Combined Bernstein Polynomial, Optimal Reciprocal Collision Avoidance, Differential Dynamic Programming for Trajectory Replanning and Collision Avoidance for UAM Vehicles

Matthew D. Houghton* †

Michael J. Acheson* ‡

Andrew Patterson* §

Alex Oshin * ¶ ‖

Irene M. Gregory*, **

This paper presents an integration of differential dynamic programming (DDP) with the optimal reciprocal collision avoidance (ORCA) algorithm as the basis for a new algorithm, titled combined Bernstein polynomial optimal reciprocal collision avoidance DDP (COBRA-DDP), for trajectory replanning and collision avoidance for urban air mobility (UAM) vehicles. State-constrained variants of DDP provide the ability to plan trajectories while avoiding obstacles, but these methods require a large increase in computational time per iteration which hinders the overall speed of the algorithm. ORCA utilizes simplified dynamics to recognize potential collisions along a trajectory and provides an optimal velocity for the avoidance of multiple vehicles and obstacles. These velocity commands, however, may not result in a dynamically feasible trajectory for DDP to plan around. As such, a Bernstein polynomial curve that considers the general dynamic constraints of the vehicle is generated to approximate a trajectory based on the velocity commands. COBRA-DDP optimizes this suggested trajectory via unconstrained DDP to provide a dynamically feasible trajectory that provides collision avoidance. This new trajectory can be applied to the vehicle or used to warm-start the state constrained DDP algorithms to decrease computation time. The benefits and effectiveness of the algorithm are demonstrated on a UAM vertical takeoff and landing (VTOL) vehicle simulation with highly nonlinear dynamics.

I. Introduction

The safety of urban air mobility (UAM) vehicles and their surroundings is crucial when considering their integration into general airspace. When interacting with manned aircraft, UAM vehicles must be both safe and predictable in their trajectories. The expected use of UAM vehicles for air travel is an excellent example of newly crowded airspace in urban environments requiring these capabilities. While the implementation of UAM vehicles into general airspace will take time, a similar style of problem is already being faced around airports and military locations where coordination of multiple vehicles may require trajectory adjustments to avoid collisions. The robotics field has advanced many algorithms to tackle the difficult challenges of trajectory optimization that are worth considering. A portion of these algorithms handle highly complex nonlinear dynamics, something that many UAM vehicles (and their vehicle concepts) are known for due to effector overactuation and aircraft transitions in their standard concept of operations. General

*NASA Langley Research Center, Hampton, VA, 23681, USA

†Research Engineer, Dynamic Systems and Control Branch, Matthew.D.Houghton@nasa.gov.

‡Senior Research Engineer, Dynamic Systems and Control Branch, Michael.J.Acheson@nasa.gov, Member AIAA.

§Research Engineer, Dynamic Systems and Control Branch, Andrew.Patterson@nasa.gov.

¶Research Engineer, Dynamic Systems and Control Branch, and Machine Learning PhD Student, School of Aerospace Engineering, Georgia Institute of Technology, alexoshin@gatech.edu.

‖Georgia Institute of Technology, Atlanta, GA, 30332, USA

**NASA Senior Technologist for Advanced Control Theory and Application, Irene.M.Gregory@nasa.gov, Fellow AIAA.

planning and replanning of trajectories involving these types of dynamics has led to increased computational complexity, a problem exacerbated by the inclusion of the state and control constraints necessary for collision avoidance.

Under standard circumstances, where a pilot is replanning a trajectory due to safety concerns, the pilot considers both the effects of the immediate maneuver and the long-term implications that it would have on the trajectory. These considerations span different time scales and detail of dynamic complexity. Locating a balance between short-term vehicle safety and long-term trajectory following is also necessary to achieve collision avoidance with reasonable trajectory replanning capabilities. These concerns and trade-offs are mirrored in autonomous flight planners. The priority in autonomous contingency management, after a contingency event occurs, is the recognition of the event and the immediate safety of the vehicle (e.g., retaining stability). The next primary consideration in the vast majority of contingencies is typically a requirement for autonomous trajectory replanning (e.g., collision avoidance maneuver, divert to alternate landing site, etc.) This paper addresses autonomous replanning, in particular, replanning for collision avoidance. Successful collision avoidance replanning has some basic tenets:

- Real-time planning that provides separation assurances
- Planning capability for a large number of stationary and/or moving obstacles
- Ability to generate dynamically feasible trajectories
- Uncooperative obstacle avoidance capability (however, cooperative capability is desirable).

In this research, the authors take a novel approach to collision avoidance replanning. The algorithm of this paper, combined Bernstein polynomial optimal reciprocal collision avoidance differential dynamic programming (DDP), or COBRA-DDP, provides immediate collision avoidance recognition across the entire trajectory and a replanned trajectory that considers the dynamic constraints of the vehicle and its nonlinear dynamics when optimizing for the new collision-free trajectory. This is made possible through the integration of optimal reciprocal collision avoidance (ORCA) with DDP. ORCA, a collision avoidance algorithm widely used in robotics, leverages its computation speed to detect collisions along the trajectory and employs simplified dynamics to calculate a new trajectory that is guaranteed to be safe. It does this by providing separation assurances. It can readily accommodate a large number of moving and stationary obstacles and has the capability for planning around cooperative/non-cooperative obstacles. The main drawback of ORCA is that it does not consider the full vehicle dynamics. DDP is an optimal control approach that generates inherently dynamically feasible optimal trajectories even for vehicles with highly nonlinear dynamics, such as UAM vertical takeoff and landing (VTOL) vehicles. Initialization with a reasonable initial trajectory (in the form of target states and controls) is crucial as DDP is a local optimization algorithm. Large changes in trajectory, as well as poor initialization conditions, can greatly affect the number of iterations for optimization.

To integrate the ORCA algorithm's collision avoidance recommendation into DDP, a method for transferring trajectory information between the two is needed. The implemented trajectory generation method converts the position and velocity target generated by ORCA into a parametric trajectory that can be sampled at any time. This parametric form allows the COBRA algorithm to compute the desired state and input targets at any time DDP needs for control or optimization. Bernstein polynomials (BPs) represent the trajectories used in this work. This polynomial representation has many qualities, as described in [1]. Piece-wise BP curves provide fast computation, compact trajectory representation, and seamless integration of position, velocity, and acceleration. The two most critical properties are that BPs have closed form, analytic derivatives and that there are efficient algorithms for computing bounds on the curves from the parameters. These properties allow the curves to be generated so that they exactly meet terminal constraints on their derivatives and their intermediate values are bounded. ORCA directly provides position and velocity values at a specific time in the future. Using the BP framework, this future position and velocity can be used to construct continuous collision avoidance maneuver. The bounds improve feasibility verification before use in DDP for optimization. A description on computing bounds on this class of curves is given in [2]. The choice of BPs will also enable connection with a wide array of aerospace algorithms, the work found in [3] as an example. The novel nature of this research is to combine the ORCA and DDP (with a trajectory backbone of piece-wise BP curves) to utilize the strengths of each technique to ameliorate the deficiencies of the other.

The paper is organized as follows: the first three sections are an explanation of the preliminary information needed to understand the COBRA-DDP algorithm. Section II provides the background and derivation for Differential Dynamic Programming and state constrained augmentations, Section III discusses the general implementation of ORCA, and Section IV defines BPs. Section V explains the improvements and interactions of the previous section's information to form the COBRA-DDP algorithm, Section VI describes the experiments and results of COBRA-DDP and its ability to enhance augmented Lagrangian differential dynamic programming (AL-DDP), and finally Section VII discusses the effectiveness and future work for the algorithm.

II. Differential Dynamic Programming

DDP is an optimal control approach, widely used in robotics, that has been shown as an effective trajectory planner for vehicle configurations envisioned for the emerging aviation sector of UAM [4] and [5]. DDP was originally developed by Jacobson and Mayne [6]. It enables planning over highly nonlinear dynamics using second-order approximations along a nominal trajectory and displays quadratic convergence to a local solution. While the original unconstrained algorithm does not handle state constraints specifically, variations of DDP such as AL-DDP and others have demonstrated this capability, though at an increased computational cost. The following section summarizes the derivation and implementation of DDP and AL-DDP. Interested readers are referred to works such as [7–9] for a more thorough derivation and discussion of DDP variants.

A. Unconstrained DDP

This work considers a nonlinear discrete-time system with dynamics that evolve according to

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t), \tag{1}$$

where $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$, $\mathbf{x}_t \in \mathbb{R}^n$ is the state, $\mathbf{u}_t \in \mathbb{R}^m$ is the control at time t, and $\mathbf{x}_0 \in \mathbb{R}^n$ denotes the initial condition. Trajectories of state and control are denoted as $\mathbf{X} := \{\mathbf{x}_0, \ldots, \mathbf{x}_T\}$ and $\mathbf{U} := \{\mathbf{u}_0, \ldots \mathbf{u}_{T-1}\}$, respectively, over a finite time horizon $T \in \mathbb{N}^+$. The discrete-time trajectory optimization problem is to find the control trajectory that minimizes a cost function of the form

$$\mathcal{J}(\mathbf{U}) = \sum_{t=0}^{T-1} \mathcal{L}(\mathbf{x}_t, \mathbf{u}_t) + \phi(\mathbf{x}_T), \tag{2}$$

given an initial state $\mathbf{x}_0$ subject to dynamics in Eq. (1). The functions $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ and $\phi : \mathbb{R}^n \to \mathbb{R}$ denote the running cost and terminal cost, respectively.

By rewriting the minimization of Eq. (2) in terms of the value function, V, describing the optimal cost-to-go, DDP solves the discrete-time trajectory optimization problem. V is defined as the cost when following the optimal control sequence $\mathbf{U}^*$ from a given state $\mathbf{x}_i$

$$V(\mathbf{x}_i) := \min_{\mathbf{U}_i} \mathcal{J}(\mathbf{x}_i, \mathbf{U}_i). \tag{3}$$

$\mathbf{U}_i := \{\mathbf{u}_i, \ldots, \mathbf{u}_{T-1}\}$ denotes the truncated control sequence from time i onward, and the cost function used to calculate the cost-to-go is the partial sum of costs from time i onwards

$$\mathcal{J}_i(\mathbf{x}_i, \mathbf{U}_i) := \sum_{t=i}^{T-1} \mathcal{L}(\mathbf{x}_t, \mathbf{u}_t) + \phi(\mathbf{x}_T). \tag{4}$$

Bellman's principle of optimality allows the minimization in Eq. (3) over the entire control sequence to be rewritten as a sequence of minimizations over each $\mathbf{u}_t$ proceeding backwards in time:

$$V(\mathbf{x}_i) = \min_{\mathbf{u}_i} \Big[\underbrace{\mathcal{L}(\mathbf{x}_i, \mathbf{u}_i) + V(\mathbf{x}_{i+1})}_{Q(\mathbf{x}_i, \mathbf{u}_i)} \Big], \tag{5}$$

with boundary condition $V(\mathbf{x}_T) = \phi(\mathbf{x}_T)$.

