Switching Protocol between controllers using Reactive Synthesis

Nitish Sharma

Abstract

The problem of synthesizing a correct by construction robust switching controller has been considered in this work. By considering the finite abstraction of a system, it is possible to use off the shelf tools from logic and automata theory to synthesize the discrete mode strategies that satisfy certain high-level specifications. In this work, the problem has been formulated as a two-player game and switching protocol has been extracted by using SLUGS tool[1]. The two-player game has been formulated between the environment and the system where the environment tries to falsify the specification and the system tries to satisfy it. By construction, the existence of a winning strategy for the system ensures that the system satisfies the given specifications against all the allowable behaviors of the environment. All the specifications have been represented using Linear Temporal Logic (LTL). A scenario consisting of a robot maneuvering in a square world with dynamic obstacles (nondeterministic) has been considered. The robot has the choice among different controllers (modeled as the system) which differ in the accuracy. All the motions of the obstacles, uncertainties in the robot motion have been modeled as the environment. The extracted strategy has been verified by studying the motion of the robot in a simulated environment. An analysis of the effect of the number of state variables of the model on the size of the extracted automaton is also performed.

Keywords: Switching Protocol, Reactive Synthesis, Temporal Logic Games, Linear Temporal Logic, Autonomous Navigation

1 Introduction

Sensors provide robotic systems with the information required to perceive the changes that happen in unstructured environments and modify their actions accordingly [2]. The information provided by each sensor and corresponding accuracy varies a lot among different control modes. For example, on one hand, vision provides the global information which is required to localize the objects in the environment and compute their relative spatial relations. On the other hand, touch provides the local information which is required to characterize the way the robot contacts the objects in the environment. Visual Servoing can be modeled as a system with large error but global information which can attribute faster motion of the system and force control can be modeled as the precise but slow control as the information is local to the sensors.

It is inevitable for a robot navigating in an environment with obstacles (potentially dynamic) to use various control modes depending upon the nature of the environment around it. For example, an autonomous car navigating on a road may have different control modes to drive on a highway and to drive on a crowded street. These modes can have different trade-offs like the speed and maneuvering accuracy. Further, there can be few specifications that our system should satisfy like the time for which it can use a particular mode or total time to reach the desired location etc.

As highlighted in two above examples, each mode can satisfy a particular set of specifications but not all the mission level specifications. It is possible to manually design a switching protocol which satisfies the specifications under certain environment behaviors but any switching strategy implemented without verification may lead to instabilities if all the possible scenarios aren't covered during testing [3]. This work is specifically concerned about this mission-level control switching such that the switched system satisfies all the mission level specifications.

Specifications have been represented in Linear Temporal Logic (LTL) [4]. The use of LTL enables to handle a wide variety of specifications beyond mere safety and reachability, as well as to account for potentially adversarial, unknown environments in which the system operates. While solving two-player games with general LTL winning conditions is known to have prohibitively high computational complexity, we are considering only Generalized Reactivity (1) [5] fragments of the LTL which are known to have polynomial complexity.

The remaining report is formulated as follows. Section 2 gives the preliminaries for LTL syntax and two player game formulation. Section 3 gives the details of the problem under consideration. Section 4 gives the simulation results and infers those. Section 5 concludes the work.

2 Preliminaries

2.1 LTL Syntax

We use linear temporal logic (LTL) [4] to formally specify the specifications. Standard LTL uses the following language:

$$\phi := p \mid \neg\phi \mid \phi 1 \vee \phi 2 \mid \phi 1 \wedge \phi 2 \mid \circ\, \phi \tag{1}$$

where p is an atomic proposition relating to the system, and symbol $\circ$ is operator applied in formulas to signify "next". In addition to logical operators ($\vee$, $\wedge$ and $\rightarrow$), there are two temporal modal operators (eventually($\Diamond$) and always($\Box$)) used in LTL formulas. The eventually operator $\Diamond$ is used to signify that something will happen at some point in the future and the always operator $\Box$ is used when a condition must be met at every stage.

2.2 Two player game and switching synthesis

The synthesis problem is modeled as a two-player game between the environment and the plant where the environment attempts to falsify the specification and the plant tries to satisfy it. This work is based on reactive synthesis [5] of controllers for systems interacting with adversarial environments, where a control protocol is synthesized to generate a sequence of control signals to ensure that a plant meets its specification for all allowable behaviors (even nondeterministic) of the environment.

