Trajectory Optimization and Time-Varying LQR Stabilization of Airplane Longitudinal Dynamics

Benjamin Thomsen
Underactuated Robotics Class Project, Spring 2018
Code at: https://github.mit.edu/thomsen/mit-6832-project

***Abstract*— This core of this project is trajectory optimization and time-varying LQR stabilization of the nonlinear longitudinal dynamics of a small, light, fixed-wing aircraft. First, a six-state nonlinear model of the longitudinal aircraft dynamics, with elevator and thrust inputs, is developed and parameterized. Trajectory optimization with input and state constraints is carried out using both direct transcription and direct collocation methods. Time-varying LQR stabilization around this nominal trajectory is carried out, to account for discretization error and for parametric uncertainties in the vehicle dynamics. Region of attraction estimation, via sums-of-squares verification is attempted but not fully implemented in its current state. Kinodynamic sampling-based motion planning, via an RRT*-like method is carried out to find dynamically feasible trajectories which avoid obstacles in the vertical plane.**

I. Introduction

This project attempts to find solutions to several of the problems involved with optimally navigating an aerial vehicle which is subject to model uncertainty, state and input constraints, and regions in space to avoid (obstacles). The major topics covered in this project are dynamics modeling, trajectory optimization, time-varying stabilization, sums-of-squares verification, and kinodynamic motion planning. These are each described in the following sections. These were all done in Python using `Pydrake` [1] when applicable.

II. Dynamics Model

The methods discussed in the following sections are applied to a nonlinear model of the longitudinal dynamics of a fixed-wing aircraft. The aircraft is assumed to be small and light, operating near sea level and constrained in a vertical plane. The roll angle and sideslip angle are assumed to be zero, and the yaw angle is assumed to be constant. The dynamics model, which is based on the nonlinear dynamics derived in [2], is defined by the following nonlinear dynamics $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$:

$$\begin{bmatrix} \dot{x} \\ \dot{z} \\ \dot{V} \\ \dot{\gamma} \\ \dot{\theta} \\ \dot{Q} \end{bmatrix} = \begin{bmatrix} V\cos\gamma \\ V\sin\gamma \\ \frac{1}{m}(F_T\cos\alpha - D - mg\sin\gamma) \\ \frac{1}{mV}(F_T\sin\alpha + L - mg\cos\gamma \\ Q \\ \frac{M}{I_y} \end{bmatrix} \tag{1}$$

The variables x and z represent horizontal and vertical positions of the vehicle in the world frame, V is the vehicle airspeed, γ is the flight path angle (angle of velocity relative to horizontal), θ is the pitch angle (angle of vehicle body relative to horizontal), and Q is the pitch rate. The angle of attack is given by $\alpha = \theta - \gamma$. L, D, F_T, and M are the aerodynamic lift force, aerodynamic drag force, thrust force, and pitching moment, respectively. The vehicle mass is m, its moment of inertia is I_y, and g is the gravitational constant. The control inputs to the model are the thrust force F_T and elevator deflection angle δ_e. The forces and moments acting on the vehicle are defined by the following:

$$L = qC_L, \quad D = qC_D, \quad M = q\bar{c}C_M \tag{2}$$

where the dynamic pressure, q, is given by $q = \frac{1}{2}\rho V^2 S$, where ρ is the air density (assumed to be constant at sea level), $\bar{c}$ is the mean aerodynamic chord, and S is the wing planform area (total area of lifting surface). The coefficients C_L, C_D, and C_M are given by the following relations:

$$\begin{aligned} C_L &= 2\cos\alpha\sin\alpha, \quad C_D = 2\sin^2\alpha \\ C_M &= C_{M,\alpha}\alpha + C_{M,Q}Q + C_{M,\delta_e}\delta_e. \end{aligned} \tag{3}$$

C_L and C_D are derived using flat-plate theory, and $C_{M,\alpha}$, $C_{M,Q}Q$, and C_{M,δ_e} are assumed to be constant.

The parameterization of the dynamics model is based on the perching foam glider presented in [3]. The mass and moment of inertia are higher than those of the perching glider due to the addition of a battery, motor controller, small brushless DC motor and propeller to provide thrust. These parameters are summarized in Table I.

Parameter	Value
g	9.81 m/s
m	0.150 kg
S	0.120 m^2
ρ	1.20 kg/m^3
$\bar{c}$	0.120 m
I_y	0.004 $kg\,m^2$
$C_{M,\alpha}$	-0.08
$C_{M,Q}$	-0.2
C_{M,δ_e}	-0.04

TABLE I
Parameters of dynamics model

III. Trajectory Optimization

Trajectory optimization is carried out with the objective being to reach a goal region in the state space while minimizing a quadratic cost functional of control effort and electric energy use. In the continuous-time formulation, the cost function can be represented by

$$J = \int_0^{t_f} \left[\mathbf{x}'Q\mathbf{x} + \mathbf{u}'R\mathbf{u} + \mathbf{x}'N\mathbf{u}\right] dt. \tag{4}$$

Note that instead of a cost on the state at $t = t_f$, the goal region is embedded into an optimization problem using constraints. In this particular problem, Q is zero (state is not penalized) and N is chosen so that the cost function includes the component $\int_0^{t_f} F_T V dt$, which is the energy input to the system via electric propulsion.