Solving for V in Eq. (5) backwards in time for all timesteps t is equivalent to solving the optimal control problem that minimizes the cost in Eq. (2). For this optimization to be tractable, optimal variations in state $\delta\mathbf{x}_t$ and control $\delta\mathbf{u}_t$ are solved for such that the cost is minimized. To do this, DDP considers the quadratic approximations of the value function about a nominal state and control trajectory given by $\bar{\mathbf{x}}_t := \mathbf{x}_t - \delta\mathbf{x}_t$ and $\bar{\mathbf{u}}_t := \mathbf{u}_t - \delta\mathbf{u}_t$:

$$V(\mathbf{x}_i) \approx V_i^0 + (V_i^x)^\top \delta\mathbf{x} + \frac{1}{2} \delta\mathbf{x}^\top V_i^{xx} \delta\mathbf{x}, \tag{6}$$

where the superscripts denote the partial derivatives of V evaluated at $\bar{\mathbf{x}}_i$. Taking the quadratic expansion of the Q-function defined in Eq. (5) gives

$$\delta Q(\mathbf{x}_i, \mathbf{u}_i) \approx \frac{1}{2} \begin{bmatrix} 1 \\ \delta\mathbf{x}_i \\ \delta\mathbf{u}_i \end{bmatrix}^\top \begin{bmatrix} 0 & (Q_i^x)^\top & (Q_i^u)^\top \\ Q_i^x & Q_i^{xx} & (Q_i^{ux})^\top \\ Q_i^u & Q_i^{ux} & Q_i^{uu} \end{bmatrix} \begin{bmatrix} 1 \\ \delta\mathbf{x}_i \\ \delta\mathbf{u}_i \end{bmatrix}, \tag{7}$$

where

$$Q_i^x = \mathcal{L}_i^x + (\mathbf{f}_i^x)^\top V_{i+1}^x, \tag{8a}$$
$$Q_i^u = \mathcal{L}_i^u + (\mathbf{f}_i^u)^\top V_{i+1}^x, \tag{8b}$$
$$Q_i^{xx} = \mathcal{L}_i^{xx} + (\mathbf{f}_i^x)^\top V_{i+1}^{xx} \mathbf{f}_i^x + V_{i+1}^x \cdot \mathbf{f}_i^{xx}, \tag{8c}$$
$$Q_i^{ux} = \mathcal{L}_i^{xu} + (\mathbf{f}_i^u)^\top V_{i+1}^{xx} \mathbf{f}_i^x + V_{i+1}^x \cdot \mathbf{f}_i^{ux}, \tag{8d}$$
$$Q_i^{uu} = \mathcal{L}_i^{uu} + (\mathbf{f}_i^u)^\top V_{i+1}^{xx} \mathbf{f}_i^u + V_{i+1}^x \cdot \mathbf{f}_i^{uu}. \tag{8e}$$

The right-most terms in Eqs. (8c) to (8e) denote contraction with a tensor. The second order derivatives of the dynamics are often neglected, leading to the iterative linear quadratic regulator (iLQR) algorithm [7]. In practice, ignoring these higher-order terms leads to a more computationally efficient algorithm without affecting the quality of the solution [10].

After plugging Eq. (8) into Eq. (5), the optimal control updates can be solved by setting the gradient equal to zero and solving for $\delta\mathbf{u}_i^*$, which is given by

$$\delta\mathbf{u}_i^* = \mathbf{k}_i + \mathbf{K}_i \delta\mathbf{x}_i, \tag{9}$$

with

$$\mathbf{k}_i := -(Q_i^{uu})^{-1} Q_i^u, \tag{10a}$$
$$\mathbf{K}_i := -(Q_i^{uu})^{-1} Q_i^{ux}. \tag{10b}$$

Substituting this optimal control variation into the quadratic expansion of the value function yields

$$V_i^0 = \mathcal{L}_i^0 + V_{i+1}^0 - \frac{1}{2}(Q_i^u)^\top (Q_i^{uu})^{-1} Q_i^u, \tag{11a}$$
$$V_i^x = Q_i^x - (Q_i^{ux})^\top (Q_i^{uu})^{-1} Q_i^u, \tag{11b}$$
$$V_i^{xx} = Q_i^{xx} - (Q_i^{ux})^\top (Q_i^{uu})^{-1} Q_i^{ux}. \tag{11c}$$

This provides equations for propagating the value function backwards in time to the initial condition with boundary conditions $V_T^0 = \phi_T^0$, $V_T^x = \phi_T^x$, and $V_T^{xx} = \phi_T^{xx}$. Solving for Eqs. (8) and (11) backwards in time up to the initial condition $\mathbf{x}_0$ at time $t = 0$ constitutes the backward pass of DDP.

Due to the nonlinearities of the dynamics and inaccuracies in the quadratic approximation, applying the control update in Eq. (9) may increase the cost. This problem can be handled via a line search being performed on the feedforward gain. This process is explained more fully in Appendix A .

After the backward pass is complete, the dynamics are simulated forward in time starting from the initial state and applying the locally-linear feedback policy obtained by Eq. (9) to receive a new nominal state and control trajectory. This is referred to as the forward pass of DDP. The backward and forward passes are repeated until a predetermined convergence criteria is met. Algorithm 1 provides a pseudocode breakdown of the DDP algorithm for a problem with nonlinear dynamics.

B. AL-DDP

While DDP explicitly handles the dynamic constraints through its problem formulation and allows for straightforward handling of box control limits [11], it is unable to account for state constraints, such as obstacle avoidance. While various approaches to adding state constraints to DDP have been proposed in the past, including active set methods and addition

Algorithm 1: Differential Dynamic Programming

Input: Nominal state and control trajectory $\bar{\mathbf{x}}_t, \bar{\mathbf{u}}_t$
while *not converged* **do**
  // Backward pass
  $V_T^0 \leftarrow \phi_T^0$
  $V_T^x \leftarrow \phi_T^x$
  $V_T^{xx} \leftarrow \phi_T^{xx}$
  for $t = T-1, \ldots, 1$ **do**
    Calculate $Q_t^x, Q_t^u, Q_t^{xx}, Q_t^{xu}, Q_t^{uu}$ according to Eq. (8) and/or Eq. (27)
    if Q_t^{uu} *not invertible* **then**
      Increase regularization parameters μ_1, μ_2
      Restart backward pass
    Calculate gains $\mathbf{k}_t, \mathbf{K}_t$ according to Eq. (10)
    Calculate V_t^0, V_t^x, V_t^{xx} according to Eq. (11) and/or Eq. (28)
  // End backward pass
  Decrease regularization parameters μ_1, μ_2
  // Line search
  $\epsilon \leftarrow 1$
  while *cost decrease not sufficient* **do**
    // Forward pass
    $\mathbf{x}_1 \leftarrow \bar{\mathbf{x}}_1$
    for $t = 1, \ldots, T-1$ **do**
      $\delta\mathbf{x}_t \leftarrow \mathbf{x}_t - \bar{\mathbf{x}}_t$
      $\mathbf{u}_t \leftarrow \bar{\mathbf{u}}_t + \epsilon\mathbf{k}_t + \mathbf{K}_t\delta\mathbf{x}_t$, from Eq. (26)
      $\mathbf{x}_{t+1} \leftarrow \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t)$, from Eq. (1)
    // End forward pass
    $\epsilon \leftarrow \rho\epsilon$ // Reduce ϵ
  // End line search
  $\bar{\mathbf{x}}_t \leftarrow \mathbf{x}_t$
  $\bar{\mathbf{u}}_t \leftarrow \mathbf{u}_t$
return *optimal controls* $\mathbf{u}_t$

of Karush-Kuhn-Tucker (KKT) conditions [12], the most straightforward extension uses an augmented Lagrangian (AL) approach [9], sometimes called the *method of multipliers*. In particular, the goal is to solve the following constrained optimal control problem:

$$\begin{aligned} \min_{\mathbf{U}} \mathcal{J}(\mathbf{U}) = \min_{\mathbf{U}} \sum_{t=0}^{T-1} \mathcal{L}(\mathbf{x}_t, \mathbf{u}_t) + \phi(\mathbf{x}_T), \\ \text{subject to } \mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t),\ \forall t = 0, \ldots, T-1, \\ \mathbf{g}_t(\mathbf{x}_t) \leq \mathbf{0},\ \forall t = 0, \ldots, T, \end{aligned} \tag{12}$$

where $\mathbf{g}_t(\mathbf{x}_t) = (g_{t,1}(\mathbf{x}_t), \ldots, g_{t,c}(\mathbf{x}_t))^\top$ is a vector-valued function of c state constraints that should hold over the entire time horizon T. To this end, the next subsection will explore some basics of constrained optimization, and the following subsection will apply those methods to solve (12).

1. Constrained Optimization

AL methods reformulate a single constrained optimization problem into a series of unconstrained problems that are easier to solve [13]. Mathematically, the constrained optimization problem is given as:

$$\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) \text{ subject to } \mathbf{g}(\mathbf{x}) \leq \mathbf{0}, \tag{13}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function to minimize, and $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^c$ is a vector-valued function of inequality constraints $\mathbf{g}(\mathbf{x}) = (g_1(\mathbf{x}), \ldots, g_c(\mathbf{x}))^\top$. The *Lagrangian* for the problem is given as

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \boldsymbol{\lambda}^\top \mathbf{g}(\mathbf{x}), \tag{14}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^c$ is the vector of Lagrange multipliers for the constraints. Solving this problem involves alternating between minimizing the Lagrangian with respect to $\mathbf{x}$ while keeping $\boldsymbol{\lambda}$ fixed. Then the Lagrange multipliers are updated through gradient ascent on the dual function for the problem, which usually results in a closed-form expression that is easily computable. This process finds a saddle point of the Lagrangian, which is a solution to the original constrained problem and satisfies the KKT conditions.

Optimization of this Lagrangian can be *augmented* by introducing a penalty term that makes the problem smoother without changing the solution:

$$L(\mathbf{x}, \boldsymbol{\lambda}, \mu) = f(\mathbf{x}) + \boldsymbol{\lambda}^\top \mathbf{g}(\mathbf{x}) + \frac{\mu}{2} \|\mathbf{g}(\mathbf{x})\|_2^2, \tag{15}$$

for $\mu > 0$, which is known as the penalty parameter. This is known as the *Augmented Lagrangian* for the original constrained problem in (13).

In principle, one can replace the Lagrange multiplier and penalty term with any smooth *penalty function* $\mathcal{P}(g_i(\mathbf{x}), \lambda_i, \mu)$, such that

$$L_{\mathcal{P}}(\mathbf{x}, \boldsymbol{\lambda}, \mu) = f(\mathbf{x}) + \sum_{i=1}^{c} \mathcal{P}(g_i(\mathbf{x}), \lambda_i, \mu). \tag{16}$$

Once again, this problem is solved by alternating between minimizing $L_{\mathcal{P}}$ with the Lagrange multipliers and penalty parameter fixed, then updating the Lagrange multipliers through the update $\lambda_i = \frac{\partial}{\partial g_i(\mathbf{x})} \mathcal{P}(g_i(\mathbf{x}), \lambda_i, \mu)$. The penalty parameter is fixed but can be increased when the constraint improvement is not satisfactory. This corresponds to making the problem smoother at the cost of slower convergence [14].

2. Applying the AL method

In practice, applying the AL method with DDP to solve the constrained optimal control problem in (12) is straightforward; the constrained OC problem in (12) is replaced with an unconstrained problem through the addition of a penalty function $\mathcal{P}$:

$$\begin{aligned} \min_{\mathbf{U}} \tilde{\mathcal{J}}(\mathbf{U}) = \min_{\mathbf{U}} \sum_{t=0}^{T-1} \tilde{\mathcal{L}}(\mathbf{x}_t, \mathbf{u}_t, \boldsymbol{\lambda}_t, \mu_t) + \tilde{\phi}(\mathbf{x}_T, \boldsymbol{\lambda}_T, \mu_T), \\ \text{subject to } \mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t), \ \forall t = 0, \ldots, T-1, \end{aligned} \tag{17}$$

where each of the running cost and terminal cost functions have been augmented via

$$\tilde{\mathcal{L}}(\mathbf{x}_t, \mathbf{u}_t, \boldsymbol{\lambda}_t, \mu_t) = \mathcal{L}(\mathbf{x}_t, \mathbf{u}_t) + \sum_{i=1}^{c} \mathcal{P}(g_{t,i}(\mathbf{x}), \lambda_{t,i}, \mu_t), \tag{18}$$

$$\tilde{\phi}(\mathbf{x}_T, \boldsymbol{\lambda}_T, \mu_T) = \phi(\mathbf{x}_T) + \sum_{i=1}^{c} \mathcal{P}(g_{T,i}(\mathbf{x}), \lambda_{T,i}, \mu_T). \tag{19}$$

The optimization process can be thought of as being divided into an inner and outer loop. The inner loop of the optimization fixes $\boldsymbol{\lambda}_t, \mu_t, \forall t = 0, \ldots, T$ and minimizes $\tilde{\mathcal{J}}$ with respect to $\mathbf{U}$ using unconstrained DDP, which implicitly handles the dynamics constraints. The outer loop updates the Lagrange multipliers and checks the constraints, updating the penalty parameters if necessary. This approach will be referred to as AL-DDP.

