Fly-by-Logic: A Tool for Unmanned Aircraft System Fleet Planning using Temporal Logic

Yash Vardhan Pant *[1], Rhudii A. Quaye *[1], Houssam Abbas[2], Akarsh Varre[1], and Rahul Mangharam[1]

[1] Dept. of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia PA 19104, USA
{yashpant,quayerhu,akarshv,rahulm}@seas.upenn.edu
[2] Dept. of Electrical Engineering and Computer Science, Oregon State University, Corvallis OR 97330, USA
houssam.abbas@oregonstate.edu

Abstract. Safe planning for fleets of Unmaned Aircraft Systems (UAS) performing complex missions in urban environments has typically been a challenging problem. In the United States of America, the National Aeronautics and Space Administration (NASA) and the Federal Aviation Administration (FAA) have been studying the regulation of the airspace when multiple such fleets of autonomous UAS share the same airspace, outlined in the Concept of Operations document (ConOps). While the focus is on the infrastructure and management of the airspace, the Unmanned Aircraft System (UAS) Traffic Management (UTM) ConOps also outline a potential airspace reservation based system for operation where operators reserve a volume of the airspace for a given time interval to operate in, but it makes clear that the safety (separation from other aircraft, terrain, and other hazards) is a responsibility of the drone fleet operators. In this work, we present a tool that allows an operator to plan out missions for fleets of multi-rotor UAS, performing complex time-bound missions. The tool builds upon a correct-by-construction planning method by translating missions to Signal Temporal Logic (STL). Along with a simple user interface, it also has fast and scalable mission planning abilities. We demonstrate our tool for one such mission.

Keywords: UAS mission planning · Signal Temporal Logic · Correct-by-construction planning · Multi-rotor UAS

1 Introduction

It is inevitable that autonomous UAS will be operating in urban airspaces [1]. In the near future, operators will increasingly rely on fleets of multiple UAS to perform a wide variety of complicated missions which could consist of a combination of: 1) spatial objectives, e.g. geofenced no fly zones, or delivery zones, 2) temporal objectives, e.g. a time window to deliver a package, 3) reactive objectives, e.g. action when battery is low.

* These authors contributed equally

In this paper, we present a tool [3] that allows an operator to specify such requirements over a fleet of UAS operating in a bounded workspace and generates trajectories for all UAS such that they all satisfy their given mission in a safe manner. In order to generate these flights paths, or trajectories, our tool relies on interpreting the mission objectives as Signal Temporal Logic (STL) specifications [2]. We then formulate the problem of mission satisfaction as that of maximizing a notion of *robustness* of STL specifications [3]. Using the approach of [4], we generate trajectories for all the UAS involved such that they satisfy the given mission objectives.

1.1 Related work

Existing mission planner software for autonomous drone operations like ArduPilot mission planner [5] and QGroundControl [6] offer UAS enthusiasts the ability to quickly plan out autonomous UAS flights by sequencing multiple simple operations (like take-off, hover, go to a way-point, land) together. However these planners either cannot handle missions involving multiple UAS and complicated requirements like co-ordination between UAS or completing tasks within given time intervals, or require hand-crafted sequences of maneuvers to meet the requirements in a safe manner. We propose a tool that can inherently deal with multi-agent missions as well as timing constraints on completion of tasks while guaranteeing that planned flight paths are safe. As opposed to existing mission planning software, our tool does not require the user to explicitly plan out maneuvers for the drones to execute to follow out a mission, e.g. in the case where two UAS have to enter the same region during the same time interval, our method generates trajectories that ensure the two UAS do so without crashing into each other without any user based scheduling of which drones enters first.

The tool presented here relies on interpreting a mission as a STL specification and generating trajectories that satisfy it. While there are multiple methods and tools that aim to solve such a problem, e.g. Mixed Integer Programming-based [7] and based on stochastic heuristics [8], we use an underlying method [4] that is tailored for generating trajectories for multi-rotor UAS, including those that allow hovering, to satisfy STL specifications in continuous-time. A detailed comparison can be found in [4, 9].