Trajectory optimization is done by discretizing the trajectory into equally spaced points in time (knot points), such that for N_T knot points, the time between the knot points is $h = \frac{t_f}{N_T - 1}$. In this discrete formulation, the cost function is approximated as

$$J = \sum_{i=0}^{N_T} h\left[\mathbf{u}_i'R\mathbf{u}_i + \mathbf{x}_i'N\mathbf{u}_i\right]. \tag{5}$$

The optimization problem will use the final time, t_f, along with the state $\mathbf{x}()$ and input $\mathbf{u}()$ at all knot points as optimization variables, subject to constraints bounding t_f, $\mathbf{x}()$, and $\mathbf{u}()$, and constraints which force the trajectory to roughly match the dynamics of the airplane (these are different depending on whether direct transcription or direct collocation is used). The following linear constraints are used in the optimization problem:

$$\begin{aligned} \mathbf{x}_{i=0} &= \mathbf{x}_0 \\ \mathbf{x}_{G,min} \leq \mathbf{x}_{i=N_T} &\leq \mathbf{x}_{G,max} \\ \mathbf{x}_{min} \leq \mathbf{x}_i &\leq \mathbf{x}_{max} \\ \mathbf{u}_{min} \leq \mathbf{u}_i &\leq \mathbf{u}_{max} \\ t_{f,min} \leq t_f &\leq t_{f,max} \end{aligned} \tag{6}$$

Dynamic constraints are nonlinear for both direct transcription and direct collocation, and defined as follows.

a) Direct Transcription: A first-order integration of dynamics is used, where the constraints become

$$\mathbf{x}_{i+1} = hf(\mathbf{x}_i, \mathbf{u}_i) \tag{7}$$

b) Direct Collocation: A more accurate way to constrain the dynamics, so that the dynamics are equivalently satisfied with fewer knot points, is to model the state trajectory $\mathbf{x}()$ as a cubic piecewise polynomial, and linearly interpolate $\mathbf{u}()$ between knot points. Direct collocation introduces *collocation points* midway between each knot point in time. Looking at each interval $[t_i, t_{i+1})$ of length h, and with collocation point $(\mathbf{x}_c, \mathbf{u}_c)$ at $t_c = t_i + h/2$, the direct collocation constraint can be formulated by constraining

$$\dot{\mathbf{x}}_c = f(\mathbf{x}_c, \mathbf{u}_c) \tag{8}$$

where

$$\mathbf{x}_c = \frac{1}{2}(\mathbf{x}_i + \mathbf{x}_{i+1}) + \frac{h}{8}(\dot{\mathbf{x}}_i - \dot{\mathbf{x}}_{i+1}) \tag{9}$$

$$\mathbf{u}_c = \frac{\mathbf{u}_i + \mathbf{u}_{i+1}}{2} \tag{10}$$

$$\dot{\mathbf{x}}_c = -\frac{3}{2h}(\mathbf{x}_i - \mathbf{x}_{i+1}) - \frac{1}{4}(\dot{\mathbf{x}}_i + \dot{\mathbf{x}}_{i+1}) \tag{11}$$

$$\dot{\mathbf{x}}_i = f(\mathbf{x}_i, \mathbf{u}_i), \quad \dot{\mathbf{x}}_{i+1} = f(\mathbf{x}_{i+1}, \mathbf{u}_{i+1}). \tag{12}$$

The use of constraint (8) is known as the compressed form of direct collocation.

For this project, I have used `Pydrake` and the `MathematicalProgram` interface which uses SNOPT to solve the nonlinear optimization problem. I found direct collocation to work fine for lower-dimensional systems, but it struggled with my dynamics. Even when providing the solver with the optimal trajectory from a direct transcription problem to use as an initial guess, direct collocation was significantly slower. This could be a bug with my code, or a subtle issue with the constraints I give to SNOPT. The following results are all generated using a direct transcription optimization as it was able to solve to a reasonable accuracy relatively quickly. There are no guarantees on global optimality with this type of optimization problem, however the results are consistent when optimizing the trajectory to a goal region from a variety of initial conditions, with a range of knot point spacing, and with a range of initial guesses for the variables. For large problems, for example when $N_T > 200$ and the goal is 200m away in space, it is likely that the resulting trajectories are just local minima.