Since DDP requires the cost function to be twice differentiable with respect to the states and controls, the penalty function must be twice differentiable with respect to the first argument. One of the most widely used penalty functions is $\mathcal{P}(y, \lambda, \mu) = \frac{1}{2\mu}(\max(0, \lambda + \mu y)^2 - \lambda^2)$, which results in the Powell-Hestenes-Rockafellar (PHR) method, but has

discontinuous second derivatives [14]. Therefore, in this paper, a smooth approximation is adopted:

$$\mathcal{P}(y,\lambda,\mu) = \frac{\lambda^2}{\mu}\phi\left(\frac{\mu}{\lambda}y\right), \tag{20}$$

$$\phi(t) := \begin{cases} \frac{1}{2}t^2 + t, & t \geq -\frac{1}{2} \\ -\frac{1}{4}\log(-2t) - \frac{3}{8} & t < -\frac{1}{2}, \end{cases} \tag{21}$$

which is the same approach adopted by [9].

III. Optimal Reciprocal Collision Avoidance

This section provides an overview of the ORCA [15] algorithm and its potential benefits. Collision avoidance is a fundamental problem in robotics that must be addressed for UAM vehicles to operate safely in crowded aerial environments. ORCA approaches the problem by viewing potential vehicles and stationary objects in terms of velocity obstacles, VO. ORCA's formulation enables the real-time collision detection calculation of multiple obstacles (both stationary and moving) with respect to a vehicle's current trajectory. Utilizing only the planning vehicle's position p, its current and desired velocity along the trajectory, v_t and v_{des}, respectively, and the estimated positions and velocities of the other objects, ORCA also determines the collision free velocity closest to v_{des} for the vehicle. The ORCA formulation can handle the communication of information to improve velocity selection between the vehicle and other obstacles as well. While communication may be feasible between some aircraft, this work assumes that all obstacles are non-communicative and will not adjust course to avoid collisions. Section V.A and Section VI.A provide a brief overview of the algorithm and the augmentations added for this work. For further interest in the full algorithm, readers are referred to [15]. For clarification on velocity objects, they are referred to [16].

Using point-mass dynamics, the algorithm checks along the trajectories of each obstacle based on its expected own-ship velocity and direction at each timestep. If a collision is predicted, it compares the trajectory of each obstacle with the trajectory of the vehicle and plans a set of permitted velocities that avoid the obstacle. The results of these calculations are half-planes in the velocity space between the vehicle and each obstacle. These half-planes are combined to produce a convex hull of permitted velocities, which are compared to a preferred velocity to find the new optimal velocity. The primary benefit of this algorithm is its speed relative to the number of obstacles that it plans around.

IV. Bernstein Polynomials

This work uses polynomials to represent vehicle trajectories. The trajectory, evaluated at a specific time in a finite time interval, $t \in [t_0, t_1]$, is denoted $\zeta(t)$. Without loss of generality, the time interval can be normalized by using the map $t' = {}^{(t-t_0)}/_{(t_1-t_0)}$ so that $t' \in [0,1]$. The polynomial is represented in Bernstein form using polynomial coefficients, c_k, and evaluated at a specific time with the equation:

$$\zeta(t') = \sum_{k=0}^{n} c_k b_k^n(t'), \tag{22}$$

where c_k is the k^{th} coefficient and $b_k^n(t')$ is a Bernstein basis function evaluated at time t'. The Bernstein basis functions are defined by

$$b_k^n(t') := \binom{n}{k}(1-t')^{n-k}t'^k, \quad k = 0,\ldots,n, \tag{23}$$

where $\binom{n}{k}$ is the binomial coefficient.

The coefficients of the polynomial, also called control points, can be freely chosen and completely define the polynomial curve. Figure 1a shows a two-dimensional polynomial evaluated using the Bernstein form, along with the associated control points.

BPs have two properties of primary interest in this work. The first property is that the curves are entirely contained inside the convex hull of the polygon made by their control points. This property enables fast constraint checking algorithms. For a spatial curve, this property can be used to guarantee that the curve does not intersect with obstacles. The second property is the analytic form for the derivatives of a curve. The derivative of an n-th order polynomial is given by the equation:

$$\dot{\zeta}(t') = \sum_{k=0}^{n-1} n\Delta c_k b_k^{n-1}(t), \tag{24}$$

where $\Delta c_k = c_{k+1} - c_k$. This equation means that the derivative of a polynomial in Bernstein form can be quickly evaluated by taking the difference between the original control points and scaling by the order of the curve. With this property, the derivatives of a trajectory can be exactly computed at any point in time, without needing to rely on finite difference approximations. Furthermore, a combination of both properties allows dynamic constraints to be quickly verified. Additional benefits of representing and evaluating polynomials in the Bernstein form are covered at length in [1].

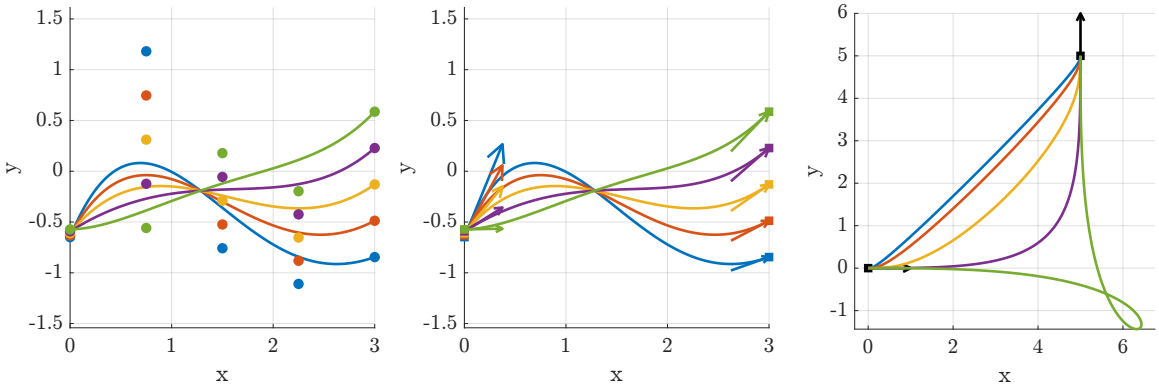
In this work, specific algorithms output trajectory information in terms of the derivatives of a curve. For example, the ORCA algorithm produces a desired position and velocity target. While Eq. (24) can evaluate the derivative of a curve from control points, use in the Bernstein polynomial framework requires a conversion from curve derivatives to control points.

To approach this problem, waypoints are defined for any time in an interval, $t \in [t_0, t_1]$, as two sets of derivatives of an underlying function, $\zeta(t)$. The waypoints are given by the equations: $\mathbf{W_0} = \left[\zeta_0^{(0)}, \ldots, \zeta_0^{(m)}\right]$ and $\mathbf{W_1} = \left[\zeta_1^{(0)}, \ldots, \zeta_1^{(m)}\right]$, where m is the highest order derivative, $\zeta_0 = \zeta(t_0)$ and $\zeta_1 = \zeta(t_1)$. To evaluate the curve at any time, a linear mapping from waypoints to control points is used: $\mathbf{c} = \mathbf{MW}$, where $\mathbf{c}$ is the vector of control points, $\mathbf{W} = [\mathbf{W_0}, \mathbf{W_1}]^\top$, and the mapping for connecting waypoints with derivatives up to order two is given by the equation

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & D/5 & 0 & 0 & 0 & 0 \\ 0 & 0 & D^2/20 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & D/5 & 0 \\ 0 & 0 & 0 & 0 & 0 & D^2/20 \end{bmatrix}, \tag{25}$$

where D is the duration of the interval over which the curve is defined. This mapping can be derived from Equation (24) and allows dynamic specifications of curves to be converted into the Bernstein polynomial framework for use throughout the vehicle architecture. The same two-dimensional curve is shown in Figure 1b. Here the curve is defined with waypoints. The waypoint positions are indicated with colored squares. The waypoint velocity magnitude and direction are indicated with colored arrows.

It is important to note that while BPs are defined with respect to a normalized time interval, the introduction of derivative specifications in the waypoint formulation requires time to be specified as it is used in the construction of the linear mapping, $\mathbf{M}$. Furthermore, adjusting the time specification can greatly change the spatial path of the curve. Figure 1c depicts colored curves that share the same waypoints but have different time intervals. A shorter time interval will lead to direct curves, while longer time intervals will cause the curve to bow out and loiter. For an interactive demonstration of these curves, see [17]



(a) Polynomial in Bernstein form. The circular markers are the control points for the associated polynomials.

(b) Curves connecting waypoints with positions and velocities shown. The curves are evaluated using the Bernstein polynomial framework.

(c) Time dependence of waypoints. Each colored line has a different duration despite connecting the same waypoints.

Fig. 1 Five different two-dimensional polynomial evaluations are shown as colored lines.

V. Algorithm Implementation

The COBRA-DDP algorithm initializes in a similar manner to other formulations of DDP by planning around an initial nominal trajectory. The algorithm can be viewed as the combination of two main components: an ORCA algorithm collision avoidance section and a DDP optimization section. These two components transfer their trajectory information via BP trajectories. To begin, BP waypoints defining a trajectory are transferred to ORCA where collision checking occurs. It is important to note that this initial trajectory can come from a previously optimized DDP iteration or any other planner that wants to check for collision avoidance and is capable of transforming its trajectory into BPs. In the case where there are no obstacles, ORCA does not update the trajectory, and DDP continues to optimize the current trajectory, confident that the trajectory does not violate the collision safety constraints set in ORCA. If ORCA does predict a collision, it will select BP waypoints that form a new simplified trajectory that avoids collisions. This trajectory is then converted into suggested DDP states and controls and optimized to be dynamically feasible. The DDP-optimized trajectory is then converted back in to BP waypoints and confirmed to be collision-free by ORCA. Fig. 2 depicts a general flow chart of the proposed algorithm. Section V.A will discuss the implementation of ORCA and its integration with BPs. The following, Section V.B discusses the combination of DDP with BPs and a full explanation of the algorithm.

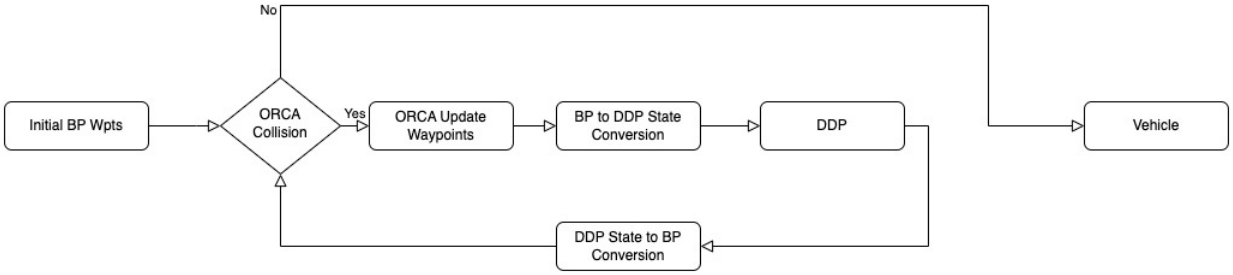


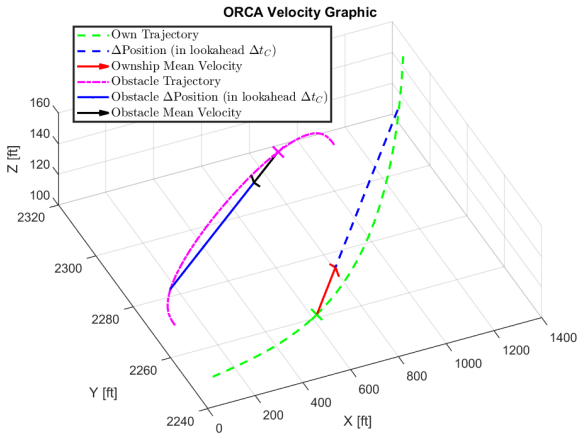
Fig. 2 COBRA-DDP Flow Chart

A. ORCA and Bernstein Polynomial Integration

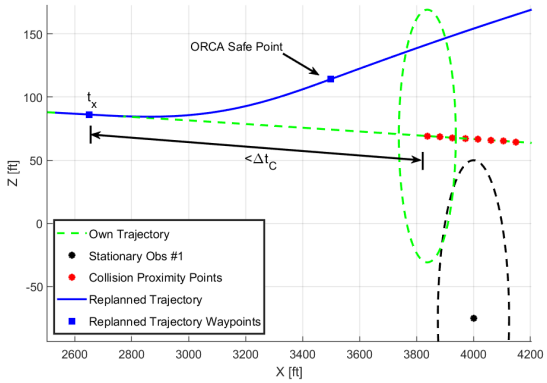
The collision avoidance replanning algorithm, ORCA, is given an own-ship trajectory (expressed as piece-wise Bernstein polynomial curves) and best-estimated obstacle trajectories over the immediate future. The own-ship trajectory is searched sequentially in time (i.e., $t_0, t_0 + \Delta t, ...t_1$) with the ORCA algorithm applied at each search step (with a specified ORCA look-ahead time Δt_C). If a collision with one or more obstacles is detected, then the ORCA-provided own-ship safety assurance velocity is utilized to update the own-ship trajectory with the addition of new waypoints which honor the ORCA position/velocity/time constraints. These concepts are depicted graphically in Fig. 3.

