Hybrid Control of a Survey Vehicle in a Constrained Environment

Ethan Burnett

May 8, 2019

1 Introduction

This project approaches the problem of designing a vehicle control system to satisfy mission objectives specified with linear temporal logic (LTL) statements. The task will be to guide and control the vehicle in an environment consisting of survey regions (S), restricted regions (K), and a home station area (R). The domain is partitioned into triangles (simplices), which enable path planning in the domain via a steering law, which provides position and velocity commands for an intermediate-level controller to track. The hybrid control design is performed on the first order "steering" system, resulting in the construction of satisfying trajectories in the domain. The triangulation of the domain is achieved by defining all points of interest (constituting the boundaries of important regions), which become vertices for the triangles. This procedure should be applicable regardless of the shape of the interest/constraint regions in the domain, so long as sufficiently many points are defined for a suitable triangulation. While this problem serves as a simple basis for the course project, additional details and concepts to be explored will scale the complexity and rigor of the work appropriately.

The project assumes simple 2D second-order vehicle dynamics with acceleration in the x and y directions freely chosen (but perhaps bounded). Such dynamics could easily describe a simplified model of a ground vehicle, or a simplified model of a UAV with dynamics linearized about the equilibrium hover condition (neglecting the effects of small change in altitude). This linearization is consistent with a medium-fidelity representation of quadcopter dynamics.[1] Quadcopters are an ideal platform for linear control methods, since their operation typically requires only small angular deviations from the hovering condition. However, this project tests hybrid control techniques and methods, so more emphasis is placed on these concepts than on high-fidelity representations of particular vehicle dynamics.

2 Defining and Discretizing the Domain

While regions to survey and avoid can generally have any 2D shape, sets of points can be defined just outside these regions such that their convex hull forms a polytope. Furthermore, the bounding region can be triangulated. In this project, the Delaunay triangulation method is used to discretize the domain. This project assumes that the regions to survey and avoid can be bounded by single triangles, as shown in Fig. 1. A fine triangular mesh of the regions is also possible – enabling trajectories to be developed to fully explore within S_1 or S_2, for example. The work in this project could be extended for application to this more complicated case without any conceptual changes.

The green nodes in Fig. 1 mark the extent of the domain and outline the survey, restricted, and home station regions. The triangularization of the domain has been performed, with the edges given by the blue lines. The code used to generate this triangularization has been written such that it works for any number of user-defined nodes in any arbitrary 2D domain. In Fig. 2, each node is a vertex of 2 or more triangles. All nodes are defined in an array, starting with v_0 and ending with v_{N-1} for N nodes. Each simplex has

been labeled with an identifier and a list of the nodes that form its vertices, ordered counter-clockwise. For example, $T_3(7,4,1)$: S_1 means that T_3 (the fourth simplex) is one of the regions to be surveyed, S_1. Furthermore, for T_3, the first vertex is the eighth node (v_7), the second vertex is the fifth node (v_4), and the third vertex is the second node (v_1). While the nodes retain the same numbering as originally entered by the user, the triangle data (ID number, counter-clockwise node order, etc.) is auto-generated by the Delaunay triangulation function.

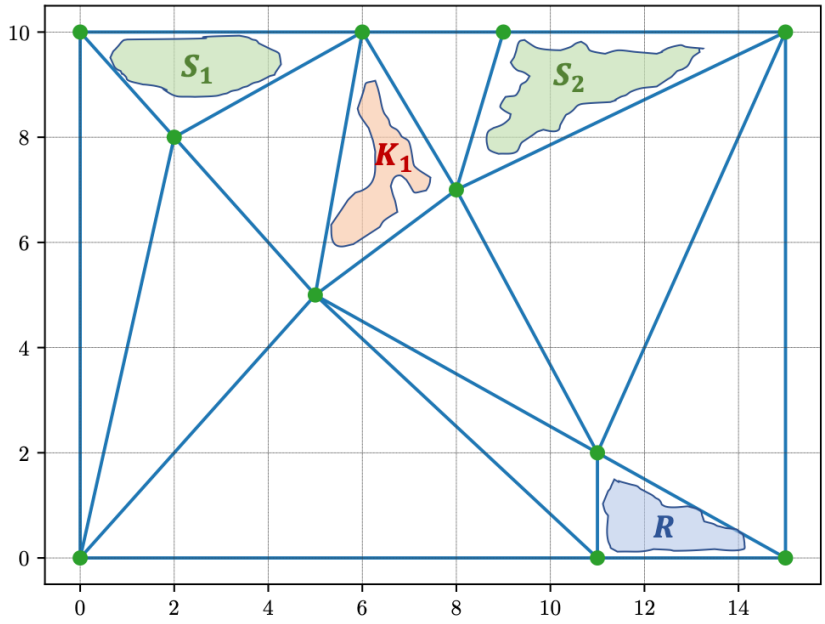

Figure 1: Survey Domain with Nodes, Mesh

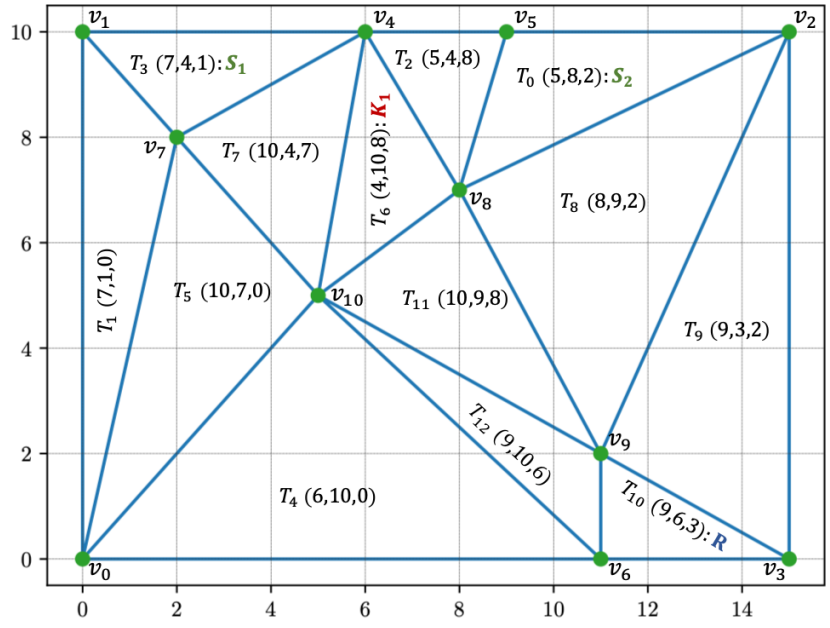

Figure 2: Survey Domain, Partitioned, Annotated

Now that the domain has been broken into triangular regions (which the lower-level controls should render

as invariants), it will be possible to describe both the system and any desired behavior of the system more simply and formally.