IV. Time-Varying LQR Stabilization

There are many factors which may cause the vehicle to deviate from the optimal trajectory, in simulation and especially in the real world. One of these factors is the discretization of the continuous-time dynamics in the trajectory optimization problem. When the knot points are spaced further apart in time, following a piecewise-linear input trajectory is likely to lead to a state trajectory which does not match the nominal trajectory. This can be seen in Figure 2, in which following the trajectory without any feedback/stabilization causes the vehicle to hit the ground halfway to the goal region (see the black lines).

Another factor which I consider in this project is that the model of the dynamics may not match the actual vehicle dynamics, in inertial properties or aerodynamic properties, for example. The green lines in Figure 2 show what the open-loop simulation looks like when in addition to the discretization errors in dynamics, the mass and moment of inertia of the vehicle are assumed to be 5% higher than what is modeled, and the wing area is assumed to be 5% smaller than what is modeled. This causes the vehicle to hit the ground even earlier.

Using state feedback to stabilize around the trajectory, both these issues can be accommodated. Time-varying linear quadratic regulation (TVLQR) is one such method to stabilize around this nominal trajectory. TVLQR is a control method which provides an optimal control policy for linear time-varying systems, so the first step is to linearize the nonlinear airplane dynamics at the knot points of the trajectory.

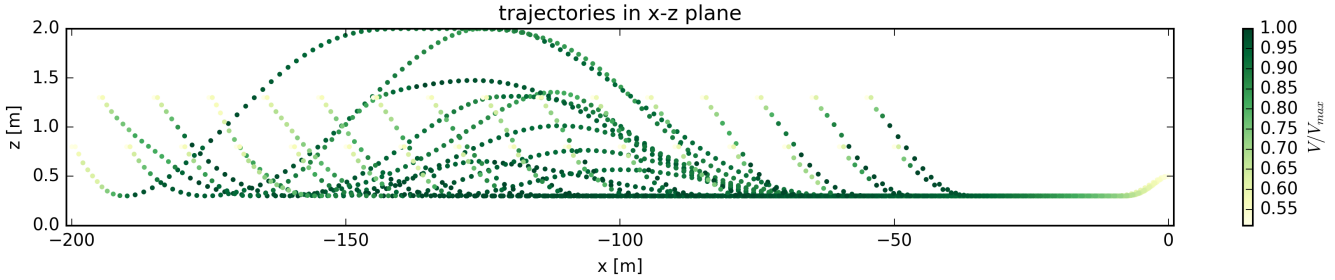

Fig. 1. Optimal trajectories in the xz-plane (vertical plane in space) from many initial conditions. Darker colors indicate higher speed. Note that the x-axis scale is 200m, while the z-axis scale is 2m.

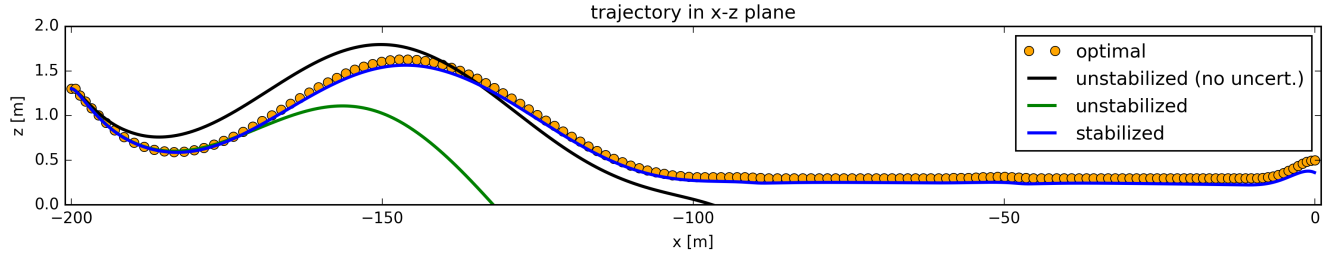

Fig. 2. One optimal trajectory, plotted in the xz-plane. The yellow dots are the knot points of the nominal trajectory, the black line is the result of a higher-fidelity simulation of the dynamics following the nominal input trajectory, the green line is the result of a higher-fidelity simulation with added modeling errors, and the blue line is the result of stabilization via TVLQR.

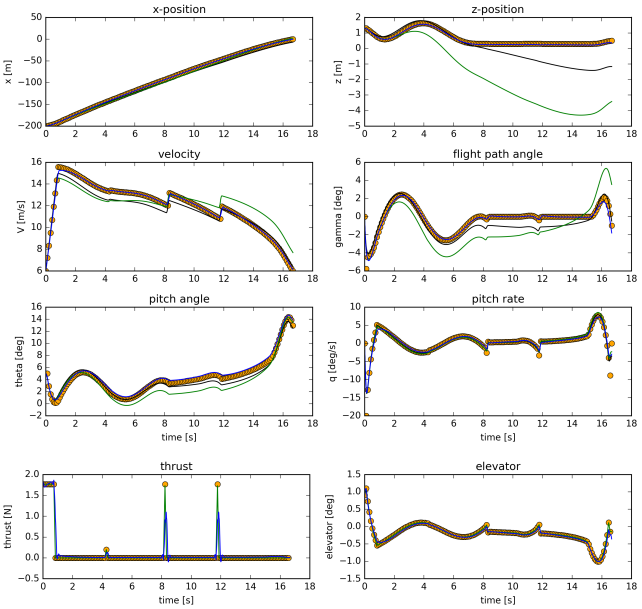

Fig. 3. The trajectory of Figure 2: six states vs. time, and the two inputs vs. time. The yellow dots are the knot points of the nominal trajectory, the black line is the result of a higher-fidelity simulation of the dynamics following the nominal input trajectory, the green line is the result of a higher-fidelity simulation with added modeling errors, and the blue line is the result of stabilization via TVLQR.