The ORCA algorithm relies on linear velocity approximations for a prescribed ORCA look-ahead time Δt_C. This concept is depicted in Fig. 3a, where portions of the own-ship and obstacle Bernstein polynomial trajectories are shown in the green and magenta lines respectively. In this work, the piece-wise linear velocities required by ORCA are approximated by interpolating the Bernstein polynomial trajectories at the current time and at one ORCA look-ahead time in the future. The corresponding change in position vectors for both of the trajectories are shown with blue lines in Fig. 3a. Using these changes in positions and the ORCA look-ahead time Δt_C, the corresponding mean velocities over Δt_C are computed. They are shown with red and black arrows for own-ship and obstacle respectively. The linearized approximation depiction in Fig. 3a is important because ORCA assurances of collision proximity are predicated on safety spheres for both own-ship and obstacles at every point along their change in position vectors over the entire Δt_C. As shown in Fig. 3a, this linearized velocity approximation does not account for actual Bernstein polynomial trajectory curvature which may potentially violate the ORCA proximity assurances. This subject is discussed further in the following section.

The ORCA algorithm's solution to collision proximity violations is depicted in Fig. 3b. The green dashed line depicts the own-ship Bernstein polynomial trajectory and the associated own-ship safety radius. Note that in this figure (since the Bernstein polynomial trajectory is very nearly linear), the linear approximation of the trajectory depicted in Fig. 3a is omitted. Along the own-ship trajectory at time t_x, the ORCA algorithm provides a continuous assessment up to time $t_x + \Delta t_c$ of contact between the own-ship safety sphere and the obstacle safety sphere (shown in this figure as a



(a) ORCA Velocity



(b) ORCA Step

Fig. 3 ORCA & Bernstein Polynomial Integration

black dashed line around a stationary obstacle). If any safety proximity violation is detected at one or more times t where $t \in [t_x, t_x + \Delta t_C]$, then using a given preferred velocity, v_{des}, input to ORCA, a resulting ORCA safe velocity is provided. Any step along this linearized velocity for any time up to and including $t_x + \Delta t_C$ is guaranteed to satisfy the safety proximity limits. In this work, the ORCA safe velocity is used to generate new waypoints for the Bernstein polynomial trajectory. The first waypoint added to the own-ship Bernstein polynomial is the own-ship point at which the proximity violation was detected (shown at time t_x). Next, an ORCA safe point is created by using the ORCA safe velocity applied at point t_x for the ORCA look-ahead time $t_x + \Delta t_C$. This information provides both the necessary position, velocity, and time (acceleration can be set to zero or some other intelligent choice) necessary to create the corresponding waypoint at the depicted ORCA Safe Point. The resulting altered Bernstein polynomial trajectory is shown in Fig. 3b as the solid blue curve. The ORCA preferred velocity, v_{des}, is an important input to the ORCA algorithm, as in the event of any proximity violations, the ORCA algorithm output is the safe velocity that is nearest to v_{des}. A typical choice for the preferred velocity is the velocity generated by the current own-ship position to the desired endpoint with the time remaining to complete the trajectory. Other choices are valid, and in many cases, preferable for the establishment of flight traffic rules. The ORCA Bernstein polynomial integration pseudo-code is shown in Algorithm 2.

Algorithm 2 is the base integration of ORCA and piece-wise Bernstein polynomial curves. This algorithm is referred to as the non-rate-limited integration since the ORCA recommended velocity (and corresponding position) is implemented without regard for whether the vehicle can perform the desired motion without violating dynamic constraints. In Appendix B, two own-ship velocity vector rate-limited enhancements to the base ORCA Bernstein polynomial algorithm are provided. Both implementations include a Newton step solver for seeking a point earlier in the own-ship trajectory that enforces velocity vector rate limits for changing the own-ship velocity vector. This achieves the ORCA generated safe collision avoidance waypoint based on these added vehicle constraints.

B. DDP and Bernstein Polynomial Integration

DDP is a trajectory optimization algorithm that is capable of planning using highly nonlinear vehicle dynamics. By optimizing over a nominal trajectory, DDP is influenced by the initial conditions relating to the nominal trajectory's states and controls. Therefore, for collision avoidance when replanning to a new trajectory, the new target states and controls provided must be carefully determined to minimize the computation time and the number of iterations necessary to converge around a locally optimal solution. Replanning to avoid collision often requires large shifts in the trajectory, potentially limiting the effectiveness of the original trajectory states and controls. Integrating standard DDP with ORCA via BPs provides the crucial exchange of collision avoidance information while also returning simplified states that can aid DDP in optimizing the newly replanned trajectory.

For this application of DDP, the set of states used for the vehicle are: inertial frame position *(x,y,z)*, body velocities *(u,v,w)*, Euler angles (ϕ, θ, ψ), and body rates *(p,q,r)*. DDP interfaces with BP in two different manners. The first is a

Algorithm 2: ORCA Bernstein Polynomial Main Loop

Input: Own-ship piece-wise BP trajectory, obstacle(s) piece-wise BP trajectory, trajectory search step Δt, ORCA look-ahead time Δt_C where $\Delta t <= \Delta t_C$

Trajectory time to be searched is the intersection of the own-ship and obstacle(s) time sets

Initialize time

`// Main Loop Trajectory Search`

while *not last time* **do**

  `// Compute function and function derivative for current point`

  Interrogate own-ship trajectory at times t_x for position $\mathbf{p_x}$, velocity $\mathbf{v_x}$ and acceleration $\mathbf{a_x}$

  Compute own-ship linearized velocity as: $(\mathbf{p_{\Delta t_C}} - \mathbf{p_{\Delta t}})/\mathbf{\Delta t_C}$

  Interrogate obstacles' trajectories at time Δt, Δt_C for positions $\mathbf{p_{\Delta t}}$, $\mathbf{p_{\Delta t_C}}$

  Compute obstacles' linearized velocities as: $(\mathbf{p_{\Delta t_C}} - \mathbf{p_{\Delta t}})/\mathbf{\Delta t_C}$

  `// Test for proximity violation using ORCA`

  Inputs to ORCA include own-ship and obstacle(s) positions and velocities, Δt_C, preferred velocity, and safety sphere radii (optional additional radius buffer)

  if *proximity violation* **then**

    Create first new BP waypoints at current time using: $\mathbf{p_{\Delta t}}$, $\mathbf{v_{\Delta t}}$ and make assumption on acceleration (e.g., set to zero)

    Given ORCA safe velocity $\mathbf{v_s}$, compute ORCA safe position as $\mathbf{p_{safe}} = \mathbf{p_{\Delta t}} + \mathbf{v_s \Delta t_C}$

    Create second new waypoint at time $t + \Delta t_C$ using $\mathbf{p_{safe}}$, $\mathbf{v_s}$ and make acceleration assumption (e.g., set to zero)

    Update time: $t = t + \Delta t$ (Take trajectory search step along the new safe segment which provides some protection against curvature induced proximity violation. Alternatively, could take entire ORCA look-ahead time step: $t = t + \Delta t_C$)

  else

    Update time using trajectory search step $t = t + \Delta t$

return *Replanned own-ship BP trajectory*

translation of the DDP trajectory into BP waypoints, pertinent states along the DDP trajectory must be converted. The implemented BP waypoints each require inertial frame position, inertial frame velocity, inertial frame acceleration, and time at each waypoint. DDP directly provides the position values and can produce the inertial frame velocities by a rotation from body frame to inertial frame. The accelerations are calculated by dividing the change in velocity by the amount of time between timesteps Δt. As such, no added information is required to provide ORCA with the necessary BP waypoints for collision avoidance detection.

Once ORCA has calculated a new trajectory, it is stored as BP waypoints and must be converted into full states and controls for DDP to optimize as an initial trajectory. The resultant BP trajectory is sampled, $\forall t, t_0 \leq t \leq T$, discretized by the timestep Δt used in DDP. Fortunately, BPs are extremely quick to sample from. This provides an exact conversion for the parameterized position, $\zeta_{x,y,z}$, and velocity, ζ_{v_x,v_y,v_z}, curves of the trajectory. The positions can be immediately incorporated into the state, while the inertial frame velocities must be rotated back into the body frame for the DDP optimization. To receive more information from the curve, an analytic conversion must occur. ζ_{v_x,v_y,v_z} is available for all timesteps and converted into ground track, χ, and body velocities, (u, v, w), based on the assumption that during flight, $\chi = \text{atan2}(v_y, v_x)$. χ coincides with heading angle, ψ, given the assumed orientation of the vehicle and is added as a state for DDP. The remaining states and controls can be intelligently selected or estimated based on the information provided by the BP trajectory. For example, the experiments in Section VI utilize a trim table to fill in the remaining states and controls. The values are linearly interpolated from a set of trim conditions for the vehicle based on the heading frame forward and vertical velocities. This provides DDP with the remaining states needed for the nominal trajectory. It then optimizes a new trajectory based on this nominal trajectory to avoid the potential collision. The newly planned trajectory can be quickly checked by ORCA for collision avoidance by converting the DDP trajectory back into the BP waypoints. A pseudocode explanation of the COBRA-DDP algorithm is provided in Algorithm 3.

While inaccurate for providing an exact dynamically-feasible collision avoidance trajectory, the ORCA-suggested trajectory serves as a strong nominal trajectory for DDP to begin optimizing over. The overall goal of DDP is to calculate a feasible trajectory that avoids collision based on the target states and controls provided by the ORCA trajectory,

Algorithm 3: COBRA-DDP Algorithm

Input: Own-ship BP waypoints, Obstacle(s) piece-wise BP trajectory, DDP timestep, Δt_{DDP}, trajectory search step Δt_{ORCA}, ORCA look-ahead time Δt_C where $\Delta t \leq \Delta t_C$

Own-ship BP waypoints can be translated from previous DDP states or a predetermined trajectory

while *not collision free* **do**

  Replanned own-ship BP trajectory = ORCA Bernstein Polynomial Main Loop(Own-ship BP waypoints, Obstacle(s) piece-wise BP trajectory, trajectory search step Δt_{ORCA}, ORCA look-ahead time Δt_C)

  if *collision detected* **then**

    $\bar{\mathbf{x}}_t, \bar{\mathbf{u}}_t$ = Conversion to DDP states and controls (Replanned own-ship BP trajectory)

    optimal state and controls $\mathbf{x}_t, \mathbf{u}_t$ = DDP($\bar{\mathbf{x}}_t, \bar{\mathbf{u}}_t$)

    Own-ship BP waypoints = Conversion to BP trajectory($\mathbf{x}_t, \mathbf{u}_t$)

  else

    collision free

return $\mathbf{x}_t$, $\mathbf{u}_t$

not follow it exactly. This approach enables the benefit of ORCA's fast collision avoidance detection to determine a suitable suggested trajectory for DDP to optimize over and converge upon quickly. The ability for ORCA to determine collision-avoiding trajectories based on a specified desired velocity, v_{des}, allows for DDP to plan feasible and locally optimal trajectories that can be based on flight traffic rules or general flight guidance. This idea is explored further in Section VI.