3 Performance Specifications

In this project, the desired behavior of the system is specified by linear temporal logic (LTL) statements. Before these statements are discussed, a quotient of the system is defined, assuming that lower-level control renders each simplex as an invariant region (in other words, the control does not permit unexpected transitions out of a simplex). In the absence of unmodeled disturbances, the quotient is a deterministic finite automaton – with at most three possible control actions per state (region). These control actions are labeled u_1, u_2, and u_3 in Fig. 3, where u_i indicates a controlled exit of the triangular region through the facet opposite the i$^{\text{th}}$ vertex of the triangle. Note that the region numbering in Fig. 3 matches the numbering in Fig. 2. Namely, S_1 is T_3, S_2 is T_0, K_1 is T_6, and R is T_{10}.

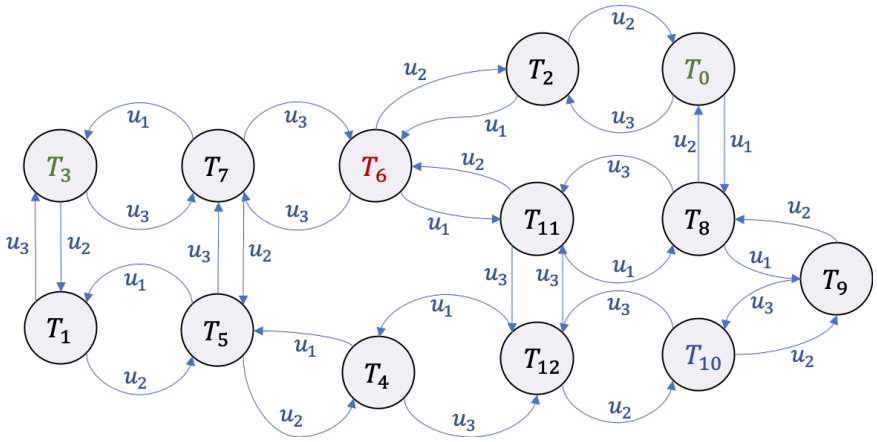

Figure 3: Quotient Induced by Partition and Polytope Control

To formalize the specification, each region of interest must be specified by a set of predicates. There are multiple ways to do this. It would be possible to describe the triangular regions of interest with 9 linear inequalities of the form $\pi_i : \ c_i a + d_i < 0, \ i = 1, 2, \ldots, 9$, but it is more straightforward to write the LTL specifications in terms of the region observations S_i, K_j, R, D for i survey regions, j restricted regions, a home area R, and accessible but otherwise unremarkable parts of the domain D. For example, the following reasonable behaviors may be described by their accompanying LTL formulas:

Specification 1: Visit S_1, S_2, and R in any order, while always avoiding K_1.

$$\varphi_1 = \diamond S_1 \wedge \diamond S_2 \wedge \diamond R \wedge \square \neg K_1 \tag{1}$$

Specification 2: Simultaneous occurrences of two or more from (S_1, S_2, R) are not permitted.

$$\varphi_2 = \square \neg (S_1 \wedge S_2) \wedge \square \neg (S_1 \wedge R) \wedge \square \neg (S_2 \wedge R) \tag{2}$$

Specification 3: Always avoid K_1, and eventually visit S_1, then S_2, then R.

$$\varphi_3 = \diamond (S_1 \wedge \diamond (S_2 \wedge \diamond (R))) \wedge \square \neg K_1 \tag{3}$$

More complex behaviors may be described using other formulas, and the combination of Specifications 1 and 2 or 3 and 2 is also possible. Note that the language of φ_3, $L(\varphi_3)$, is a subset of $L(\varphi_1)$. Note also that LTL formulations in the form of Specification 2 are often unnecessary, since the dynamic model will enforce that such simultaneous states cannot occur after the product automaton is formed.

In this project, a code was written such that the user may specify deterministic automatons based on simple, non-recurrent LTL formulas. This was tested on the following slightly simpler formula:

Specification 4: Always avoid K_1, and eventually visit S_1, then R.

$$\varphi_4 = \diamond(S_1 \wedge \diamond(R)) \wedge \square\neg K_1 \tag{4}$$

Formula φ_4 is used to generate the deterministic automaton in Fig. 4. (See *https://spot.lrde.epita.fr/app/*)

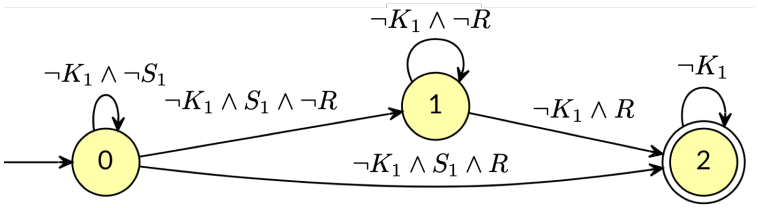

Figure 4: Deterministic Büchi Automation from LTL Formula φ_4

In Section 5 of the report, the transition system T (based on the domain partitioning) and the automaton A_{T_4} will be explicitly defined, and the product automaton will be built from these.