The linearization of the dynamics is given by:

$$\dot{\mathbf{x}} \simeq \left.\frac{df}{d\mathbf{x}}\right|_{\mathbf{x}_i,\mathbf{u}_i} (\mathbf{x}-\mathbf{x}_i) + \left.\frac{df}{d\mathbf{u}}\right|_{\mathbf{x}_i,\mathbf{u}_i} (\mathbf{u}-\mathbf{u}_i) \qquad (13)$$

when $f(\mathbf{x}_i, \mathbf{u}_i) = 0$, which can be written equivalently with $\bar{\mathbf{x}} = \mathbf{x} - \mathbf{x}_i$ and $\bar{\mathbf{u}} = \mathbf{u} - \mathbf{u}_i$ as

$$\dot{\bar{\mathbf{x}}} = A\bar{\mathbf{x}} + B\bar{\mathbf{u}}. \qquad (14)$$

These A and B matrices can be computed at each knot point by taking the Jacobian of the dynamics (1) with respect to the state vector and input vector, respectively, and evaluating the Jacobians at the knot points $\mathbf{x}_i$ and $\mathbf{u}_i$. Then the LQR method can be used to solve a Riccati equation and produce a gain matrix K which will provide the optimal control policy

$$\mathbf{u}* = \bar{\mathbf{u}}_i - K\bar{\mathbf{x}} \qquad (15)$$

for a cost function of the form

$$J = \int_0^\infty [\mathbf{x}'Q\mathbf{x} + \mathbf{u}'R\mathbf{u}]\, dt. \qquad (16)$$

Computing this policy at every knot point of the trajectory allows for stabilization around the nominal trajectory, which is seen in Figures 2 and 3 (blue lines) and is able to handle both the discretization errors and simulated modeling errors.

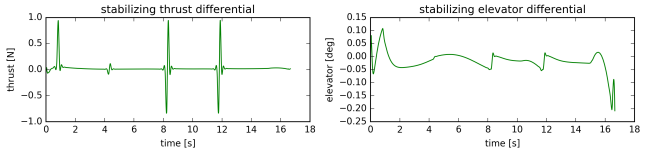

Fig. 4. The extra inputs required to stabilize around the nominal trajectory using TVLQR with simulated modeling errors.

V. Region of Attraction Verification

One interesting way to analyze the time-varying LQR stabilization is to determine how far the vehicle can deviate

from the nominal trajectory and remain stable using the LQR control policy. Because LQR is predicated on linear dynamics, while the airplane dynamics are quite nonlinear, the control policy may not have its intended effect when the vehicle is far from the operating point around which it was linearized.

This type of analysis can also be used as part of a "space-filling" algorithm, such as LQR-trees [4] so that the state space can be filled with regions in which the vehicle can be stabilized to some trajectory which will lead to the goal region.

For the case of time-invariant dynamics, with $\dot{\mathbf{x}} = f(\mathbf{x})$, we can verify stability by finding a sub-level set, B, of a positive-definite Lyapunov function, $V(\bar{\mathbf{x}})$, as in

$$B = \{\mathbf{x} \,|\, V(\bar{\mathbf{x}}) \leq \rho\}, \quad \rho > 0, \tag{17}$$

within which the following Lyapunov condition holds:

$$\frac{\partial V}{\partial \mathbf{x}} f(\mathbf{x}) < 0 \tag{18}$$

and by definition $V(\bar{\mathbf{x}}) > 0 \,\forall\, \bar{\mathbf{x}} \neq 0$. B is then a verified region of attraction for the system, and any state within the region will converge asymptotically to the origin as $t \to \infty$. For the class of systems with polynomial dynamics $f(\mathbf{x})$, this problem can be formulated as a convex optimization problem using sums-of-squares (SOS) optimization.