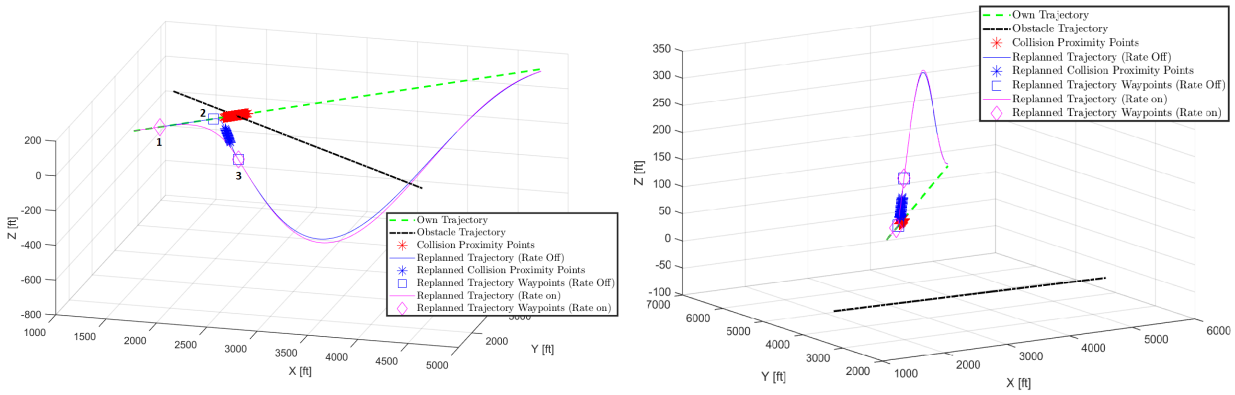
VI. Experiments and Results

For the experiments, the authors applied the above algorithms to NASA's Revolutionary Vertical Lift Technologies (RVLT) Lift+Cruise (L+C) UAM vehicle [18]. UAM VTOL vehicles typically have highly nonlinear dynamics and effectively showcase the benefits provided by both DDP and its variations. In crowded airspace and other situations where collision avoidance would be implemented for these types of vehicles, narrowly missing the object, or planning trajectories extremely close to the obstacle would be inherently dangerous. Therefore, an assumed safety radius centered around each obstacle is considered when planning vehicle trajectories. For the latter experiments in this work, all safety radii set for COBRA-DDP and AL-DDP are the same with respect to individual experiments. Similarly, both COBRA-DDP and AL-DDP are provided with the same initial trajectory information and target state prior to collision. In all of the following experiments, a North-East-Down (NED) reference frame is assumed. In the plots, the z position and w body velocity components are flipped so that they are more intuitive, i.e. z corresponds to altitude. There is no implementation of ground effects or a lower bound defining the ground in these experiments. The experiments are divided into two subsections: ORCA integration benefits and COBRA-DDP replanning in conjunction with AL-DDP as full trajectory planners.

A. ORCA Bernstein Polynomial Integration Benefits

Results for the ORCA Bernstein polynomial integration replanning algorithms (both non-rate-limited and rate-limited) are depicted in Fig. 4. Fig. 4a depicts a co-altitude pending collision with simple (linear) own-ship trajectory (green) and a co-altitude linear obstacle trajectory in black.

In the Fig. 4a scenario, the ORCA search along the green own-ship trajectory detects pending collision proximity violations (denoted by the red asterisks along the green trajectory) after waypoint 2. Given a preferred velocity towards the own-ship trajectory end waypoint, ORCA prescribes the safe point waypoint 3, that results in a vertical maneuver downward to avoid collision (replanned trajectory shown in the blue solid line). This replanned BP trajectory is curved between waypoints 2 and 3, which violates the ORCA assumption of linear tracking between these waypoints. As a result, proximity violations still occur, as denoted with the blue asterisks. Next, the velocity vector rate-limiting algorithm described in Appendix B is performed to replan for collision avoidance. The resulting magenta trajectory not only satisfies velocity vector rate constraints, it also resolves the replanned trajectory collision proximity issue. This is due to the earlier turn initiated at waypoint 1 (instead of waiting until waypoint 2), which approaches waypoint 3 tangentially. Figure 4b depicts similar trajectory replanning (rate limits off and on) but for a non-co-altitude intersection which results in a pure vertical (upward) maneuver.

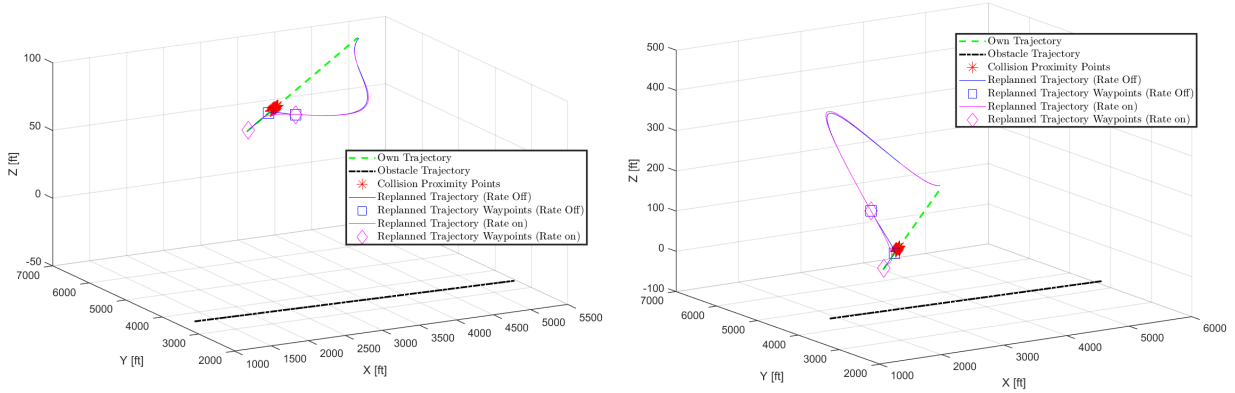


(a) Co-altitude Collision

(b) Vertical Separation Collision

Fig. 4 Rate-Limited and Non-Rate-Limited ORCA Bernstein polynomial

One of the powerful features of the basic ORCA algorithm is the capability to specify the preferred velocity when a collision proximity is detected. This capability is quite important for trajectory replanning (e.g., traffic rules), and a demonstration of this is given in Fig. 5. This figure depicts two alternate preferred velocity selections applied to the non-co-altitude scenario of Fig. 4b. In particular, Figure 5a rotates the ORCA input preferred velocity by 90 degrees to the right (horizontally), and the resulting replanned trajectory is a pure right-hand turn avoidance maneuver. Similarly, Fig. 5b requests a pure left-hand turn, but results in a climbing left-hand avoidance maneuver. This occurs as ORCA outputs the closest velocity to the preferred velocity request that still satisfies collision proximity. Since the obstacle trajectory is to the left at the relevant trajectory time, a pure left-hand avoidance maneuver would not be collision-proximity free.



(a) Right Turn Avoidance

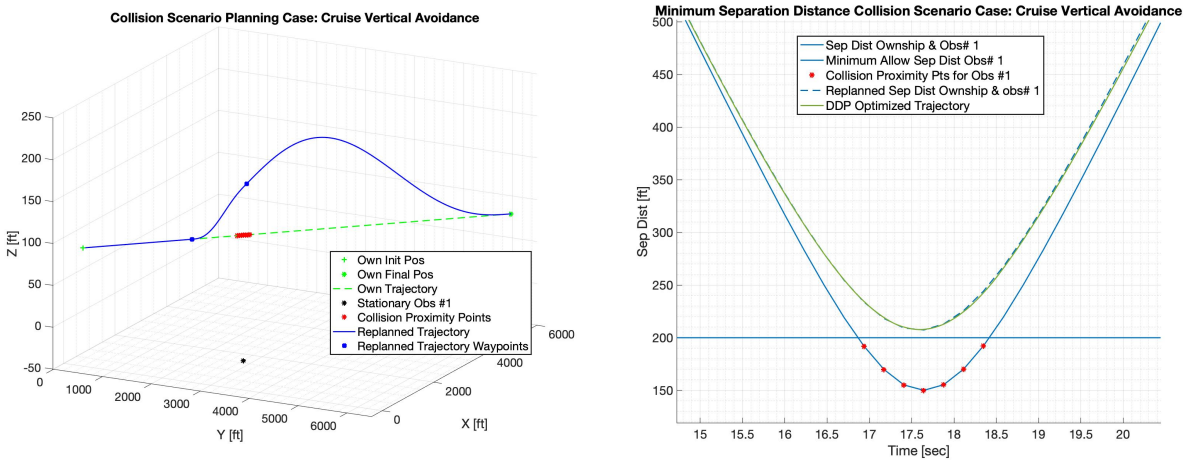
(b) Left Turn Avoidance

Fig. 5 Preferential Avoidance Direction

B. COBRA-DDP and AL-DDP Comparison

To discern the effectiveness of COBRA-DDP's collision avoidance capabilities, COBRA-DDP and AL-DDP were implemented using the L+C vehicle dynamics. Both were presented with two cruising flight cases that would result in a breach of the safety radii via a static object. In the first experiment, the vehicle is in level flight cruise at 170 ft/s with a safety radius of 200 ft when sensors recognize an obstacle 150 ft below the vehicle's trajectory. Fig. 6a depicts the trajectory COBRA-DDP plans via BP waypoints to avoid the obstacle where the original trajectory is in green and the BP trajectory is in blue. The vehicle ascends 100 ft in total to avoid the obstacle before returning to the original altitude and completing the trajectory. Fig. 6b presents the minimum distance of the original, the BP, and the DDP optimized trajectories during the time when they are nearest to the obstacle. From the plot, a portion of the original

trajectory, in solid blue, fails to meet the minimum allowable distance denoted by the red asterisks. Both the replanned ORCA component trajectory, in dashed blue, and the optimized DDP trajectory, in solid green, are able to prevent the proximity breach. Fig. 7 highlights the steps of the algorithm, where the ORCA-suggested trajectory is updated by the DDP optimized trajectory to be dynamically feasible. This comparison of the ORCA-suggested and DDP-optimized trajectories makes it clear that COBRA-DDP was able to effectively optimize to a dynamically feasible trajectory from the simplified suggested trajectory.



(a) Cruise: Vertical Collision Avoidance

(b) Collision Avoidance minimum Distance

Fig. 6 COBRA-DDP Vertical Avoidance

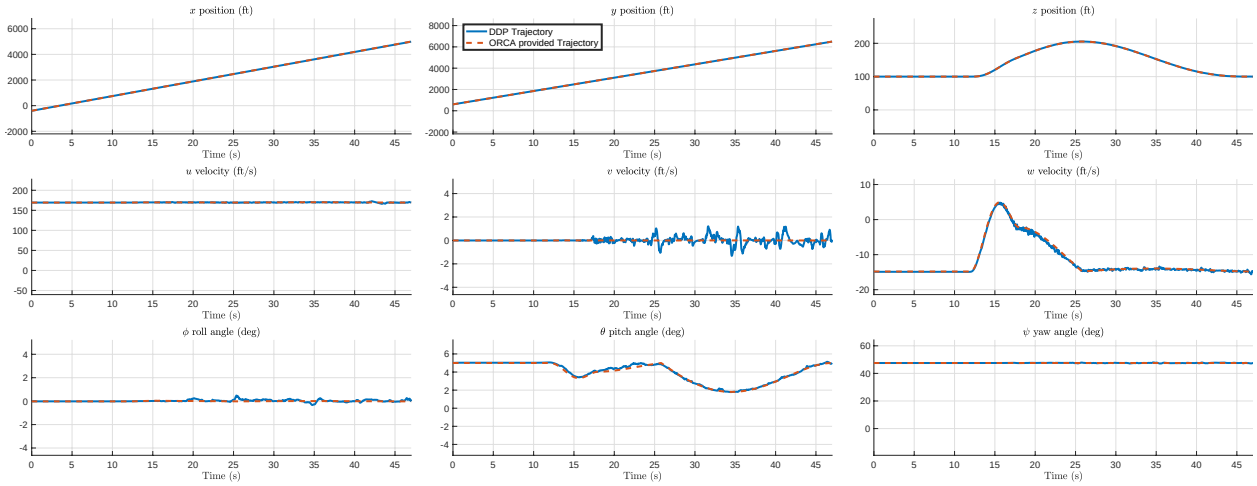
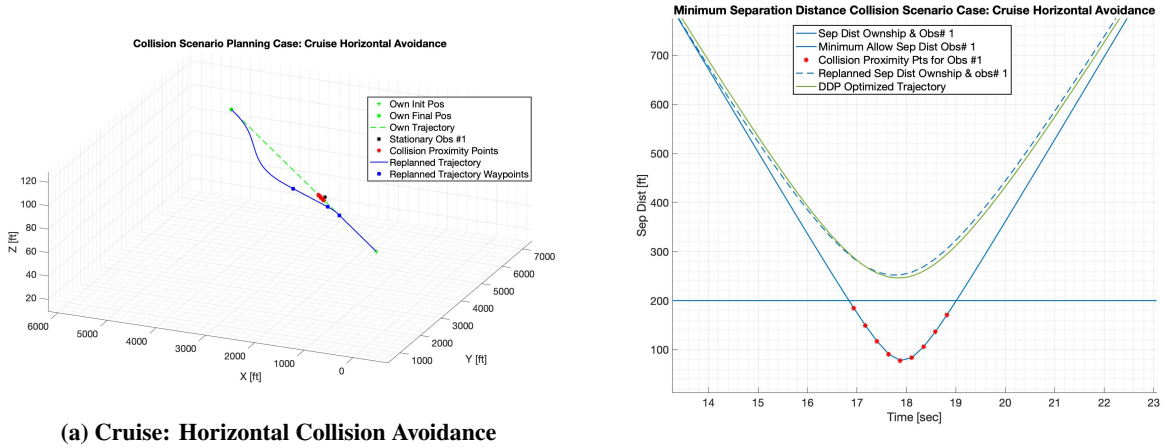