We consider the GR(1) specifications of the following form:

$$\begin{gathered}\phi = ((\phi_q \wedge \phi_e) \rightarrow \phi_s), \\ \phi_\alpha = \phi^\alpha_{initial} \wedge \Box\phi^\alpha_a \wedge \Box\Diamond\phi^\alpha_2\end{gathered} \tag{2}$$

where ϕ_q characterizes all the non deterministic transitions of the system, ϕ_e captures all the environment behaviors and ϕ_s is the expected behavior of the switched system. More precisely, all GR(1) formulas used in this work are of the form given by ϕ_α where $\alpha \in (q, e, s)$. $\phi^\alpha_{intiial}$ is the propositional formula characterizing the initial conditions. ϕ^α_1 is the set of the transition relations characterizing safe, allowable moves and propositional formulas characterizing invariants; all the states that are invariant and ϕ^α_2 is the set of the propositional formulas characterizing all the states that should be reached infinitely often.

Any state of the game can be represented as ((e,q),s). The transition of the game is the move of the plant and the environment followed by the controller (switching mode) move. A switching strategy chooses a mode such that the specifications are satisfied. For a winning strategy to exist (ϕ is realizable), the specification should be met for all the behaviors of the environment and plant. At each state, the system executes a switching mode, which drives the system to a number of possible states due to nondeterminism. If the specification is realizable, solving the two-player game gives a finite automaton that effectively gives a state-feedback switching protocol. By observing which state the system enters, the next switching mode is chosen accordingly by reading the finite automaton. Given two-player game and GR(1) specifications, by using any tool like SLUGS, one can obtain a finite automaton that represents a switching strategy for the system.

3 Problem Formulation and SLUGS encoding

Consider a robot navigating in an environment and can use 3 modes which vary in robot speed and accuracy. More formally, robot's dynamics are controlled by the following set of equations.

$$\dot{\mathbf{x}} = \mathbf{f_d}(\mathbf{u_p}(\mathbf{t}), \mathbf{e_p}(\mathbf{t})) \tag{3}$$

where $\mathbf{f_d}$ is the linear function of $\mathbf{u_p}$ and $\mathbf{e_p}$ and $d \in D$ denotes the set of maneuvering directions of the robot. $\mathbf{u_p}$ is the control signal and $\mathbf{e_p}$ is the corresponding error associated with the p^{th} controller; $p \in P$ denotes the set of control actions.

Though not handled in this work, discrete states can be abstracted from the continuous dynamics by introducing an abstraction mapping between the continuous and the discrete domain and further can be formally defined as a transition system by over-approximation of the system [6]. Roughly speaking, over-approximation takes into account all the possible transitions from a given initial state to all reachable states.

Figure 1 shows the discrete world our robot wants to navigate in (from the start location (red block) to goal(green block)). The dotted region represents the wall. Dark blocks represent the external obstacles which can navigate in the environment. The robot receives 2 signals: controller number ($p \in 0, 1, 2$) and direction ($d \in toStay, North, East, South, West$. For our analysis 3 controllers are considered such that $|u_0| < |u_1| < |u_2|$ and $|e_0| < |e_1| < |e_2|$. In our case, u_p refers to the number of cells that the robot can traverse in one time-step. Hence, the controller 2 can provide the highest speed but has poor accuracy compared to other 2 controllers. Similarly, controller 1 can provide higher speed compared to controller 0 but is less accurate compared to 0. Controller 1 and 2 can be considered two different modes of servoing control and controller 0 is equivalent to force control.

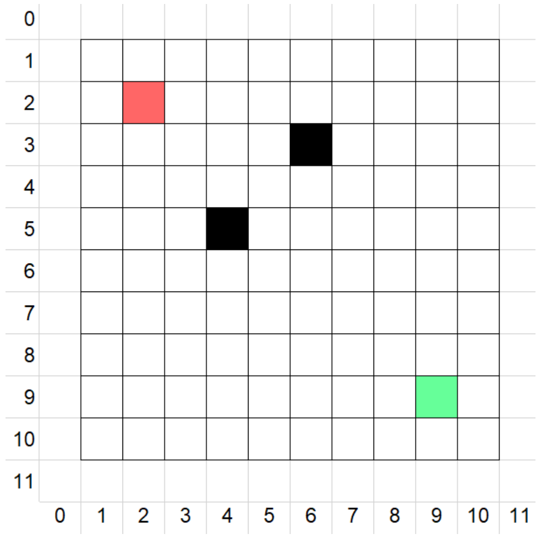


Figure 1: Discrete world where robot is maneuvering. Red block is the start location, green block is the goal, dark blocks represent the obstacles. Dotted region denotes the wall

Obstacles are modeled as moving along the specified path non deterministically. x-coordinate of the obstacles are fixed and 3-bit variables are used to capture the y-coordinate. a_x and a_y denote the coordinates of the robot. The transition in robot co-ordinates There is discrete time variable $t \in 0...31$ which keeps track of the time elapsed since it first left the start location.