Taking the Lyapunov function candidate $V(\bar{\mathbf{x}}) = \bar{\mathbf{x}}' S \bar{\mathbf{x}}$, which is the cost-to-go for the LQR stabilization and is positive definite, all that remains is to find the sub-level set B where condition (18) holds. This could be demonstrated by the following optimization problem:

$$\begin{aligned} \max_{\rho,\lambda} \quad & \rho \\ \text{subject to} \quad & -\dot{V} + \lambda(V - \rho) \text{ is SOS} \\ & \lambda \text{ is SOS} \\ & \rho > 0 \end{aligned} \tag{19}$$

where $\lambda(\bar{\mathbf{x}})$ is a SOS polynomial multiplier. This optimization is not linear in the decision variables, so the problem can be solved by splitting into two alternating problems, as

a) Problem 1:

$$\begin{aligned} \max_{\gamma,\lambda} \quad & \gamma \\ \text{subject to} \quad & -\dot{V} + \lambda(V - \rho) - \gamma \text{ is SOS} \\ & \lambda \text{ is SOS} \\ & \gamma > 0 \end{aligned} \tag{20}$$

where we note that ρ is a constant rather than a decision variable. The second problem will then be to maximize ρ for the fixed polynomial $\lambda(\bar{\mathbf{x}})$, whose coefficients are solved in the first problem and passed to the second.

b) Problem 2:

$$\begin{aligned} \max_{\rho} \quad & \rho \\ \text{subject to} \quad & -\dot{V} + \lambda(V - \rho) \text{ is SOS} \\ & \lambda \text{ is SOS} \\ & \rho > 0 \end{aligned} \tag{21}$$

In the time-invariant case,

$$\dot{V}(\bar{\mathbf{x}}) = 2\bar{\mathbf{x}}' S f(\mathbf{x}_0 + \bar{\mathbf{x}}, \mathbf{u}_0 - K\bar{\mathbf{x}}) \tag{22}$$

which can be substituted into the above problems along with a specified order of the polynomial $\lambda(\bar{\mathbf{x}})$, in order to find ρ which defines the maximal certifiable region of attraction for the given Lyapunov function, through iterations until convergence.

When the system dynamics are non-polynomial, as is the case here, the problem can no longer be solved as a convex optimization problem. One approach to this is to approximate the system dynamics as polynomial, by taking an arbitrarily-high order Taylor series expansion. For this system, I found a third-order Taylor expansion to give very good approximations of the non-polynomial dynamics across a range of operating conditions. Defining the closed-loop system dynamics as $f_{cl}(\mathbf{x}) = f(\bar{\mathbf{x}} + \mathbf{x}_0, \mathbf{u}_0 - K\bar{\mathbf{x}})$, the third-order Taylor expansion is given (per element ℓ of the vector $f_{cl}(\mathbf{x})$) by

$$\begin{aligned} \hat{f}_{cl,\ell}(\bar{\mathbf{x}}) = f_\ell(\mathbf{x}_0, \mathbf{u}_0) + \\ \sum_i \bar{x}_i \frac{\partial f_{cl}}{\partial x_i}\bigg|_{\mathbf{x}_0,\mathbf{u}_0} + \sum_{i,j} \bar{x}_i \bar{x}_j \frac{\partial f_{cl}}{\partial x_i \partial x_j}\bigg|_{\mathbf{x}_0,\mathbf{u}_0} + \\ \sum_{i,j,k} \bar{x}_i \bar{x}_j \bar{x}_k \frac{\partial f_{cl}}{\partial x_i \partial x_j \partial x_k}\bigg|_{\mathbf{x}_0,\mathbf{u}_0}. \end{aligned} \tag{23}$$

I computed these through recursive use of the `symbolic::Jacobian` method using `pydrake` and was able to get quite accurate results. In order to verify a region of attraction around the trajectory (a "funnel"), several additional changes need to be made. First is that with the time-varying LQR controller, S is not time-invariant, so the derivative of the Lyapunov function becomes

$$\dot{V}(\bar{\mathbf{x}}) = 2\bar{\mathbf{x}}' S f(\mathbf{x}_0 + \bar{\mathbf{x}}, \mathbf{u}_0 - K\bar{\mathbf{x}}) + \bar{\mathbf{x}}' \dot{S} \bar{\mathbf{x}}. \tag{24}$$

The next change is that ρ is no longer time-invariant, and the constraint (18) is modified to become

$$\dot{V}(\bar{\mathbf{x}}, t) = \frac{\partial V}{\partial \mathbf{x}} f(\mathbf{x}) < \dot{\rho}(t). \tag{25}$$

We can sample at points along the trajectory (the same knot points used in optimization, for example), and take $\rho(t)$ to be a piecewise linear function which interpolates between the value at knot points. The SOS optimization problem corresponding to (19) for each knot point (at t_i) becomes

$$\begin{aligned} \max_{\rho_i,\lambda_i} \quad & \rho_i \\ \text{subject to} \quad & -(\dot{V} - \dot{\rho}_i) + \lambda_i(V - \rho_i) \text{ is SOS} \\ & \lambda_i \text{ is SOS} \\ & \rho_i > 0 \end{aligned} \tag{26}$$

according to the method outlined in [3]. This is again be solved using bilinear alternations similar to the time-invariant case, and solved backwards from the goal, so that $\dot{\rho}_i$ is the slope of the linear interpolation between ρ_i and ρ_{i+1}.