4 Steering Law and Control on Polytopes

4.1 Path-Planning for the Vehicle

The concept of a steering law is to use a lower-order system to develop desirable trajectories to be tracked by a higher-order controller. The simplified vehicle obeys second-order dynamics of the form $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, t)$ with $\boldsymbol{x} = (x, y, \dot{x}, \dot{y})$ and $\boldsymbol{u} = (u_1, u_2)$ representing control accelerations in the x and y directions. The simplest form of such a system obeys the linear form $\dot{\boldsymbol{x}} = [A]\boldsymbol{x} + [B]\boldsymbol{u}$:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{pmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{pmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{pmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \tag{5}$$

The generalized linear second-order 2D dynamics are consistent with a simple model of ground vehicle dynamics. While 2D vehicle dynamics are usually described in a polar form with variable velocity v and steering angle θ, the simultaneous control of both of these quantities ($\dot{v} = a$, $\dot{\theta} = \omega$) still yields a variable control acceleration vector. Eq. 5 is also consistent with a linearized model of quadcopter dynamics.

4.1.1 Quadcopter Dynamics

For the quadcopter, the nonlinear dynamics are briefly described below:[1]

$$\begin{aligned}
\ddot{x} &= (\sin\psi \sin\phi + \cos\psi \sin\theta \cos\psi)\,\frac{U_1}{m} \\
\ddot{y} &= (-\cos\psi \sin\phi + \sin\psi \sin\theta \cos\phi)\,\frac{U_1}{m} \\
\ddot{z} &= -g + (\cos\theta \cos\psi)\,\frac{U_1}{m} \\
\ddot{\phi} &= \frac{I_{yy} - I_{zz}}{I_{xx}}\dot{\theta}\dot{\psi} - \frac{I_p}{I_{xx}}\dot{\theta}\tilde{\Omega} + \frac{U_2}{I_{xx}} \\
\ddot{\theta} &= \frac{I_{zz} - I_{xx}}{I_{yy}}\dot{\phi}\dot{\psi} - \frac{I_p}{I_{yy}}\dot{\phi}\tilde{\Omega} + \frac{U_3}{I_{yy}} \\
\ddot{\psi} &= \frac{I_{xx} - I_{yy}}{I_{zz}}\dot{\phi}\dot{\theta} + \frac{U_4}{I_{zz}}
\end{aligned} \tag{6}$$

where the I_{kk} terms are the body moments of inertia about the corresponding k body axis, I_p is the moment of inertia of the propeller, and ψ, θ, ϕ are the yaw, pitch and roll angles. The control quantities U_i and angular velocity grouping term $\tilde{\Omega}$ are defined in terms of the 4 propeller angular velocities Ω_i below:

$$\begin{aligned} U_1 &= b\left(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2\right) \\ U_2 &= bl\left(\Omega_4^2 - \Omega_2^2\right) \\ U_3 &= bl\left(\Omega_3^2 - \Omega_1^2\right) \\ U_4 &= d\left(\Omega_2^2 - \Omega_1^2 + \Omega_4^2 - \Omega_3^2\right) \end{aligned} \tag{7}$$

$$\tilde{\Omega} = \Omega_2 - \omega_1 + \Omega_4 - \Omega_3 \tag{8}$$

where b, d represent the lift and drag factors, and l is the distance between two opposite propellers.

If it is assumed that the maneuvers are sufficiently gentle that all desired attitude maneuvers can be performed by the motors, then consideration of the motor dynamics is unnecessary. Then, the unperturbed dynamics linearized about the hover condition have the following simple form in terms of the pitch and roll θ and ϕ and gravitational acceleration g:

$$\begin{aligned} \ddot{x} &= g\theta \\ \ddot{y} &= -g\phi \\ \ddot{z} &= 0 \end{aligned} \tag{9}$$

The similarity of Eq. 9 to Eq. 5 is obvious (indeed, the same form can be obtained by a change of variables to absorb the g terms). Note that the non-dominant effect of small attitude control variation on the vertical motion is not captured by the linearization, and the inclusion of state-dependent perturbations would result in the appearance of additional terms in the linearized dynamics (i.e. nonzero kinetic terms in the $[A]$ matrix in the linearized equations $\dot{\boldsymbol{x}} = [A]\boldsymbol{x} + [B]\boldsymbol{u}$).

Overall, the primary purpose of this project is to demonstrate concepts from the Hybrid Systems course, so high-fidelity system dynamics are not considered. The 2D double-integrator dynamical system in Eq. 5 serves as a suitable first-order analog for more complex vehicles for the purposes of this project.

4.1.2 Tracking the Steering Law

The task of the steering law is to obtain a 2D trajectory to track: $(\boldsymbol{r}(t),\ \boldsymbol{v}(t))$, where $\boldsymbol{r} = [x^*, y^*]^\top$ and $\boldsymbol{v} = \dot{\boldsymbol{r}}$. By design, the system obeys simple first-order integrator dynamics: $\dot{\boldsymbol{r}} = [0_{2\times2}]\boldsymbol{r} + [I_{2\times2}]\boldsymbol{v}$, where $\boldsymbol{v}$ is the "control" velocity of this first-order system. This steering system is used with the triangulated domain and the polytope control strategy (to be discussed) to obtain deterministic velocity laws in each simplex $\boldsymbol{v} = [F]\boldsymbol{x} + \boldsymbol{g}$ driving the position to exit through a specified facet from anywhere within the simplex. Then, the deterministic desired trajectory is tracked in each simplex by an intermediate-level controller (i.e. second-order dynamics with linear tracking control law for this project). The task of the tracking control is to drive the error dynamics to zero for the generalized system in Eq. 5:

$$\boldsymbol{e} = \boldsymbol{x} - \begin{pmatrix} \boldsymbol{r} \\ \boldsymbol{v} \end{pmatrix},\ \boldsymbol{v}(t) = [F]\boldsymbol{r}(t) + \boldsymbol{g},\ \boldsymbol{r}(t) = [\Phi_F(t,0)]\boldsymbol{r}_0 + \int_0^t [\Phi_F(\tau,0)]d\tau \cdot \boldsymbol{g} \tag{10}$$

$$\dot{\boldsymbol{e}} = [A]\boldsymbol{x} + [B]\boldsymbol{u} - \begin{pmatrix} \boldsymbol{v} \\ \dot{\boldsymbol{v}} \end{pmatrix} = [A]\boldsymbol{e} + [B]\boldsymbol{u} - \begin{pmatrix} \boldsymbol{0}_{2\times1} \\ \dot{\boldsymbol{v}} \end{pmatrix} \tag{11}$$

The stabilizing control to track the reference is given by:

$$\boldsymbol{u} = -[K]\boldsymbol{e} + \dot{\boldsymbol{v}} \tag{12}$$

The gain matrix $[K]$ can be chosen such that the linearized error dynamics are critically damped. Note that these results assume that $[A]$ and $[B]$ are as given in Eq. 5. Also, in this representation, $t = 0$ when the simplex is entered (control is discontinuous across the facets). Since the system is linear, the steering law and control performance criteria can be chosen to guarantee mitigation of the error discontinuities at the transition between triangular regions. Now, it becomes necessary to discuss the trajectory design within each simplex.