Fig. 7 Vertical Avoidance: DDP and ORCA Trajectory Comparison

The second experiment again presents the vehicle with a planned trajectory cruising at 170 ft/s with a 200 ft safety radius. In this case, the static obstacle is at the same altitude as the vehicle, but to the right of the trajectory such that the vehicle would violate the minimum allowable distance by 125 ft. COBRA-DDP replans a level turning maneuver to avoid the obstacle. The plotted trajectories and minimum separation distance are presented in Fig. 8. Again, the suggested ORCA trajectory and optimized DDP trajectory are able to replan and avoid collision. In this case however, when comparing the trajectory states in Fig. 9, the limitations of a purely forward flight trim table are more visible. While COBRA-DDP is able to closely match the position and velocity values of the simplified ORCA trajectory, adjustments in the roll and yaw angles more noticeably differ. These deviations from the ORCA suggestion are necessary for a dynamically feasible trajectory. Because these state values are pulled directly from a trim table for

forward flight, the values provided do not account for any adjustments in roll or yaw necessary to implement a turn. Despite the simplicity of the trim table however, COBRA-DDP still manages to optimize the suggested trajectory and provide a collision-avoiding trajectory.



(a) Cruise: Horizontal Collision Avoidance

(b) Collision Avoidance minimum Distance

Fig. 8 COBRA-DDP Horizontal Avoidance

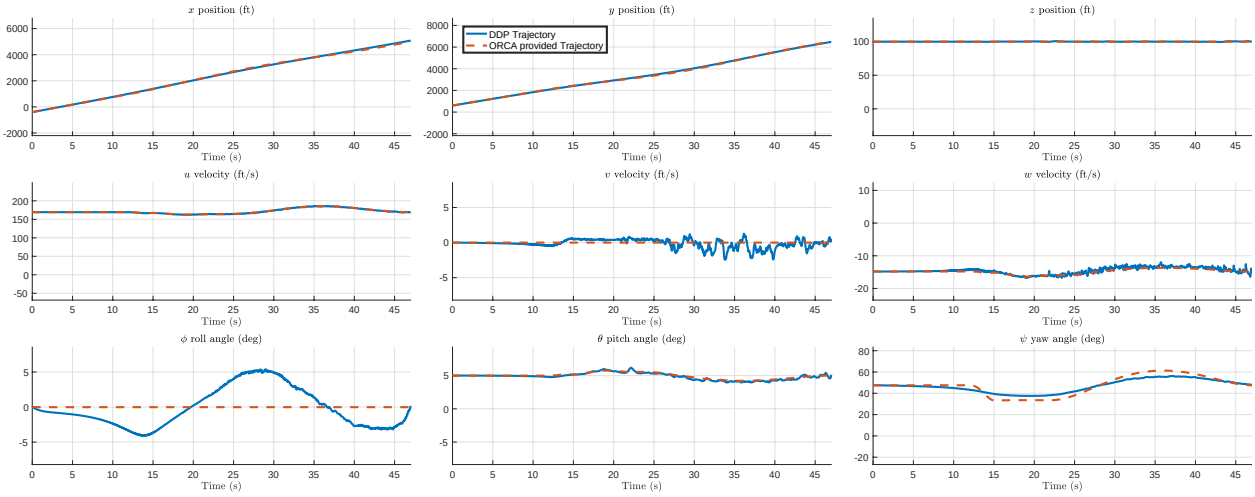


Fig. 9 Horizontal Avoidance: DDP and ORCA Trajectory Comparison

Finally, to check if the COBRA-DDP implementation was capable of improving upon the AL-DDP's collision avoidance capabilities, AL-DDP was given the same two example cases to optimize. Unfortunately, AL-DDP when given the same initial state and controls of level cruise was not able to produce a trajectory that was collision-free. In both cases, regardless of penalty design, AL-DDP was unable to avoid the obstacle. It is hypothesized that AL-DDP was unable to escape the local minimum of the trimmed cruise trajectory with minor changes to the controls. Large changes in the controls, due to the highly nonlinear dynamics of the vehicle, would also not minimize the cost of the overall trajectory, forcing an increase in penalty function cost that prevented convergence. In Fig. 10 the orange line depicts an example trajectory where the vehicle does not turn to avoid the obstacle.

When AL-DDP was provided initial states and control values of zero, the algorithm was able to avoid the obstacles. However, AL-DDP would consistently optimize for a trajectory that avoided the obstacle by flying underneath it. The magenta line in Fig. 10 shows an example case for the turn experiment. While this is not an inherently incorrect trajectory, the inability for AL-DDP to plan a trajectory other than flying below the obstacle poses a concerning limitation for its implementation for UAM VTOL vehicles.

AL-DDP was lastly provided with the COBRA-DDP simplified trajectory as initial states and controls, to warm-start the optimization. It was able to optimize for a feasible trajectory similarly to the unconstrained DDP approach in COBRA-DDP. The blue line in Fig. 10 shows the resulting warm-started AL-DDP trajectory in comparison to the other warm-start methods. This makes intuitive sense, as once AL-DDP confirms that the state constraints are satisfied, the inner loop implements unconstrained DDP to optimize the controls. Fig. 11 depicts similar results for the altitude changing collision avoidance example. While AL-DDP is able to directly consider the state constraints during optimization, COBRA-DDP is able to check the unconstrained DDP trajectory due to the speed of ORCA and BP. COBRA-DDP is also able to adjust the direction of the obstacle avoidance via the desired velocity value, an advantage for obeying standardized flight rules in a potentially crowded aerial environment.

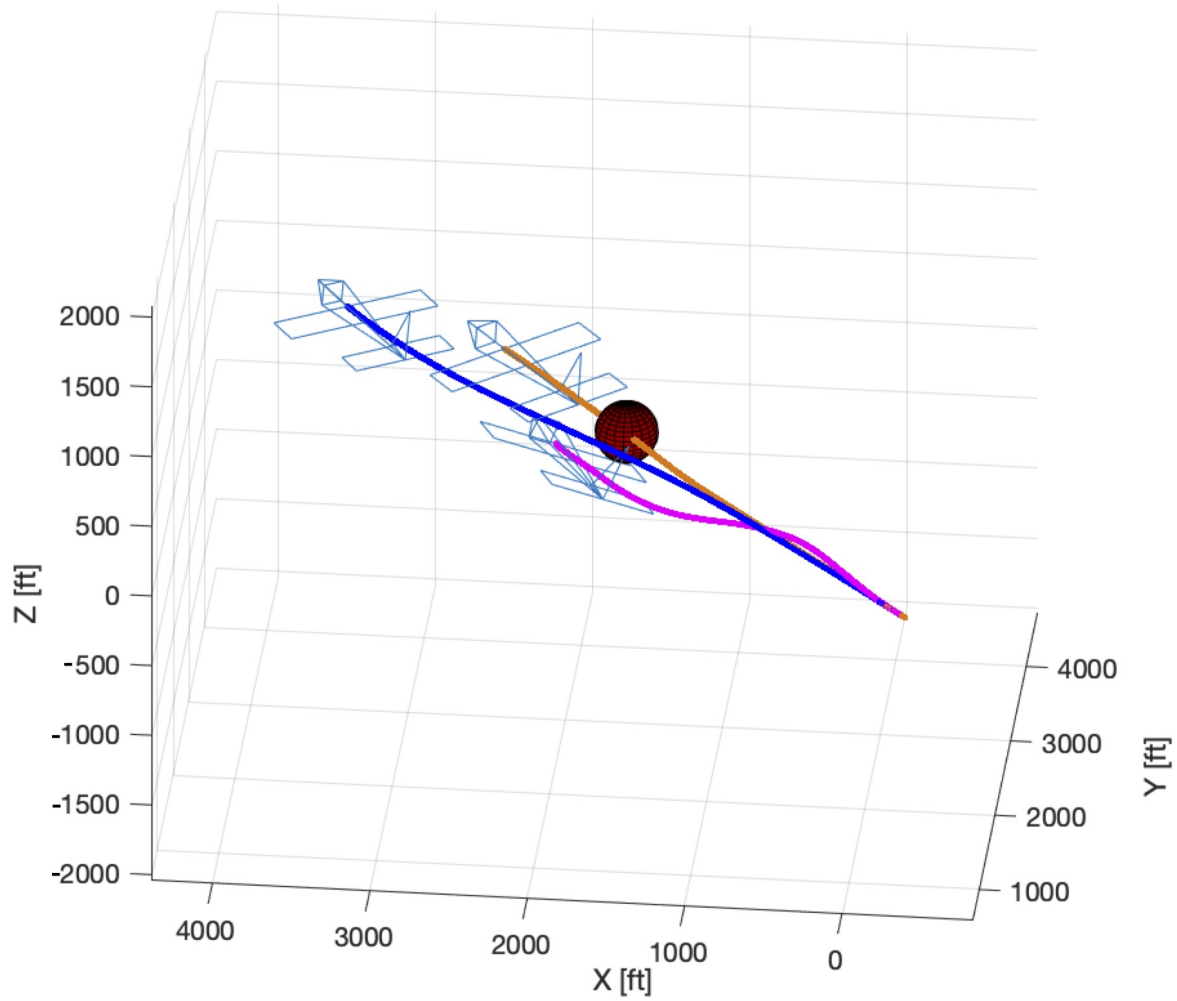


Fig. 10 AL-DDP cruise turn with various warm-start conditions:
Orange: initial cruise trajectory warm-start
Magenta: no warm-start information
Blue: COBRA-DDP warm-start trajectory

VII. Discussion and Future Work

COBRA-DDP provides the capability for UAM VTOL vehicles to plan full trajectories around both stationary and moving obstacles by considering their trajectories. In this paper, it has demonstrated the capability to select and plan collision avoidance trajectories that are dynamically feasible. It has also been shown to provide useful warm-start states