c) Time-Varying Problem 1:

$$\begin{aligned} \max_{\gamma_i,\lambda_i} \quad & \gamma_i \\ \text{subject to} \quad & -(\dot{V}-\dot{\rho}_i)+\lambda_i(V-\rho_i)-\gamma_i \text{ is SOS} \\ & \lambda_i \text{ is SOS} \\ & \gamma_i > 0 \end{aligned} \tag{27}$$

d) Time-Varying Problem 2:

$$\begin{aligned} \max_{\rho_i} \quad & \rho_i \\ \text{subject to} \quad & -(\dot{V}-\dot{\rho}_i)+\lambda_i(V-\rho_i) \text{ is SOS} \\ & \lambda_i \text{ is SOS} \\ & \rho_i > 0 \end{aligned} \tag{28}$$

These problems can be alternated until convergence to a maximum value of ρ_i. My current implementation of this verification along the trajectory does not appear to be giving the intended results. The time-varying problem formulations given in [4] and [5] use a slightly different description of the convex optimization problems, and it is not clear to me whether those methods might provide better results. My main issue is that the second of the alternating problems should be able to strictly increase the value of ρ, however it often returns that the problem is infeasible for the given constraints and coefficients of λ_i passed from the first problem, and it also takes much longer to solve than the first problem. One way around this was to iterate over the first problem only, increasing ρ by a set amount when the problem is feasible and decreasing ρ when it is infeasible, until convergence of ρ to a maximum feasible value. This produces the results of Figure 5, which do not show the "funneling" behavior towards the goal that would be expected.

If I had more time, I would consider changing the way I solve for $S(t_i)$ and $\dot{S}(t_i)$ to a square-root method integrating the differential Riccati equation backwards from the goal, rather than solving for $S(t_i)$ at each of the knot points. Other issues may be with picking the order of λ_i (currently 4), or with the high-order polynomial $\dot{V}(\bar{\mathbf{x}})$ which arises when taking the third-order Taylor expansion of the dynamics.

Another approach to the problem would be to use a grid-based discretization of the state-space and computations of $\dot{V}(\bar{\mathbf{x}})$ as was done in Problem Set 3 of the class. One more approach would be to use a simulation-based approach instead of the formal approach via verification of Lyapunov conditions, such as the method of [6].

VI. Kinodynamic Motion Planning

The above trajectory optimization problem becomes more difficult when there are convex regions in space which need to be avoided (obstacles). Avoidance of these obstacles could be embedded into a cost function, but this would increase the nonlinearity of the problem and make it very difficult to solve. Instead, the approach I am exploring is to sample obstacle-free points in the state-space and try to find dynamically feasible connections between these points, which satisfy input and state constraints and avoid collisions with obstacles. The aim is that eventually a path of connections is found which reaches the goal, from which additional sampling can be done to try and improve the path from start to goal. The dynamically feasible connections between two points in space can be computed using the same trajectory optimization method as above, and make use of TVLQR stabilization as well. There is no guarantee on the overall path from start to goal being optimal according to the cost function of the individual trajectory optimization problems, although some algorithms have the property of *asymptotic optimality* as infinitely many points are sampled.

The method I implement here is similar to the RRT* algorithm [7], [8], where the *steering* function between two points performs a trajectory optimization. The original RRT* algorithm for motion planning under differential constraints is reproduced in Figure 6 directly from [8]. There are several modifications I make to this method for use in my motion planning problem, summarized as follows.

a) Sampling function: As with the standard RRT* algorithm, I sample random points from a bounded region of the state-space and ensure it is not within an obstacle. Before accepting the sample and trying to connect it, however, I compute the (zero-input) accelerations in the x, z, and θ directions and weight the probability of accepting the sample based on the magnitude of these accelerations. This functions as a heuristic on how close the sample is to a "trim" condition, and speeds up the algorithm as it spends less time trying to compute trajectories between points which are likely to be dynamically infeasible.

b) Distance function: My distance function is an asymmetric function, which uses the states $(x,\, z,\, \gamma)$ at the origin node $(\mathbf{x}_0)$ and the states $(x,\, z)$ at the destination node $(\mathbf{x}_1)$.

$$\begin{aligned} d &= \sqrt{(x_1-x_0)^2+(z_1-z_0)^2} \\ \beta(\mathbf{x}_0,\mathbf{x}_1) &= \tan^{-1}\left(\frac{z_1-z_0}{x_1-x_0}\right) \end{aligned} \tag{29}$$

$$\text{Dist}(\mathbf{x}_0,\mathbf{x}_1) = d(\cos(\beta-\gamma_0)+C\sin^2(\beta-\gamma)) \tag{30}$$

where C is a scaling coefficient. This distance function looks for points that are close both in the xz-plane and near the axis of the vehicle's travel. The distance function also adds some stochasticity to the choice of nearest neighbor, by applying a Bernoulli process to the ordered list of nearest points (i.e. flipping a biased coin) until one is selected. This was found to help the algorithm converge to better solutions.

c) Nearby vertices: The set of nearby vertices is split into two groups, one of which is denoted the *backward neighbors* and one of which is denoted the *forward neighbors*, which are classified based on the angles $(\beta-\gamma_0)$. The backward neighbors are used for the loop starting on Line 9 of the RRT* algorithm in Figure 6, and the forward neighbors are used for the loop starting on Line 16 of the RRT* algorithm.