4.2 Control on Polytopes

For affine dynamical systems, it is often possible to design control to particular facets of a polytope. Previous work[4] studied the affine control system $\dot{\boldsymbol{x}} = [A]\boldsymbol{x} + [B]\boldsymbol{u} + \boldsymbol{a}$, $\boldsymbol{x}(0) = \boldsymbol{x}_0$ with $[A] \in \mathbb{R}^{N\times N}$, $[B] \in \mathbb{R}^{N\times m}$, and $\boldsymbol{a} \in \mathbb{R}^N$, and at every time instant in a certain interval, the state $x \in \mathbb{R}^N$ is assumed to be contained in the polytope P_N. The work studied the control problem of steering the state to a particular facet of P_N from any point within P_N. The work also showed that if the polytope is a full-dimensional simplex, the control-to-facet input at every point in the simplex can be written as a convex combination of the control at the vertices, and provided a strategy for obtaining the vertex control and resultant convex combination. For systems with $N = 2$, this becomes the problem of controlling to a particular triangular facet using a convex combination of the control at the three vertices. While the formulas given in the paper are for an N-dimensional polytope, for a triangle they result in the following conditions for control out of facet F_1, with vertices numbered $\boldsymbol{v}_i$, $i = 1, 2, 3$, and control at each vertex indicated by $\boldsymbol{u}_i$:

$$\begin{aligned}
&\boldsymbol{n}_2^\top [B]\boldsymbol{u}_1 \leq -\boldsymbol{n}_2^\top ([A]\boldsymbol{v}_1 + \boldsymbol{a}) \\
&\boldsymbol{n}_3^\top [B]\boldsymbol{u}_1 \leq -\boldsymbol{n}_3^\top ([A]\boldsymbol{v}_1 + \boldsymbol{a}) \\
&\boldsymbol{n}_2^\top [B]\boldsymbol{u}_1 + \boldsymbol{n}_3^\top [B]\boldsymbol{u}_1 < -\boldsymbol{n}_2^\top ([A]\boldsymbol{v}_1 + \boldsymbol{a}) - \boldsymbol{n}_3^\top ([A]\boldsymbol{v}_1 + \boldsymbol{a})
\end{aligned} \tag{13}$$

$$\begin{aligned}
&\boldsymbol{n}_1^\top [B]\boldsymbol{u}_2 > -\boldsymbol{n}_1^\top ([A]\boldsymbol{v}_2 + \boldsymbol{a}) \\
&\boldsymbol{n}_3^\top [B]\boldsymbol{u}_2 \leq -\boldsymbol{n}_3^\top ([A]\boldsymbol{v}_2 + \boldsymbol{a})
\end{aligned} \tag{14}$$

$$\begin{aligned}
&\boldsymbol{n}_1^\top [B]\boldsymbol{u}_3 > -\boldsymbol{n}_1^\top ([A]\boldsymbol{v}_3 + \boldsymbol{a}) \\
&\boldsymbol{n}_2^\top [B]\boldsymbol{u}_3 \leq -\boldsymbol{n}_2^\top ([A]\boldsymbol{v}_3 + \boldsymbol{a})
\end{aligned} \tag{15}$$

Then, the control $\boldsymbol{u} = [F]\boldsymbol{x} + \boldsymbol{g}$ is obtained by solving the following linear equation for $[F]$ and $\boldsymbol{g}$:

$$\begin{bmatrix} \boldsymbol{v}_1^\top & 1 \\ \boldsymbol{v}_2^\top & 1 \\ \boldsymbol{v}_3^\top & 1 \end{bmatrix} \begin{bmatrix} F^\top \\ \boldsymbol{g}^\top \end{bmatrix} = \begin{pmatrix} \boldsymbol{u}_1^\top \\ \boldsymbol{u}_2^\top \\ \boldsymbol{u}_3^\top \end{pmatrix} \tag{16}$$

The procedure for control to other facets is unchanged, and can be obtained by permutation of the facet indices in Eq. 13 - 15. For the steering law system, $[A] = [0_{2\times 2}]$, thus the inequalities can be put in the following linear constraint form, with the constraint matrix also given for F_1:

$$[C_{F_i}]\boldsymbol{u} > \boldsymbol{0}_{7\times 1} \tag{17}$$

$$[C_{F_1}] = \begin{bmatrix} -\hat{\boldsymbol{n}}_2^\top [B] & \boldsymbol{0}_{1\times 2} & \boldsymbol{0}_{1\times 2} \\ -\hat{\boldsymbol{n}}_3^\top [B] & \boldsymbol{0}_{1\times 2} & \boldsymbol{0}_{1\times 2} \\ -\hat{\boldsymbol{n}}_2^\top [B] - \hat{\boldsymbol{n}}_3^\top [B] & \boldsymbol{0}_{1\times 2} & \boldsymbol{0}_{1\times 2} \\ \boldsymbol{0}_{1\times 2} & \hat{\boldsymbol{n}}_1^\top [B] & \boldsymbol{0}_{1\times 2} \\ \boldsymbol{0}_{1\times 2} & -\hat{\boldsymbol{n}}_3^\top [B] & \boldsymbol{0}_{1\times 2} \\ \boldsymbol{0}_{1\times 2} & \boldsymbol{0}_{1\times 2} & \hat{\boldsymbol{n}}_1^\top [B] \\ \boldsymbol{0}_{1\times 2} & \boldsymbol{0}_{1\times 2} & -\hat{\boldsymbol{n}}_2^\top [B] \end{bmatrix} \tag{18}$$

Similar forms for facets F_2 and F_3 may be found in the codes also submitted. This system of inequalities is solved to obtain satisfactory $\boldsymbol{u}_1$, $\boldsymbol{u}_2$, $\boldsymbol{u}_3$ by performing constrained optimization (from the Python SciPy package, using *optimize.minimize*) of the following cost function:

$$J = \sum_{i=1}^{3} (c - \|\boldsymbol{u}_i\|)^2 \tag{19}$$

where c is the desired reference velocity at all three of the vertices. In general, the simultaneous zeroing of these three velocity conditions is not possible, and the optimal $J > 0$. This cost function is only introduced to enable a unique solution to the control problem, and to scale the norm of the spatially-dependent velocity in the triangle to be similar to c.