1.2 Contributions

With this proposed tool we aim to bridge the gap between the ease-of-use of the UAS mission planning software popular among amateur drone enthusiasts, and the capabilities of academic tools [7, 8] for control/planning with STL specifications. By doing this, we generate trajectories for multi UAS fleets that can satisfy complicated mission requirements while providing strong guarantees on mission satisfaction as well as the ability of the multi-rotor UAS to follow out their planned trajectories [4]. The main contributions of our tool are:

[3] `https://github.com/yashpant/FlyByLogic`

1. An easy to use graphical interface to specify mission requirements for multi-rotor UAS fleets,
2. The ability to interpreting these as missions as STL specifications and automatically generate an optimization to maximize a notion of robustness of this STL specification,
3. By interfacing to an off-the-shelf optimization solver, generation of trajectories that satisfy the mission requirements, are optimal with respect to minimizing jerk [10], and respect (potentially different) kinematic constraints for all UAS.
4. Does not require the UAS fleet operator to know how to write specifications in STL, but through an object-oriented C++ library allows the advanced user to generate custom missions specifications with even more flexibility than the graphical interface.

‘

2 Fly-by-Logic: The tool

2.1 Architecture and outline

Figure 1 shows the architecture of the Fly-by-Logic tool. Through the user interface in MATLAB, the user defines the missions (more details in section 2.2). The mission specific spatial and temporal parameters are then read in by the Fly-by-Logic C++ back-end. Here, these parameters are used to generate a function for the continuously differentially approximation of the robustness of the STL specification associated with the mission. An optimization to maximize this function [4] value is then formulated in Casadi [11]. Solving this optimization via IPOPT [12] results in a sequence of way-points for every UAS (uniformly apart in time). Also taken into account in the formulation is the motion to connect these way-points, which is via jerk-minimizing splines [10] and results in trajectories for each UAS. Through the Fly-by-Logic library, the (original non-smooth) robustness of these trajectories is evaluated for the mission STL specification and displayed back to the user via the MATLAB interface. A positive value of this robustness implies that the generated trajectories satisfy the mission and can be flown out, while a negative value (or 0) implies that the trajectories do not satisfy the mission [13] and either some additional parameters need to be tweaked (e.g. allowable velocity and acceleration bounds for the UAS, time intervals to visit regions, or a constant for the smooth robustness computation) or that the solver is incapable of solving this particular mission from the given initial positions of the UAS.

2.2 The mission template

Through the interface, the user starts by defining the number of way-points N (same number for each drone), as well well as the (fixed) time, T that the UAS take to travel from one way-point to the next. These way-points are the variables that the tool optimizes over, and the overall duration of the mission is