The results of this RRT* algorithm are shown in Figure 7. Note that the plot covers 200m in the x-axis and 2m in the z-axis, so variations in the height of the vehicle have a relatively small effect on optimality compared to velocity, for example. The obstacle avoidance problems which I've simulated are relatively simple (it is sparsely-cluttered), but

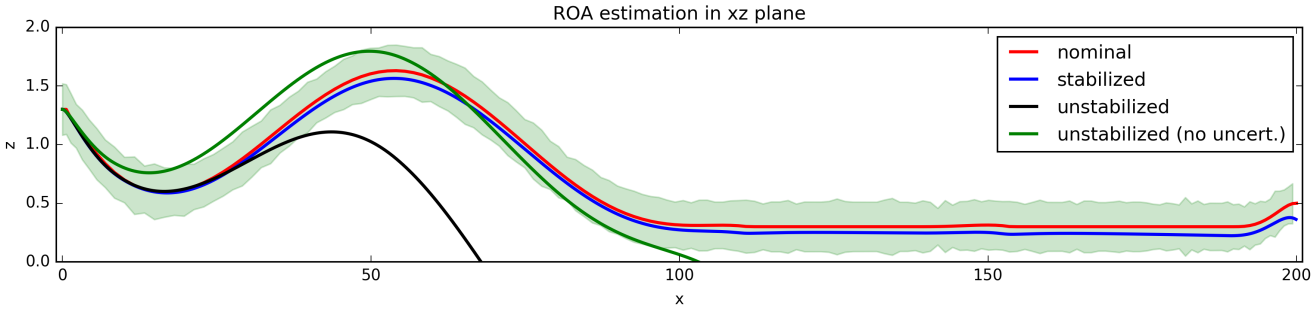

Fig. 5. Region of attraction as computed at each knot point, projected as a magnitude onto the z-axis by using the shape (eigenvalues and eigenvectors) of $S(t)$.

Algorithm 1: The RRT* Algorithm

$V \leftarrow \{z_{\text{init}}\}$; $E \leftarrow \emptyset$; $i \leftarrow 0$;
while $i < N$ **do**
  $G \leftarrow (V, E)$;
  $z_{\text{rand}} \leftarrow \texttt{Sample}(i)$; $i \leftarrow i + 1$;
  $(V, E) \leftarrow \texttt{Extend}(G, z_{\text{rand}})$;

Algorithm 2: The Extend Procedure

$V' \leftarrow V$; $E' \leftarrow E$;
$z_{\text{nearest}} \leftarrow \texttt{Nearest}(G, z)$;
$(x_{\text{new}}, u_{\text{new}}, T_{\text{new}}) \leftarrow \texttt{Steer}(z_{\text{nearest}}, z)$;
$z_{\text{new}} \leftarrow x_{\text{new}}(T_{\text{new}})$;
if $\texttt{ObstacleFree}(x_{\text{new}})$ **then**
  $V' := V' \cup \{z_{\text{new}}\}$;
  $z_{\min} \leftarrow z_{\text{nearest}}$; $c_{\min} \leftarrow \texttt{Cost}(z_{\text{new}})$;
  $Z_{\text{nearby}} \leftarrow \texttt{NearVertices}(G, z_{\text{new}}, |V|)$;
  for *all* $z_{\text{near}} \in Z_{\text{nearby}}$ **do**
    $(x_{\text{near}}, u_{\text{near}}, T_{\text{near}}) \leftarrow \texttt{Steer}(z_{\text{near}}, z_{\text{new}})$;
    if $\texttt{ObstacleFree}(x_{near})$ *and* $x_{\text{near}}(T_{\text{near}}) = z_{\text{new}}$ **then**
      if $\texttt{Cost}(z_{\text{near}}) + J(x_{\text{near}}) < c_{\min}$ **then**
        $c_{\min} \leftarrow \texttt{Cost}(z_{\text{near}}) + J(x_{\text{near}})$;
        $z_{\min} \leftarrow z_{\text{near}}$;
  $E' \leftarrow E' \cup \{(z_{\min}, z_{\text{new}})\}$;
  for *all* $z_{near} \in Z_{\text{nearby}} \setminus \{z_{\min}\}$ **do**
    $(x_{\text{near}}, u_{\text{near}}, T_{\text{near}}) \leftarrow \texttt{Steer}(z_{\text{new}}, z_{\text{near}})$;
    if $x_{\text{near}}(T_{\text{near}}) = z_{\text{near}}$ *and* $\texttt{ObstacleFree}(x_{\text{near}})$ *and* $\texttt{Cost}(z_{\text{near}}) > \texttt{Cost}(z_{\text{new}}) + J(x_{\text{near}})$ **then**
      $z_{\text{parent}} \leftarrow \texttt{Parent}(z_{\text{near}})$;
      $E' \leftarrow E' \setminus \{(z_{\text{parent}}, z_{\text{near}})\}$;
      $E' \leftarrow E' \cup \{(z_{\text{new}}, z_{\text{near}})\}$;
return $G' = (V', E')$