It is also possible to implement control to remain inside the simplex, by solving for unique values of $\boldsymbol{u}_1$, $\boldsymbol{u}_2$, $\boldsymbol{u}_3$ satisfying the following constraints:[3]

$$\forall i, j,\ i \neq j:\ \hat{\boldsymbol{n}}_i^\top \left([A]\boldsymbol{v}_j + [B]\boldsymbol{u}_j + \boldsymbol{a}\right) \leq 0 \tag{20}$$

For the steering law, the following simpler form is obtained:

$$\forall i, j,\ i \neq j:\ \hat{\boldsymbol{n}}_i^\top [B]\boldsymbol{u}_j \leq 0 \tag{21}$$

This can be used to construct a constraint matrix similar to Eq. 18, for use with an identical constrained optimization procedure as for solving the control-to-facet problem.

4.3 Planned Path Smoothing and Fourier Series Fitting

One problem that was originally encountered in this project was that the generated trajectories had discontinuities in velocity at the transition between simplices, or between modes within a simplex. For realistic applications, this may not be acceptably tracked by the intermediate-level controller due to dynamics or control-authority constraints. This could result in poor tracking and overshoot at the simplex/mode transitions – even entering of violating regions in extreme cases. Fortunately, the problem of smoothing planned paths is a somewhat well-studied problem, and multiple techniques have been proposed and implemented for taking a non-continuously differentiable path and correcting it (using interpolation or splines).[5] It is even feasible to do this in a constrained manner such that the trajectory is not smoothed into a violating region.

In this project, a Fourier series fit of the planned path was used to do path smoothing. The number of frequencies to use is set by the user, but the code is written such that any number may be used. In general, using 20 – 30 distinct frequencies is sufficient to closely reproduce the planned path while still smoothing the discontinuities. By the definition of the Fourier series, the smoothed trajectory analytically approaches the sharp generating trajectory as the order of the series fit is increased. While Fourier series are typically used to fit periodic data, the generally non-periodic path states $x(t)$ and $y(t)$ may also be fit for $t \in [0, T]$ by duplicating the signal, flipping it from left to right, and appending it to the original signal $x(t)$ or $y(t)$. This results in the construction of a symmetric $2T$-periodic signal, where T is the original planned traverse time from the discontinuous planned path data, and the Fourier series need only be evaluated on $t \in [0, T]$. The time $t \in [T, 2T]$ generates the reversed path signals $x'(t) = x(2T - t)$ and $y'(t) = y(2T - t)$, and these may be ignored. The trajectory signal will generally always be well-fit by a sufficiently extensive Fourier series, regardless of its shape. A (typically minor) consequence of this approach is that the initial path point (x_0, y_0) is moved from the point specified by the initial conditions to the point on the Fourier series fit at the initial time. This issue can be completely avoided by an alternate flipping and appending signal lengthening strategy, but there is insufficient time or need to make that change for this course project.

One last benefit of the Fourier series smoothing of the steering law path is that the resulting smoothed path can be mathematically described at any evaluation time using the Fourier coefficients, current time, and path duration (T) alone. This removes any error that could result in tasking the higher-level controller with tracking a path only represented with discretized signal data. This also removes the need to consider discrete transitions between simplices, so for the higher-order system control, the problem is reduced to tracking an analytic time-explicit path description. The important safety properties of the first-order steering law system are inherited by the higher-order closed-loop controlled system, assuming that it has sufficient control authority and disturbance-rejection qualities to follow the planned safe path.

4.4 Additional Notes

Since the nodes in the domain are defined based on the boundaries of regions of interest, the natural question arises of how this work generalizes if the nodes move in particular ways, corresponding to changing conditions in the domain. For the sake of this argument, assume that the mapping of node number to triangle ID is unchanged – e.g. the domain is not "re-meshed". If the time-varying behavior of the node positions

is known and sufficiently well-behaved (i.e. positions are not swapped), there should exist a time-varying coordinate transformation for any given triangle from an invariant master coordinate system to the 2D space, in which the triangle distorts and translates. Perhaps it would be possible to perform the facet-based control design in the master coordinate system, and the control-to-facet and remain-inside control laws could be extended to a system with moving geometry through the known coordinate transformation. If time permits, this concept would be interesting to investigate.

For systems with sufficient control authority and reasonably small unmodeled disturbances, the tracked steering law strategy used in this project is a potentially attractive way to generate satisfying behavior in a computationally efficient way. It seems efficient since the control in each simplex can be obtained by solving a low-dimensional, linear-constrained, quadratic-cost optimization problem. Examples of potentially appropriate systems for application are (1) quadcopters in relatively calm weather, (2) ground vehicles, and (3) spacecraft in planar terminal rendezvous conditions (i.e. just before docking, or while inspecting/servicing another spacecraft).

5 Formal Definitions, Product Automaton, and Control Strategy

5.1 Transition System, Büchi Automaton, and Product Automaton

The transition system T is now formally defined:[2]

$$T = (X, \Sigma, O, o, \delta) \tag{22}$$

where X is the set of states, Σ is the set of input symbols (controls), O is the set of observations, $o : X \to O$ is the observation map, and $\delta : X \times \Sigma \to X$ is a transition function. Note that in this project, T is deterministic, so $|\delta(x, \sigma)| = 1$ i.e. for every state x, every non-blocking control action σ results in a unique transition to a particular state in X.