In order to make the specifications realizable, we are introducing two control variables: *controller* and *direction*. *direction* can take value from 0 to 4 (0 = no motion, 1 = North, 2 = East, 3 = South, 4 = West) and *controller* gives the choice to the system to choose among 3 controllers. 0 refers to the controller with zero error and can move the robot to the adjacent cell depending on the *direction* $\in 1, 2, 3, 4$. controller 1 can move the robot in the assigned direction to the 3rd cell from its current location but will have error of ± 1 in the perpendicular direction of motion.

Similarly, controller 2 can move to the 4^{th} cell in the direction of motion and error of ± 2 in the perpendicular direction. As we are considering GR(1) representation and need a strategy which assures the specification is met, all these transitions are modeled as non deterministic.

$$\begin{aligned} &[INPUT] && [OUTPUT] \\ o1_y &: 4...10 & controller &: 0...2 \\ o2_y &: 1...7 & direction &: 0...4 \\ a_x &: 0...11 \\ a_y &: 0...11 \\ time &: 0...31 \end{aligned} \tag{4}$$

We want to extract the strategy such that the robot never collides with any obstacle or get into restricted region $((x = 0)|(y = 0)|(x = 11)|(y = 11))$ and should always eventually visits $(2, 2)$ and $(9, 9)$. $time$ is reset to 0 when robot visits $(2, 2)$. We want the $time \leq t_{threshold}$. Please refer the attached files for exact transitions and further details.

4 Results and Discussion

In this work, two types of encodings have been tried to capture the transitions of the robot. One is the symbolic representation of the coordinates of the robot (refer projectSymbolic.structuredslugs) and the other one uses all the states transitions explicitly (refer projectState.structuredslugs). As expected, the time taken by the tool to extract the winning strategy from each representation is same because the symbolic files aren't directly parsed to the tool but are converted to .slugsin format before extracting strategy. Moreover, both the encodings are equivalent. Explicit strategy extracted from both the encodings lead to same finite automaton with 33622 states with $t_{thrshold} = 25$.

4.1 Strategy Extraction

Table 1 shows the number of states of the automaton extracted for different scenarios. From the table, it can be easily inferred that size of the automaton of the realizable strategy increases exponentially with the increase in the encoded states of the system. It requires 31867 states when both the obstacles are moving compared to 256 when there are no obstacles in the environment. Minimum time to satisfy all the specifications also increases with dynamic obstacles in the environment.

It is interesting to note the difference in the cases when one of the obstacles is fixed in contrast to the case when that obstacle is not present at all. For example, compare the case 5 and 7 where obstacle 1 is moving. The minimum time is same for both the cases but when obstacle 2 is present(fixed), size of the automaton is much smaller than the case when obstacle 2 is not present. Presence of obstacle 2 can be seen as restricting the allowed behavior of the system (controller and direction). In other words, all the states allowed in case 5 forms the subset of the states allowed in case 7.