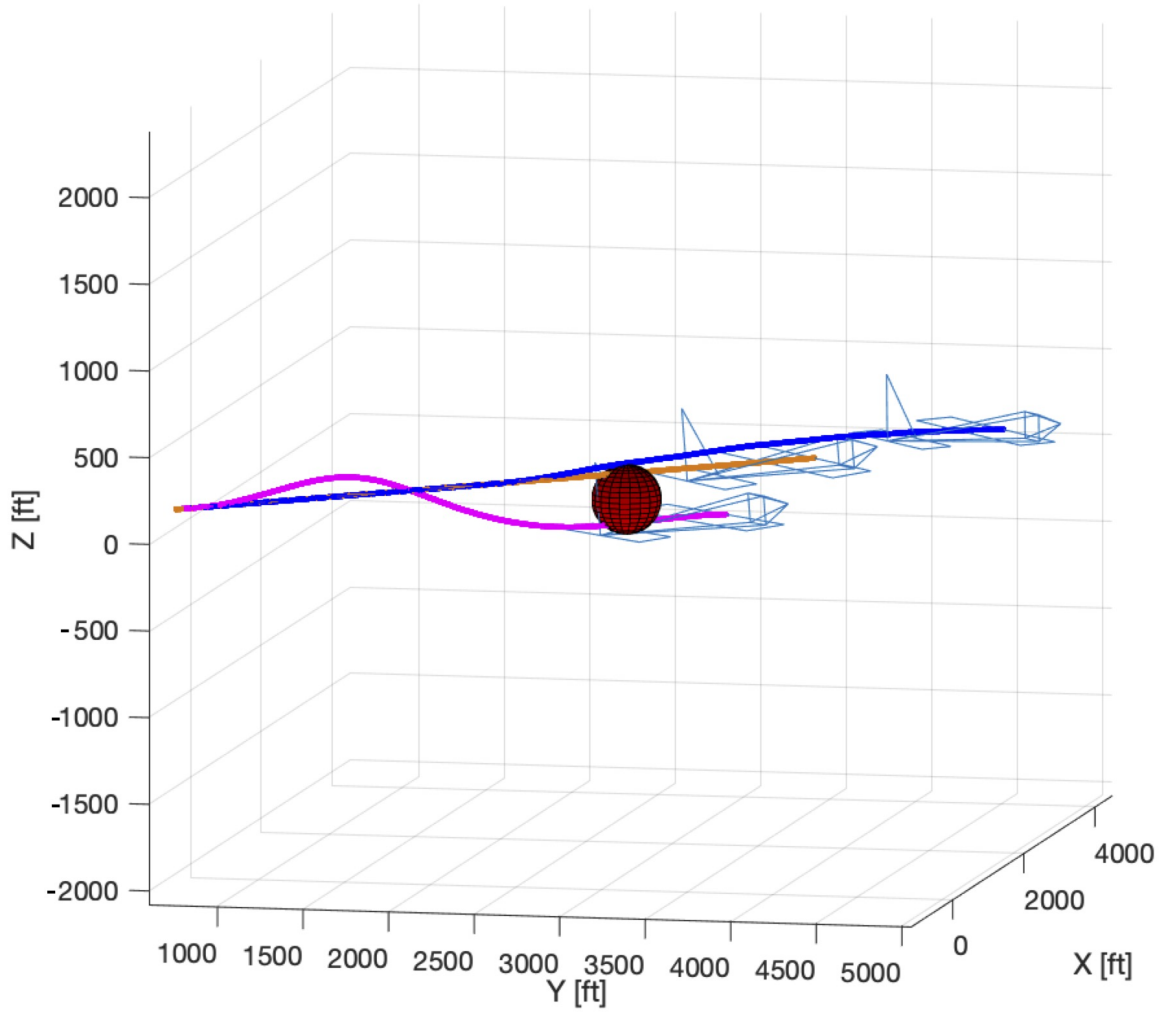


Fig. 11 AL-DDP cruise altitude change with various warm-start conditions:
Orange: initial cruise trajectory warm-start
Magenta: no warm-start information
Blue: COBRA-DDP warm-start trajectory

and controls for state-constrained DDP variants such as AL-DDP. While the case can always be made for more cost function tuning to improve AL-DDP's capability to plan using state constraints, COBRA-DDP utilizes the advantages of ORCA, BPs, and DDP to improve the recognition and optimization of dynamically feasible collision-avoiding trajectories.

Future research efforts would involve the implementation of COBRA-DDP in a model predictive control receding horizon architecture, as well as demonstrating COBRA-DDP's advantage when considering trajectories involving multiple objects. Efforts to augment the ORCA algorithm with a neural network or simplified algorithm to recognize when other vehicles are turning (or performing other specific maneuvers) may help to better advise the costs and warm-started trajectory of the integrated algorithm. Similarly, incorporating more information regarding vehicle turning capabilities and other maneuvers will also help to generate suggested trajectories that minimize the computational cost of DDP. The goal of these formulations and improvements would be to utilize COBRA-DDP for real-time planning and provide realistic separation assurances during flight in congested airspace.

Acknowledgments

This research was supported by the NASA Aeronautics Research Mission Directorate (ARMD), Transformative Tools and Technologies (TTT) project, under the Autonomous Systems / Intelligent Contingency Management subproject.

Appendix

A. Differential Dynamic Programming Further Explanation

Due to the nonlinearities of the dynamics and inaccuracies in the quadratic approximation, applying the control update in Eq. (9) may increase the cost. To resolve this, a line search is performed on the feedforward gain $\mathbf{k}_i$ to ensure that the cost is reduced at every iteration:

$$\delta \mathbf{u}_i^* = \epsilon \mathbf{k}_i + \mathbf{K}_i \delta \mathbf{x}_i. \tag{26}$$

The line search parameter $0 < \epsilon \leq 1$ is initialized to 1 and iteratively reduced if the new trajectory does not reduce the cost. Additionally, regularization is employed to help in the convergence of the algorithm [19]:

$$Q_i^{xx} = \mathcal{L}_i^{xx} + (\mathbf{f}_i^x)^\top (V_{i+1}^{xx} + \mu_1 \mathbf{I}) \mathbf{f}_i^x, \tag{27a}$$
$$Q_i^{xu} = \mathcal{L}_i^{xu} + (\mathbf{f}_i^x)^\top (V_{i+1}^{xx} + \mu_1 \mathbf{I}) \mathbf{f}_i^u, \tag{27b}$$
$$Q_i^{uu} = \mathcal{L}_i^{uu} + (\mathbf{f}_i^u)^\top (V_{i+1}^{xx} + \mu_1 \mathbf{I}) \mathbf{f}_i^u + \mu_2 \mathbf{I}. \tag{27c}$$

The parameter μ_1 can be interpreted as adding a quadratic cost so that the new trajectory stays near the nominal (where the quadratic approximation is accurate), while μ_2 ensures that Q_i^{uu} is positive definite and thus invertible. These modified derivatives are used to calculate the feedforward and feedback gains in Eq. (9). However, the same cancellations of Q_i^{uu} and its inverse are not possible in Eq. (11), so the new value function derivatives are given by

$$V_i^x = Q_i^x + \mathbf{K}_i^\top Q_i^{uu} \mathbf{k}_i + \mathbf{K}_i^\top Q_i^u + (Q_i^{ux})^\top \mathbf{k}_i, \tag{28a}$$
$$V_i^{xx} = Q_i^{xx} + \mathbf{K}_i^\top Q_i^{uu} \mathbf{K}_i + \mathbf{K}_i^\top Q_i^{ux} + (Q_i^{ux})^\top \mathbf{K}_i. \tag{28b}$$

B. Velocity Vector Rate Limiting Algorithm

This appendix details a variant of the basic ORCA Bernstein Polynomial Integration algorithm that enables enforcement of a basic vehicle dynamic limitation (velocity vector rate) into the trajectory replanning process. The variant is vehicle agnostic in that no vehicle dynamic properties are required to be known *a priori*. Instead, the modified algorithm enforces a user-provided angular rate limitation only when a proximity violation is observed. This limitation is implemented by applying the real-world principle that starting an avoidance maneuver earlier typically leads to a more benign maneuver when the desired end state is held constant. The algorithm variant and this principle are graphically depicted in Fig. 12.

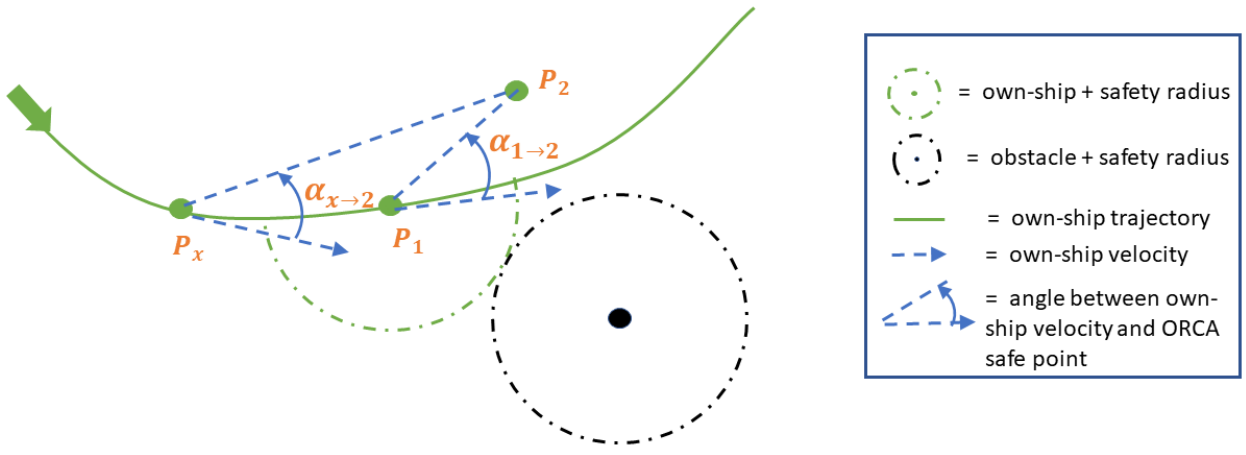


Fig. 12 Velocity Vector Rate Limitation Search Algorithm

Figure 12 depicts an arbitrary own-ship Bernstein polynomial trajectory (green solid line), own-ship safety radius (green dashed semicircle) and a single stationary obstacle with its corresponding safety radius (dashed black line).

The nominal integration of ORCA and BP finds the point P_1 along the trajectory for which a proximity violation occurs (within a prescribed ORCA look-ahead time) and updates the own-ship Bernstein polynomial trajectory with additional waypoints P_1 and the ORCA generated safe point P_2. The angle $\alpha_{1\to 2}$ is the angle between the velocity vector of own-ship at P_1 of the original Bernstein polynomial trajectory and the position of the ORCA safe point P_2. Additionally, a nominal corresponding velocity vector rate can be found as approximately $\alpha_{1\to 2}/(t_2 - t_1)$ where t_1, t_2 are the trajectory times at P_1 and P_2 respectively. The magnitude of this angle and the corresponding velocity vector rate is greatly impacted by the relative velocities (own-ship and obstacles), the various safety radii, and substantially by the ORCA look-ahead time Δt_C. If the nominal velocity vector rate (computed above) is greater than a prescribed body-rate limitation, then a new trajectory point P_x is sought for to replace P_1 such that the new angular rate $\alpha_{x\to 2}/(t_2 - t_x)$ is identically the prescribed velocity vector rate limit. While the angle $\alpha_{x\to 2}$ is highly dependent on the particular Bernstein polynomial curve under examination, in general choosing a t_x earlier in the trajectory and thus increasing the magnitude of the nominal velocity vector rate equation denominator, generally lowers the body-rate.