For the system defined in this project, $X = \{T_0, T_1, T_2, \ldots, T_{11}, T_{12}\}$. These states are represented in the code by the indices j of T_j, i.e. $X = [0, 1, 2, \ldots, 11, 12]$. These indices also correspond to the rows in the 13×3 array of triangles defined by their vertices, *tri.simplices*. Furthermore, for the system defined in this project, $\Sigma = \{u_0, u_1, u_2, u_3, u_4\}$, where u_0 represents control to remain in a simplex (self-transition) and $u_j, j = 1, 2, 3$ represents control to facet F_j. The facets F_j are defined to be opposite vertex v_j, with both auto-numbered by the triangulation code. The controls are represented by the indices j of u_j in the code. The observations are defined $O = \{R, S_1, S_2, K_1, D\}$. In the code, they are represented by string entries of the form "S1", etc.

For the system defined in this project, the observation relation is $o(T_0) = S_2$, $o(T_3) = S_1$, $o(T_{10}) = R$, $o(T_6) = K_1$, and $o(T_j) = D$ for all otherwise unlisted $j \in [0, 12]$. Lastly, the transition function is defined as the mapping $x' = \delta(x, u)$ for all admissible u (those that for a given x yield a transition that remains in X). In the code, the mapping is defined as an ordered array of all possible combinations of x, u and the resultant x', if it exists (or "-1" if it does not). The option is available for the user to show such full transition maps or to suppress their display.

The deterministic Büchi automaton A_φ is now defined:

$$A_\varphi = (Q, Q_0, F, \Sigma, R) \tag{23}$$

where Q is the set of states, Q_0 is the set of initial states, F is the set of accepting (final) states, Σ is the set of input symbols (alphabet), and $R : Q \times \Sigma \to Q$ is the transition function.

For the system defined in this project (see Fig. 4), $Q = \{q_0, q_1, q_2\}$. These states are represented in the code by the indices j of q_j, i.e. $Q = [0, 1, 2]$. In the project system, $Q_0 = \{q_0\}$ and $F = \{q_2\}$. There is only one accepting state in F allowed for the codes in this project, as well as only one initial state in Q_0. Again, the initial and final states are represented by an index, a string is not needed. In this project, Σ is the set of all 6 Boolean formulas in Fig. 4, and the code is generalized so that the user may specify the formulas

that define the transitions in Σ. Lastly, the transition function is defined as the mapping $q' = R(q, \sigma)$ for all admissible σ (those that for a given q yield either a self transition or a transition to another state in Q). In the code, the mapping is again defined as an ordered array of all possible combinations of q, σ and the resultant q', if it exists (or "-1" if it does not).

Lastly, the product automaton $P = T \otimes A_\varphi$ is defined:

$$P = (Q_P, Q_{0P}, F_P, \delta_P) \tag{24}$$

where $Q_p = X \times Q$ is the set of states of the product automaton, $Q_{0P} = X \times Q_0$ is the set of initial states, and $F_P = X \times F$ is the set of accepting (final) states. For the project, there are 156 states in the product automaton (of the form (x, q)), of which 13 are initial states and 13 are accepting states. Lastly, for the product automaton, δ_P is the transition function defined below:

$$\delta_P\left((x, q), u\right) = \{(x', q') \in Q_P | x' \in \delta(x, u) \wedge q' \in R(q, o(x'))\} \tag{25}$$

Again, all items defined in this subsection are explicitly represented and easily visualized in the code.

5.2 Solving For Satisfying Runs

To obtain trajectories that satisfy φ, we initialize in the set of accepting states F_P and perform backwards reachability analysis using the transition function δ_P to inform which predecessors in the product states are possible and which are impossible. To improve performance and avoid an explosion of the number of saved previous paths, a simple shortest-path algorithm (similar to Dijkstra's algorithm) is implemented to retain only the "product state history" branches that most quickly reach Q_{0P}.

The code is written so that a user specifies the initial second-order system state $(x_0, y_0, \dot{x}_0, \dot{y}_0)$ which is automatically converted to the proper state in Q_{0P}, along with the final simplex number T_k, for which the corresponding accepting (final) state (T_k, q_2) is obtained. The search function computes the shortest path (all variants, since typically there are multiple equal-length paths) in the domain connecting the user-specified initial condition and final state. Thus, the trajectory to be generated by the steering law will be that which leads the system from the initial condition $\{[x_0, y_0, \dot{x}_0, \dot{y}_0], q_0\}$ to a final position inside the specified end simplex, in a manner which satisfies the LTL formula φ along the way. Once the set of shortest satisfactory paths are obtained, one of the paths is picked by the code or by the user, and the data is converted to a set of instructions for the steering law trajectory-generation functions to interpret and use. The output of these functions is a smoothed satisfactory and analytic steering law trajectory through the domain for the higher-level controller to track. The tracking control is then simulated, and plots of the relevant results are displayed to the user.

6 Simulation Results

In this section, the results of simulation are presented for initial conditions $(x_0, y_0, \dot{x}_0, \dot{y}_0) = (11.0, 9.5, 0.02, 0.0)$. This puts the system in T_0 initially. The gain matrix $[k]$ is chosen such that the settling times are both $\tau = 5.0$ seconds, with closed-loop critically damped behavior to prevent overshoot. The chosen satisfying trajectory traverses through the following order of states: $(T_0, T_8, T_{11}, T_{12}, T_4, T_5, T_7, T_3, T_1, T_5, T_4, T_{12}, T_{10})$. The product automaton also yields three other unique orderings of states, j for T_j given in each of the other three cases: $[0, 8, 11, 12, 4, 5, 1, 3, 1, 5, 4, 12, 10]$, $[0, 8, 11, 12, 4, 5, 1, 3, 7, 5, 4, 12, 10]$, $[0, 8, 11, 12, 4, 5, 7, 3, 7, 5, 4, 12, 10]$.

For the chosen path, the ordering of control actions is: $(u_1, u_3, u_3, u_1, u_1, u_3, u_1, u_2, u_2, u_2, u_3, u_2, u_0)$. By inspection with Fig. 2, it is clear that these control signals given by the code correctly yield the expected ordering of states.

Figs. 5 - 10 show the successive results of the code, starting with the trajectory design and smoothing in Figs. 5 - 7 and proceeding to results from the full control of the true system. It is clear from Figs. 8 - 10 that the convergence of the true system to the desired trajectory is successful and rapid.

