reached. The best trajectory is selected and then sent to the controller and the tree expansion is resumed after updating the vehicle states and the situational awareness.

Algorithm 3: RRT Based planning Algorithm

```
1:  repeat
2:    Receive current vehicle states and environment
3:    Propagate states by computation time limit
4:    repeat
5:      Take a sample for input to controller
6:      select a node in tree using heuristics
7:      propagate from selected node to the sample until the vehicle stop
8:      Add branch node on the path
9:      if Propagated path is feasible with the drivability map then
10:       Add sample and branch nodes to tree
11:     else
12:       if all the branch nodes are feasible then
13:         Add branch nodes to tree and mark them as unsafe
14:       end if
15:     end if
16:     for for each newly added node v do
17:       Propagate the target
18:       if propagate path is feasible with drivability map then
19:         Add path to tree
20:         set cost of propagated path as upper bound of cost-to-go at v
21:       end if
22:     end for
23:   until The time limit is reached
24:   choose best safe trajectory in tree and check feasibility with latest
      drivability map
25:   if best Trajectory is infeasible then
        remove the infeasible portion from tree and go to line 24
26:   end if
27:   send the best trajectory to controller
28: until Vehicle reaches the target
```

This plan which is being executed is not perfect as collisions may still occur or it becomes infeasible and no feasible trajectory is found while the car is moving. In such cases a warning must be issued to the Human controller and he may take appropriate action. Such conditions may arise only as a result of mismatches between the real world and its representation in the previous planning cycles. It

may happen that an undetectable object suddenly appears in front of the vehicle.
Extensions made in RRT : In order to efficiently generate the path in real world, several extensions have been made to the standard RRT approach.
Biased sampling : In line 5 we have used biased sampling. The physical and logical structure of environment is used to bias the sampling, which increases the probability of generating feasible trajectories.
For example, the sample (xsample,ysample) is taken randomly. But we can use some parameters to bias its shape or location :

$$\begin{pmatrix} \mathrm{x}_{sample} \\ \mathrm{y}_{sample} \end{pmatrix} = \begin{pmatrix} \mathrm{x}_0 \\ \mathrm{y}_0 \end{pmatrix} + r \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix}$$

Where r, are the resulting distributions after offsetting Gaussian distributions. (x0,y0) is the center of the Gaussian cloud. By varying bias values, which can be set based on the situational information, our planner can generate various maneuvers like lane following, U-turn and parking etc.

Tree expansion Heuristics : In line we used heuristics. The Euclidean distance between two points, does not take into account the minimum turn radius of the vehicle. Thus the Euclidean distance between two points can be a poor estimate of the achievable path length.
Thus we use Dubins Path Length from node to sample. Dubins path is the shortest path between two points with a constrained curvature of path and predefined initial and final tangents. This is discussed in detail in [8] Safety as an invariant property : Safety is important feature of any system. We define a state to be safe if vehicle can stay in that state for an indefinite time, without violating any rule. All leaves in the tree are safe states, guarantees that there is always a feasible way to stop if the car is moving and unless there is a safe stopping node at the end of path, the vehicle will not execute it.

In this section, in order to make our Human in the Loop (HuIL) control system more robust, we discussed using the RRT algorithm for our autonomous controller. Due to the nature of the algorithm, the path planning is easy to visualize and the human can also easily decide when to take over the control.

11 Conclusion:

In this report we have described a method to synthesize a controller which takes into consideration the interaction with human operator. We characterized the h-CPS system and described the challenges associated with h-CPS design and the specifications that are to be satisfied by h-CPS. By taking semi-autonomous driving as a motivating example we described the algorithm which can synthesize semi-autonomous systems using GR(1) synthesis and generated counter-strategies. We then described methods to model the driver using data driven approaches. We also discussed briefly the requirement of interface with the human operator. In the final section we discussed about RRT based motion planning for semi-autonomous cars.

There can be some possible extensions to the above discussed methods. While synthesizing the semi-autonomous controller authors haven't considered the uncertainties in the transition of the environment. We can design probabilistic transition functions for environment and make the system more robust.While modelling the driver behaviours we can also consider the drivers possible choices and use more complex model for the driver by utilizing the recent advances in computer vision and machine learning techniques. We can also use online feedback mechanism while modelling the driver. Complex models of the vehicle can also be considered to make the system robust.

References

[1] Wenchao Li, Dorsa Sadigh, S. Shankar Sastry, and Sanjit A. Seshia
Synthesis for Human-in-the-Loop Control Systems

[2] Sanjit A. Seshia, Dorsa Sadigh, and S. Shankar Sastry
Formal Methods for Semi-Autonomous Driving

[3] Nir Piterman, Amir Pnueli and Yaniv Saar
Synthesis of Reactive(1) Designs

[4] D. Sadigh, K. Driggs-Campbell, A. Puggelli, W. Li, V. Shia, R. Bajcsy, A. L. Sangiovanni-Vincentelli, S. S. Sastry, S. A. Seshia
Data-Driven Probabilistic Modeling and Verification of Human Driver Behavior

[5] Ram Vasudevan, Victor Shia, Yiqi Gao, Ricardo Cervera-Navarro, Ruzena Bajcsy, and Francesco Borrelli
Safe Semi-Autonomous Control with Enhanced Driver Modeling

[6] D. Sadigh, K. Driggs-Campbell, R. Bajcsy, S. S. Sastry, S. A. Seshia
User Interface Design and Verification for Semi-Autonomous Driving

[7] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M. Murray
Receding Horizon Temporal Logic Planning for Dynamical Systems

[8] Y. Kuwata, G. A. Fiore, J. Teo, E. Frazzoli and J. P. How,
Motion planning for urban driving using RRT

[9] J. J. Enright, E. Frazzoli, K. Savla, and F. Bullo
On multiple UAV routing with stochastic targets: Performance bounds and algorithms

Contribution

1. I G Prasad - 35 %
2. Prafulla Saxena - 35%
3. Vasu Bansal - 30 %