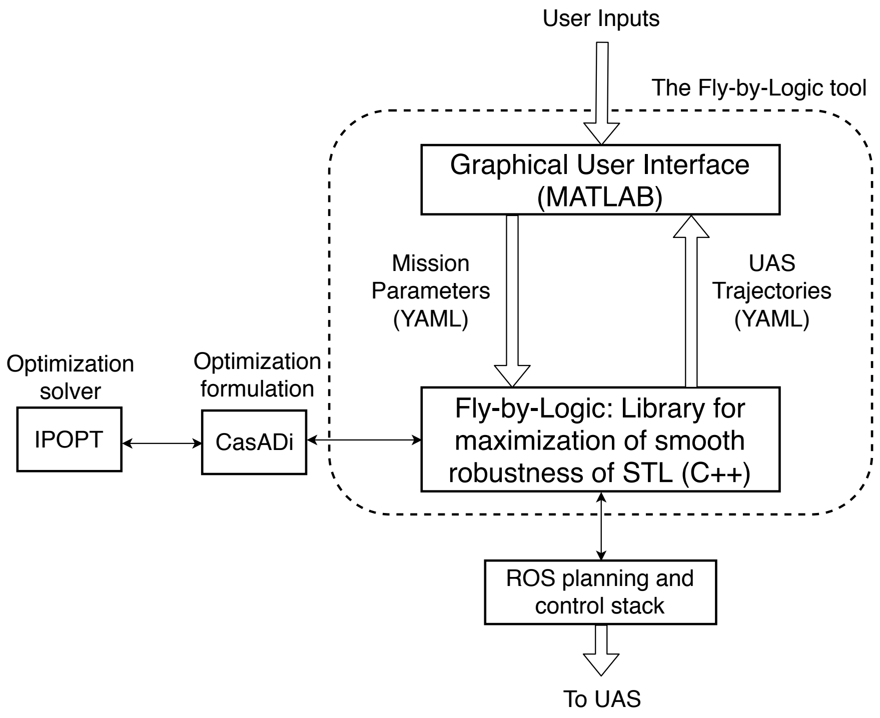
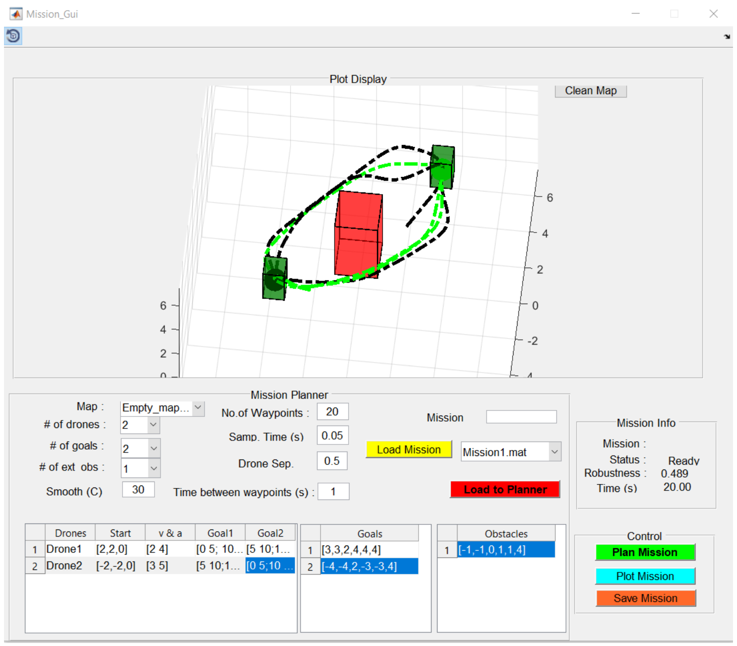

User Inputs
The Fly-by-Logic tool
Graphical User Interface (MATLAB)
Mission Parameters (YAML)
UAS Trajectories (YAML)
Optimization solver
Optimization formulation
IPOPT
CasADi
Fly-by-Logic: Library for maximization of smooth robustness of STL (C++)
ROS planning and control stack
To UAS

Fig. 1. The Fly-by-Logic tool-chain. Through a MATLAB-based graphical interface (figure 2), the user defines the workspace and the multi UAS mission. This mission is interpreted as an STL specification (of the form in equation 1), the parameters of which are passed from the interface to the Fly-by-Logic C++ library. Through interfacing with off-the-shelf optimization tools, trajectories that satisfy the mission are generated for each UAS and visualized through the user interface. The way-points that generate these trajectories can also be sent to a Robot Operating Systems (ROS) implementation of trajectory following control to be deployed on board actual robots (e.g. `bit.ly/varvel8`).

then $H = NT$ seconds. Next, the user defines regions in a bounded 3-dimensional workspace (see figure 2). These regions are axis-aligned hyper-rectangles and can be either *Unsafe* no-fly zones (in red), or *Goal* regions that the UAS can fly to. For each UAS, the user specifies their starting position in the workspace, as well as the velocity and acceleration bounds that their respective trajectories should respect. Finally, the user also specifies the time intervals within which the UAS need to visit some goal sets.

Through the user interface, the user-defined missions result in specifications corresponding to the following fragment of STL:

$$\begin{aligned}\varphi = &\wedge_{u=1}^{U} \wedge_{d=1}^{D} (\Box_I \neg(p_d \in \text{Unsafe}_u)) \wedge \wedge_{d \neq d'} (\Box_I (||p_d - p_{d'}||_2 \geq d_{\min})) \wedge \\ &\wedge_{g=1}^{G} \wedge_{d=1}^{D} (\Diamond_{I^1_{g,d}} (p_d \in \text{Goal}_g) \wedge \ldots \wedge \Diamond_{I^c_{g,d}} (p_d \in \text{Goal}_g))\end{aligned} \tag{1}$$



Map : Empty_map...
# of drones : 2
# of goals : 2
# of ext obs : 1
Smooth (C) 30
No.of Waypoints : 20
Samp. Time (s) 0.05
Drone Sep. 0.5
Time between waypoints (s) : 1
Mission
Mission1.mat

Mission :
Status : Ready
Robustness : 0.489
Time (s) 20.00

	Drones	Start	v & a	Goal1	Goal2
1	Drone1	[2,2,0]	[2 4]	[0 5; 10...	[5 10;1...
2	Drone2	[-2,-2,0]	[3 5]	[5 10;1...	[0 5;10 ...