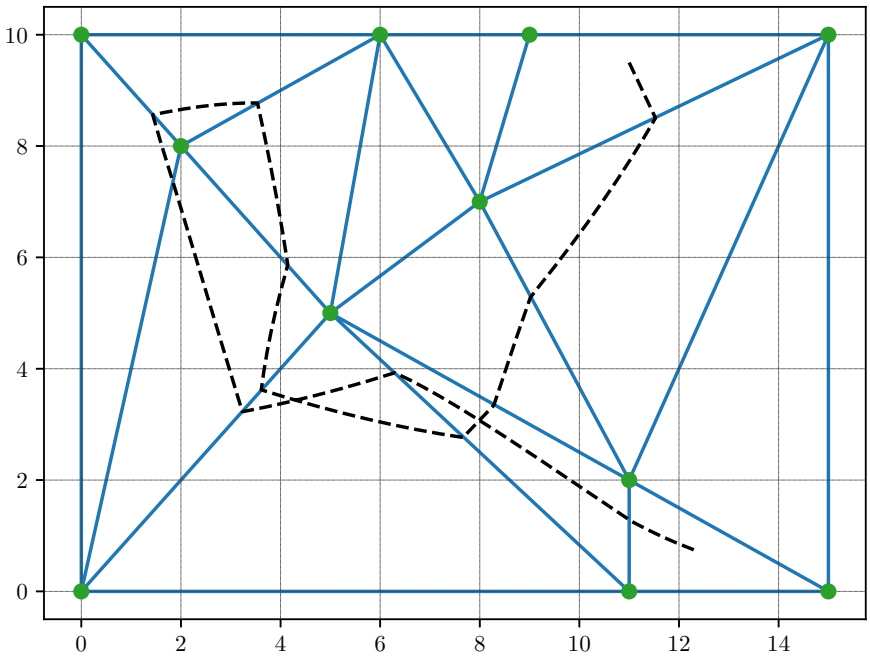

Figure 5: Partitioned Domain with Steering Law Path, Before Smoothing

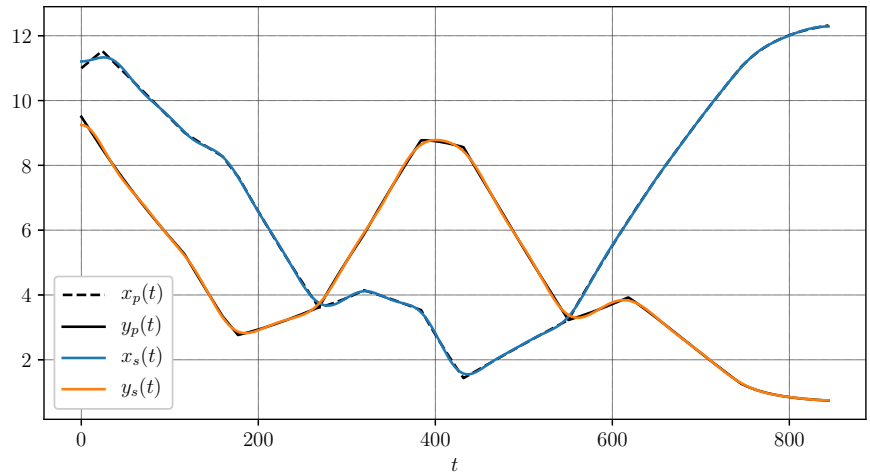

Figure 6: Initial Planned and Smoothed Steering Law Trajectory

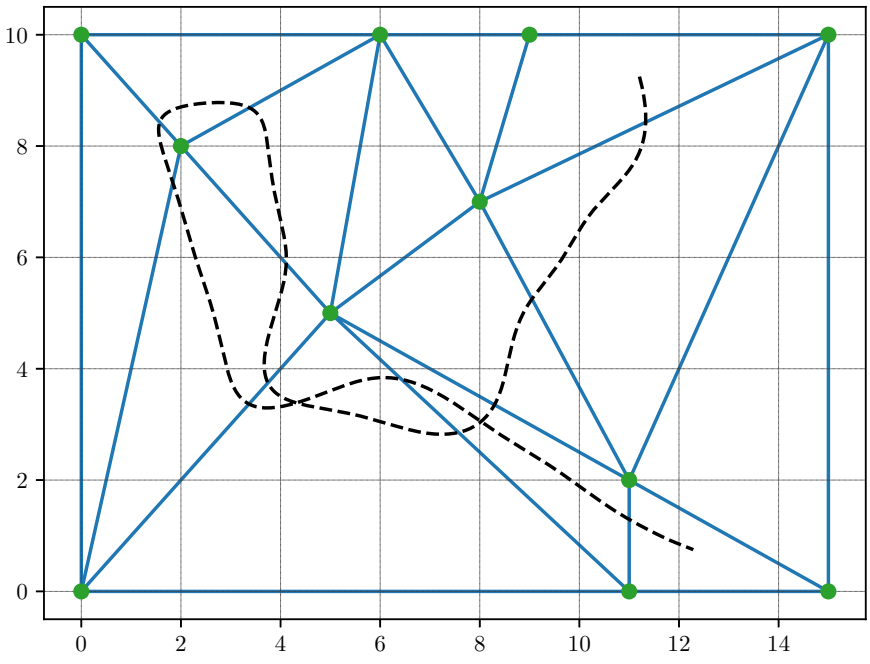

Figure 7: Partitioned Domain with Smoothed Steering Law Path

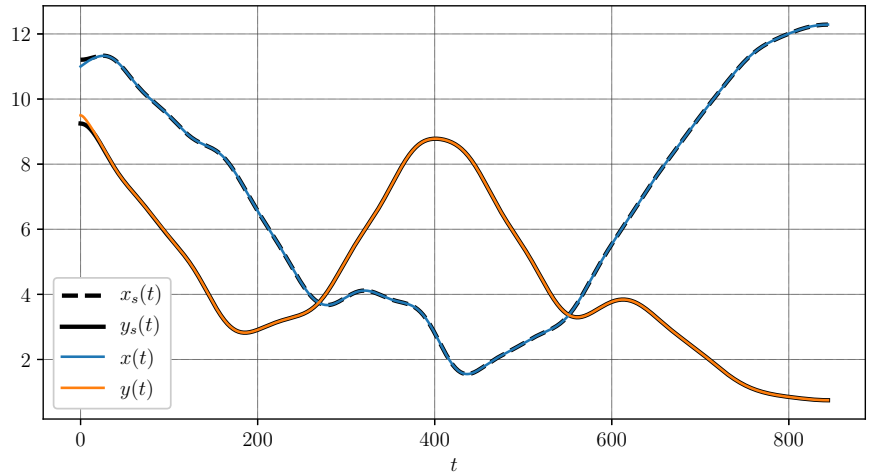

Figure 8: Steering Law Trajectory and True States

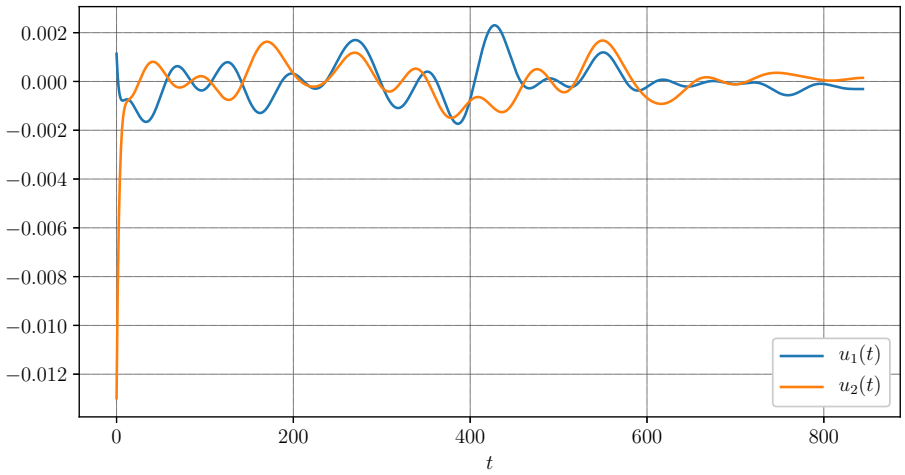

Figure 9: Control Signals

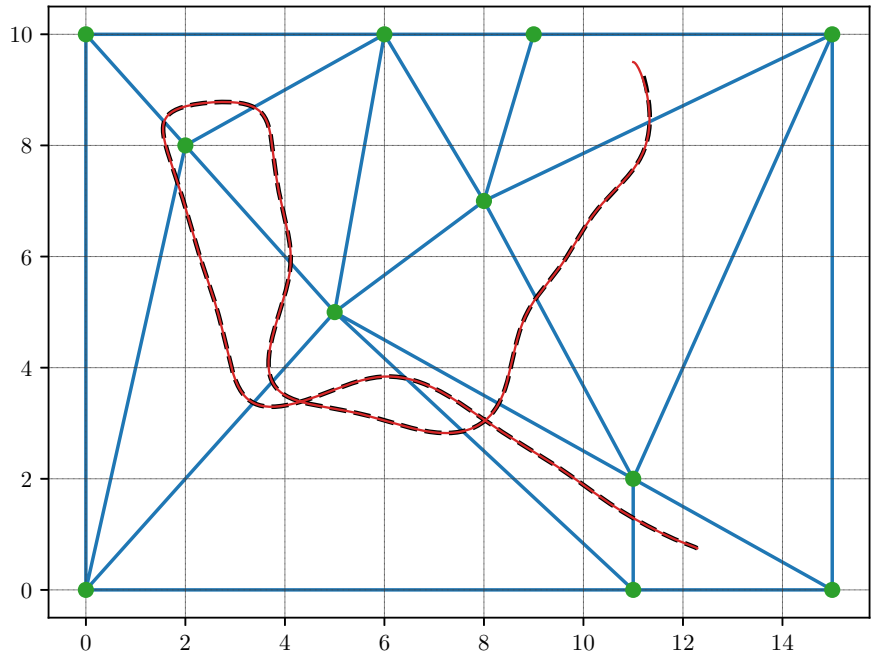

Figure 10: Partitioned Domain with Planned Path and True Trajectory

7 Conclusion

This project demonstrates the successful application of hybrid systems concepts to the problem of controlling a vehicle in a constrained domain. A simulation was written which enables a user to specify nodes in a domain, regions of interest ("survey", "avoid", and "home") as well as LTL formulas making use of the observations (labels) of the states of interest. The application of this code was demonstrated, and the results agree with expectations. Future work could include generalizing the project code further, and improving usability, along with extending the applicability to additional types of LTL formulas. Furthermore, control of more advanced systems could be demonstrated using this steering law approach, and the effect of uncertainties, unmodeled disturbances, and the problem of mitigating these could be investigated.