The algorithm variation was generated by expressing the geometric relationship and then setting up a resulting analytic equality in t_x. The derivative of this function in t_x is then used to perform a Newton step iteration to satisfy the equality. First, from the geometric relationship, the equality is defined as:

$$tan\left(q_{max}\left(t_2 - t_x\right)\right) = \frac{\|l_\perp\|_2}{\|l_\parallel\|_2} \implies \tag{29}$$

$$0 = f\left(t_x\right) = tan^2\left(q_{max}\left(t_2 - t_x\right)\right) {l_\parallel}^T l_\parallel - {l_\perp}^T l_\perp \tag{30}$$

where q_{max} is the desired velocity vector rate limit, t_2 is the time at the ORCA safe point, $l_\parallel$ is the parallel component of the projection of the position vector $\mathbf{p_{x\to 2}}$ onto the local velocity $\mathbf{v_x}$, and $l_\perp$ is the component of $\mathbf{p_{x\to 2}}$ perpendicular to $\mathbf{v_x}$, and t_x is the sought trajectory time. Now define the projection operator of an arbitrary vector $\mathbf{u}$ onto the local velocity $\mathbf{v_x}$ as:

$$\mathcal{P}_{\mathbf{v_x}}\left(\mathbf{u}\right) = \mathbf{v_x}\left(\mathbf{v_x}^\mathbf{T}\mathbf{v_x}\right)^{-1}\mathbf{v_x}^\mathbf{T}\left(\mathbf{u}\right) \tag{31}$$

So that

$$\mathbf{l}_\parallel = \mathbf{v_x}\left(\mathbf{v_x}^\mathbf{T}\mathbf{v_x}\right)^{-1}\mathbf{v_x}^\mathbf{T}\mathbf{p_{x\to 2}}, \quad \text{where } \mathbf{p_{x\to 2}} = \mathbf{p_2} - \mathbf{p_x} \tag{32}$$

$$\mathbf{l}_\perp = \left(I - \mathbf{v_x}\left(\mathbf{v_x}^\mathbf{T}\mathbf{v_x}\right)^{-1}\mathbf{v_x}^\mathbf{T}\right)\mathbf{p_{x\to 2}} \tag{33}$$

The next step is to obtain the analytic derivative of Eq. (30), so examining the derivatives of each of the terms results in:

$$\frac{d}{dt}tan^2\left(q_{max}\left(t_2 - t_x\right)\right) = \frac{-2q_{max}tan\left(q_{max}\left(t_2 - t_x\right)\right)}{cos^2\left(q_{max}\left(t_2 - t_x\right)\right)}, \tag{34}$$

$$\frac{d}{dt}l_\parallel^T l_\parallel = 2l_\parallel^T\frac{d}{dt}l_\parallel, \tag{35}$$

$$\frac{d}{dt}l_\perp^T l_\perp = 2l_\perp^T\frac{d}{dt}l_\perp. \tag{36}$$

$$\tag{37}$$

Now using the piece-wise Bernstein polynomial curve one can obtain the own-ship quantities:

$$\mathbf{p_x} = \text{ position vector at time } t_x \tag{38}$$

$$\mathbf{v_x} = \text{velocity vector at time } t_x \tag{39}$$

$$\mathbf{a_x} = \text{acceleration vector at time } t_x \tag{40}$$

which together with Eq. (32) obtains

$$\frac{d}{dt}\mathbf{l}_\parallel = \mathbf{a_x}\left(\mathbf{v_x^T}\mathbf{v_x}\right)^{-1}\mathbf{v_x^T}\left(\mathbf{p_2} - \mathbf{p_x}\right) - \mathbf{2v_x}\left(\mathbf{v_x^T}\mathbf{v_x}\right)^{-2}\mathbf{v_x^T}\mathbf{a_x}\mathbf{v_x^T}\left(\mathbf{p_2} - \mathbf{p_x}\right) + \mathbf{v_x}\left(\mathbf{v_x^T}\mathbf{v_x}\right)^{-1}\mathbf{a_x^T}\left(\mathbf{p_2} - \mathbf{p_x}\right) - \mathbf{v_x}. \tag{41}$$

Now Eq.(33) together with Eq.(41) shows that

$$\begin{aligned}\frac{d}{dt}\mathbf{l}_\perp &= -\mathbf{a_x}\left(\mathbf{v_x^T}\mathbf{v_x}\right)^{-1}\mathbf{v_x^T}\left(\mathbf{p_2} - \mathbf{p_x}\right) + \mathbf{2v_x}\left(\mathbf{v_x^T}\mathbf{v_x}\right)^{-2}\mathbf{v_x^T}\mathbf{a_x}\mathbf{v_x^T}\left(\mathbf{p_2} - \mathbf{p_x}\right) - \mathbf{v_x}\left(\mathbf{v_x^T}\mathbf{v_x}\right)^{-1}\mathbf{a_x^T}\left(\mathbf{p_2} - \mathbf{p_x}\right) \\ &= -\frac{d}{dt}\mathbf{l}_\parallel - \mathbf{v_x}.\end{aligned} \tag{42}$$

So the derivative of the function in Eq.(30) is found to be

$$\frac{d}{dt}f\left(t\right)=\frac{-2tan\left(\Theta\right)}{cos^{2}\left(\Theta\right)}\mathbf{l}_{\parallel}^{\mathbf{T}}\mathbf{l}_{\parallel}+\mathbf{2tan^{2}}\left(\mathbf{\Theta}\right)\mathbf{l}_{\parallel}^{\mathbf{T}}\frac{\mathbf{d}}{\mathbf{dt}}\mathbf{l}_{\parallel}-\mathbf{2}\left(\mathbf{p_{2}p_{x}}-\mathbf{l}_{\parallel}\right)^{\mathbf{T}}\left(-\frac{\mathbf{d}}{\mathbf{dt}}\mathbf{l}_{\parallel}-\mathbf{v_{x}}\right)\text{ where }\mathbf{\Theta}=\mathbf{q_{max}}\left(\mathbf{t_{2}}-\mathbf{t_{x}}\right)\tag{43}$$

A pseudo-code algorithm for seeking the time (position) on the own-ship trajectory that satisfies the velocity vector rate limit is given in Algorithm 4. In particular, this algorithm returns the Bernstein polynomial waypoints $\mathbf{p_1}$ and $\mathbf{p_2}$ if the maximum velocity vector rate $\dot{\alpha}_{max}$ is satisfied. If the velocity vector rate limit is exceeded, then the algorithm seeks the position on the own-ship trajectory $\mathbf{p_x}$ for which the maximum velocity vector rate is identically satisfied. Then the Bernstein polynomial trajectory is updated with the dynamically constrained waypoints $\mathbf{p_x}$ and $\mathbf{p_2}$.

Algorithm 4: Velocity Vector Rate Limiting

Input: Own-ship piece-wise Bernstein polynomial trajectory, obstacle(s) piece-wise Bernstein polynomial trajectory, collision proximity detection point $\mathbf{p_1}$ and $\mathbf{v_1}$, ORCA generated safe point $\mathbf{p_2}$ and $\mathbf{v_2}$ and desired max velocity vector rate $\dot{\alpha}_{max}$

Compute angle α_1 between position vector $\mathbf{p_{1\to 2}}$ and velocity vector $\mathbf{v_1}$: $\alpha_1=\cos^{-1}\left(\frac{\mathbf{p_{1\to 2}}\circ\mathbf{v_1}}{\|\mathbf{p_{1\to 2}}\|_2\|\mathbf{v_1}\|_2}\right)$

`// NOTE above angle assumes rotation of velocity vector from current velocity` $\mathbf{v_1}$ `to velocity vector pointing at` $\mathbf{p_2}$`.`

Compute actual (mean) velocity vector rate $\dot{\alpha_1}=\alpha_1/(t_2-t_1)$

Initialize $\mathbf{p_x}=\mathbf{p_1}$, $t_x=t_1$, and prescribe velocity vector rate limit convergence tolerance

if *actual velocity vector rate not less than max rate* **then**

  while *not converged* **do**

    Interrogate own-ship trajectory for $\mathbf{p_x}$, $\mathbf{v_x}$, and $\mathbf{a_x}$

    `// Compute quantities for` $f\left(t_x\right)$ `Eq.30 and` $\frac{d}{dt}f\left(t_x\right)$ `Eq.43`

    Compute $\mathbf{l}_{\parallel}$ using Eq. 32

    Compute $\mathbf{l}_{\perp}$ using Eq. 33

    Compute $\frac{d}{dt}\mathbf{l}_{\parallel}$ using Eq. 41

    Compute $\frac{d}{dt}\mathbf{l}_{\perp}$ using Eq. 42

    Solve for $f\left(t_x\right)$ and $\frac{d}{dt}f\left(t_x\right)$

    Compute angle α_x between position vector $\mathbf{p_{x\to 2}}$ and velocity vector $\mathbf{v_x}$: $\alpha_x=\cos^{-1}\left(\frac{\mathbf{p_{x\to 2}}\circ\mathbf{v_x}}{\|\mathbf{p_{x\to 2}}\|_2\|\mathbf{v_x}\|_2}\right)$

    `// Update time using Newton step`

    $t_{x+}=t_x-\frac{\lambda}{d/dtf\left(t_x\right)}f\left(t_x\right)$, where λ is step scale factor

    Compute updated actual (mean) velocity vector rate $\dot{\alpha_x}=\alpha_x/(t_2-t_x)$

return *Update Bernstein polynomial trajectory with waypoints:* $\mathbf{p_x}$, $\mathbf{v_x}$, $\mathbf{a_x}=\mathbf{0}$ *and* $\mathbf{p_2}$, $\mathbf{v_2}$, $\mathbf{a_2}=\mathbf{0}$

Note, an alternative formulation of Algorithm 4 is available that uses the angle between $\mathbf{v_x}$ and $\mathbf{v_2}$. This formulation is more applicable in some situations, as it compares angles between own-ship velocity along the safe trajectory and the ORCA safe prescribed velocity $\mathbf{v_2}$.

References

[1] Farouki, R. T., "The Bernstein Polynomial Basis: A Centennial Retrospective," *Computer Aided Geometric Design*, Vol. 29, No. 6, 2012, pp. 379–419.

[2] Lakshmanan, A., Patterson, A., Cichella, V., and Hovakimyan, N., "Proximity Queries for Absolutely Continuous Parametric Curves," *Robotics: Science and Systems XV*, Freiburg im Breisgau, Germany, 2019.

[3] Kaminer, I., Pascoal, M. A., Xargay, E., Hovakimyan, N., Cichella, V., and Dobrokhodov, V., *Time-Critical Cooperative Control of Autonomous Air Vehicles.*, 1st ed., Butterworth-Heinemann, Jordan Hill, Oxford, UK, 2017.

[4] Houghton, M. D., Oshin, A. B., Acheson, M. J., Theodorou, E. A., and Gregory, I. M., "Path Planning: Differential Dynamic Programming and Model Predictive Path Integral Control on VTOL Aircraft," *AIAA Scitech Forum*, 2022.

[5] Oshin, A. B., Houghton, M. D., Acheson, M. J., Gregory, I. M., and Theodorou, E. A., "Parameterized Differential Dynamic Programming," *Robotics: Science and Systems*, 2022.

[6] Jacobson, D. H., and Mayne, D. Q., *Differential Dynamic Programming*, American Elsevier Pub. Co., 1970.

[7] Todorov, E., and Li, W., "A Generalized Iterative LQG Method for Locally-optimal Feedback Control of Constrained Nonlinear Stochastic Systems," *Proceedings of the 2005, American Control Conference, 2005.*, IEEE, 2005, pp. 300–306.

[8] Pan, Y., Boutselis, G. I., and Theodorou, E. A., "Efficient Reinforcement Learning via Probabilistic Trajectory Optimization," *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 29, No. 11, 2018, pp. 5459–5474.

[9] Aoyama, Y., Boutselis, G., Patel, A., and Theodorou, E. A., "Constrained Differential Dynamic Programming Revisited," *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 9738–9744.

[10] Van Den Berg, J., Patil, S., and Alterovitz, R., "Motion Planning under Uncertainty using Iterative Local Optimization in Belief Space," *The International Journal of Robotics Research*, Vol. 31, No. 11, 2012, pp. 1263–1278.

[11] Tassa, Y., Mansard, N., and Todorov, E., "Control-Limited Differential Dynamic Programming," *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2014, pp. 1168–1175.

[12] Xie, Z., Liu, C. K., and Hauser, K., "Differential Dynamic Programming with Nonlinear Constraints," *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 695–702.

[13] Boyd, S. P., and Vandenberghe, L., *Convex optimization*, Cambridge university press, 2004.

[14] Birgin, E. G., Castillo, R. A., and Martínez, J. M., "Numerical Comparison of Augmented Lagrangian Algorithms for Nonconvex Problems," *Computational Optimization and Applications*, Vol. 31, No. 1, 2005, pp. 31–55.

[15] van den Berg, J., Guy, S. J., Lin, M., and Manocha, D., "Reciprocal n-body Collision Avoidance," *In Proc. of the IEEE International Conference on Robotics and Automation*, 2010.

[16] Fiorini P, S. Z., "Motion Planning in Dynamic Environments Using Velocity Obstacles," *The International Journal of Robotics Research*, 1998.

[17] Kamermans, M., "A Primer on Bézier Curves," , 2022. Https://pomax.github.io/bezierinfo/.

[18] Silva, C., Johnson, W. R., Solis, E., Patterson, M. D., and Antcliff, K. R., "VTOL Urban Air Mobility Concept Vehicles for Technology Development," *Aviation Technology, Integration, and Operations Conference*, 2018.

[19] Tassa, Y., Erez, T., and Todorov, E., "Synthesis and Stabilization of Complex Behaviors through Online Trajectory Optimization," *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012, pp. 4906–4913.