	Goals
1	[3,3,2,4,4,4]
2	[-4,-4,2,-3,-3,4]

	Obstacles
1	[-1,-1,0,1,1,4]

Fig. 2. The user interface and the planned trajectories for a two UAS patrolling mission (see example 1). Real-time playback can be seen at `http://bit.ly/fblguiexmpl`

Here, D, U, G are the number of UAS, *Unsafe* sets and *Goal* sets in the mission respectively. $I = [0, NT]$ is an interval that covers the entire mission duration, while $I^i_{g,d} \subseteq I$, $\forall i = 1, \dots, c$ is the i^{th} interval in which UAS d must visit *Goal* g. $\neg$ is the boolean negation operator. p_d is the position of UAS d.

The symbol $\Box_I \phi$ corresponds to the *Always* operator of STL and encodes the requirement that a boolean formula ϕ should be *true* through the time interval I. We use this operator to enforce that the UAS never enter the *Unsafe* zones or get closer than $d_{\min}$ meters of each other. Similarly, $\Diamond_I \phi$ corresponds to the *Eventually* operator which encodes the requirement that ϕ should be true at some point in time in the interval I. We use this to capture the requirement that the a UAS visits a *Goal* region within the user defined interval I. More details on STL and its grammar can be found in [14].

Example 1 ***Two UAS patrolling mission.*** *Two UAS, starting off at positions* $[2, 2, 0]$ *and* $[-2, -2, 0]$*, are tasked with patrolling two sets (in green), while making sure not to enter the set in red, and also maintaining a minimum distance of* $0.5m$ *from each other. For a mission of time* $20s$*, we set the number of way-points to* 20*, and the time between them to be* $1s$*. The timing constraints on the patrolling are as follows: UAS 1 has to visit the first set in green in an interval of time* $[0,\ 5]$ *seconds from the missions starting time, has to visit the other green set in the interval* $[5,\ 10]$ *seconds, re-visit the first set in the interval* $[10,\ 15]$*, and the second set again in the interval* $[15,\ 20]$*. UAS 2 has a similar mission, visiting the first set in the intervals the UAS 1 has to visit the* *second set and so on. Figure 2 shows the trajectories generated by our method, and* ***http://bit.ly/fblguiexmpl*** *shows a real-time playback of the planned trajectories visualized through the user interface.*

For the mission of example 1, the temporal logic specification is:

$$\begin{aligned}\varphi = &\wedge_{u=1}^{2} \wedge_{d=1}^{2} (\square_{[0,20]} \neg (p_d \in \text{Unsafe}_u)) \wedge \square_{[0,20]} (||p_1 - p_2||_2 \geq 0.5) \wedge \\ &\Diamond_{[0,5]} (p_1 \in \mathit{Goal}_1) \wedge \Diamond_{[5,10]} (p_1 \in \mathit{Goal}_2) \wedge \Diamond_{[10,15]} (p_1 \in \mathit{Goal}_1) \wedge \\ &\Diamond_{[15,20]} (p_1 \in \mathit{Goal}_2) \wedge \Diamond_{[0,5]} (p_2 \in \text{Goal}_2) \wedge \Diamond_{[5,10]} (p_2 \in \text{Goal}_1) \wedge \\ &\Diamond_{[10,15]} (p_2 \in \text{Goal}_2) \wedge \Diamond_{[15,20]} (p_2 \in \text{Goal}_1)\end{aligned} \tag{2}$$

The tool comes pre-loaded with some example missions, and offers the user the ability to save new missions, as well save and load workspaces as text files. More details on the usage of the tool are in [15].