Note that while only one set of simulation results is discussed in this report, the codes are submitted with a README.txt file that provides instructions on applying new initial conditions to the same problem, as well as guidance on how to use the code to solve other problems with a different set of nodes/simplices or a different LTL formula.

References

[1] Oualid Araar and Nabil Aouf. Full Linear Control of a Quadrotor UAV, LQ vs H_∞. *UKACC International Conference on Control*, pages 133–138, 2014.

[2] Calin Belta, Boyan Yordanov, and Ebru Aydin Gol. *Formal Methods for Discrete Time Dynamical Systems*, volume 89 of *Studies in Systems, Decision, and Control*. Springer, 2017.

[3] L.C.G.J.M. Habets, P. J. Collins, and J. H. van Schuppen. Reachability and Control Synthesis for Piecewise-Affine Hybrid Systems on Simplices. *IEEE Transactions on Automatic Control*, 51(6):938–948, 2006.

[4] L.C.G.J.M. Habets and J.H. van Schuppen. A control problem for affine dynamical systems on a full-dimensional polytope. *Automatica*, 40:21–35, 2004.

[5] A. Ravankar, A. A. Ravankar, Y. Kobayashi, Y. Hoshino, and C. Peng. Path Smoothing Techniques in Robot Navigation: State-of-the-Art, Current and Future Challenges. *Sensors*, 18, 2018.