Fig. 6. Original RRT* algorithm, copied from [8] for reference.

the algorithm appears to work well. The algorithm could be improved through improving the heuristics used in the sampling and distance functions, but for the simple problems it is adequate as-is.

VII. Concluding Remarks

Throughout this project, I have run into various issues while trying to implement the methods described above. I was never able to determine why my direct collocation implementation was much slower than the `DirectCollocation` implementation in `Drake`. Unfortunately I haven't gotten satisfying results for the computation of invariant "funnels" around the trajectory, and there are potentially several issues which I have overlooked with the formulation of the SOS programs. I have had success comparing the region of attraction estimates around a time-invariant fixed point from these SOS programs to the results of a grid-based discretization, using a simpler system with two states, a single input, but was unable to apply the same algorithms to the longitudinal dynamics of the airplane following a trajectory.

Overall, working on applying these various computational approaches has been a great learning experience. I am happy to have gained experience with several useful `Drake` features (through the `Pydrake` Python bindings), such as the symbolic computation library and the mathematical programming interface, as well as several simulation and visualization tools.

References

[1] R. Tedrake and the Drake Development Team, "Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems," 2016.
[2] B. L. Stevens, F. L. Lewis, and E. N. Johnson, *Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems*. John Wiley & Sons, 2015.
[3] J. Moore, R. Cory, and R. Tedrake, "Robust post-stall perching with a simple fixed-wing glider using LQR-trees," *Bioinspiration & Biomimetics*, vol. 9, no. 2, p. 025013, 2014.
[4] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "LQR-trees: Feedback motion planning via sums-of-squares verification," *The International Journal of Robotics Research*, vol. 29, no. 8, pp. 1038–1052, 2010.
[5] M. M. Tobenkin, I. R. Manchester, and R. Tedrake, "Invariant funnels around trajectories using sum-of-squares programming," *arXiv preprint arXiv:1010.3013*, 2010.

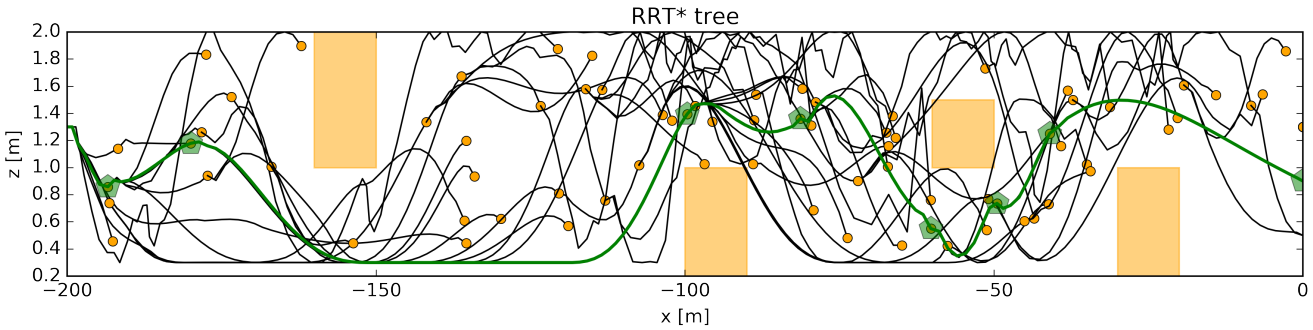

Fig. 7. A run of the RRT* algorithm, showing the entire tree (as well as the solution in green) after 350 sample points generated, with 80 successfully added to the tree.

[6] P. Reist, P. Preiswerk, and R. Tedrake, "Feedback-motion-planning with simulation-based LQR-trees," *The International Journal of Robotics Research*, vol. 35, no. 11, pp. 1393–1416, 2016.

[7] S. Karaman, "Incremental sampling-based algorithms for optimal motion planning," *Robotics Science and Systems VI*, vol. 104, p. 2.

[8] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *Decision and Control (CDC), 2010 49th IEEE Conference on*, pp. 7681–7687, IEEE, 2010.

[9] A. Majumdar, A. A. Ahmadi, and R. Tedrake, "Control design along trajectories with sums of squares programming," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 4054–4061, IEEE, 2013.