Note: Through the C++ library that forms the back-end for the tool, specifications involving the nested operators $\square_{I_1} \Diamond_{I_2}$ and $\Diamond_{I_1} \square_{I_2}$ can be used in conjunction with the template of equation 1. This functionality will be added to the user interface at a later time.

2.3 Behind-the-scenes: Generating the trajectories

In order to generate the trajectories that satisfy the mission specification, an optimization is solved (in the C++ back-end) to maximize, over N way-points for each drone, the smooth robustness of the mission STL specification evaluated for the UAS trajectories of NT seconds in duration. The constraints in the optimization ensure that the resulting trajectories are such that the resulting trajectories have velocity and accelerations within the user-defined bounds for each UAS, i.e. are kinematically feasible for the UAS to fly. See [4] for details.

3 Conclusions and ongoing work

In this paper we presented Fly-by-Logic, a tool for planning for multi-rotor UAS missions. By interpreting the missions as STL specifications, the underlying method generates kinematically feasible trajectories to satisfy missions with complicated spatial and temporal requirements while ensuring safety. Through an example, we introduce the kind of missions that can be specified in the tool. At the time of writing this paper, the tool is suitable only for offline trajectory generation for UAS missions. In [4] the underlying method has been shown to work in an online manner as well (see `bit.ly/varvel2`), and current work on the tool is focused on wrapping the Fly-by-Logic C++ library as a ROS package to seamlessly integrate with off-the-shelf planning and control implementations. Also planned is a method to import 3-d maps for actual geographical locations with *Unsafe* zones covering landmarks.

References

1. Federal Aviation Administration, “Concept of operations v1.0,” https://utm.arc.nasa.gov/docs/2018-UTM-ConOps-v1.0.pdf, 2018, accessed: 2018-11-19.
2. O. Maler and D. Nickovic, *Monitoring Temporal Properties of Continuous Signals.* Springer Berlin Heidelberg, 2004.
3. G. Fainekos, “Robustness of temporal logic specifications,” Ph.D. dissertation, University of Pennsylvania, 2008. [Online]. Available: http://www.public.asu.edu/~gfaineko/pub/fainekos_thesis.pdf
4. Y. V. Pant, H. Abbas, R. A. Quaye, and R. Mangharam, “Fly-by-logic: control of multi-drone fleets with temporal logic objectives,” in *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems.* IEEE Press, 2018, pp. 186–197.
5. “Ardupilot mission planner,” ardupilot.org/planner/, accessed: 2018-12-15.
6. “QGROUNDCONTROL intuitive and powerful ground control station for px4 and ardupilot uavs,” qgroundcontrol.com, accessed: 2018-12-15.
7. V. Raman, A. Donze, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia, “Model predictive control with signal temporal logic specifications,” in *53rd IEEE Conference on Decision and Control*, Dec 2014, pp. 81–87.
8. Y. Annapureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, “S-taliro: A tool for temporal logic falsification for hybrid systems,” in *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems: Part of the Joint European Conferences on Theory and Practice of Software*, ser. TACAS’11/ETAPS’11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 254–257. [Online]. Available: http://dl.acm.org/citation.cfm?id=1987389.1987416
9. Y. V. Pant, H. Abbas, and R. Mangharam, “Smooth operator: Control using the smooth robustness of temporal logic,” in *Control Technology and Applications (CCTA), 2017 IEEE Conference on.* IEEE, 2017, pp. 1235–1240.
10. M. W. Mueller, M. Hehn, and R. DÁndrea, “A computationally efficient motion primitive for quadrocopter trajectory generation,” in *IEEE Transactions on Robotics*, 2015.
11. J. Andersson, “A General-Purpose Software Framework for Dynamic Optimization,” PhD thesis, Arenberg Doctoral School, KU Leuven, 2013.
12. A. Wächter and L. T. Biegler, “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming,” *Mathematical Programming*, 2006.
13. G. Fainekos and G. Pappas, “Robustness of temporal logic specifications for continuous-time signals,” *Theoretical Computer Science*, 2009.
14. A. Donzé and O. Maler, *Robust Satisfaction of Temporal Logic over Real-Valued Signals.* Springer Berlin Heidelberg, 2010.
15. “Fly-by-logic: User documentation,” https://github.com/yashpant/FlyByLogic, accessed: 2018-12-15.