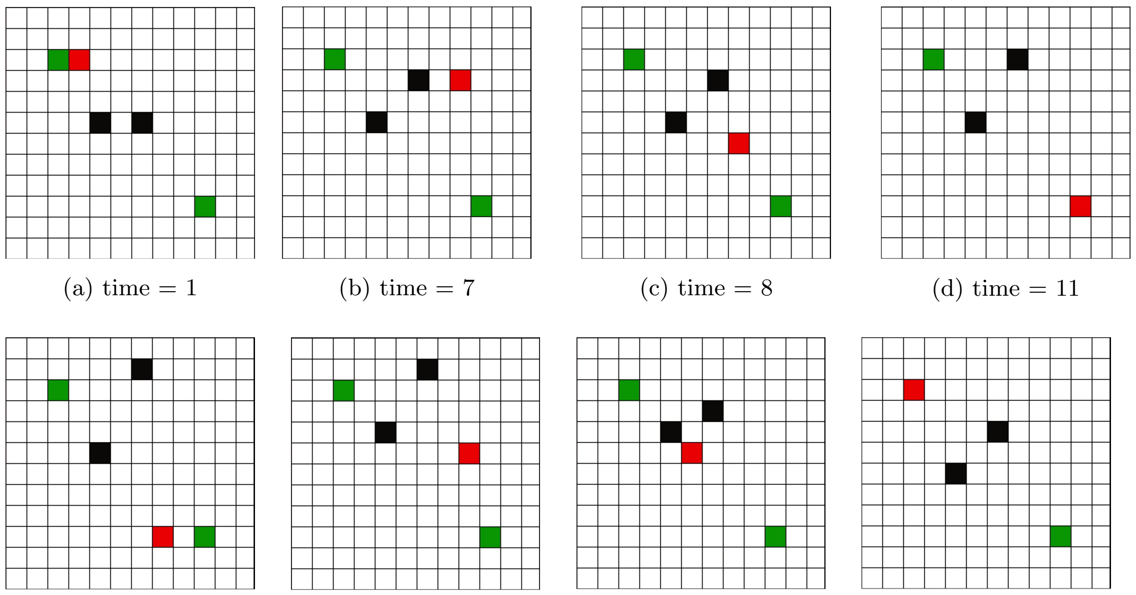
Table 1: Automaton size Comparison

Sr.no.	Obstacle 1	Obstacle 2	Minimum time	Size of automaton
1	not present	not present	15	256
2	not present	fixed	15	237
3	fixed	not present	15	233
4	fixed	fixed	16	163
5	moving	fixed	16	1418
6	fixed	moving	17	2197
7	moving	not present	16	2130
8	not present	moving	17	2425
9	moving	moving	22	31867

4.2 Sample Run

Figure 2 shows the state of the robot and the obstacles at various time steps. It can be easily seen that robot never collides with any obstacle and never enters the restricted region $((x = 0)|(y = 0)|(x = 11)|(y = 11))$. Also, the systems is able to reach back to its starting zone $(x = 2)\&(y = 2))$ within the threshold time ($time = 25$ in this case).

A tuple $Control = (direction, controller)$ can be used to capture the system choices and will be used from now onwards. At $time = 1$, robot leaves the start zone (2,2) by using $Control = (2, 0)$. By continuing for 5 more time steps in $Control = (2, 0)$ and taking first $Control = (1, 0)$ the robot and the obstacles reach the state shown in figure 2b. At $time = 7$, system chooses $Control = (1, 1)$ leads the robot at state shown in figure 2c. At $time = 11$, robot satisfies its first LIVENESS property as shown in figure 2d. At $time = 14$, system chooses to move the robot by using $Control = (3, 2)$ as shown in figure 2e to figure 2f transition. As $controller = 2$ has highest error value associated with it, one would generally expect system never taking this choice. But as can be seen at $time = 14$, no matter how the obstacles behave, choice of $controller = 2$ is still a safe choice (no collision possible) for the system. At $time = 20$, robot returns back to its starting zone which is when under specification limits. Please refer the demo video attached with this report for more details. Please also refer README.txt file to rerun the simulation.



(a) time = 1 (b) time = 7 (c) time = 8 (d) time = 11

(e) time = 14 (f) time = 15 (g) time = 18 (h) time = 20

Figure 2: Position of the robot and the obstacles at various time steps

5 Conclusion and Future Work

Autonomous navigation of a robot in a nondeterministic environment has been considered in this work. The control modes in which the robot can operate can satisfy a set of specifications but are not assured to satisfy mission level specifications which makes it inevitable to switch between these modes. The switching protocol is extracted as a finite automaton by modeling the problem as two player game and encoded in SLUGS. System (switching protocol among available control modes) tries to satisfy the specification and environment (uncertainty in the robot and obstacles motion) tries to falsify it. By construction, the existence of a winning strategy for the system ensures that the system satisfies the given specifications against all the allowable behaviors of the environment. Increase in the size of the automaton with encoding size is also reported.

In this work, the robot and obstacles are assumed to take discrete steps of motion and there mapping to continuous variables has been established. Further, no effort has been made to understand the practical implementation of the extracted protocol. Nevertheless, this work motivates to carry out the similar analysis on a real system which will involve abstraction of the finite state representation of the continuous system. As shown in this work, discrete states can be used to formulate and solve the discrete synthesis problem to extract switching protocol using any GR(1) synthesis tool. Further, testing it on real hardware will be something worth looking into. We are planning to test this approach on NRG vaultbot.

Figure 3: NRG VaultBot mobile dual manipulator

References

[1] Rüdiger Ehlers and Vasumathi Raman. Slugs: Extensible gr (1) synthesis. In *International Conference on Computer Aided Verification*, pages 333–339. Springer, 2016.

[2] Ming Xie. *Fundamentals of robotics: linking perception to action*, volume 54. World Scientific Publishing Co Inc, 2003.

[3] Daniel Liberzon and A Stephen Morse. Basic problems in stability and design of switched systems. *IEEE Control systems*, 19(5):59–70, 1999.

[4] Amir Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977.

[5] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive (1) designs. In *VMCAI*, volume 3855, pages 364–380. Springer, 2006.

[6] Jun Liu, Necmiye Ozay, Ufuk Topcu, and Richard M Murray. Synthesis of reactive switching protocols from temporal logic specifications. *IEEE Transactions on Automatic Control*, 58(7): 1771–1785, 2013.